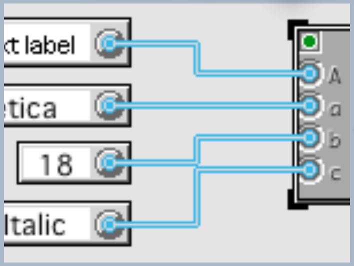
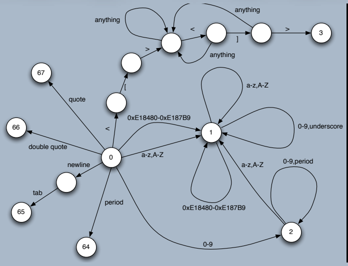
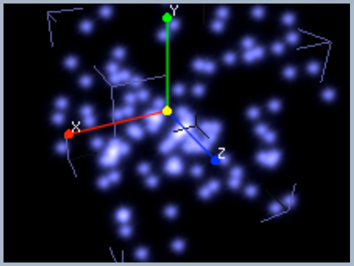
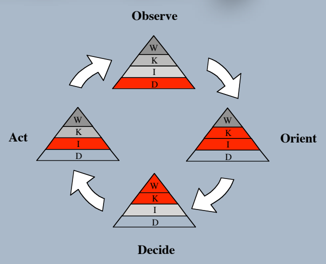
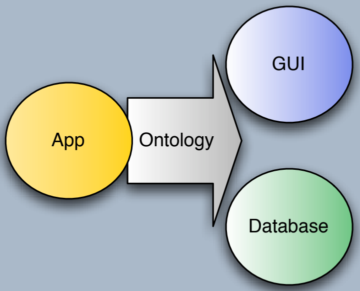
Mitopia® Technical Library

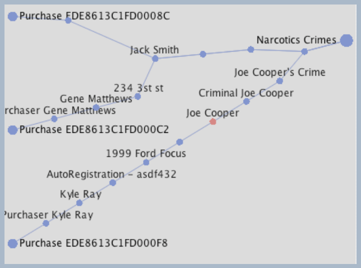


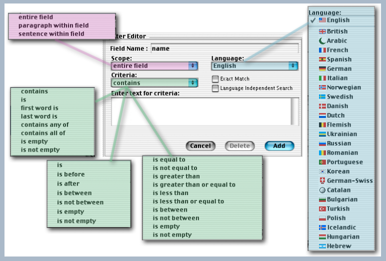


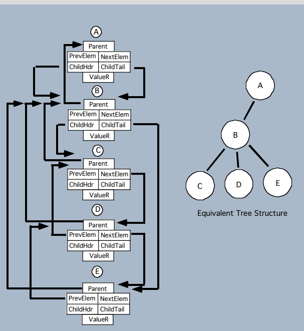








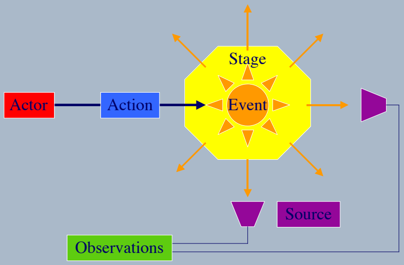


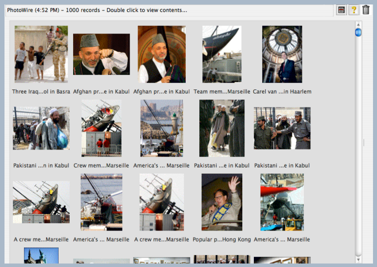


The Carmot Ontology Definition Language

Rev. 1.3

January 2012

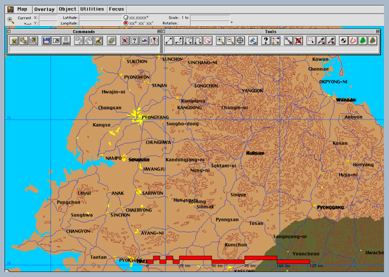


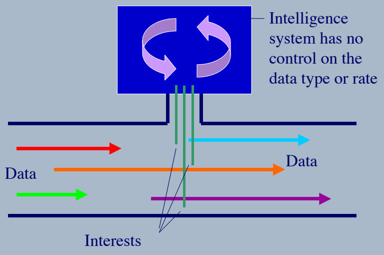

PhotoWire (4:52 PM) ~ 1000 records ~ Double click to view contents...

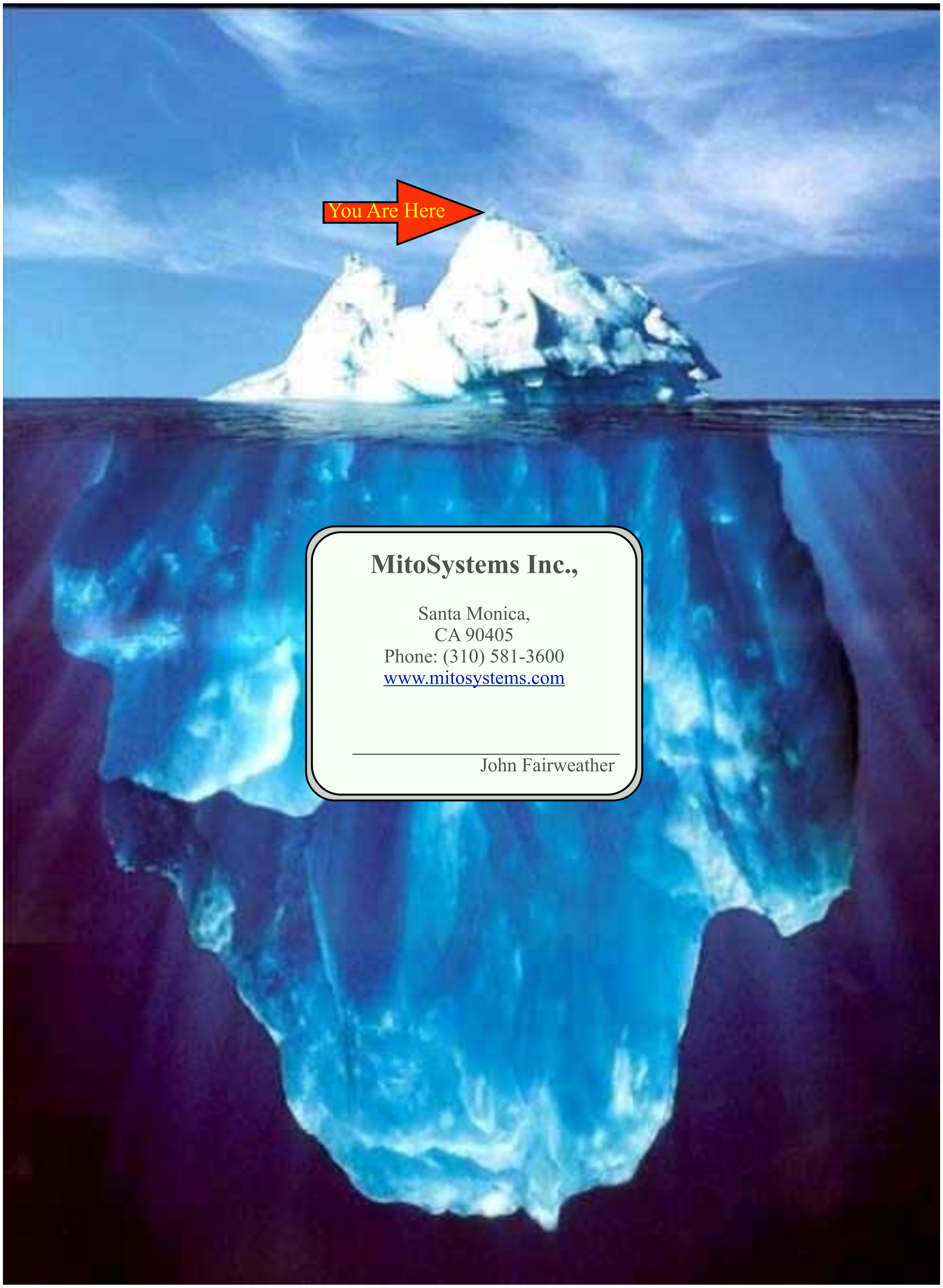
John Fairweather, President

MitoSystems, Inc.

Santa Monica, CA 90405
Tel: (310) 581-3600
www.mitosystems.com





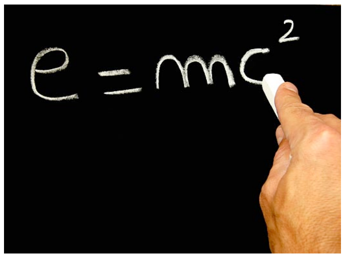


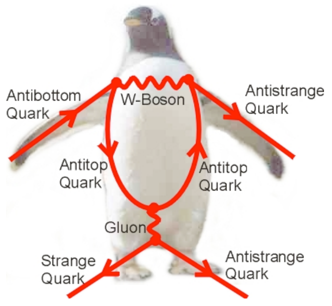
MitoSystems Inc.,

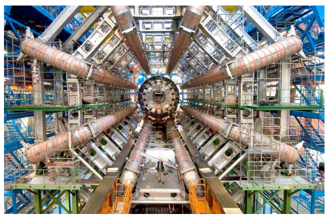
Santa Monica,
CA 90405
Phone: (310) 581-3600
www.mitosystems.com

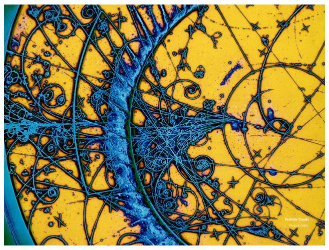
John Fairweather

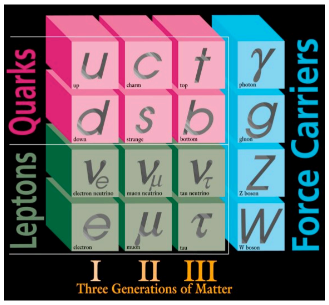
Table of Contents

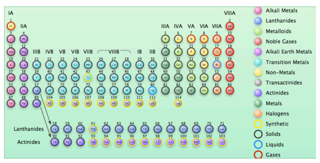



Antibottom Quark
W-Boson
Antistrange Quark
Antitop Quark
Antitop Quark
Gluon
Strange Quark
Antistrange Quark

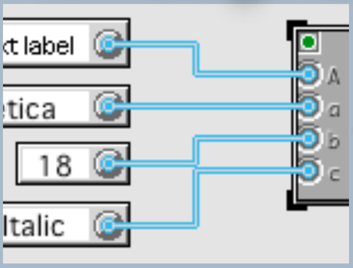


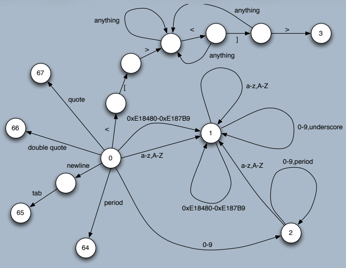


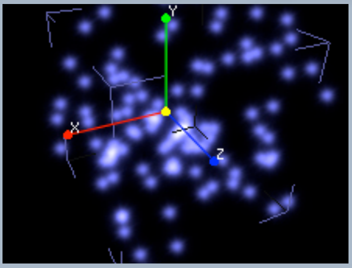

Quarks
Leptons
Force Carriers
u c t γ
d s b g
ν_e ν_μ ν_τ Z
e μ τ W
I II III
Three Generations of Matter

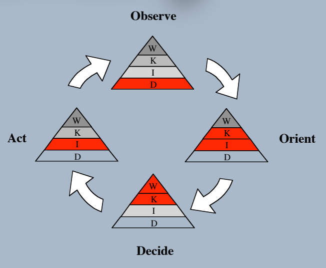

Alkali Metals
Lanthanides
Metalloids
Noble Gases
Alkali Earth Metals
Transition Metals
Non-Metals
Transactinides
Actinides
Metals
Halogens
Synthetic
Solids
Liquids
Gases

Mitopia® Technical Library









Introduction

This document describes the philosophy and implementation of Mitopia's unique Carmot Ontology Definition Language (ODL), and serves as a guide for those wishing to use, understand, or extend Mitopia's default/base ontology.

Definition

In philosophy, ontology (from the Greek ὄν, genitive ὄντος: of being (part. of εἶναι: to be) and -λογία: science, study, theory) is the most fundamental branch of metaphysics. Ontology is the study of being or existence and its basic categories and relationships. It seeks to determine what entities can be said to "exist", and how these entities can be grouped according to similarities and differences.

In both computer science and information science, an ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain.

Ontology Background

The concept of ontology is generally thought to have originated in early Greece and occupied Plato and Aristotle. While the etymology is Greek, the oldest extant record of the word itself is the Latin form ontologia, which appeared in 1606, in the work Ogdoas Scholastica by Jacob Lorhard (Lorhardus) and in 1613 in the Lexicon philosophicum by Rudolf Göckel (Goclenius). The first occurrence in English of "ontology" as recorded by the OED appears in Bailey's dictionary of 1721, which defines ontology as 'an Account of being in the Abstract'.

Students of Aristotle first used the word 'metaphysica' (literally "after the physical") to refer to the work their teacher described as "the science of being qua being". The word 'qua' means 'in the capacity of'. According to this theory, then, ontology is the science of being, in as much as it is being, or the study of beings insofar as they exist. Take anything you can find in the world, and look at it, not as a puppy or a slice of pizza or a folding chair or a president, but just as something that is. More precisely, ontology concerns determining what categories of being are fundamental and asks whether, and in what sense, the items in those categories can be said to "be".

Ontological questions have also been raised and debated by thinkers in the ancient civilizations of India and China, in some cases perhaps predating the Greek thinkers who have become associated with the concept.

Ontologies are used in artificial intelligence, the Semantic Web, software engineering, biomedical informatics, library science, and information architecture as a form of knowledge representation about the world or some part of it. Common components of ontologies include:

- **Individuals**: instances or objects (the basic or "ground level" objects)
- **Classes**: sets, collections, concepts or types of objects
- **Attributes**: properties, features, characteristics, or parameters that objects (and classes) can have
- **Relations**: ways that classes and objects can be related to one another
- **Function terms**: complex structures formed from certain relations that can be used in place of an individual term in a statement
- **Restrictions**: formally stated descriptions of what must be true in order for some assertion to be accepted as input
- **Rules**: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form
- **Axioms**: assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of "axioms" in generative grammar and formal logic. In these disciplines, axioms include only statements asserted as a priori knowledge. As used here, "axioms" also include the theory derived from axiomatic statements.
- **Events**: the changing of attributes or relations

Ontologies are commonly encoded using ontology languages. Examples of traditional ontology languages are:

CycL, DOGMA, F-Logic, KIF, KL-ONE, OCML, OKBC, and RACER

Examples of markup ontology languages (i.e., that use a markup scheme commonly XML):

DAML+OIL, OIL, OWL (Web Ontology Language), RDF, and SHOE

In general, ontology languages fall into three basic types:

- Frame-based languages (e.g., F-Logic, OKBC, and KM) in which the focus is primarily on the description of objects and classes, while relations and interactions are considered “secondary”. In this sense object-oriented programming languages are frame languages.

- Description logic Languages (e.g., K:-ONE, RACER and OWL) extend frame languages to include logic-based semantics to allow reasoning about items and relations described in the language, generally by mapping description logic into first-order predicate calculus/logic.

- First-order logic-based Languages (e.g., CycL and KIF) go all the way to extend frame languages by directly supporting first-order logic within the language itself.

Of all the ontology languages mentioned above, the most dominant at present is the Web Ontology Language (OWL). For the purposes of brevity, the discussions that follow will compare and contrast Mitopia's approach to ontologies with that of OWL, since this language is the evolutionary pinnacle of what we will refer to as "semantic ontologies". All other languages mentioned above are also semantic ontology languages.

The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies, and is endorsed by the World Wide Web Consortium. This family of languages is based on two (largely, but not entirely, compatible) semantics: OWL DL and OWL Lite semantics are based on Description Logics, which have attractive and well-understood computational properties, while OWL Full uses a novel semantic model intended to provide compatibility with RDF Schema. OWL ontologies are most commonly serialized using RDF/XML syntax. OWL is considered one of the fundamental technologies underpinning the Semantic Web, and has attracted both academic and commercial interest. The character Owl from Winnie the Pooh wrote his name WOL.

The data described by an OWL ontology is interpreted as a set of "individuals" and a set of "property assertions" which relate these individuals to each other. An OWL ontology consists of a set of axioms which place constraints on sets of individuals (called "classes") and the types of relationships permitted between them. These axioms provide semantics by allowing systems to infer additional information based on the data explicitly provided. For example, an ontology describing families might include axioms stating that a "hasMother" property is only present between two individuals when "hasParent" is also present, and individuals of class "HasTypeOBlood" are never related via "hasParent" to members of the "HasTypeABBlood" class. If it is stated that the individual Harriet is related via "hasMother" to the individual Sue, and that Harriet is a member of the "HasTypeOBlood" class, then it can be inferred that Sue is not a member of "HasTypeABBlood". A full introduction to the expressive power of the OWL language(s) is provided in the W3C's OWL Guide.

The Knowledge Pyramid

In order to understand where ontologies fit into the range of computing systems, and most particularly how ontology-based systems differ fundamentally from todays taxonomy level systems, we need to examine the information systems "knowledge pyramid".

Systems that operate on the Data level may be characterized as those that contain or acquire large amounts of measurements or data points concerning the target domain but have not yet organized this data into a human-useable form. Data-level systems are most frequently found during the ingestion or data-acquisition phase.

Information-level systems can be characterized as having taken the raw data and placed it into tables or structures that can then be searched, accessed and displayed by the system users. The overwhelming majority of information systems out there today operate in this realm.

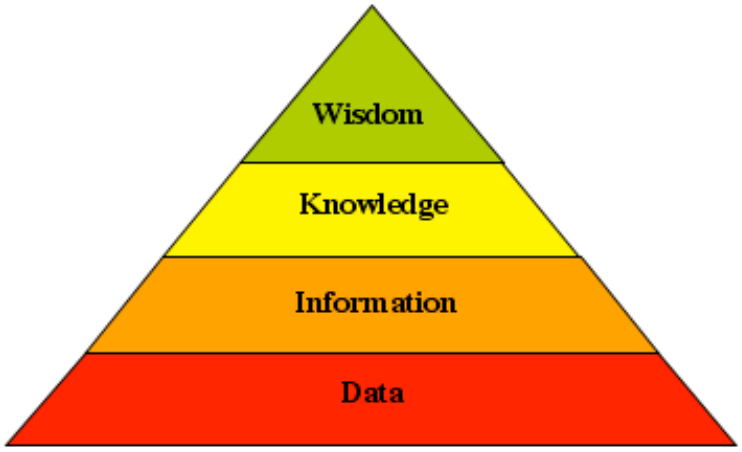


Figure 1 - The Knowledge Pyramid

A system that operates at the Knowledge level has organized the information into richly interrelated forms that are tied directly to a mental model or ontology that expresses the kinds of things that are being discussed in the information, and the kinds of interactions that are occurring between them. An ontology is a formalization of the "mental model" for the types of items that exist in the target domain, and the types of interactions that can occur between them. Few systems today operate at this level, but those that do allow their users to find 'meaning' in the information they contain, and see information and relationships directly in terms of a mental model that relates to real word items of interest.

Finally, we have the level of Wisdom. In this domain it is all about patterns within the knowledge. A system operating at the Wisdom level allows its users to view new knowledge in terms of their entire repository of past knowledge, to see patterns in that knowledge, and to predict the intent of known or inferred entities of interest that those patterns imply. The key to a Wisdom level system is its ability to model what is truly going on, and to predict, by comparison with past patterns, what may be about to happen. Unfortunately, there are no information systems in existence that operate at the Wisdom level in any large or generalized domain. As a result, wisdom remains the exclusive purview of the people that use our current information systems.

We can associate other adjectives with the various layers of the information pyramid, for example if we consider the needs of how information is organized we can divide the pyramid as follows:

Data – FORMAT,STORAGE. At this level we must know only the format (from the Latin forma meaning shape) of the data. The data is stored but retrieval is ad-hoc and search is not supported.

Information – TAXONOMY,ACCESS. At this level we must organize the data using a taxonomy (from the Greek 'taxis' meaning to arrange or order), nothing more. An information level system provides access to the data but not integration or meaning.

Knowledge – ONTOLOGY,CONNECTIONS. At this level, an ontology (from the Greek 'ont' meaning to be) is required since we are interested in connections as well as content. Understanding what is going on requires a rich web of connections associated with each item of any significance that is persistently stored.

Wisdom - COGNOLOGY,PATTERNS. At this level, to see patterns and trends, our ontologically organized data must be stored as a continuum, that is, we can simultaneously and graphically view the state of all records and their interconnections over time, and from differing perspectives (which may alter the content or connections for any record). We will call such a data substrate a "Cognology" (from the Latin 'cognitum' meaning to know) since no word exists to describe this level of organization in the computer science literature.

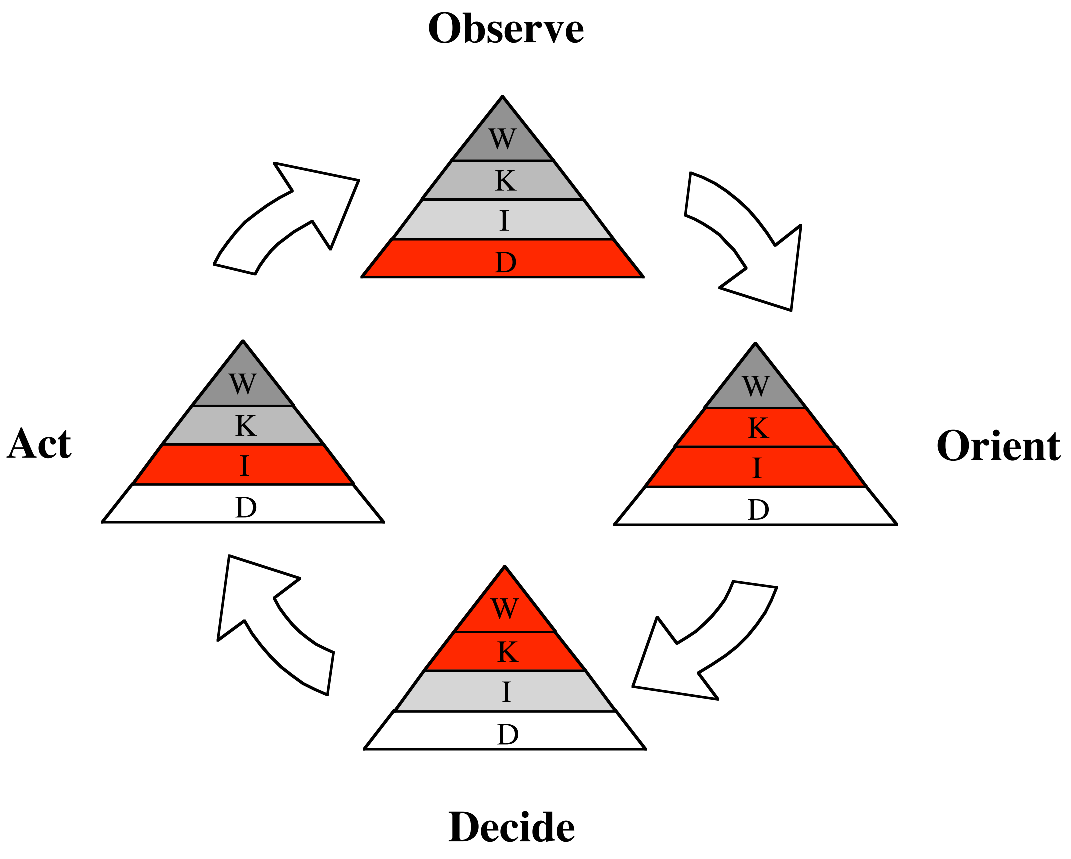


Figure 2 - Organizational OODA Loop

The diagram above illustrates a large Organization's OODA loop (decision cycle) in terms of the levels of the knowledge pyramid that are required to facilitate each step in the cycle. As can be seen, to close the cycle requires knowledge level activities in the 'orient' and 'decide' steps, and wisdom level (i.e., human in the loop) at the 'decide' step. Since we cannot currently create wisdom level software systems, all we can hope to do is provide extensive tools at this step to facilitate human decisions, while automating to the maximum degree possible the 'orient' step. To 'orient' generally involves integrating massive amounts of disparate data, and hence without automating this step within a System, it will be difficult if not impossible for human beings to keep up with ongoing events, thus making the remaining steps of the cycle moot. Current generation information systems and design techniques fail to adequately address even the integration step required to 'orient', and thus in reality they cannot keep up with evolving events, and so they are at best retroactive tools to explain what went wrong.

Taxonomy vs. Ontology

As has been stated above, current generation software systems, databases, and design methodologies, are all based on an information level or taxonomic approach. Mathematically, a taxonomy is a tree structure (or containment hierarchy) of classifications for a given set of objects. Taxonomies are constrained to this tree form, and any references between one node in the tree and another that is not either a parent, a sibling, or a child is 'out of scope' for a pure taxonomic system. Of course it is these arbitrary connections that contain most of the interesting information in the real world, not just the contents of record fields.

If we look at the technological underpinnings of today's information systems, at the base level we can identify just two technologies that are fundamental to virtually all that we do in these systems, namely object oriented programming, and relational databases. Object oriented programming (OOP) languages are frame-based and taxonomic in nature, that is, the focus is on creating classes and sub-classes with which to inherit functionality and fields. The OOP programming languages themselves provide no support for creating connections between instances other than the basic pointer mechanism which means that there is no systematic support for knowledge level operations. Even the pointer mechanism can only be used within the current process, and so to persist any relationship beyond the current program run, the problem must generally be offloaded to the database using an entirely different and incompatible programming model (SQL).

A relational database can be visualized as a collection of tables, each of which comprises a number of columns and a number of rows. The tables are called relations, and the rows are called tuples. Each relation records data about a particular type of thing such as customers, products, or orders, and each tuple in a relation records data about one individual instance of the appropriate type. The names of each relation's columns must be set up by a database administrator before the database is first set up. Thereafter, new tuples can be added and existing tuples can be altered or deleted to reflect changes in the data. The SQL language provides the interface between code in an application program and the operations or searches that it wishes to do on the tuples of the database. In every relation, one or more columns together are designated as the primary key which can be used elsewhere to reference individual tuples. There must be exactly one primary key, no more, no less. A relation may also contain one or more foreign keys, which are basically references to the primary key of some other relation and it is through this mechanism that it is possible to link one table with others thus creating a complete cross-referenced database. It is important to note that the term "relational" is somewhat of a misnomer since it refers to the mathematical concept of a relation, it does not imply that relational databases are actually designed to represent the relationships between different things. In relational databases, relationships between things must be inferred by someone (or some code) with sufficient knowledge of the database schema. In real life situations, tables can contain thousands of columns each added by different programmers with different naming conventions and agendas, and thus it is often very difficult to determine what relationships exist within the data, and is certainly not something that can be accomplished by any generalized piece of code.

Contrary to its name, the relational model makes it extremely difficult to represent relationships between different things, and almost impossible to dynamically create new types of relationships as they are discovered in incoming data. Since a knowledge level system is focused primarily on the relationships between things, we can effectively rule out the use of relational databases to implement such a system.

The ontological approach applies not only to where a given item can be found and what ancestral types it derives from, but far more importantly, it determines the very nature and format of the 'fields' within any given record. In a taxonomic database (e.g., a relational database) or information repository, the system implementer is free to define a field in isolation from any consideration of the rest of the database structure. In an ontological database, where the focus is not only on the content, but more importantly, on the interconnections, this is no longer the case. Thus for example if one had a database field for a person's titles (e.g., Professor or President), in a classic taxonomic database or system one would simply create a text field called "titles" and rely on user's latent understanding of such concepts to ensure that they entered the appropriate text (which would to the system have no inherent 'meaning'). In an ontological database, where meaning and connections are the focus, one is forced to ask "what is a person's title…what does it mean?" If one thinks about it in these terms, it is clear that a title represents a three way relationship between an individual (the incumbent holder of the title), the organization that confers that title upon that individual, and a particular title (e.g., "Professor") which has some inherent meaning in a wider scope such that use of the title immediately conveys meaning in some larger context. Coveting and acquiring titles is fundamental to the character of many people and tells us much about them. The title explicitly defines rank and privilege within the conferring organization, but more importantly the title, size, and importance of the conferring organization strongly impacts the behavior and allegiances of the individual and his social status and interactions with others. There is much to be understood from the simple nature of the title web. Thus in an ontological system, a title must be represented as a three-way linking record between a conferring organization, a person, and a constrained domain of generally understood titles conveying some functional authority. Not only that, but we must consistently handle the 'echo' fields implied by such relationships in the other types involved. Thus it is clear that the concept of people having titles implies that organizations must have something like a 'key personnel" field which provides the reverse connection to the incumbents through the linking record. It should be obvious that this is fundamentally a more correct and certainly a more 'meaningful' and thus computable representation of what a title represents, and yet few if any current systems go even this far, so such systems can never represent knowledge, only information. Thinking of information in these terms is rare, indeed software courses teach the exact opposite localized taxonomic approach, and our databases and tools provide only for such localized

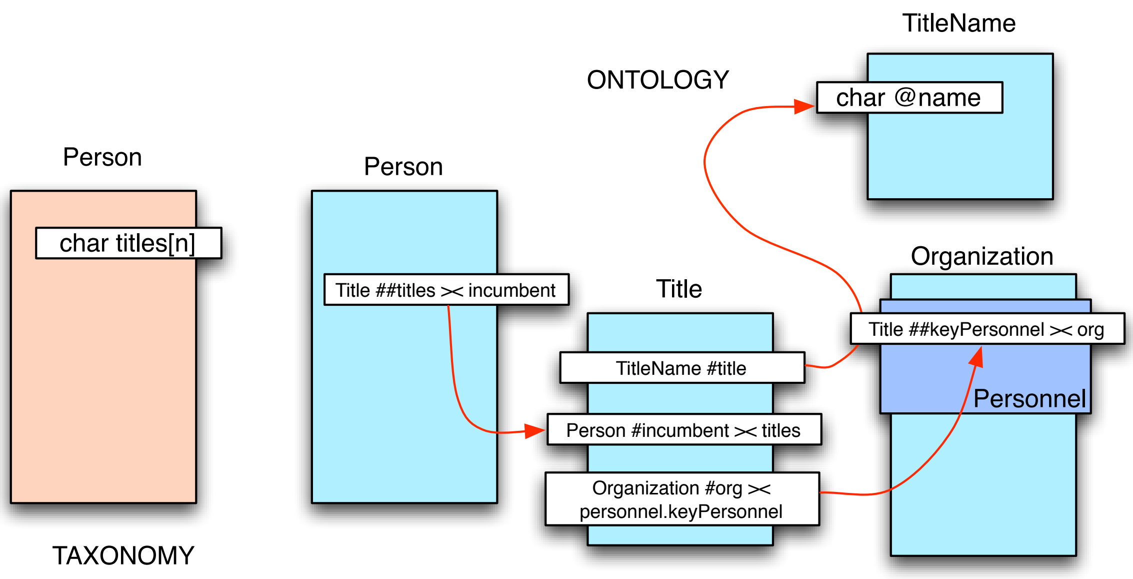

TitleName
ONTOLOGY
char @name
Person
char titles[n]
Person
Title ##titles >< incumbent
Title
Organization
Title ##keyPersonnel >< org
TitleName #title
Personnel
Person #incumbent >< titles
Organization #org >< personnel.keyPersonnel
TAXONOMY

Figure 3 - Taxonomy vs. Ontology

thinking. It is hard, very hard, to implement systems that truly operate at the knowledge level, and it requires almost a philosophy degree to think this way, which explains why we don't see such systems out there.

Clearly also, this is not the way people normally seek information, so any web site (or other digital information repository or source) externally organized in this ontological manner would certainly go out of business in short order. Many people find such an ontological approach unnatural because it challenges their own poorly thought-out models of what things mean in a way that a bland taxonomic approach does not, since it leaves the interpretation of meaning up to the reader. For these reasons, resistance to ontological thinking is widespread and the approach clearly faces significant educational hurdles. However, the truth is that though we might shy away from these concepts when they are made explicit in our communications, the human mind at its lowest, often subconscious, level, operates exclusively by means of ontologies. The process by which a child grows up and experiences the world is one in which that child builds mental models (or ontologies) of how things in the world work and interrelate, and refines these models to the point where they become powerful enough to allow that person to make informed and weighted decisions on what actions to take in any given situation and what the possible consequences might be (by modeling the ontologies of others). It is these ontologies that allow us to understand new and unfamiliar information by converting it to parallels or transient metaphors based on existing internal models and the connections between them. It is our internal 'ontologies' that drive us to seek particular answers, but our spoken language and other communication techniques require us to translate these into the taxonomy of others, who cannot possibly know or understand our internal models, for the purposes of communicating. Human beings, the consummate communicators, do this so well that most of us are completely unaware of our internal ontology unless closely questioned by experts, a thing that rarely happens unless we visit a psychiatrist or are suspected of a crime. Exposing our ontology in communication leads to conflict and we are all loath to do so.

Thus we can say that in a very real sense, an ontology is the more fundamental, and ultimately more useful way, of organizing information for connections and computability, and yet there are almost no information systems in existence that adopt this approach to data and its organization. This is either because of the difficulty of bridging the communication gap with the user, or more likely, simply a matter of convenience or lack of mental rigor on the part of our software vendors. The fact of the matter is, that only an ontology based software architecture can serve as a generalized platform on which to build true knowledge level systems, since that ontology (and the technology to define and access it) is, as we ourselves show, the key to spanning and unifying diverse sources. It is this lack of a meaningful underpinning that gives rise to the belief that computer systems are stupid, and will never rise to the level of comprehension and flexibility exhibited by the human mind. For existing systems this belief is well-founded. Clearly the goal of future intelligence systems must be, if not to rise to the level of human understanding, then at least to rise to a sufficient level of understanding that these systems can serve to augment human beings and act as their proxies in the torrent of information, drawing those things that seem most relevant to the attention of the appropriate person. Thus not only must the system operate ontologically, but it must also express that ontology (perhaps filtered) to the user so that he can express his interests and desires in a form that the system can reliably, and as far as possible, unambiguously automate.

Mitopia® is fundamentally a suite of technologies necessary to rapidly build systems based on an application (and possibly user specific) configurable ontology, layered on top of a more fundamental ontology designed to understand the nature of the world and to facilitate meaningful exchange of knowledge between systems. We have stated above that information organized as an ontology is inherently more 'computable' and meaningful than information stored in taxonomies, but why and how is this so? To illustrate this with a trivial example, let us return to the question of how a person's title(s) are represented in the two approaches. In the taxonomic approach, the 'titles' field contains a presumably comma-separated list of titles (hopefully spelled correctly!). It is basically free-form text. Now imagine we wish to create an analytical process that operates on the data, and is designed to extract some kind of predictive model for the probable habits, lifestyle, and social impact of a set of persons in storage.

In a taxonomic system, the best we could do is create a process with its own internal list of common titles and associated with each title we might define some sort of weighting to indicate certain social characteristics that such a title might imply. Of course, our title list would only be partial, would not adapt to new input, and we could hope at best to recognize a tiny fraction of the actual titles. More importantly, since there is no relationship between the titles field and the organization that confers it, we cannot in any real sense evaluate the significance of the title. For example, the President of the United States and the President of a one-man corporation are indistinguishable by taxonomy, the title "President" thus becomes meaningless. The same is actually true of almost all titles, so in fact the best our process could do in this situation, is make some kind of guess as to the nature of the job that person had, but not in any way the significance, allegiances or other implications. Titles themselves are used inconsistently and thus require context. For example the title Secretary generally connotes a low paid clerical worker with little or no authority. But consider the Secretary of State, or the Secretary of Defense.

In an ontological system, the situation is quite different. Firstly, our process needs no disconnected internal list of titles since it can retrieve the current complete list simply by accessing persistent storage. The fact that the titles field is implemented by reference also removes all issues with misspelling, updates and consistency between the reference field and the referenced items (via the echo fields). Also now that there is an ontological type to hold title names, we have a place in persistent storage to put the fields defining any characteristics or information associated with particular titles. This means that not only is this information available to our analysis process, it is also available to all other algorithms in the system and indeed can be viewed and/or updated directly by any system user. It is clear that we are breaking down the stovepipe 'hidden' nature of the algorithm that our taxonomic process would have exhibited, and instead are publishing it in ontological form for all to see (or ignore). Moreover, since the title link is between an individual and an organization, we can now examine the organization to see what it is, and how big it is, and thereby we can get a far more accurate measure of the significance of each particular use of a given title such as "President". We will no longer confuse the President of the United States with just another guy on the street. Furthermore, if we wish to know more about the job a person might do, we can examine the conferring organization, perhaps we know its SIC or NAICS code, or perhaps we have a list of ontologically described products. We can probably examine the number of employees, or the annual income, and thereby compute the social significance of the particular title when used to interact with outsiders. From this we may be able to compute certain constraints on that individual's public positions and behavior. We can probably also discover or approximate the individual's income and living standard, and we can certainly track allegiances. The list of possible computations that such a process might do based solely on the 'titles' field could be quite extensive.

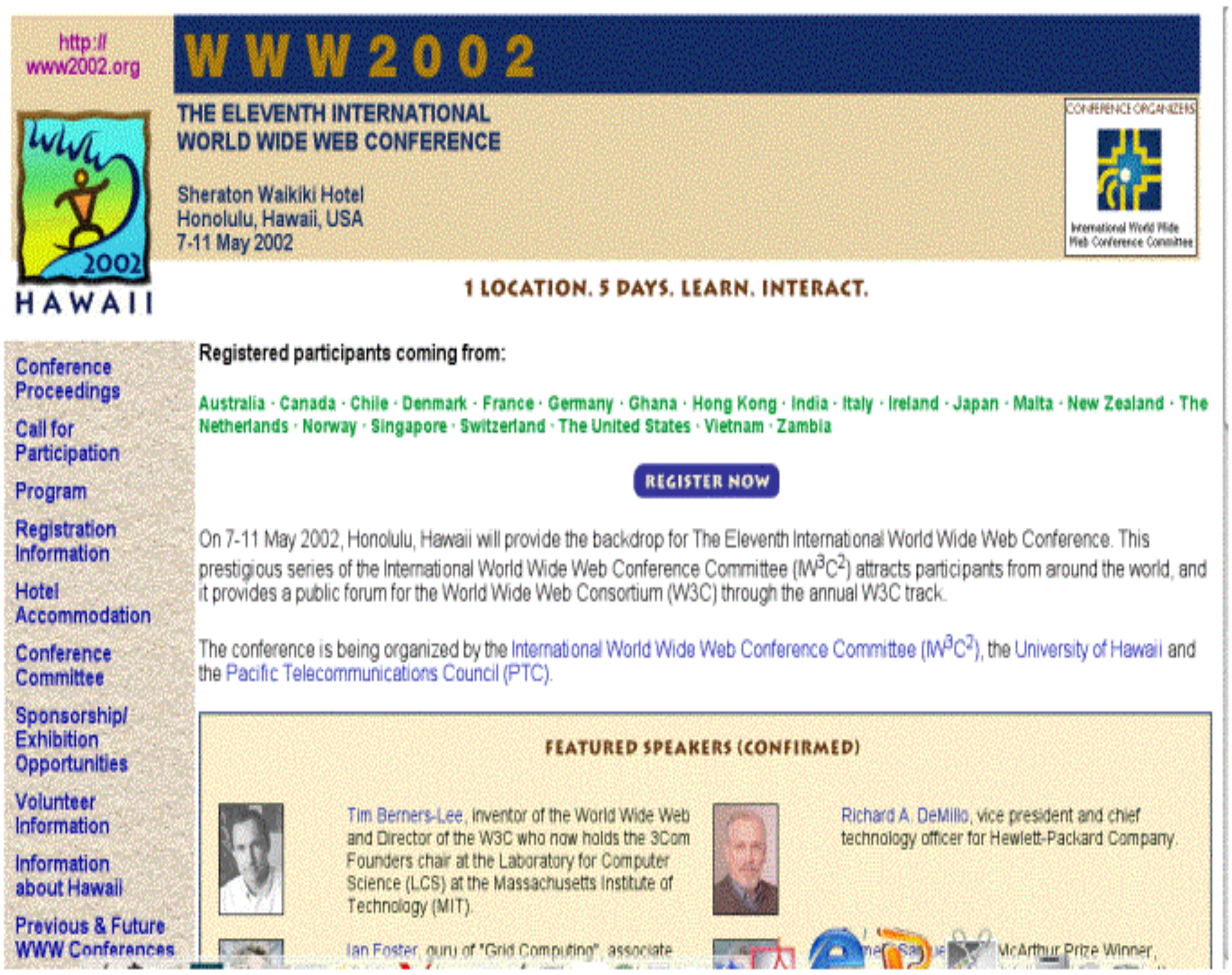
Now imagine this same improvement in computability repeated for all the possible fields in a Person record (perhaps hundreds) and the various institutions, persons, possessions and insights they might lead us to. Then further imagine a suite of similar processes each using ontologically-based algorithms to establish some kind of quantitive or qualitative measure of understanding, all of which can share in drawing an analytical conclusion, and we begin to see the difference in power between the two approaches. More importantly, in the ontology approach, since algorithms can be written to operate on higher level types (e.g., entity) from which specific types derive, the algorithmic smarts can be re-purposed easily and, if well written, may be largely independent of the specific ontology and certainly should be common across the underlying base ontology.

Unfortunately the word ontology has now been completely hijacked in the computer field to mean ‘semantic ontology’, that is an ontology of language and sentence understanding. In pure semantic ontology, everything is simply a “document” together with a set of graphs representing the “meaning” of the sentences within the document. Specialized AI-style code is required to manipulate these graphs and tie them to the source documents, and the utility of such a system is restricted to querying based on the graphs that are present in those documents. The other major shortcoming of this approach is that while it deals well with document understanding, it cannot represent or leverage in any meaningful way an encyclopedic store of data derived from external databases in order to improve the power of its analysis. To add more power, one must add more semantic knowledge (or rules) to the ontology itself. This is why researchers have spent so many man-years developing more and more elaborate semantic ontologies in order to improve performance in sentence understanding. Many such projects can be found on the web. Moreover, such ontologies provide no means to unify information from divergent sources into an explicit representational form that can serve as a generalized platform on which all computations and data accesses can be built. Only the semantic graphs are provided, and thus it is meaningless for example to discuss the use of semantic ontologies to represent and handle business process since there is no support for anything but the manipulation of semantic graphs in text.

The semantic web, which is largely based on OWL, seeks to extend the semantics into the XML tags associate with the text on the web. As a result, we must first examine the semantic web and contrast it with the Mitopia® approach in order to clarify exactly what we mean by the word ontology.

Why Semantic Ontologies? (illustrations from Semantic Web lecture by John Davies, BT)

The motivation behind the semantic web, for which OWL is the current underpinnings, was essentially to find some way for a computer to understand 'meaning' in the vast textual information content of the world wide web in order to assist computer users to answer more complex questions than those that could be answered by simple keyword text searches. This goal is still largely unrealized, although some progress has been made in limited areas. To understand why this is such a serious problem consider the appearance of a typical web page such as that shown below:



http://www2002.org

WWW2002

THE ELEVENTH INTERNATIONAL WORLD WIDE WEB CONFERENCE

Sheraton Waikiki Hotel
Honolulu, Hawaii, USA
7-11 May 2002

1 LOCATION. 5 DAYS. LEARN. INTERACT.

- Conference Proceedings
- Call for Participation
- Program
- Registration Information
- Hotel Accommodation
- Conference Committee
- Sponsorship/ Exhibition Opportunities
- Volunteer Information
- Information about Hawaii
- Previous & Future WWW Conferences

Registered participants coming from:

Australia · Canada · Chile · Denmark · France · Germany · Ghana · Hong Kong · India · Italy · Ireland · Japan · Malta · New Zealand · The Netherlands · Norway · Singapore · Switzerland · The United States · Vietnam · Zambia

REGISTER NOW

On 7-11 May 2002, Honolulu, Hawaii will provide the backdrop for The Eleventh International World Wide Web Conference. This prestigious series of the International World Wide Web Conference Committee (IW^3C^2) attracts participants from around the world, and it provides a public forum for the World Wide Web Consortium (W3C) through the annual W3C track.

The conference is being organized by the International World Wide Web Conference Committee (IW^3C^2), the University of Hawaii and the Pacific Telecommunications Council (PTC).

FEATURED SPEAKERS (CONFIRMED)

Tim Berners-Lee, inventor of the World Wide Web and Director of the W3C who now holds the 3Com Founders chair at the Laboratory for Computer Science (LCS) at the Massachusetts Institute of Technology (MIT).

Richard A. DeMillo, vice president and chief technology officer for Hewlett-Packard Company.

Ian Foster, guru of "Grid Computing", associate

McArthur Prize Winner,

This page can be quite readily understood by a human being, however the page contains a vast amount of information that is communicated by such things as page layout, font sizes, color, sequence and organization as well as hyperlinks to related content. This information is largely held in the HTML markup tags used to lay out the page which are not displayed to the user but are nonetheless critical to understanding, thus the semantic content is not easily accessible to computers.

The human sees:

WWW2002
The eleventh international world wide web conference
Sheraton waikiki hotel
Honolulu, hawaii, USA
7-11 may 2002
1 location 5 days learn interact
Registered participants coming from
australia, canada, chile denmark, france, germany, ghana, hong kong, india, ireland, italy, japan, malta, new zealand, the netherlands, norway, singapore, switzerland, the united kingdom, the united states, vietnam, zaire
Register now
On the 7th May Honolulu will provide the backdrop of the eleventh international world wide web conference. This prestigious event …
Speakers confirmed
Tim berners-lee
Tim is the well known inventor of the Web, …
Ian Foster
Ian is the pioneer of the Grid, the next generation internet …

Whereas the computer from a cognitive point of view does not understand either the text, or the markup, and so it sees:

🕈🕈🕈📄📁📁📄
❄♒♏ ♏●♏❖♏■⧫♒ ♓■⧫♏❒■♋⧫♓□■♋● ⬥□❒●♎ ⬥♓♎♏ ⬥♏♌ ♍□■♐♏❒♏■♍♏
💧♒♏❒♋⧫□■ ⬥♋♓🙵♓🙵♓ ♒□⧫♏●
☟□■□●◆●◆📪 ♒♋⬥♋♓♓📪 🕆💧✌
🖮📫📂📂 ❍♋⍓ 📄📁📁📄
📂 ●□♍♋⧫♓□■ 🗄 ♎♋⍓⬧ ●♏♋❒■ ♓■⧫♏❒♋♍⧫
☼♏♑♓⬧⧫♏❒♏♎ ◻♋❒⧫♓♍♓◻♋■⧫⬧ ♍□❍♓■♑ ♐❒□❍
♋◆⬧⧫❒♋●♓♋📪 ♍♋■♋♎♋📪 ♍♒♓●♏ ♎♏■❍♋❒🙵📪 ♐❒♋■♍♏📪 ♑♏❒❍♋■⍓📪 ♑♒♋■♋📪 ♒□■♑ 🙵□■♑📪 ♓■♎♓♋📪 ♓❒♏●♋■♎📪 ♓⧫♋●⍓📪 🙰♋◻♋■📪 ❍♋●⧫♋📪 ■♏⬥ ⌘♏♋●♋■♎📪 ⧫♒♏ ■♏⧫♒♏❒●♋■♎⬧📪 ■□❒⬥♋⍓📪 ⬧♓■♑♋◻□❒♏📪 ⬧⬥♓⧫⌘♏❒●♋■♎📪 ⧫♒♏ ◆■♓⧫♏♎ 🙵♓■♑♎□❍📪 ⧫♒♏ ◆■♓⧫♏♎ ⬧⧫♋⧫♏⬧📪 ❖♓♏⧫■♋❍📪 ⌘♋♓❒♏
☼♏♑♓⬧⧫♏❒ ■□⬥
🏳■ ⧫♒♏ 🖮⧫♒ 💣♋⍓ ☟□■□●◆●◆ ⬥♓●● ◻❒□❖♓♎♏ ⧫♒♏ ♌♋♍🙵♎❒□◻ □♐ ⧫♒♏ ♏●♏❖♏■⧫♒ ♓■⧫♏❒■♋⧫♓□■♋● ⬥□❒●♎ ⬥♓♎♏ ⬥♏♌ ♍□■♐♏❒♏■♍♏📬 ❄♒♓⬧ ◻❒♏⬧⧫♓♑♓□◆⬧ ♏❖♏■⧫ ⑤
💧◻♏♋🙵♏❒⬧ ♍□■♐♓❒❍♏♎
❄♓❍ ♌♏❒■♏❒⬧📫●♏♏
❄♓❍ ♓⬧ ⧫♒♏ ⬥♏●● 🙵■□⬥■ ♓■❖♏■⧫□❒ □♐ ⧫♒♏ 🕈♏♌📪 ⑤
✋♋■ ☞□⬧⧫♏❒
✋♋■ ♓⬧ ⧫♒♏ ◻♓□■♏♏❒ □♐ ⧫♒♏ ☝❒♓♎📪 ⧫♒♏ ■♏⌧⧫ ♑♏■♏❒♋⧫♓□■ ♓■⧫♏❒■♏⧫ ⑤

The solution that the web has adopted to this problem is to use XML markup with meaningful tags:

```
<name>🕈🕈🕈📄📁📁📄

❄♒♏ ♏●♏❖♏■⧫♒ ♓■⧫♏❒■♋⧫♓□■♋● ⬥□❒●♎ ⬥♓♎♏ ⬥♏♌♍□■</name>
<location>💧♒♏❒♋⧫□■ ⬥♋♓🙵♓🙵♓ ♒□⧫♏●
🏱□■□●◆●◆📪 ♒♋⬥♋♓♓📪 🕆💧✌</location>
<date>🖮📫📂📂 ❍♋⍓ 📄📁📁📄</date>
<slogan>📂 ●□♍♋⧫♓□■ 🗄 ♎♋⍓⬧ ●♏♋❒■ ♓■⧫♏❒♋♍⧫</slogan>
<participants>☼♏♑♓⬧⧫♏❒♏♎ ◻♋❒⧫♓♍♓◻♋■⧫⬧ ♍□❍♓■♑ ♐❒□❍
♋◆⬧⧫❒♋●♓♋📪 ♍♋■♋♎♋📪 ♍♒♓●♏ ♎♏■❍♋❒🙵📪 ♐❒♋■♍♏📪 ♑♏❒❍♋■⍓📪 ♑♒♋
    ■♋📪 ♒□■♑ 🙵□■♑📪 ♓■♎♓♋📪 ♓❒♏●♋■♎📪 ♓⧫♋●⍓📪 🙰♋◻♋■📪 ❍♋●⧫♋📪
    ■♏⬥ ⌘♏♋●♋■♎📪 ⧫♒♏ ■♏⧫♒♏❒●♋■♎⬧📪 ■□❒⬥♋⍓📪 ⬧♓■♑♋◻□❒♏📪 ⬧⬥♓⧫⌘♏❒
    ●♋■♎📪 ⧫♒♏ ◆■♓⧫♏♎ 🙵♓■♑♎□❍📪 ⧫♒♏ ◆■♓⧫♏♎ ⬧⧫♋⧫♏⬧📪 ❖♓♏⧫■♋❍📪
    ⌘♋♓❒♏</participants>
<introduction>☼♏♑♓⬧⧫♏❒ ■□⬥
🏱■ ⧫♒♏ 🖮⬧♒ ♐♋⍓ 🏱□■□●◆●◆ ⬥♓●● ◻❒□❖♓♎♏ ⧫♒♏ ♌♋♍🙵♎❒□◻ □♐ ⧫♒♏ ♏
    ●♏❖♏■⧫♒ ♓■⧫♏❒■♋⧫♓□■♋● ⬥□❒●♎ ⬥♓♎♏ ⬥♏♌ ♍□■♐♏❒♏■♍♏📬 ❄♒♓⬧ □□
    ♏⬧⧫♓♌♓⧫♓□■⬧ ♏❖♏■⧫ ⑤
💧◻♏♋🙵♏❒⬧ ♍□■♐♓❒❍♏♎</introduction>
<speaker>❄♓❍ ♌♏❒■♏❒⬧📫●♏♏</speaker>
<bio>❄♓❍ ♓⬧ ⧫♒♏ ⬥♏●● 🙵■□⬥■ ♓■❖♏■⧫□❒ □♐ ⧫♒♏ 🕈♏♌📪</bio>...
```

But of course, the next problem is how the computer understands the meaning of the tags themselves in any standardized and cognitive way across multiple web sites, all with different content and focus. The truth is that without addressing this issue, to the computer, the web page still looks as follows:

```
<■♋❍♏>🕈🕈🕈📄📁📁📄
❄♒♏ ♏●♏❖♏■⧫♒ ♓■⧫♏❒■♋⧫♓□■♋● ⬥□❒●♎ ⬥♓♎♏ ⬥♏♌</■♋❍♏>
<●□♍♋⧫♓□■>💧♒♏❒♋⧫□■ ⬥♋♓🙵♓🙵♓ ♒□⧫♏●
🏱□■□●◆●◆📪 ♒♋⬥♋♓♓📪 🕆💧✌</●□♍♋⧫♓□■>
<♎♋⧫♏>🖮📫📂📂 ❍♋⍓ 📄📁📁📄</♎♋⧫♏>
<⬧●□♑♋■>📂 ●□♍♋⧫♓□■ 🗄 ♎♋⍓⬧ ●♏♋❒■ ♓■⧫♏❒♋♍⧫</⬧●□♑♋■>
<◻♋❒⧫♓♍♓◻♋■⧫⬧>☼♏♑♓⬧⧫♏❒♏♎ ◻♋❒⧫♓♍♓◻♋■⧫⬧ ♍□❍♓■♑ ♐❒□❍
♋◆⬧⧫❒♋●♓♋📪 ♍♋■♋♎♋📪 ♍♒♓●♏ ♎♏■❍♋❒🙵📪 ♐❒♋■♍♏📪 ♑♏❒❍♋■⍓📪 ♑♒♋
    ■♋📪 ♒□■♑ 🙵□■♑📪 ♓■♎♓♋📪 ♓❒♏●♋■♎📪 ♓⧫♋●⍓📪 🙰♋◻♋■📪 ❍♋●⧫♋📪
    ■♏⬥ ⌘♏♋●♋■♎📪 ⧫♒♏ ■♏⧫♒♏❒●♋■♎⬧📪 ■□❒⬥♋⍓📪 ⬧♓■♑♋◻□❒♏📪 ⬧⬥♓⧫⌘♏❒
    ●♋■♎📪 ⧫♒♏ ◆■♓⧫♏♎ 🙵♓■♑♎□❍📪 ⧫♒♏ ◆■♓⧫♏♎ ⬧⧫♋⧫♏⬧📪 ❖♓♏⧫■♋❍📪
    ⌘♋♓❒♏</◻♋❒⧫♓♍♓◻♋■⧫⬧>
<♓■⧫❒□♎◆♍⧫♓□■>☼♏♑♓⬧⧫♏❒ ■□⬥
🏱■ ⧫♒♏ 🖮⬧♒ ♐♋⍓ 🏱□■□●◆●◆ ⬥♓●● ◻❒□❖♓♎♏ ⧫♒♏ ♌♋♍🙵♎❒□◻ □♐ ⧫♒♏ ♏
    ●♏❖♏■⧫♒ ♓■⧫♏❒■♋⧫♓□■♋● ⬥□❒●♎ ⬥♓♎♏ ⬥♏♌ ♍□■♐♏❒♏■♍♏📬 ❄♒♓⬧ □□
    ♏⬧⧫♓♌♓⧫♓□■⬧ ♏❖♏■⧫ ⑤
💧◻♏♋🙵♏❒⬧ ♍□■♐♓❒❍♏♎</♓■⧫❒□♎◆♍⧫♓□■>
<⬧◻♏♋🙵♏❒>❄♓❍ ♌♏❒■♏❒⬧📫●♏♏</⬧◻♏♋🙵♏❒>
<♌♓□>❄♓❍ ♓⬧ ⧫♒♏ ⬥♏●● 🙵■□⬥■ ♓■❖♏■⧫□❒ □♐ ⧫♒♏ 🕈</♌♓□>
<⬧◻♏♋🙵♏❒>✋♋■ ☞□⬧⧫♏❒</⬧◻♏♋🙵♏❒>
<♌♓□>✋♋■ ♓⬧ ⧫♒♏ ◻♓□■♏♏❒ □♐ ⧫♒♏ ♑❒♓♎📪 ⧫♒♏ ■♏</♌♓□>
```

We need to add the semantics of the tags so that they can meaningfully be used to interpret the content that they enclose. One approach to this would be to get global agreement on the meaning of such annotation tags but this approach is fraught with problems and even if agreement could be reached, some formalism is needed to extend the annotation tags as necessary.

The solution chosen was to use semantic ontologies to specify the meaning of the annotations via a formalized ontology language thus allowing different systems to interpret data from other sources by examining the published ontology for that site and then interpreting the site contents which is fully tagged according to the site ontology. Ontologies allow new terms to be defined by combining existing ones as well as formally defining the meaning of such terms. Just as importantly, it is now possible to specify the relationships between terms in multiple ontologies thus allowing sharing of knowledge.

The resultant architecture required in order to get semantic ontologies to perform the goal of the semantic web appears as shown in Figure 4 below. Note that the entire architecture is built directly on top of Unicode and XML, that is, this is an architecture for which all the data, the ontology itself, and everything it can be used to express and manipulate, is based on text. This is as one might expect given the original target, which was to understand the textual content of the web pages that make up the world wide web.

Note also that various portions of this architecture are still in flux and much development work needs to be done before it is practical for any realistic ontological systems based on this approach to operate in unrestricted domains.

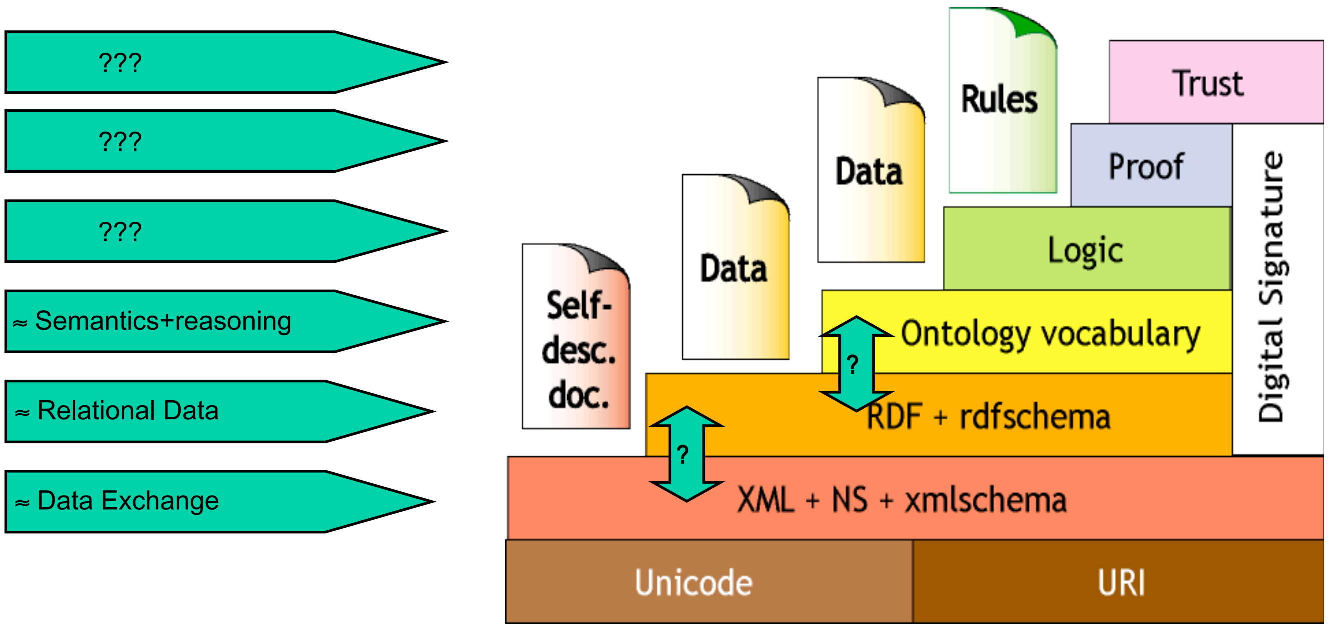


Figure 4 - Architecture of the Semantic Web

Of critical importance is that this entire approach does not address in any way the representation and direct manipulation of binary data. Since the incredible performance of today's processors, software systems and databases, is driven primarily by their ability to rapidly manipulate binary data representing the problem domain, we can see that a text-based semantic ontology approach is going to introduce all kinds of performance limitations and bottlenecks. This occurs as the textual representation of the data (e.g., “3.14159”) is continually and pervasively converted back and forth to/from the appropriate binary forms, in order to get anything done (including of course sharing any data with other systems). To see the impact of this, consider the timing for the two C code snippets shown below which represent the direct addition of two numbers held either directly or in textual form (an operation that at the lowest level represents the performance difference between the two approaches):

```
// addition of numbers held in binary:
    a = 3.14159; b = 1.61803; N = 1000000;
    for ( i = 0 ; i < N ; i++ )
        c = a + b;

// addition of numbers held as text:
    a = “3.14159”; b = “1.61803”; N = 1000000;
    for ( i = 0 ; i < N ; i++ )
        sprintf(c,”%f”,strtod(a,&cp) + strtod(b,&cp));
```

The second code snippet takes 275 times longer than the first to execute, so we can anticipate a similar performance problem will strike any semantic ontology-based system when we attempt to scale it. Not surprisingly therefore, one is hard pushed to find any successful truly large-scale deployment of semantic ontology technologies.

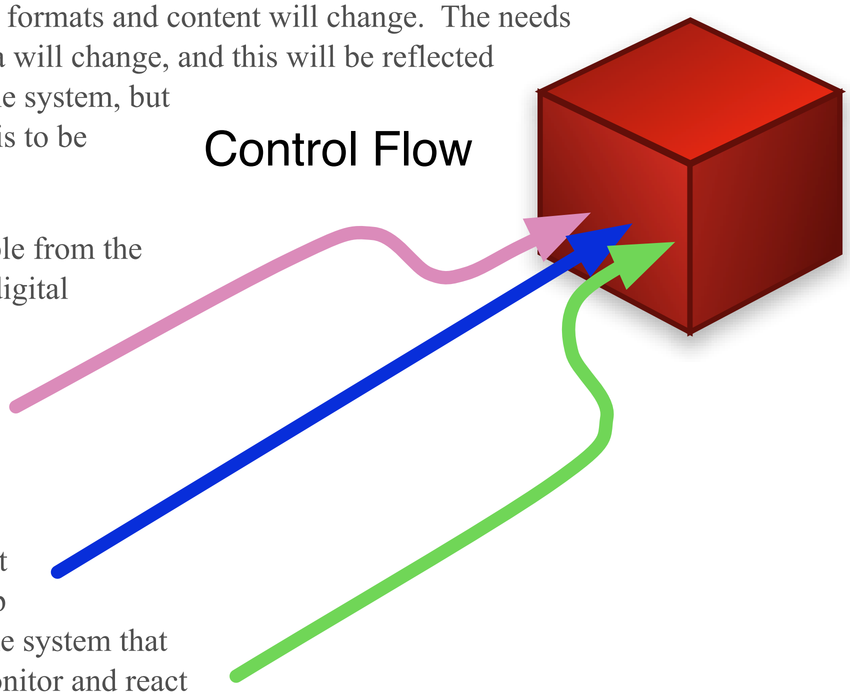
Why Mitopia's Ontology?

In contrast to the development of semantic ontologies and ultimately OWL, the Mitopia® approach followed a very different evolutionary path, and for very different motives. Indeed, throughout the early history of Mitopia® development, the fundamental technology that underpins Mitopia's ontology was (and still is) known as the "Type Manager". I first learned the word "ontology" during a presentation to a group of approximately 40 individuals at a US Intelligence Agency during December 2000 . At that presentation, I was describing the need for a dynamic and adaptive approach to knowledge representation (like Mitopia's) in any large and complex intelligence system, when one of the audience members pointed out that my approach had parallels with the use of an ontology. Not knowing what an ontology was at the time, I could not comment on this statement. However, in researching the meaning of the word (and thence the world of semantic ontologies) after the presentation, I came to realize that Mitopia's approach was indeed "ontological" though at a very fundamental level quite different (having been in development in complete ignorance of ontologies for nearly 10 years). Since the word ontology appeared to be understood by others, I resolved at that time to call the relatively small portion of known types (defined using the "type manager") that related primarily to 'persistent data' (i.e., the database in conventional terminology) the system 'Ontology'. The larger aspects of type management was, and still are, referred to simply using the term "type manager". Given the different motivations behind, and the 'clean room' development history of the Mitopia® ontology technology, it is not surprising therefore, that there are huge, and very significant, differences between Mitopia's unique ontology system, and any conventional semantic ontologies with which the reader may be familiar.

The motivations that drove the specification of Mitopia's type manager (Mitopia's ontology 'engine') in the original requirements document (SDS) for the first Mitopia® based system (written in 1991-3) were unrelated to ontological thinking, but were instead caused by two main drivers. The first driver was the goal of making Mitopia® a data-flow based rather than a normal control-flow based system, and the second was the goal of system adaptability and creating a strategy to overcome the "Software Bermuda Triangle" effect. Fully 2/3 of the software pages and requirements in the SDS were directly targeted at these two goals, and were built on the type manager requirement. Both of these design goals resulted from an in-depth analysis of the reasons for the failure of an earlier system (developed from 1983-1988). In mid 1988 I was brought in to try to salvage that earlier system, a task which it soon became clear was impossible. As a result, during the entirety of 1989, I had the opportunity and mandate to do nothing but reflect on the causes for that failure and think/experiment on how one might design a system that could overcome these limitations, albeit with a delivery date that might extend into the next century. The result of this process was a presentation and a series of study documents that eventually led in 1991 to the first version of the SDS. To understand the motivations underlying Mitopia's ontological approach, it is therefore necessary to understand the two primary drivers that led to it.

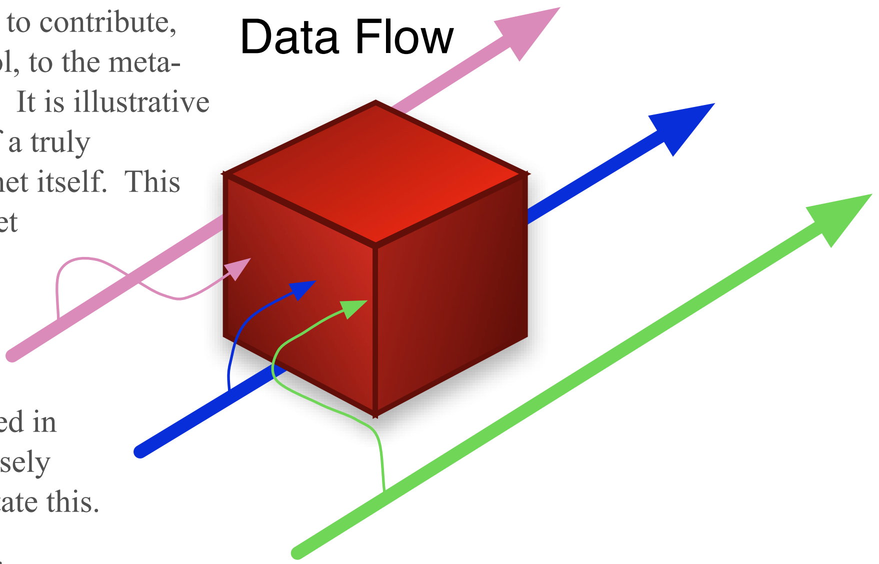
Data Flow vs. Control Flow

To a large extent, the decision to specify a data-flow based system as opposed to a control-flow based one was driven by a common higher level "adaptability" goal that also led to identifying and addressing the "Software Bermuda Triangle" issue. In particular, for complex systems, such as those designed for multimedia intelligence and knowledge management applications, there is a fundamental problem with current 'control flow' based design methods that renders them totally unsuited for these kinds of systems. Once the purpose of a system is broadened to acquisition of unstructured, non-tagged, time-variant, multimedia information (much of which is designed specifically to prevent easy capture and normalization by non-recipient systems), a totally different approach is required. In this arena, many entrenched notions of information science and database methodology must be discarded to permit the problem to be addressed. We will call systems that attempt to address this level of problem 'Unconstrained Systems' (UCS). A UCS is one in which the source(s) of data have no explicit or implicit knowledge of, or interest in, facilitating the capture and subsequent processing of that data by the system. To summarize the principal issues that lead one to seek a new paradigm to address unconstrained systems, they are as follows:



- Change is the norm. The incoming data formats and content will change. The needs and requirements of the users of the data will change, and this will be reflected not only in their demands of the UI to the system, but also in the data model and field set that is to be captured and stored by the system.
- An unconstrained system can only sample from the flow going through the pipe that is our digital world. It is neither the source nor the destination for that flow, but simply a monitoring station attached to the pipe capable of selectively extracting data from the pipe as it passes by.
- The system cannot 'control' the data that impinges on it. Indeed, we must give up any idea that it is possible to 'control' the system that the data represents. All we can do is monitor and react to it. This step of giving up the idea of control is one of the hardest for most people, especially software engineers, to take. After all, we have all grown up learning that software consists of a 'controlling' program which takes in inputs, performs certain predefined computations, and produces outputs. Every installed system we see out there complies with this world view, and yet it is obvious from the discussion above that this model can only hold true on a very localized level in a UCS. The flow of data through the system is really in control. It must trigger execution of code as appropriate depending on the nature of the data itself. That code must be localized and autonomous. It cannot cause or rely upon tendrils of dependency without eventually clogging up the pipe. The concept of data initiating control (or program) execution rather than the other way is alien to most programmers, and yet it becomes fundamental to addressing unconstrained systems.

We cannot in general predict what algorithms or approaches are appropriate to solving the problem of 'understanding the world'. The problem is simply too complex. Once again we are forced away from our conventional approach of defining processing and interface requirements, and then breaking down the problem into successively smaller and smaller sub-problems. Again, it appears that this uncertainly forces us away from any idea of a control-based system and into a model where we must create a substrate through which data can flow and within which localized areas of control flow can be triggered by the presence of certain data. The only practical approach to addressing such a system is to focus on the requirements and design of the substrate and trust that by facilitating the easy incorporation of new plug-in control flow based 'widgets' and their interface to data flowing through the data-flow based substrate, it will be possible for those using the system to develop and 'evolve' it towards their needs. In essence, the users, knowingly or otherwise, must teach the system how they do what they do as a side effect of expressing their needs to it. Experience shows that any more direct attempt to extract knowledge from users or analysts to achieve computability, is difficult, imprecise, and in the end contradictory and unworkable. No two analysts will agree completely on the meaning of a set of data, nor will they concur on the correct approach to extracting meaning from data in the first place. Because all such perspectives and techniques may have merit, the system must allow all to co-exist side by side, and to contribute, through a formalized substrate and protocol, to the meta-analysis that is the eventual system output. It is illustrative to note that the only successful example of a truly massive software environment is the Internet itself. This success was achieved by defining a rigid set of protocols (IP, HTML etc.) and then allowing Darwinian-like and unplanned development of autonomous but compliant systems to develop on top of the substrate. A similar approach is required in the design of unconstrained systems. Loosely coupled data-flow based approaches facilitate this.



The most basic change that must be made is to create an environment that operates according to data-flow rules, not those of a classic control-flow based system. At the most fundamental operating system scheduling level, we need an environment where presence of suitable data initiates program execution, not the other way round.

Data-flow based software design and documentation techniques have been in common usage for many years. In these techniques, the system design is broken into a number of distinct processes and the data that flows between them. This breakdown closely matches the perceptions of the actual system users/customers and thus is effective in communicating the architecture and requirements. Unfortunately, due to the lack of any suitable data-flow based substrate, even software designs created in this manner are invariably translated back into control-flow methods, or at best to message passing schemes, at implementation time. This translation begins a slippery slope that results in such software being of limited scope and largely inflexible to changes in the nature of the flow. This problem is at the root of why software systems are so expensive to create and maintain.

A critical realization when following the reasoning above, was that the islands of computation that are triggered by the presence of matching data in a functional data-flow based system must express their data needs to the outside world through some kind of formalized pin-based interface such that the 'types' of the data that is required to cause an arriving 'token' to appear on any given pin must be discovered at run-time by the system substrate responsible for triggering the code island when suitable data arrives. This clearly means that such a system will require a run-time discoverable types system as opposed to a compile-time types system as found in common programming languages including OOP. Since we might want to pass any type whatsoever to a given code island ('widget' in Mitopia® parlance), this implies that the run-time type system must be capable of compiling all the C header files for the underlying OS in addition to any additional types that might be specific to any given application of the system (i.e., the system ontology). The bottom line then is that such a system must implement a full C type compiler together with a type manager API that is capable of describing, discovering, and manipulating the type of any data occurring within system data flows. The result being that compiled structure types defined using C itself would be identical and interchangeable in binary form to those generated by the type manager. If such a system were to be created, and if all system code were to access data through the type manager, rather than by direct compiled access in the code, then one would have a substrate for which it were possible to have ZERO compiled types in the code that had anything to do with the actual application (or its ontology) and which would thus be completely generic and highly adaptive. Moreover, such a substrate is a necessary precursor to building a data-flow based system capable of dynamic run-time wiring and adaptability mediated by the underlying architecture without any need for individual widgets to know or care what they are connected to in the larger system.

This then set the stage for the first radical departure in Mitopia's type manager (i.e., ontology system), which is that the ontology description language (ODL) must manipulate binary data through run-time type discovery, and must include and extend the C language itself. This is unlike any other ontology language in existence. Thus C headers specifying C binary structures can be handled natively by type manager based code, that is, there is no need for any intermediate textual forms associated with the ontology or the data it describes. All operations, including textual operations, operate on data held natively in its binary form. This of course provides massive performance benefits at the cost of introducing considerable complexity into the substrate, including the need to be aware of tricky issues such as alignment and byte ordering (e.g., big endian vs. little endian) when operating in a heterogeneous data-flow based network. Conventional semantic markup ontology languages, as stated previously, do not address actual storage or implementation of data, but instead allocate "attributes" to a "class", where these attribute are textual and have no relation to any existing programming language, or any means of actually storing them in a binary form. Thus Mitopia's type manager is not a markup language, it is instead a run-time typed interpreted language that includes and extends upon the C programming language.

Initial experimentation on this concept of data-flow in 1989 and 1990 using a transputer-based co-processor card led to additional realizations regarding the needs of such a data-flow system that further focused the evolution of Mitopia's type system and ODL. The primary realization was that in any large scale data-flow based system, there is a need to pass not only isolated structures, but also whole collections of structures all cross referencing each other in various ways. Conventionally, this would require the duplication of any complex collection by following all the pointers buried within it and duplicating them while updating the references, so that the passed copy of the collection would be functionally identical though completely separate from the original (i.e., pass by value not reference). It rapidly became clear that this duplication and replication process becomes the dominant constituent of CPU load in any data-flow system passing significant numbers of such compound structures across its flows. The problem is exacerbated dramatically as the data is passed from one CPU/address space to another, since this requires that the data be serialized into an intermediate textual form and then de-serialized and re-assembled in the new address space. This issue is faced to a much lesser degree by conventional systems that share data, and is usually addressed by serializing to/from XML as the textual intermediate. Unfortunately the timing overhead for this is unacceptable for a data-flow based system. The binary/text based performance slowdown factor of 275 discussed earlier rears its ugly head in this serialization scenario.

As a result of thinking about this issue, Mitopia's unique memory model as embodied most visibly in the "Type Collections" technology eventually emerged. The basic technique was to discourage the use of pointers and replace them with 'relative' references so that all data in a collection, including all the cross references, could be contained and manipulated within a single variable sized memory allocation using the "type collections" abstraction. Nodes within such a collection have associated types as discovered and defined by the type manager. This allows the entire binary collection to be passed across data flows (local or remote) without any modification whatsoever, while still being completely functional at the destination thereby eliminating the performance bottleneck. The full details of this technology and approach are described more fully in other Mitopia® documentation and are beyond the scope of this discussion, however, the result of this extension to the type system led to the initial extensions to the C programming language to describe references between items (e.g., '@' for a relative reference field).

The final piece of the puzzle dropped into place once one adds the need for an external database in order to store, query, and recover the actual data that was part of the system persistent data content (what would later be referred to as the system ontology). At the time, the approach was to use the Oracle relational database as the principal member of a federation of different database containers (e.g., inverted file text engine, GIS etc.), and thus it became necessary to create a large and complex 'glue' layer that converted types defined in the system ontology (I.DEF), and discovered using the type manager, to/from the corresponding relational database tables and cells. This glue also implemented the querying capability, again by field type discovery using the type manager, followed by conversion to the equivalent SQL. The glue layer became one of the largest blocks of code in the system outside the core Mitopia® technology and was known as MitoSQL. MitoSQL disappeared completely in 2003 as all server functionality was subsumed within Mitopia®, and the relational database became the last container to be eliminated.

The MitoSQL layer needed to be able to fetch a set of data from the database and put it into a collection using the type collections abstraction, so that it could be passed across data flows and manipulated by other widgets. However, the original SDS database specifications included many fields containing explicit references to data items of different types (e.g., from a person's employment field to an employer organization) and these were described as "unique database ID refs". This concept is of course quite clear in relational database, and in conventional systems, this is where the system 'glue' code gets involved in order to handle the incompatible data model between in-memory operations and those of a relational database. It is this glue code or 'dark matter' that eventually spells the demise of conventional systems in the face of subsequent change. However, in the Mitopia® approach, the data-flow and type manager requirements did not permit this kind of 'glue' to be passed over a data-flow boundary, and so it was necessary to find some way to describe and implement these database references within the type manager, and thence in the type collections abstraction. This would permit database derived results to be passed around and manipulated over data flows while freely intermixed with type manager data originating from other sources. In late 1992 this problem more than any other, led to the beginning of Mitopia's ontology approach (though the word was still unknown to me then).

The first realization was that to address this transformation, all records derived from the database, that is, all records that are part of the ontology, must contain a number of 'required' fields, not the least of which would be the 'unique ID' associated with persistent storage. It became clear that Mitopia's ODL must support inheritance of types and type fields, and that all persistent data must be derived directly or indirectly from an ancestral type referred to as "Datum" that would be used to hold such fields. In looking at the customer database record requirements, it became obvious that they were not organized in any general manner, but were instead totally application specific (as is normal in system specification). This of course did not fit with the needs of creating a data-flow based system, and thus instead of directly mapping database records to types, I instead set about trying to organize the types derived from Datum so that they could be used when in the collections in a meaningful way for understanding the world, rather than simply as a set of disconnected records. This was the time at which the organization of Mitopia's base ontology was defined. The tension that resulted from my desire to organize data in a 'cognitive' manner to allow a data-flow based system to be created, and the natural desire of those responsible for implementing the relational database 'glue' to keep the difference between the two representations to a minimum, and to focus on the customer specifications, led to some heated debates. Inheritance is not a concept that fits into the relational paradigm.

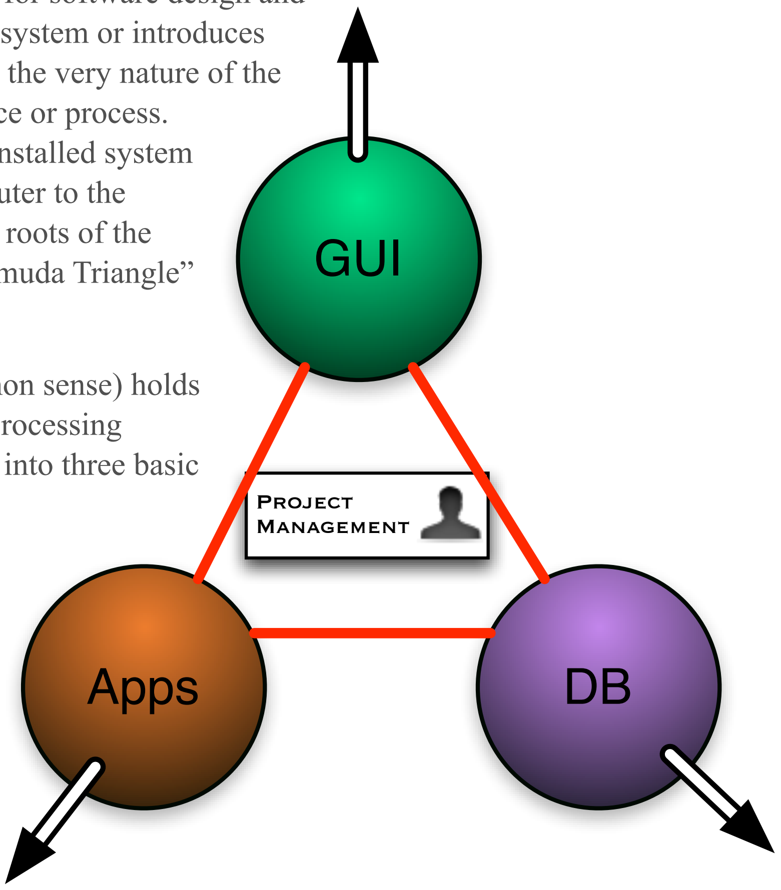
Next it was clearly necessary to extend the language to add the concept of a persistent reference (i.e., a '#' field) to implement one-to-one database references, as well as a collection reference (i.e., a '##' field) to implement one-to-many database references. Both of these constructs were added to the ODL language and implemented within the framework as a hidden reference structure containing unique ID(s) as well as a relative reference(s), so that once the data had been fetched from the database and instantiated into the collection, the relative reference field could be used to directly navigate to the collection copy. This approach ensured that code using the type collections abstraction was essentially unaware if the data was in memory, or needed to be fetched from the database (which of course happened automatically if any attempt was made to access data via the reference).

This then completed the migration of Mitopia's type manager system from a conventional (though run-time discoverable) programming language, to an ontology description language (ODL) that was also focused on the need to represent persistent data and the kinds of arbitrary relationships that can exist between persistent data records. Of course much evolution of the language and technology would follow, as additional issues were identified and overcome, but in essence by 1993, Mitopia's type manager had become an ODL. The next major leap forward in capabilities did not occur until 2002-3 when the relational database itself was eliminated and additional responsibilities moved into the ODL.

Mitopia's ontology description language is called 'Carmot'. For a long time alchemists believed that a key component of the legendary "philosopher's stone" was the mythical element carmot. The philosopher's stone, it was believed, had the power to transform base metals to gold. In an information sense, Mitopia's Carmot ODL is the key component that allows Mitopia® to accomplish the unique things it does in transforming information into knowledge. Carmot declarations are normally distinct from the Carmot code that uses them (e.g., within MitoMine™) and for this reason, the Carmot language actually has two variants, Carmot-D (for declarations - this is the ODL variant discussed in this document), and Carmot-E (for execution - the language of running Carmot code such as within MitoMine™ and other Carmot-based interpreters within Mitopia®). Carmot-E is covered by other Mitopia® documentation, and is not discussed extensively herein. The 'D' and 'E' variants are usually dropped in use and both forms are referred to simply as Carmot.

The Software Bermuda Triangle

It is obvious that for any system connected to the external world, change is the norm, not the exception. The outside world does not stand still just to make it convenient for us to monitor it. Moreover, in any system involving multiple analysts with divergent requirements, even the data models and requirements of the system itself will be subject to continuous and pervasive change. By most estimates, more than 90% of the cost and time spent on software is devoted to maintenance and upgrade of the installed system to handle the inevitability of change. Even our most advanced techniques for software design and implementation fail miserably as one scales the system or introduces rapid change. The reasons for this failure lie in the very nature of the currently accepted software development practice or process. Analysis of the causes for failure of the earlier installed system showed that this effect was a significant contributer to the problem. The graphic to the right illustrates the roots of the problem, which we shall call the "Software Bermuda Triangle" effect.



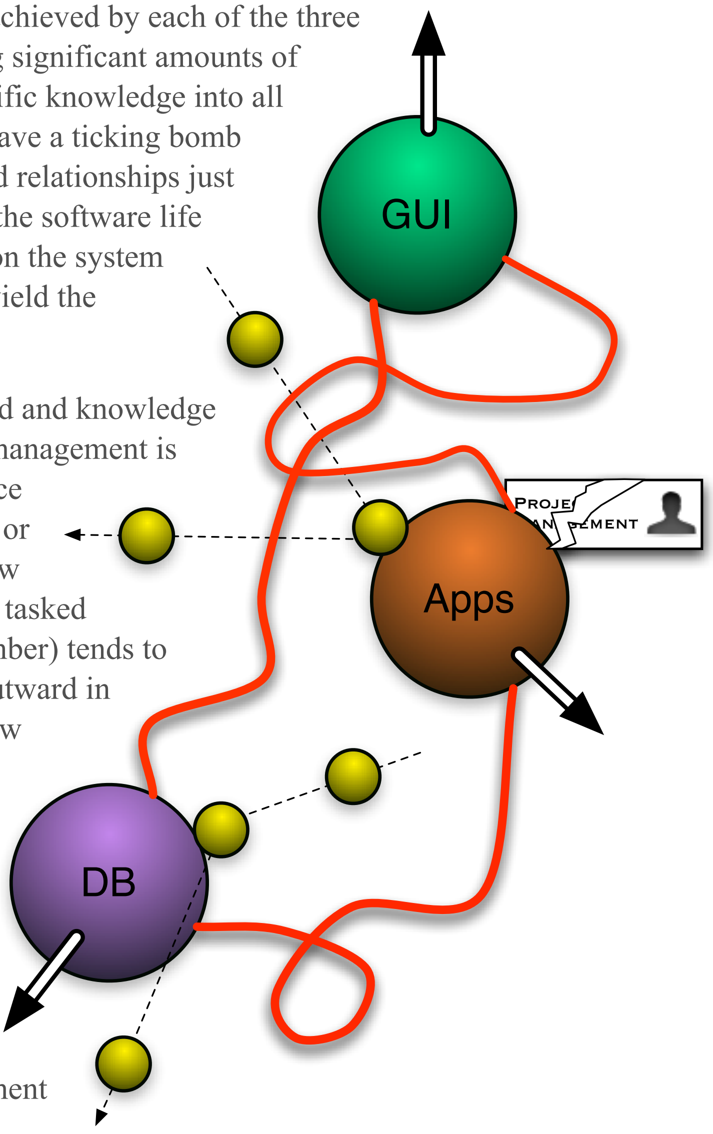
Conventional programming wisdom (and common sense) holds that during the design phase of an information processing application, programming teams should be split into three basic groups.

The first group (labeled "DB") is the database group. These individuals are experts in database design, optimization, and administration. This group is tasked with defining the database tables, indexes, structures, and querying interfaces based initially on requirements, and later, on requests primarily from the Applications ("Apps") group. These individuals are highly trained in database techniques and tend naturally to pull the design in this direction, as illustrated by the outward pointing arrow in the diagram.

The second group is the Graphical User Interface ("GUI") group. The GUI group is tasked with implementing a user interface to the system that operates according to the customer's expectations and wishes, and yet complies exactly with the structure of the underlying data (DB group) and the application behavior (Apps. group). The GUI group will have a natural tendency to pull the design in the direction of richer and more elaborate user interfaces.

Finally the Applications group is tasked with implementing the actual functionality required of the system by interfacing with both the DB and the GUI groups and Applications Programming Interfaces ("API"). This group, like the others, tends to pull things in the direction of more elaborate system specific logic.

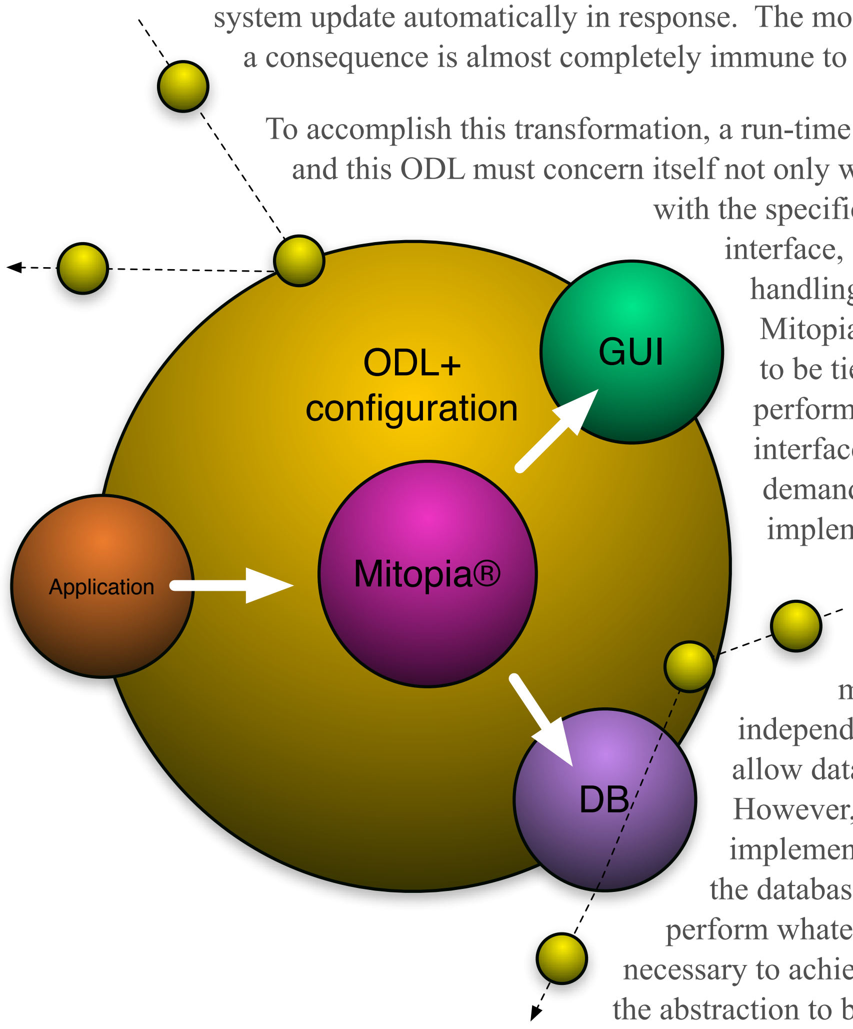
Each of these groups tends to have no more than a passing understanding of the issues and needs of the other groups. Thus, during the design phase, and assuming we have strong project and software management that rigidly enforces design procedures, we have a relatively stable triangle where the strong connections enforced between each group by management (represented by the lines joining each group in the diagram), are able to overcome the outward pull of each member of the triangle. Assuming a stable and unchanging set of requirements, such an arrangement stands a good chance of delivering a system to the customer on time. The reality, however, is that correct operation has been achieved by each of the three groups in the original development team embedding significant amounts of undocumented application, GUI, and database-specific knowledge into all three of the major software components. We now have a ticking bomb comprised of these subtle and largely undocumented relationships just waiting to be triggered. After delivery (the bulk of the software life cycle), in the face of the inevitable changes forced on the system by the passage of time, the system breaks down to yield the situation illustrated to the right.



The state now is that the original team has disbanded and knowledge of the hidden dependencies is gone. Furthermore, management is now in a monitoring mode only. During maintenance and upgrade phases, each change hits primarily one or two of the three groups. Time pressures, and the new development environment, mean that the individual tasked with the change (probably not an original team member) tends to be unaware of the constraints, and naturally pulls outward in his particular direction. The binding forces have now become much weaker and more elastic, while the forces pulling outwards remain as strong. All it takes is a steady supply of changes impacting this system for it to break apart and tie itself into knots. Some time later, the system grinds to a halt or becomes unworkable or not modifiable. The customer must either continue to pay progressively more and more outrageous maintenance costs (swamping the original development costs), or must start again from scratch with a new system and repeat the cycle. The latter approach is often much easier than the former. This effect is central to why software systems are so expensive. Since change of all kinds is pervasive in an intelligence system, an architecture for such systems must find some way to address and eliminate this Software Bermuda Triangle effect.

If we wish to tackle the Bermuda Triangle effect, it was clear from the outset that the first step is to reduce the molecule from three components to one, so that when change impacts the system, it does not force the molecule to warp and eventually break as illustrated in the preceding diagram. Clearly since application specific logic cannot be avoided, is was necessary to find a way whereby both the database (or in Mitopia® parlance, the persistent storage) and the user interface could be automatically generated from the system ontology at run time and not compile time. The strategy was thus to transform the molecule to the form depicted below.

In this approach, the application-specific requirements are specified to the central Mitopia® engine primarily through the ontology itself using the ODL, and to a lesser extent through a number of other configuration metaphors, not through code. The central Mitopia® engine is capable of automatically generating and handling the GUI as well as the database functionality entirely from the ODL. This means that change can no longer directly impact the code of the GUI and DB since these are entirely driven by Mitopia®, based on the ODL, and thus all change hits the ODL/configuration layer, so once that has been updated, all aspects of the system update automatically in response. The molecule is now rigid and highly adaptive, and as a consequence is almost completely immune to the Bermuda Triangle effect.



To accomplish this transformation, a run-time discoverable types system (ODL) is essential and this ODL must concern itself not only with basic type definition and access, but also with the specifics of how types are transformed into user interface, as well as how they specify the content and handling of system persistent storage. For this reason Mitopia's ontology, unlike semantic ontologies, had to be tied to binary data storage, had to have the performance necessary to implement complex user interfaces through run-time discovery, and most demanding of all, had to have the ability to directly implement a high performance scaleable database architecture. All code within both Mitopia®, the GUI, and the persistent storage is forced to access and handle data through the type manager abstraction in order to preserve code independence from the application specifics, and to allow data to be passed and understood across flows. However, because the Mitopia® core code itself implements the key type manager abstraction as well as the database and GUI use of that abstraction, it is able to perform whatever optimization and cacheing steps are necessary to achieve maximum performance without allowing the abstraction to be broken by external non-private code. This is a critical distinction from the object-oriented approach to data abstraction. Another critical benefit is that this approach unifies the programming model for handling data in memory with that used to access it in a database, and that associated with its display. By eliminating all the custom glue code normally associated with these transformations (which is an estimated 60% or more of any large system code base), we have drastically improved system reliability and adaptability, and reduced development time and costs associated with creating new systems.

There is considerable overlap then in the ODL requirements driven by the needs of a data-flow based system (as discussed previously), and those driven by the attempt to eliminate the Bermuda Triangle effect. The additional requirements driven by the Bermuda Triangle include the need to generate and handle user interfaces, the need to provide some means of specifying application dependent behaviors for the types and fields in the ontology (i.e., ODL based type and field scripts and annotations), and the need to specify the form and topology of the system's distributed persistent storage (i.e., database). Once again, these requirements are quite different from the focus or intent of a semantic ontology, and resulted in Mitopia's ontology being quite distinct from the main evolutionary branch of ontological technology today which is semantic/linguistic.

Towards an Ontology of Everything

As discussed in the preceding section, once having settled on an ontological approach to organizing information within a Mitopia® system around 1993, the next question that had to be addressed, given that Mitopia® is an architecture, and not focussed on any particular application, was “what kind of upper level of organization is appropriate for laying the foundations of an Ontology of Everything (OOE)”?. The corollary question “What fundamental process do we use to extract meaning from data organized in this manner”? must also be asked. Before describing the approach taken in Mitopia®, it is first perhaps instructive to review the history and current state of upper ontologies in the semantic realm for comparative purposes, even though as stated previously Mitopia’s ontology evolved in complete ignorance of other work in the ontology field. The following discussion is taken from the Wikipedia article on the subject of upper ontologies.

Upper ontologies are commercially valuable, creating competition to define them. Peter Murray-Rust has claimed that this leads to "semantic and ontological warfare due to competing standards", and accordingly any standard foundation ontology is likely to be contested among commercial or political parties, each with their own idea of 'what exists'.

No one upper ontology has yet gained widespread acceptance as a de facto standard. Different organizations are attempting to define standards for specific domains. The 'Process Specification Language' (PSL) created by the National Institute for Standards and Technology (NIST) is one example.

There is debate over whether the concept of using a single, shared upper ontology is even feasible or practical at all. There is further debate over whether the debates are valid - often leading to outright censorship and boosterism of particular approaches in supposedly neutral sources including this one. Some of these arguments are outlined below, with no attempt to be comprehensive. Please do not censor them because you promote some ontology.

Why an upper ontology is not feasible

Historically, many attempts in many societies have been made to impose or define a single set of concepts as more primal, basic, foundational, authoritative, true or rational than others. In the kind of modern societies that have computers at all, the existence of academic and political freedoms imply that many ontologies will simultaneously exist and compete for adherents. While the differences between them may be narrow and appear petty to those not deeply involved in the process, so too did many of the theological debates of medieval Europe, but they still led to schisms or wars, or were used as excuses for same. The tyranny of small differences that standard ontologies seek to end may continue simply because other forms of tyranny are even less desirable. So private efforts to create competitive ontologies that achieve adherents by virtue of better communication may proceed, but tend not to result in long standing monopolies.

A deeper objection derives from ontological constraints that philosophers have found historically inescapable. Some argue that a transcendental perspective or omniscience is implied by even searching for any general purpose ontology since it is a social / cultural artifact, there is no purely objective perspective from which to observe the whole terrain of concepts and derive any one standard.

A narrower and much more widely held objection is implicature: the more general the concept and the more useful in semantic interoperability, the less likely it is to be reducible to symbolic concepts or logic and the more likely it is to be simply accepted by the complex beings and cultures relying on it. In the same sense that a fish doesn't perceive water, we don't see how complex and involved is the process of understanding basic concepts.

There is no self-evident way of dividing the world up into concepts, and certainly no non-controversial one

There is no neutral ground that can serve as a means of translating between specialized (or "lower" or "application-specific") ontologies

Human language itself is already an arbitrary approximation of just one among many possible conceptual maps. To draw any necessary correlation between English words and any number of intellectual concepts we might like to represent in our ontologies is just asking for trouble. (WordNet, for instance, is successful and useful precisely because it does not pretend to be

a general-purpose upper ontology; rather, it is a tool for semantic / syntactic / linguistic disambiguation, which is richly embedded in the particulars and peculiarities of the English language.)

Any hierarchical or topological representation of concepts must begin from some ontological, epistemological, linguistic, cultural, and ultimately pragmatic perspective. Such pragmatism does not allow for the exclusion of politics between persons or groups, indeed it requires they be considered as perhaps more basic primitives than any that are represented.

Those who doubt the feasibility of general purpose ontologies are more inclined to ask "what specific purpose do we have in mind for this conceptual map of entities and what practical difference will this ontology make?" This pragmatic philosophical position surrenders all hope of devising the encoded ontology version of "everything that is the case," Wittgenstein, Tractatus Logico-Philosophicus).

According to Barry Smith in The Blackwell Guide to the Philosophy of Computing and Information (2004), "the project of building one single ontology, even one single top-level ontology, which would be at the same time non-trivial and also readily adopted by a broad population of different information systems communities, has largely been abandoned." (p. 159)

Finally there are objections similar to those against artificial intelligence; Technically, the complex concept acquisition and the social / linguistic interactions of human beings suggests any axiomatic foundation of "most basic" concepts must be cognitive, biological or otherwise difficult to characterize since we don't have axioms for such systems. Ethically, any general-purpose ontology could quickly become an actual tyranny by recruiting adherents into a political program designed to propagate it and its funding means, and possibly defend it by violence. Historically, inconsistent and irrational belief systems have proven capable of commanding obedience to the detriment of harm of persons both inside and outside a society that accepts them. How much more harmful would a consistent rational one be, were it to contain even one or two basic assumptions incompatible with human life?

Why an upper ontology is feasible

Most of the objections to upper ontology refer to the problems of life-critical decisions or non-axiomatized and difficult to understand problem areas such as law or medicine or politics. Some of these objections do not apply to infrastructure or standard abstractions that are defined into existence by human beings and closely controlled by them for mutual good, such as electrical power system connections or the signals used in traffic lights. No single general metaphysics is required to agree that some such standards are desirable. For instance, while time and space can be represented many ways, some of these are already used in interoperable artifacts like maps or schedules.

Most proponents of an upper ontology argue that several good ones may be created with perhaps different emphasis. Very few are actually arguing to discover just one within natural language or even an academic field. Most are simply standardizing some existing communication.

Several common arguments against upper ontology can be examined more clearly by separating issues of concept definition (ontology), language (lexicons), and facts (knowledge). For instance, people have different terms and phrases for the same concept. However, that does not necessarily mean that those people are referring to different concepts. They may simply be using different language or idiom. Formal ontologies typically use linguistic labels to refer to concepts, but the terms mean no more and no less than what their axioms say they mean. Labels are similar to variable names in software, evocative rather than definitive.

A second argument is that people believe different things, and therefore can't have the same ontology. However, people can assign different truth values to a particular assertion while accepting the validity of certain underlying claims, facts, or way of expressing an argument with which they disagree. Using, for instance, the issue/position/argument form.

Even arguments about the existence of a thing require a certain sharing of a concept, even though its existence in the real world may be disputed. Separating belief from naming and definition also helps to clarify this issue, and show how concepts can be held in common, even in the face of differing belief. For instance, wiki as a medium may permit such confusion but disciplined users can apply dispute resolution methods to sort out their conflicts, e.g. Wikipedia ArbCom.

Advocates argue that most disagreement about the viability of an upper ontology can be traced to the conflation of ontology, language and knowledge, or too-specialized areas of knowledge: many people, or agents or groups will have areas of their respective internal ontologies that do not overlap. If they can cooperate and share a conceptual map at all, this may be so very useful that it outweighs any disadvantages that accrue from sharing. To the degree it becomes harder to share concepts the

deeper one probes, the more valuable such sharing tends to get. If the problem is as basic as opponents of upper ontologies claim, then, it applies also to a group of humans trying to cooperate, who might need machine assistance to communicate easily.

If nothing else, such ontologies are implied by machine translation, used when people cannot practically communicate. Whether "upper" or not, these seem likely to proliferate.

From the discussion above it should be obvious that in the world of semantic ontologies, based as they are on human language, the prospects of ever agreeing on an upper ontology in order to enable any truly global exchange of ideas has almost been abandoned. This fact notwithstanding, there are essentially just two viable upper ontologies existing in the semantic web, the first being the Cyc Upper Ontology, and the second being the Suggested Upper Merged Ontology (SUMO). Remember that these are semantic ontologies, that is ontologies for organizing words and linguistic meaning, not for organizing or representing data. Thus we see that immediately below the root of either upper ontology, things immediately begin to split into some very esoteric sounding groups, many of which it would require a dictionary to even begin to understand. These groupings are driven by philosophical considerations of a very obscure nature, and thus it would be very hard for the average (or even far above average) person to place a given data record type anywhere within such an ontology. Try it yourself by looking at the upper ontology diagrams given on the following pages and then trying to place the following into one of the boxes shown:

(1) A dog

(2) A contract between two organizations

(3) A news story or document

(4) A country

Difficult isn't it? But these are exactly the kinds of things one might track in any system designed to understand real world events. This is one of the fundamental problems with disjoint model ontologies. Since they divorce themselves from any consideration of actually representing, storing or manipulating data, they tend to evolve on courses charted by philosophical, rather than practical, considerations, with the result that they become useless to anyone other than those few geeks that are really into ontologies. This explains the tiny adoption rates for semantic ontology-based systems, despite having been in development for more than twenty years.

The Cyc Upper Ontology

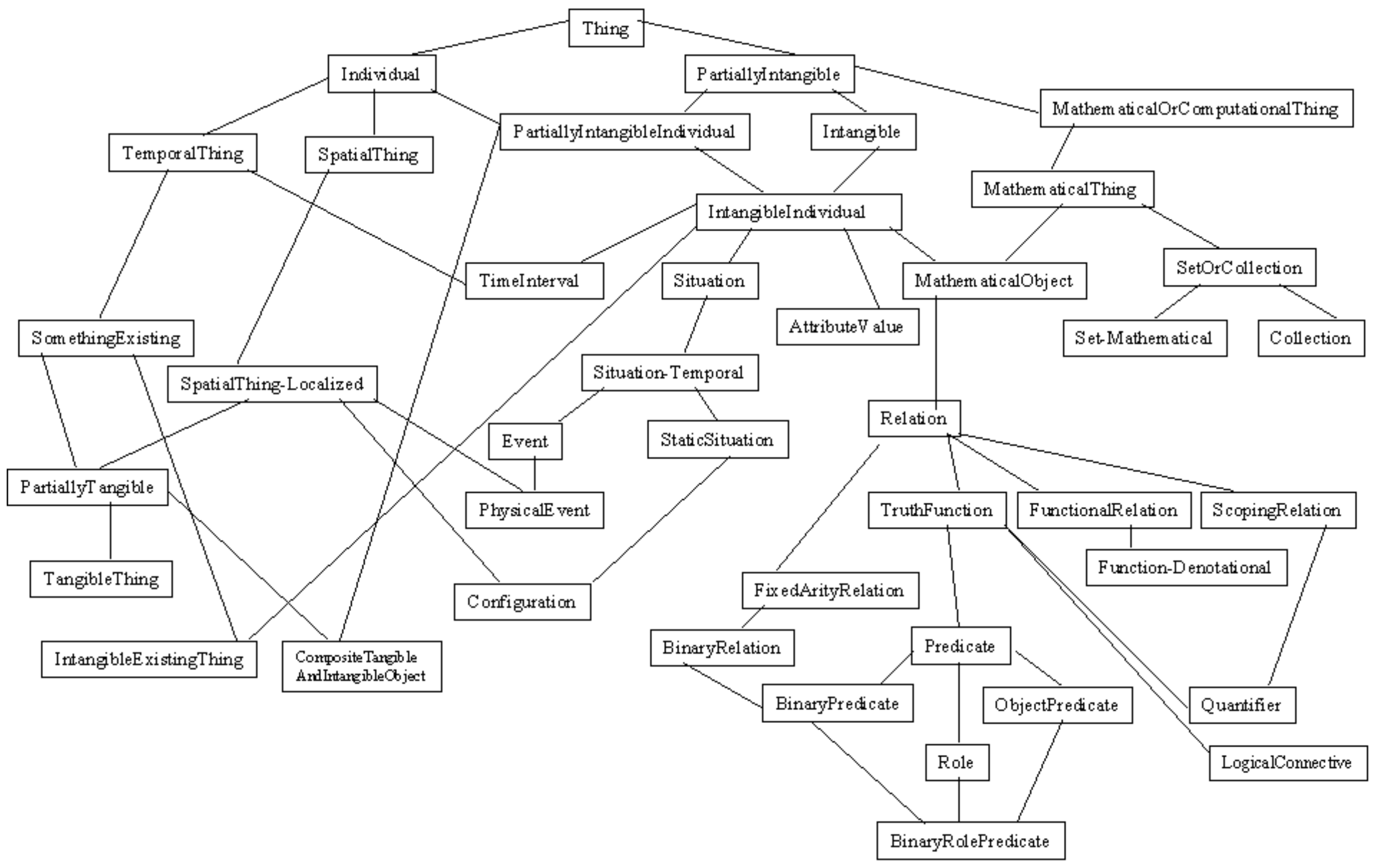


Figure 5 - The Cyc Upper Ontology

Figure 5 above shows the top level of the Cyc upper ontology. Cyc is an artificial intelligence project that attempts to assemble a comprehensive ontology and database of everyday common sense knowledge, with the goal of enabling AI applications to perform human-like reasoning.

The project was started in 1984 by Doug Lenat as part of Microelectronics and Computer Technology Corporation. The name "Cyc" (from "encyclopedia", pronounced like psych) is a registered trademark owned by Cycorp, Inc. in Austin, Texas, a company run by Lenat and devoted to the development of Cyc. The original knowledge base is proprietary, but a smaller version of the knowledge base, intended to establish a common vocabulary for automatic reasoning, was released as OpenCyc under an open source license. More recently, Cyc has been made available to AI researchers under a research-purposes license as ResearchCyc.

Typical pieces of knowledge represented in the database are "Every tree is a plant" and "Plants die eventually". When asked whether trees die, the inference engine can draw the obvious conclusion and answer the question correctly. The Knowledge Base (KB) contains over a million human-defined assertions, rules or common sense ideas. These are formulated in the language CycL, which is based on predicate calculus and has a syntax similar to that of the Lisp programming language.

Much of the current work on the Cyc project continues to be knowledge engineering, representing facts about the world by hand, and implementing efficient inference mechanisms on that knowledge. Increasingly, however, work at Cycorp involves giving the Cyc system the ability to communicate with end users in natural language, and to assist with the knowledge formation process via machine learning.

The Suggested Upper Merged Ontology (SUMO)

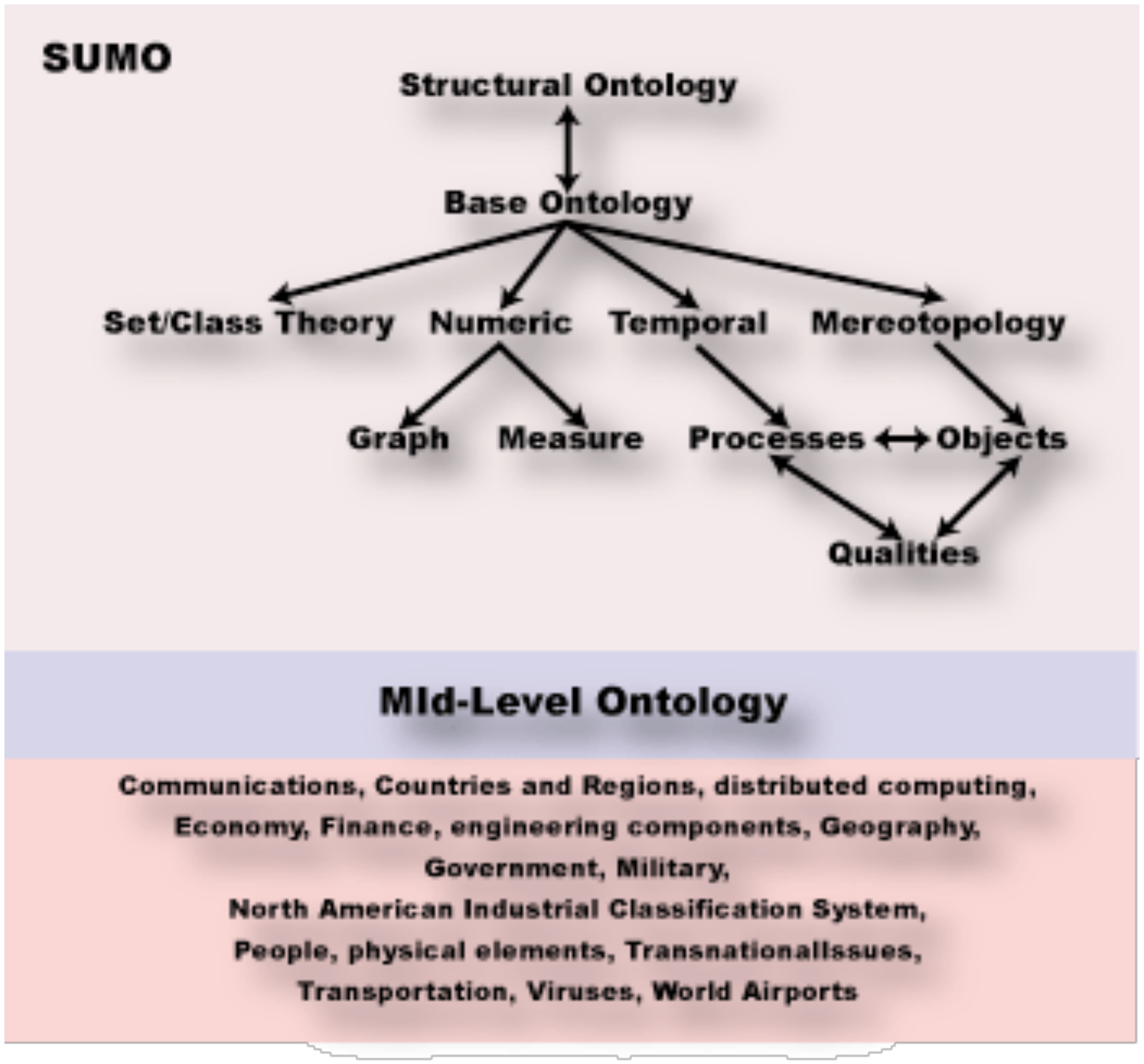

SUMO
Structural Ontology
Base Ontology
Set/Class Theory
Numeric
Temporal
Mereotopology
Graph
Measure
Processes
Objects
Qualities
MId-Level Ontology
Communications, Countries and Regions, distributed computing,
Economy, Finance, engineering components, Geography,
Government, Military,
North American Industrial Classification System,
People, physical elements, TransnationalIssues,
Transportation, Viruses, World Airports

Figure 6 - The Suggested Upper Merged Ontology (SUMO)

The Suggested Upper Merged Ontology or SUMO is an upper ontology intended as a foundation ontology for a variety of computer information processing systems. It was originally developed by the Teknowledge Corporation and now is maintained by Articulate Software. It is one candidate for the "standard upper ontology" that IEEE working group 1600.1 is working on. It can be downloaded and used freely.

SUMO originally concerned itself with meta-level concepts (general entities that do not belong to a specific problem domain), and thereby would lead naturally to a categorization scheme for encyclopedias. It has now been considerably expanded to include a mid-level ontology and dozens of domain ontologies.

SUMO was first released in December 2000. It defines a hierarchy of SUMO classes and related rules and relationships. These are formulated in a version of the language SUO-KIF which has a LISP-like syntax. A mapping from WordNet synsets to SUMO has also been defined.

SUMO is organized for interoperability of automated reasoning engines. To maximize compatibility, schema designers can try to assure that their naming conventions use the same meanings as SUMO for identical words, (eg: agent, process). SUMO has an associated open source Sigma knowledge engineering environment.

As can be seen from Figure 6, SUMO is really more of a grab bag of individual ontologies that loosely fit into parts of an upper framework. It is ontology by accretion, not design, and lacks much of the consistency of the Cyc Upper Ontology because of the open source style in which it has evolved. SUMO and its domain ontologies form the largest formal public ontology in existence today. They are being used for research and applications in search, linguistics and reasoning. SUMO is the only formal ontology that has been mapped to all of the WordNet lexicon. The ontologies that extend SUMO are available under GNU General Public License.

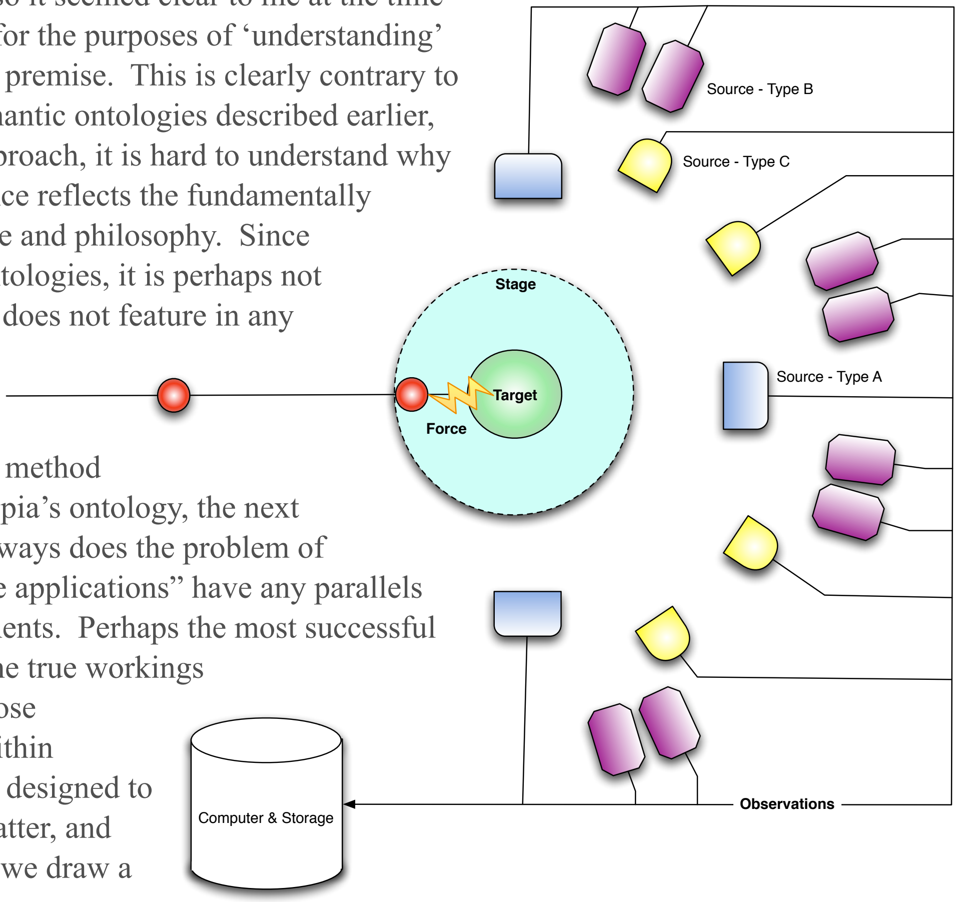
The ontology comprises 20,000 terms and 70,000 axioms when all domain ontologies are combined. These consist of SUMO itself, the MId-Level Ontology (MILO), and ontologies of Communications, Countries and Regions, distributed computing, Economy, Finance, engineering components, Geography, Government, Military (general, devices, processes, people), North American Industrial Classification System, People, physical elements, Transnational Issues, Transportation, Viruses, World Airports A-K, World Airports L-Z, WMD, and terrorism.

Philosophy of Mitopia's Base Ontology

As stated previously, Mitopia's base Carmot ontology evolved primarily out of pragmatic and implementation considerations regarding the creation of a data-flow based system, and the necessity to efficiently represent and organize data in a generalized form that could be discovered and leveraged in a reusable manner over data flows, by individual and unconnected islands of computation. By training, I am a physicist, thus, given my ignorance of ontologies at the time (1993), when faced with the question of how to organize and represent information about anything in existence for the purposes of understanding world events, there was a natural tendency to gravitate towards thinking of things in terms of the scientific method. The scientific method, as everyone knows, can be summarized as follows:

(1) Identify a problem
(2) Form a hypothesis
(3) Design and perform Experiments
(4) Collect and Analyze the experimental data
(5) Formulate conclusions about the hypothesis
(6) Repeat (2)-(5) until reality appears to match theory in all test cases.

Essentially the scientific method is the only process/tool that has ever succeeded in explaining how the world works and what kinds of interactions can occur between things in the world, in a manner that gives any predictive power to allow us to reason what might be the results of some new situation in the absence of experimental results to tell us. Everything we take for granted in the modern world was developed by this process, and so it seemed clear to me at the time that any system for organizing data for the purposes of 'understanding' anything, must also be based on this premise. This is clearly contrary to the approach taken in any of the semantic ontologies described earlier, and yet it seems such an obvious approach, it is hard to understand why this is the case. Perhaps the difference reflects the fundamentally different approaches between science and philosophy. Since philosophy has spawned all other ontologies, it is perhaps not surprising that the scientific method does not feature in any fundamental way in conventional ontologies.



Given the premise that the scientific method should be the underpinnings of Mitopia's ontology, the next question was to ask in exactly what ways does the problem of "observing the world for intelligence applications" have any parallels with conventional scientific experiments. Perhaps the most successful scientific experiments in exposing the true workings of the world around us have been those performed in high energy physics within accelerators. These experiments are designed to investigate the building blocks of matter, and the forces that act between them. If we draw a generic diagram of any accelerator experiment, it would consist of firing a stream of particles into a target located in an environment or 'stage' that is ringed by a set of different detectors, each capable of detecting different kinds of particles resulting from the 'event' that happens when one of the particles strikes the target. The output from all detectors results in a stream of 'observations' of differing types and accuracy (depending on the detector or 'source' accuracies). These observations are recorded into the computer system for later analysis to determine what 'event' might have happened.

The situation after such an event occurs appears similar to that shown in Figure 7. The colliding particle and the target have exchanged some 'forces' between them in an 'event', with the result that the target has been broken up into its constituents which we can observe as they are detected by the various sources arranged around the 'stage' in which the 'event' occurred. Of course not all the resultant particles are detected by our sensor array; we get only a partial snapshot of what actually happened. The goal of the analysis is to postulate a model for what might have happened and examine the observation stream for events that might consist of the interaction we are interested in. We then total up all the resultant bits we see in these events, and deduce what other bits we must be missing for the total energy to be conserved. By sampling multiple candidate events we are able to get a complete picture of what is actually happening in the 'event' and determine if the experimental results match our theoretical model. If they do, we pronounce our theory 'good' and we use it to model other as yet unseen situations until such time as the theory fails to match the results of some new experiment at which time we start over again on a new theory.

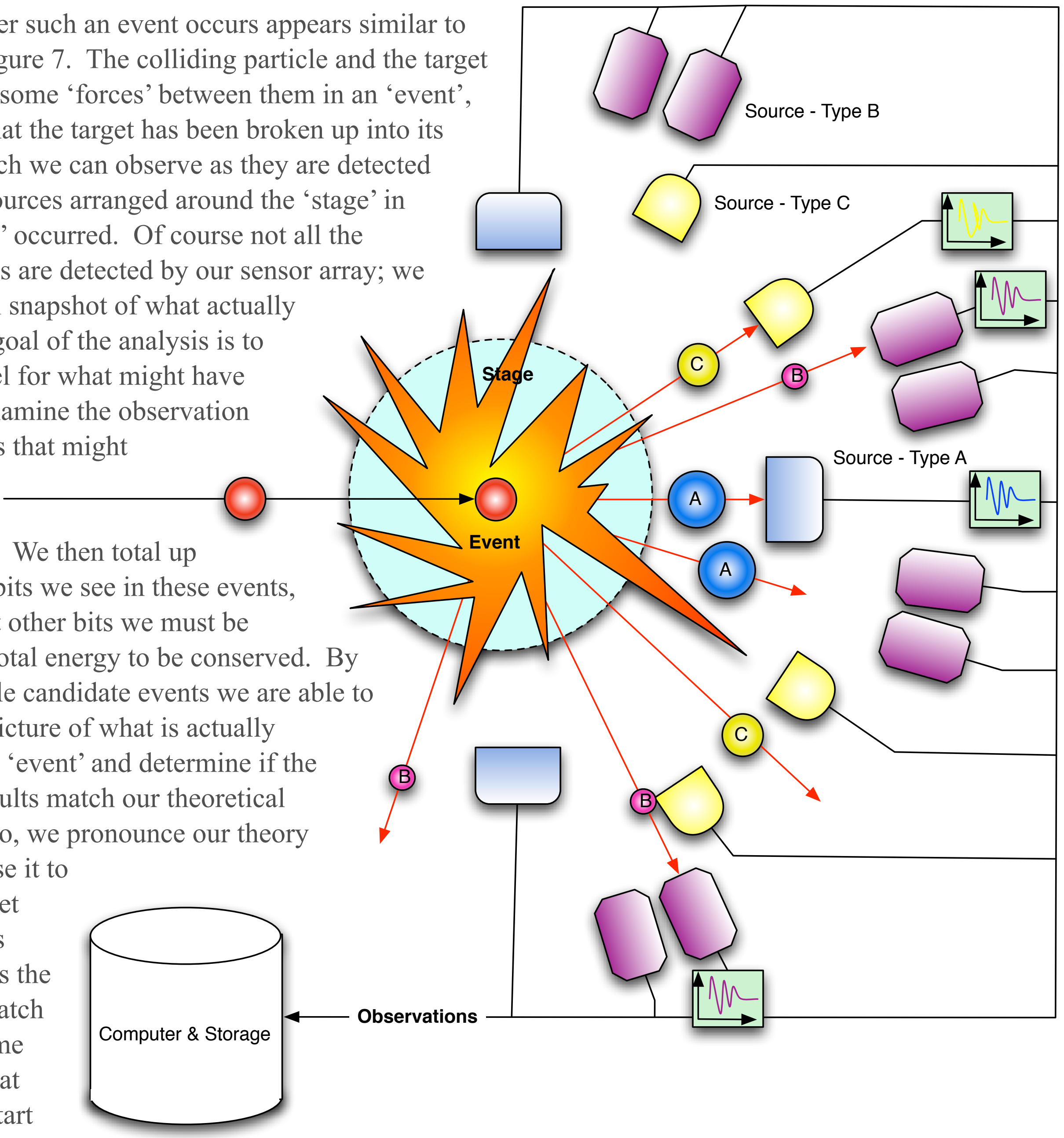


Figure 7 - Elements of an Accelerator Experiment

This then is the process that physicists use to ‘understand’ the world around us. A very similar approach can be found in virtually all science experiments. Thus if we are to begin to categorize the various distinct things that make up what we need to track to ‘understand’ the world in a physics sense, we come up with the following top level list, nothing more, nothing less:

(1) Particles (beam, target, debris)
(2) Forces
(3) Stage/Environment
(4) Observations
(5) Sources
(6) Events

If we now consider the intelligence process, that is what do we have to track to ‘understand world events’, we could once again draw an idealized diagram of such an experiment and it would appear as shown in Figure 8.

Essentially the problem is the same. Again we are interested in tracking the ‘events’ that happen, again those events happen in a ‘stage’ or environment that effects the behavior of the players and must thus be tracked carefully. Again we have a stream of ‘observations’, but this time from a much wider variety of ‘sources’ of vastly differing reliabilities which must be tracked carefully if they lead to any analytical conclusions. In the real world, we are interested not in particles and forces , but instead in ‘actors’ (mostly ‘entities’) and ‘actions’ which are conceptually analogous.

In other words, all we have to do is modify our terms slightly from

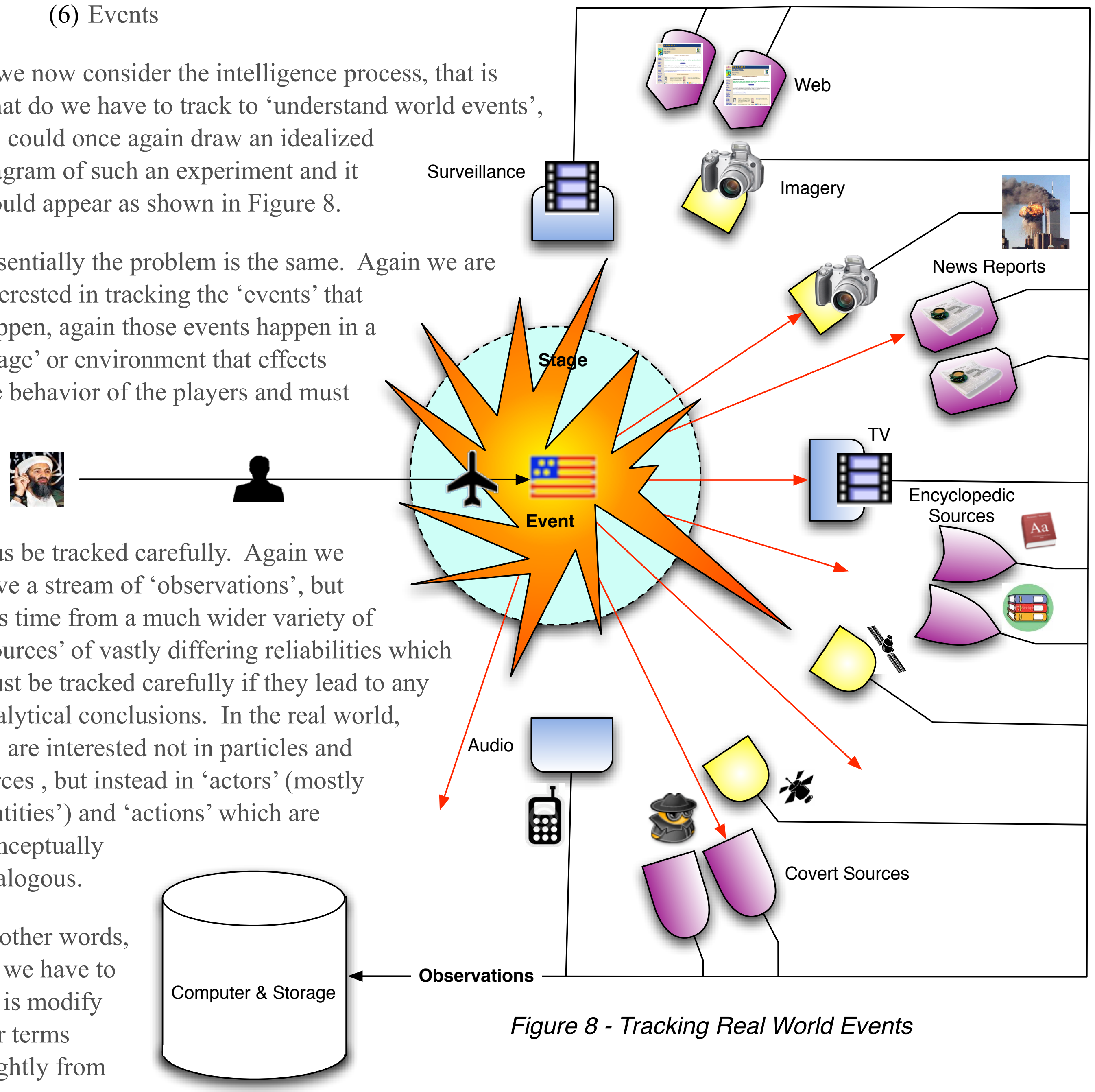


Figure 8 - Tracking Real World Events

the physics experiment and we have the set of things that must be tracked to ‘understand’ real world events. They are:

(1) Actors
(2) Actions
(3) Stages
(4) Observations
(5) Sources
(6) Events

Clearly then we have found the fundamental groupings which should form the top level of our ontology of everything as shown in Figure 9.

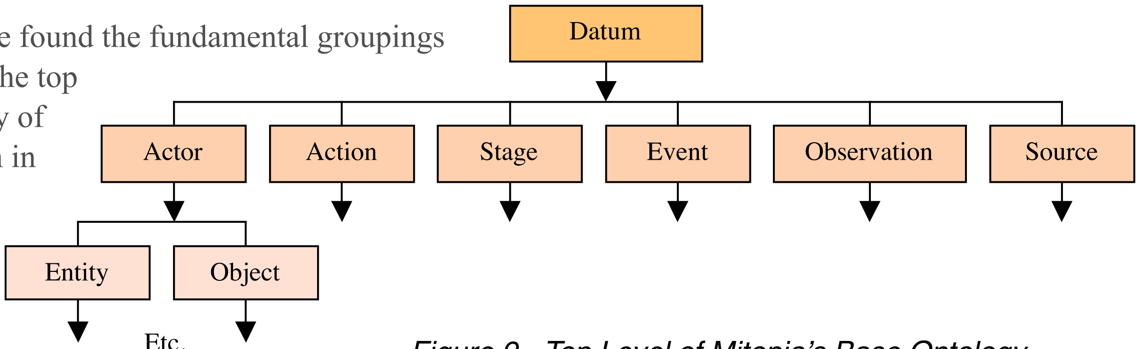


Figure 9 - Top Level of Mitopia’s Base Ontology

Remember, this ontology is based on the scientific method and is intended to allow organization and interrelation of actual data gleaned from the real world. We intend to use this ontology to generate and maintain our ‘database’ and the queries on it. This is in stark contrast to semantic ontologies which are targeted primarily at the problem of understanding the meaning of human written communication and the ideas expressed in that communication. Given this difference in intent, it seems hard to argue with the the top level arrangement used in Mitopia’s base ontology. With this choice of upper ontology, we have in one step chosen to organize our data in a manner that is consistent with ‘understanding’ it and we have simultaneously chosen the fundamental method we will use to analyze the data for meaning, that is the scientific method. This ontology has been chosen to facilitate the extraction of meaning from world events, and does not necessarily correspond to any functional, physical or logical breakdown chosen for other purposes, though given the discussion above, it is expected that the overwhelming majority of phenomena can be broken down according to this scheme.

Datum	The ancestral type of all persistent storage.
Actor	Actors participate in Events, perform Actions on Stages and can be observed.
Entity	Any 'unique' Actor that has motives and/or behaviors, i.e., that is not passive. People and Organizations are Entities, and understanding what kind of Entity they are and their interdependence is critical to understanding observed content.
Event	Events are conceptual groupings of Actors and Actions within a Stage, about which we receive a set of Observations from various Sources. It is by categorizing types of Events into their desirability according to the perceiving organization, modeling the steps necessary to cause that Event to happen, and looking in the data stream for signs of similar Events in the future, that a knowledge level intelligence system performs its function.

Object	A passive non-unique actor, i.e., a thing with no inherent drives or motives such as a piece of machinery. Entities must acquire and use precursor Objects in order to accomplish their goals and thus we can use objects to track intent and understand purpose.
Stage	This is the platform or environment where Events occur, often a physical location. Stages are more that just a place. The nature and history of a stage determine to a large extent the behavior and actions of the Actors within it. What makes sense in one Stage may not make sense in another.
Action	Actions are the forces that Actors exert on each other during an Event. All actions act to move the Actor(s) involved within a multi-dimensional space whose axes are the various motivations that an Entity can have (greed, power, etc.). By identifying the effect of a given type of Action along these axes, and, by assigning entities 'drives' along each motivational axis and strategies to achieve those drives, we can model behavior.
Observation	An Observation is a measurement of something about a Datum, a set of data or an Event. Observations come from sources. Observations represent the inputs to the system and come in a bewildering variety of formats, languages, and taxonomies. The ingestion process is essentially one of breaking raw Source Observations into their ontological equivalent for persisting and interconnecting.
Source	A Source is a logical origin of Observations or other Data. Knowledge of the source of all information contributing to an analytical conclusion is essential when assigning reliabilities to such conclusions.

The goal of an intelligence system is still to reconstruct what event has occurred by analysis of the observation data streams coming from the various sources/feeds. The variety of feed and sensor types is infinitely larger than in the particle physics case, however, as for the particle physics case, many effects of the event are not observed. The major difference between the two systems is simply the fact that in the intelligence system, the concept of an event is distributed over time and detectable particles are emitted a long time before what we generally think of as the event itself. This is simply because the interacting ‘particles’ are intelligent entities, for which a characteristic is forward planning, and which as a result give off ‘signals’ that can be analyzed via such a system in order to determine intent. For example in the 9/11 attacks, there were a number of prior indicators (e.g., flight training school attendance) that were consistent with the fact that such an event was likely to happen in the future, however, the intelligence community failed to recognize the emerging pattern due to the magnitude of the search, correlation, and analysis task. This then is the nature of the problem that must be addressed and as mentioned previously, we refer to systems attempting to address this challenge as “Unconstrained Systems”. In an Unconstrained System (UCS), the source(s) of data have no explicit knowledge of, or interest in, facilitating the capture and subsequent processing of that data by the system.

The later section of this document entitled “The Default/Base Ontology” gives extensive descriptions of Mitopia’s base Carmot ontology including descriptions of additional levels within each of the six upper concepts. The intent of the current section is simply to present the philosophy behind this upper level arrangement, since it is critical that the reader or user of the ontology learn to adapt his way of thinking at a fundamental level to match this world view.

Now that we have chosen the basic organization of our ontology, we must address one last issue which is that the fundamental purpose of an intelligence system is to recognize the patterns that lead up to a particular type of event, and if that event is undesirable, to provide warning before it occurs, so that steps can be taken to avoid it. In this regard an intelligence system is quite different from a physics experiment in that we seek to recognize the signature of an event before it happens rather than after. As mentioned above, we can address this issue quite simply by recognizing that thinking entities (the types of actors we are most interested in) take actions before an event they are planning to participate in, in order to get ready for that participation. Thus we must

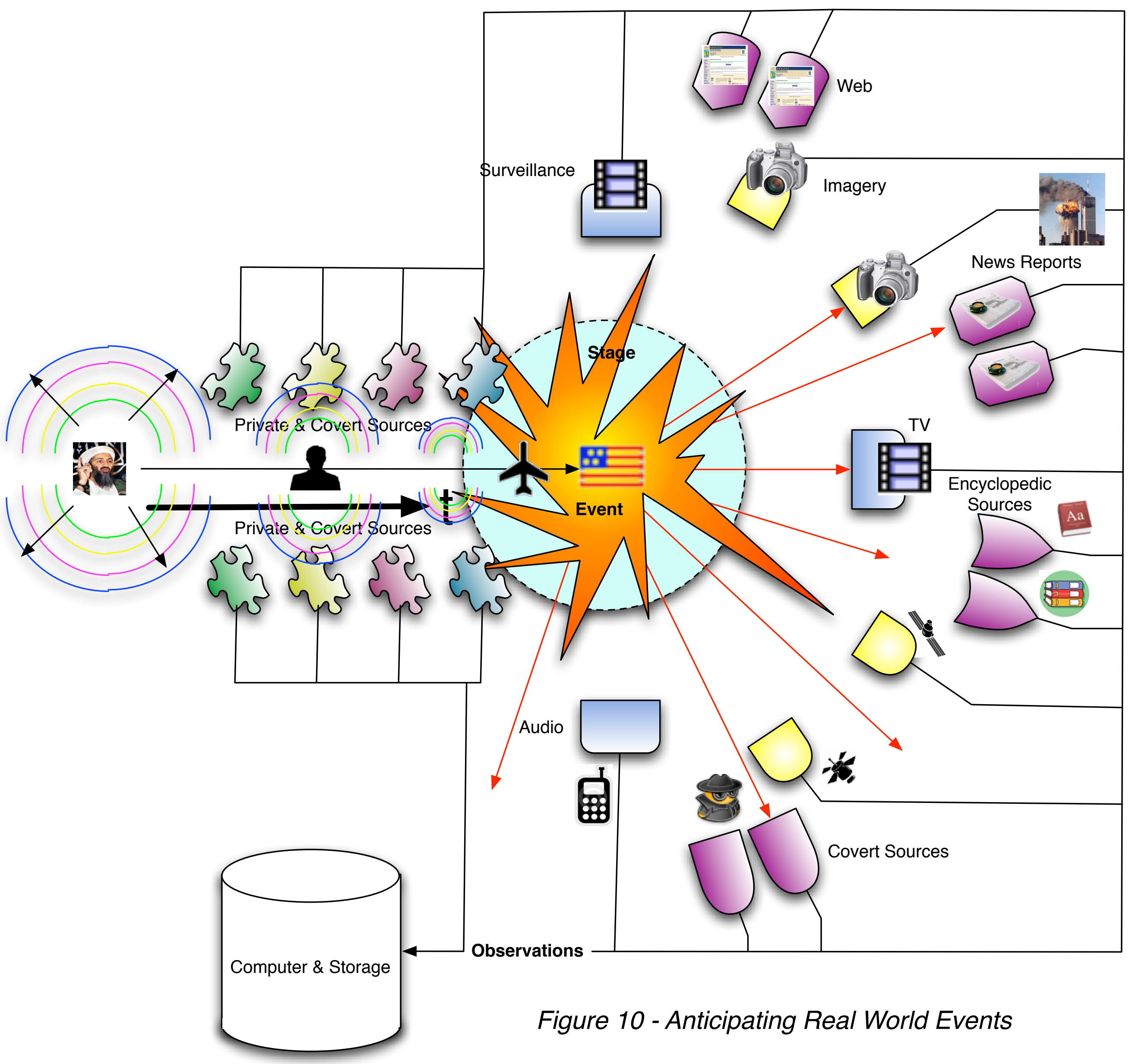
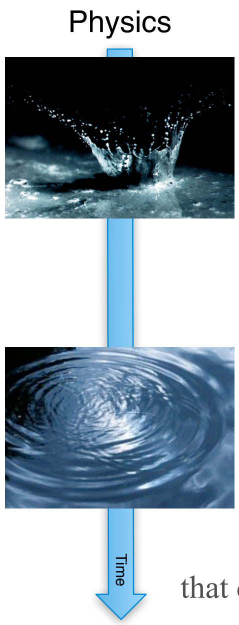
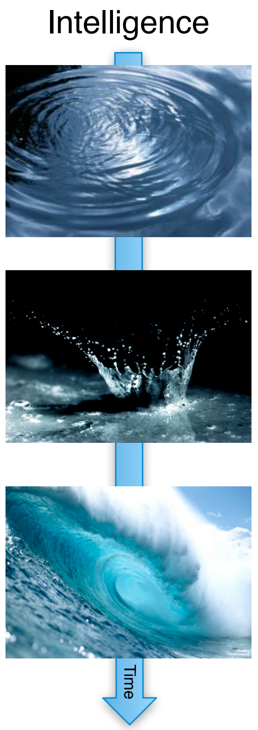


Figure 10 - Anticipating Real World Events

redesign our experiment slightly to gather information on a continuous basis up-stream in the time line as shown in Figure 10.





These up-stream sensors tend to be private or covert feeds rather than open source and thus they are things like passport and travel data, financial transactions, licenses and police reports, communications intercepts, HUMINT, etc. The nature of these feeds in general is that they are more reliable and contain data with high semantic content that can be connected via link analysis to other encyclopedic data. This is important since these upstream sources must be sensitive to far lower 'signal levels' than those typically associated with an actual publicly perceptible event which includes media attention. These sensors are examining the 'ripples' in the pond and looking for correlations between them whereas the post-event sensors are examining the actual splash. It is the degree to which a system is able correlate between the various types of low level ripples that precede an event in order to recognize a significant pattern of intent, that determines how well the system itself functions in a predictive capacity.

Unfortunately, most of the effort and expenditure exerted by intelligence agencies today is focussed on gathering more different types of data, with higher accuracy, while little if any progress is made on the far more important problem of seeking coherence of intent in the ripples from these up-stream feeds. We can now read license plates from space, but we still rarely know where to point such amazing sources before the event. The mistake is to treat the problem in the same way as one conventionally treats almost every other real world problem. To improve understanding post-event, all one really needs is more data with higher accuracy, and so it is easy to fall into the trap of thinking that this will also solve the pre-event issue, especially when demonstrations of new technology to potential purchasers are given using post-event historical data in order to clarify what difference the technology might have made in a known historical event. When seeking coherence and correlation in low intensity pre-event data streams, the issue is primarily one of removing the overwhelming amount of noise or insignificant data, from the tiny amount of significant data that may be mixed into the stream. In any real world scenario looking for correlation up stream, the noise is likely to outweigh the signal by many orders of magnitude, and so our analysis techniques must be able to operate in this setting. Only by organizing and interconnecting data in a rigorous ontological manner that is firmly based on an upper level ontology tied to the scientific method, is it plausible that the highly indirect series of connections that make up significance in the presence of overwhelming noise, can be discovered and extracted in a reliable manner. Our approach to ontology must focus first on assembling the content and connections implied in the stream of observations and meta-data that make up our limited picture of the world. Much of this information is extracted by non-linguistic techniques from meta-data or by inter-source combination. Only once this firm representation of the variety of observations and sources is in place should we consider refinement of our ontology based on linguistic and/or semantic processing of individual textual observations to understand the intent of human speech. Conventional systems cannot unify multiple sources and thus are restricted to attempting understanding based on semantic analysis of text, but this misses virtually all of the most reliable and telling connections that come from a fully contiguous ontological approach (see following section for definition). This then is the philosophical reason behind the unique organization of Mitopia's base ontology.

Carmot vs. Semantic Ontologies

The preceding sections have described the motivations behind both the semantic approach to ontologies and that used by Mitopia® and it's Carmot ODL. In effect we have to go back to our fundamental definition of what an ontology is, and recognize that what makes an ontology different from a taxonomy is simply the focus on supporting and handling relationships and connections between data, in addition to field content. The representation, discovery, and manipulation of relationships must be entirely based on the ontology, not on knowledge latent in the application code. A taxonomic or information level system uses a language and programming model(s) that focus on field content, which essentially means that any relationship knowledge must be embedded in the application code, and hence rigid and hidden from examination. Both approaches, Semantic ontologies (e.g., OWL), and Carmot, meet this definition of an ontological system, and yet they are fundamentally different. We will therefore have to invent some new language for defining the two types of ontology definition languages:

A **contiguous-model** ODL (of which Carmot is the only example) is one for which the ontological aspects of the language are integrated directly with the normal programming data model, that is, they can occur directly within the programming language used to access data held in that ontology, and furthermore where accessing code uses the same programming data model for all data be it in memory, on disk, or in a database. The ontological aspects are thus 'contiguous' with all other aspects of accessing data including binary compatibility with type declarations from the underlying platform headers.

A **disjoint-model** ODL (all other ODLs, of which OWL is just one) is one in which the ontological aspects are functionally and syntactically separated from the details of program access to and manipulation of data, be it in memory, disk, or in a database. All semantic ontologies are disjoint and make no attempt to unify programming models, the ontological aspects are 'disjoint' with normal program data access.

There are some fundamental differences in features and benefits between **contiguous** ontologies (i.e., Carmot), and **disjoint** ontologies (e.g., OWL), as summarized by the Table 1 below:

Feature	Carmot/Mitopia®	Semantic (OWL)
Run-time discovery of types, fields and links.	**YES**	**YES**
Unifies in-memory and in DB data access and programming models	**YES.** All operations are unified through Carmot APIs (TypeMgr, TypeCollections, etc.).	**NO**. Does not address either in memory (binary) form, or how the data is stored and accessed in a DB. Simply a text and semantics interchange format.
Unifies in-memory and in GUI programming models	**YES.** All operations are unified through Carmot APIs (TypeMgr, TypeCollections, etc.).	**NO**. Does not formally address the mapping to GUI layout although third party tools exist for this purpose.

Feature	Carmot/Mitopia®	Semantic (OWL)
Supports Multilingual Text Understanding.	**PARTIAL.** Mitopia® uses the names and aliases of persistent data to automatically identify known items in multilingual text, but it does not directly contain linguistic code to parse sentence structure using the ODL.	**PARTIAL.** All semantic ontologies are directed at the problem of text understanding, although since they are linguistic, they only work in one language (English).
Binary data and structures supported.	**YES**. Functional superset of C.	**NO.** Does not address storage. All operations are textual. Ontology is distinct from programming model.
Direct language support for logic and reasoning.	**PARTIAL.** Using Carmot, data-flow based widgets can navigate through links and examine types to perform reasoning, but this ability is not formalized into the syntax of the ODL.	**YES.** Although an external technology is required to map to first-order logic and implement the actual reasoning. Not many convincing examples of actual complex reasoning using OWL.
Designed for performance and scaleability (distribution).	**YES**	**NO**. Text-based and thus slow to handle and share.
Syntax compatible with standard programming language.	**YES.** Based on extensions to C.	**NO.** The ontology and tools to use it represent yet another incompatible programming model/language with a massively different syntax.
Associate scripts and behaviors with ontological types.	**YES**	**YES**
Support for Database auto-generation and query.	**YES**	**NO**
Support for GUI generation and handling.	**YES**	**PARTIAL**. Using 3rd party tools.
Built-in & unified programming model for manipulating collections of related data.	**YES**. Based on the TypeCollections area of the Carmot API.	**NO**. Implementation detail how references are resolved and unified.
Web Standards Based, Open Source.	**NO**	**YES**
Automatic data migration when ontology changes.	**YES**. Mitopia's Types server handles this through Carmot automatically when old data is accessed, regardless of source (disk, communications or DB).	**NO**. Still a subject of research.
Integrated support for federation and multimedia data and containers.	**YES**. The MitoPlex™ framework provides this with MitoQuest™ handling most non-multimedia types and fields.	**NO**. Only textual data is covered by the ODL.

Table 1 - Carmot vs. OWL Comparison

There are in fact two distinct parts to the Carmot language. The Carmot-D variant discussed in this document is a language of data and type declaration and subsequent dynamic run-time discovery. The Carmot-E variant which is discussed elsewhere (see MTL 'yellow' book), is the run-time executable language and environment, which is utilized in many aspects of Mitopia® (e.g., MitoMine™) to access and compute using data described by the ontology. Thus we see that unlike all conventional languages, which combine the declaration and execution aspects of the language, Carmot takes a unique 'split' approach to language definition, and it is perhaps important to point out why this is so.

The first and foremost reason for the split is that as stated earlier, Carmot is designed to allow code to discover all the required data and data types as run-time as opposed to compile-time, in support of a data-flow and data-driven system. The fact that conventional languages allow and indeed encourage intermingling of type and data declarations with the executable code that manipulates them, is to a very real extent encouraging programmers to create software that contains fragile embedded assumptions that cross the data-code membrane and thus contribute to the Bermuda Triangle effect. For this reason, the Carmot-D language was implemented as a pure declaration language and all the Carmot APIs and abstractions were built upon it independent of the need for a run-time executable language.

The second requirement that drove the development of the Carmot-E variant was the need to match 'impedance' between the Carmot-D ontology of the system and the taxonomies/formats of the large numbers of sources that must be combined into an ontological representation in order to perform useful analysis. This interface between source formats and the ontology must be accomplished by allowing source data to drive the conversion state without explicit hard-coded knowledge in the conversion layer of the input or indeed the output formats. This realization in turn led to the concept of entangled parsers (i.e., nested parsers where each can influence the other) and the patented mechanism to accomplish this feat. MitoMine™ (see MTL 'red' book) is the premier example of this approach. The key aspect of such entangled parsers is that the order of execution of statements in the 'inner' parser is determined not by the order they occur in the script, but rather by the source data itself, as reflected in the evolution of the 'outer' parser state. We refer to such unique and unusual languages as 'heteromorphic' languages (see MTL 'yellow' book for details). There is a massive cognitive difference between the programming model of a conventional language, and that of a heteromorphic language. Learning to embrace this approach can be difficult for those trained in conventional programming languages.

Since in all cases within Mitopia®, either the source, or more often the target, of any conversion is data described by the Carmot-D ontology, it was clear that the Carmot-E language should be defined as service abstraction, just like Carmot-D. The fact that Carmot-E program state is driven by source data format and content, and not the programmer's 'algorithm', and that this occurs as a result of many small isolated snippets of Carmot-E code rippling to the top of a parser stack, means that the Carmot-E language has little use for type declarations of its own, discovering all needed types dynamically from the Carmot-D type information.

In conclusion then, it is clear that the splitting of the underlying Carmot language into two distinct aspects, one handling type declaration, and the other type access and manipulation, was a necessary step in the creation of a truly data-driven environment such as Mitopia®, and in encouraging the development of truly adaptive code.