Building an Enterprise Ontology in Less Than 90 Days

Dave McComb, Dan Carey & Todd Schneider
Semantic Arts
www.semanticarts.com

November 18, 2015
SEMANTIC TECHNOLOGY FOR
INTELLIGENCE, DEFENSE, AND SECURITY
George Mason University, Fairfax, VA

Building an Enterprise Ontology in 3 Acts

- Act 1 – Do we need to do something different?
- Act 2 – What is it about semantics that will make a difference?
- Act 3 - 6 – "How to's" for getting this done in 90 days (or less!)

Semantic Arts

- Over the last 15 years, we've been designing and building ontologies for a number of large firms in many different industries

Dan Carey

- Ontologist at Semantic Arts.
- Dan has 30 years of consulting experience, 25 of it designing application databases, logical and physical data models, and data strategies with major IT service firms.
- He has designed semantic technology products to assist in military human resources management, and data exchange standards using OWL and XSD.
- He holds a bachelor's degree in Applied Physics.

Todd Schneider

- Todd has 25 years of experience as a systems engineer and ontologist, primarily in the defense industry
- Lead several initiatives within Raytheon and their clients to integrate semantic technology with enterprise architecture
- Early and frequent participant and contributor to the Net Centric Industry Consortium, and other Net Centric initiatives throughout the federal space.
- He holds a PhD in Mathematics

Please introduce yourselves

- We can still tailor this presentation based on your needs and backgrounds
  - Name
  - Organization
  - Experience with Semantic Technology
  - Experience with other information/data technologies
  - Your specific interest in this topic

Dave McComb

- Founded Semantic Arts in 2000
- Co-founded Semantic Technology conference in 2004
- Wrote Semantics in Business Systems
- Four patents in software engineering including first patent on model driven development
- Worked with dozens of large enterprises at the Enterprise Architecture and Enterprise Application level

Information Systems Cost

- Our information systems cost 10 - 100x what they could or should cost

Why is that?

▪Legacy Systems?	▪Certainly a contributor
▪Vendor Lock in?	▪Adds to it
▪Solving the wrong problem?	▪Frequently
▪Undue Complexity?	▪Getting at the root

Architect, Prostitute & System Analyst

Yeah, Where did that chaos come from?

- Root-cause analysis lead us to one of the most damaging phrases ever uttered in the Corporate world:

"Let's not reinvent the wheel"

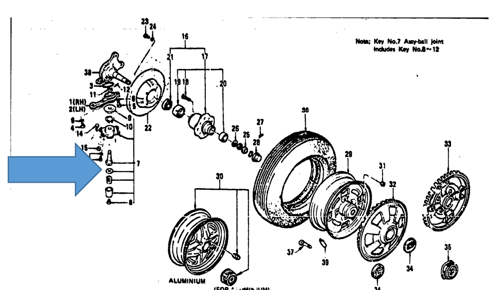
Sounds innocent, helpful even

- Let's explore how it works its way through a decision process
- Someone says "we need x feature"
- "Surely we're not the first firm to want that. Let's not reinvent the wheel"
- And a search begins to find a product that can be acquired that has "x"

As if…

- …You needed a washer,

- …And you discovered this wheel has a washer!



Before you know it

Surely they don't…

- …Implement large monolithic applications just to handle a small variation in data or function.
- Yep, they do

Examples

- Washington State – Paying employees v. paying w2 providers
- Washington State – 6 referral systems
- Washington State Labor & Industries – 23 systems with Accounts Receivable functionality

Net result

- Most large firms have thousands of major applications
- Each has its own, arbitrarily different data model
  - With thousands of tables and attributes
- Each at an arbitrary level of abstraction,
  - With an arbitrary data structure
  - With arbitrary names
- Leading to millions of distinctions to be mastered
- The implicit and explicit relationships between them are vastly complex

For Example

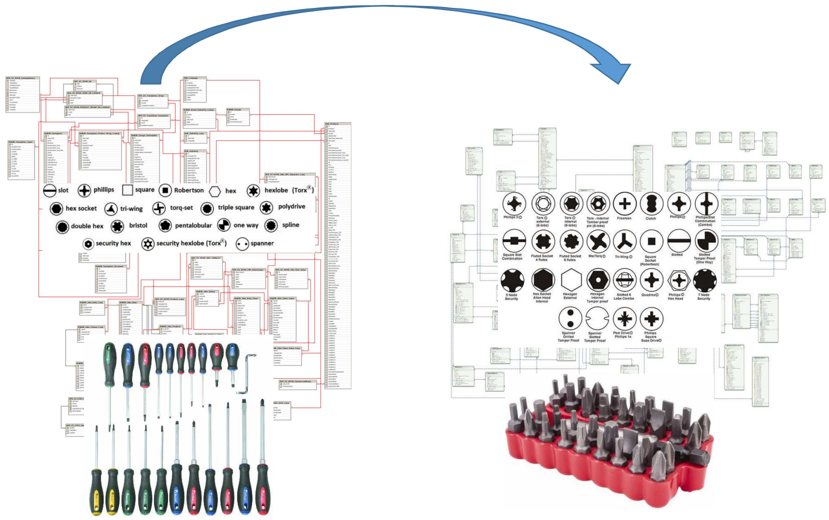
- SAP - Average SAP Install has 95,000 tables and well over 1 million attributes
- EPIC (Electronic Medical Record) - has 210,000 attributes

Replacing one of these systems

- Most of the time people don't retire systems, they just build additional ones
- Let's look at what happens when the project is to actually retire an existing application

20

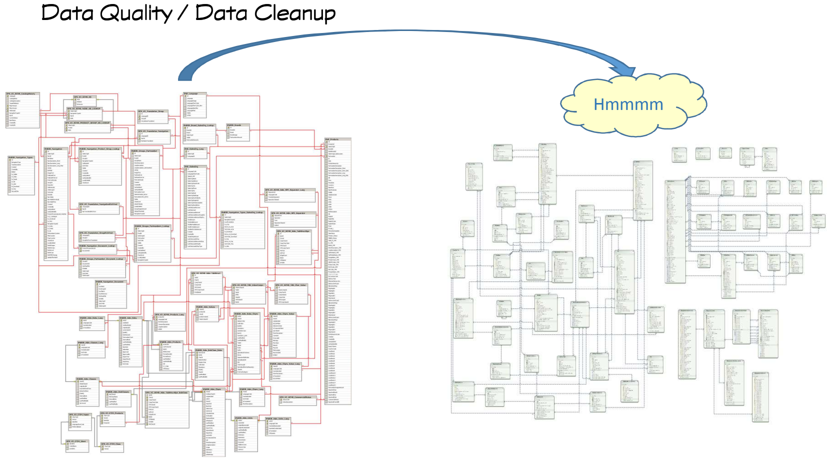
Implementing a new system



21

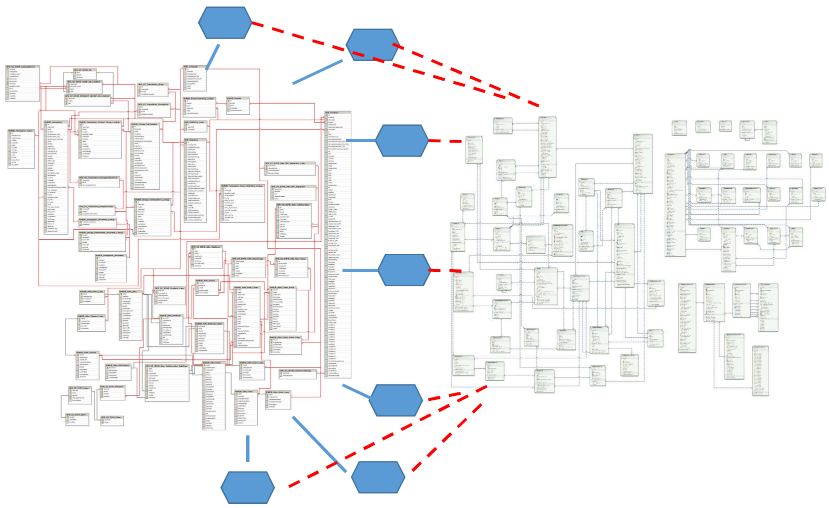
“Data Quality Problems”

Data Quality / Data Cleanup

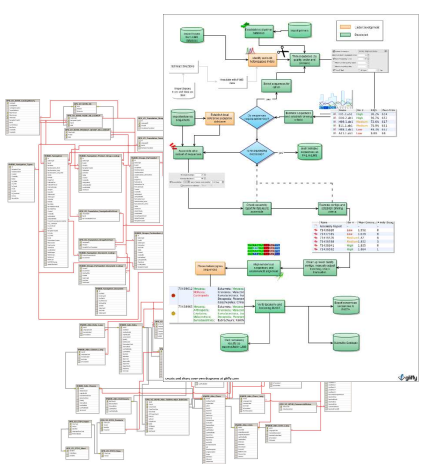


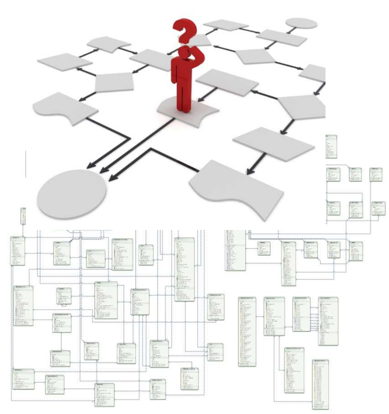
22

Integration



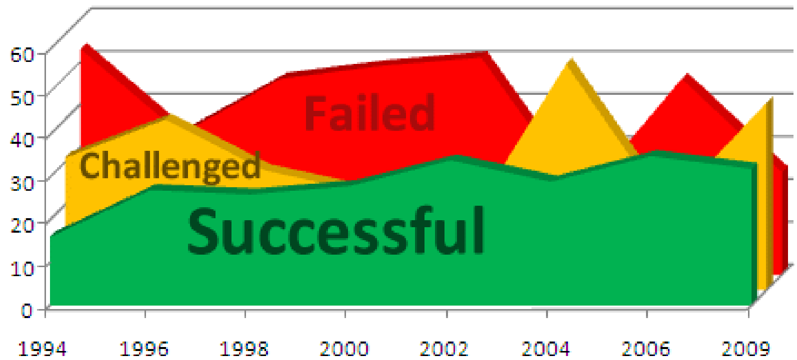
Those darn users





"Change Management"

Most of these projects end poorly



Healthcare.gov

	HealthCare.gov	HealthSherpa
Cost	$1 billion to date	Initial Build: 3 guys, 3 weeks. Probably a person year since then
Finding plans	Allows you to search for plans, some eligibility checking, enrollment	Same
Eligibility	Verify income and calculate subsidies	Subsidiary calculator, but doesn't check eligibility
User experience	Healthcare.gov has adopted some of healthsherpa's UI ideas	Initially much better, browse first, then register later
Users enrolled	7.1 million	120,000

Further evidence that these economics are not necessary

- Pinterest – 0 - 10 billion page views/month in 2 years and 40 engineers

http://highscalability.com/blog/2013/4/15/scaling-pinterest-from-0-to-10s-of-billions-of-page-views-a.html

- Instagram – 30 million users in 2 years, from 2 engineers to 5

http://techcrunch.com/2012/04/12/how-to-scale-a-1-billion-startup-a-guide-from-instagram-co-founder-mike-krieger/

The Question

- Is not: "Is this approach to building and deploying systems dysfunctional?"
  - (It is.)
- The question is: "Why does this approach persist?"

How does the bad idea of adding yet another application to our architecture persist?

Because
- It's in the interest of the vendors
- It's budget-able
- It's what we know
- A credible alternative hasn't been put forward

29

Steam v. Electricity

30

We are putting forward two related ideas

- 1) The data-centric approach to systems implementation is the only viable way to break this pattern

And

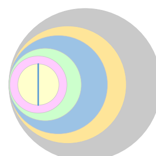
- 2) The application of semantic technology and enterprise ontologies is crucial to the success of the data-centric approach

Data-Centric

- In a data-centric enterprise, the data is the main asset
- Application functionality comes and goes
- The data remains and is added to
- It does not need to be converted

In order for this to work

- There needs to be an architecture…
- …Which replaces some of the key functions currently being performed by applications

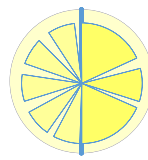


33

At the architecture's heart is the data

34

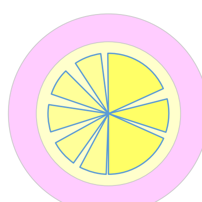
Really a set of coordinated data repositories



Some of which are internally generated and curated and others are not

35

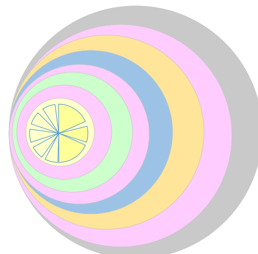
With shared meaning



This is where the enterprise ontology comes in

36

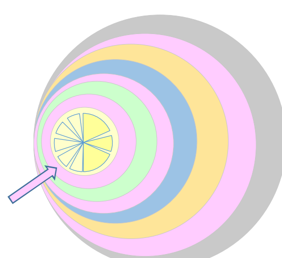
Other layers



Include such things as
identity management, security,
common services,
producer and consumer ontologies

What is an Enterprise Ontology?

- Definition of a common core of concepts that expose the hidden sameness in what people are talking about,
- That identifies some agreed-upon terms,
- And is represented in a formal notation to support automation.

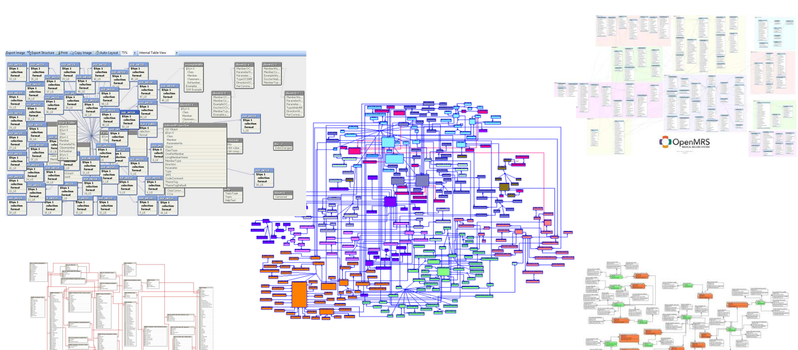


Sounds like a Data Dictionary / Controlled Vocabulary?

- It is
- But it differs from DD/CV in that a DD/CV is primarily for human consumption (users or system designers)
- An Enterprise Ontology is meant for human consumption, plus to be used directly for system building and system integration, and operation
- It's developed using ontological principals and analyses

Sounds Like an Enterprise Data Model?

- But if each application data model is vastly complex?



- Wouldn't logic suggest that a model to cover them all would be far more complex than any of the individual models?

In this case…

- The logic is wrong
- It is possible to model an enterprise's information with hundreds of core concepts, plus a manageable constellation of fine-grained distinctions
- Without losing any fidelity

An Enterprise Ontology

- Can capture the fidelity of the distinctions in an enterprise
- Without succumbing to the temptation to recreate the complexity

What are the Key Difference between a traditional data model and an ontology that allow this?

- Structure
- Complexity
- Explicitness
- Flexibility
- Reusability

Structure

- It's surprising how much the structure of a structured database contributes to its complexity
- The slightest addition, variation, or change in cardinality, seems to give rise to more tables and more attributes

Complexity

- IRS - EDM 30,000 Entities
- AT&T - have been working on an EDM for over a decade
- Most Defense Contractors – Multi-year, Multi-thousand entity efforts

Explicitness

- In traditional models, the meaning is implicit
- It lives in design documents and people's heads
- There is nothing in the "Employee Master Table" to tell you what an employee is
- Only, to tell you a few key attributes we've decided were of interest to us
- In an Ontology the meaning can be explicit

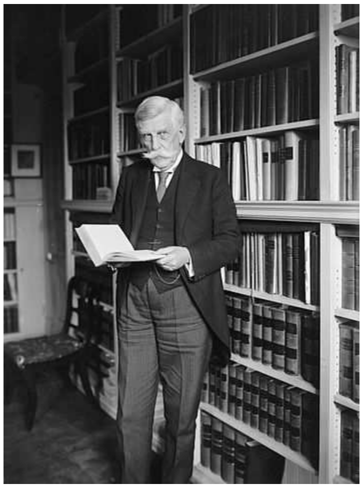
Flexibility

- The structure of a data model is rigid, and programmers rely on that rigidity
- Many programming idioms rely on the structure of the data being recapitulated in the code
- This means that data models are easy to re-factor before development begins, and very hard to re-factor afterward
- This promotes a design style of adding on rather than refactoring, and at some point it is easier to create a new system than to add on
- Ontology implementation is through a graph database that is inherently flexible

Reuseability

- There is no extension mechanism for Data Models
- There is no inheritance model in relational, and while some developers have developed patterns for simulating inheritance it is not widespread
- Most modelers add additional tables and columns in order to extend a model, or create an entirely new model and application
- Modularity, inheritance and importation are baked into Ontologies
- As a result it is usually easier to make small extensions, without disruption

Oliver Wendell Holmes



"I would not give a fig for the simplicity this side of complexity, but I would give my right arm for the simplicity on the other side of complexity."

Some Evidence that Semantics Helps

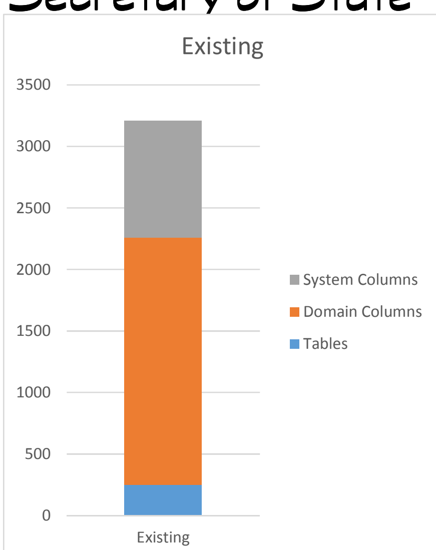
- Sallie Mae
- Secretary of State (WA)
- Sentara
- Schneider-Electric

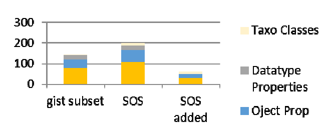
Case Study: Reducing Complexity at Sallie Mae

	tables	attributes
Class	582	10,230
LoanCons	133	15,295
Eagle I	356	13,538
Eagle II	464	12,502
	1,535	51,565

Category	Metric
Classes	366
Object Properties	224
Datatype Properties	13
Individuals (Categories mostly)	227
Total TBox Axioms	1,030

Case Study: Reducing Complexity at WA Secretary of State





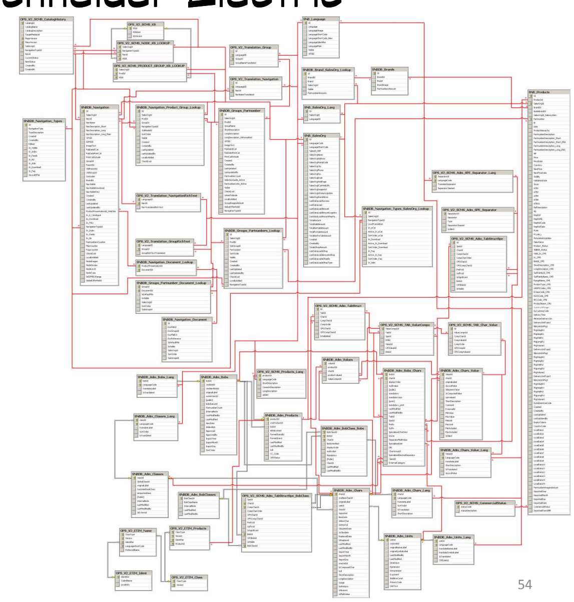
52

Case Study: EO for Sentara Healthcare

- Enterprise Ontology
- Scope: entire enterprise: healthcare, assisted living, insurance and internal systems
- Size: 1,276 classes/397 properties
- Applications:
  - We co-developed a Proof of Concept in Asthma and COPD, and the newly uncovered concepts we are finding are directly derivable from the core.
  - Currently being used as basis for semantic enterprise search. More than adequate coverage.
  - Flexible way to manage Physicians data across multiple sources

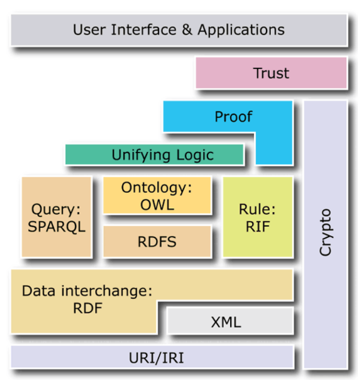
Case Study: Schneider Electric

- Existing product catalog systems has about 700 tables and 7000 attributes in total
- To date the new system has populated 46 classes and used 36 properties.
- We expect this to slightly more than double, to include the full scope

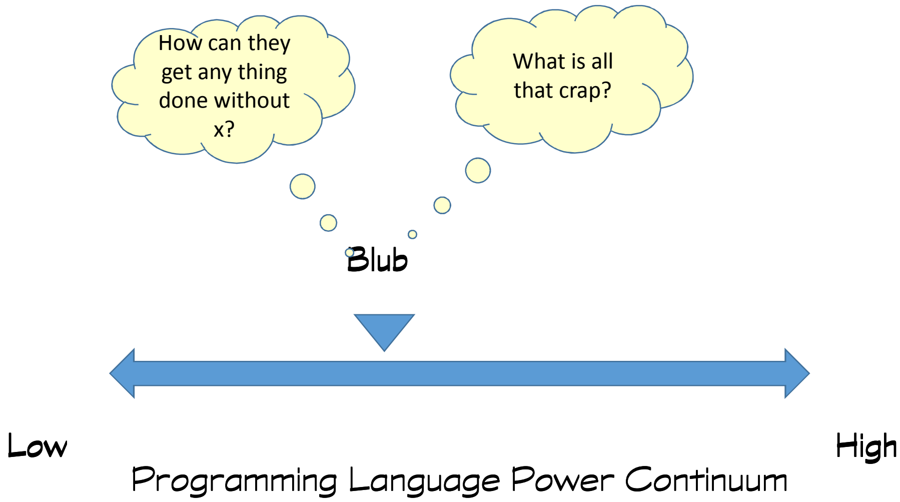


Semantic Technology

- Semantics – Pertaining to the study of meaning
- Semantic Technology – software and methods that rely on representing meaning, especially those that are based on the Semantic Web stack as standardized by the W3C
- However, information systems don't understand 'meaning'; they interpret symbols.
  - OWL and ontologies built using it provide a way to better control how they interpret the symbols.



Blub Paradox

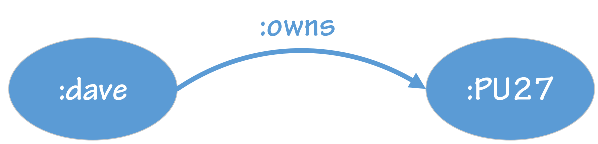


Developers and Semantics

- Most enterprise developers have come from either Relational, Object-Oriented or JSON-style document backgrounds
- When building ontologies, they tend to build them to resemble what they are used to
- Then they wonder why it doesn't do X as well as their approach of choice
- And what's all this "inference" and "open world" stuff, anyway?

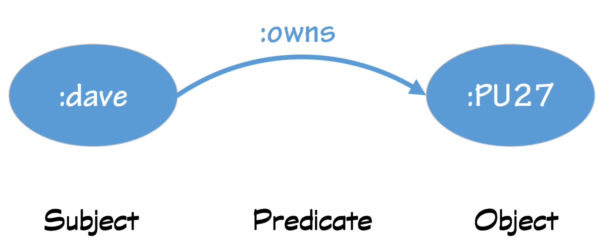
What Differentiates Ontology?

- All information representation is reduced to a single data structure: the "triple"



By convention

- The three parts of a triple are called…



Useful Global Identifiers

- All systems create identifiers for the many things they need to identify and distinguish
- The typical approach is to create a primary key on a table

SecretAgent	
ID	Name
007	Bond, James
002	Fairbanks, Bill

There are problems with this approach

- These numbers only mean anything in the context of:
  - This database
  - This table
  - This column
- They embed context and hide usage

SecretAgent	
ID	Name
007	Bond, James
002	Fairbanks, Bill

This is one reason existing systems are complex

- If you want to get information from multiple different tables
- You have to write a query that "joins" the tables
- … WHERE SegretAgent.ID = Gadgets.Assignee And SecreteAgent.ID = FemmeFatale. Foe …

Villian		
Code	Name	Protag
BB104	No, Dr. Julius	007
BB105	Goldfinger, Auric	007

FemmeFatale		
Code	Name	Foe
HB	Brandt, Helga	007
FV	Volpe, Fiona	007

SecretAgent	
ID	Name
007	Bond, James
002	Fairbanks, Bill

Gadgets		
Number	Desc	Assignee
001	Shoe Dagger	002
002	Rocket Belt	007

This is one reason existing systems are complex

- A human has to know all the (implicit) metadata to construct this join for each use
- This only works between tables in the same database

Villian		
Code	Name	Protag
BB104	No, Dr. Julius	007
BB105	Goldfinger, Auric	007

FemmeFatale		
Code	Name	Foe
HB	Brandt, Helga	007
FV	Volpe, Fiona	007

SecretAgent	
ID	Name
007	Bond, James
002	Fairbanks, Bill

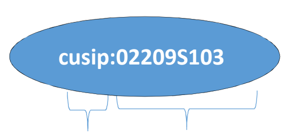
Gadgets		
Number	Desc	Assignee
001	Shoe Dagger	002
002	Rocket Belt	007

We need context-independent identifiers

- GUIDs
- May be globally unique, but not globally resolvable
- Nor are they easily interpretable
  - What do you do when someone sends you a message with this in it?
  - {21EC2020-3AEA-1069-A2DD-08002B30309D}

W3C's Semantic Technology Stack Finesses this with the "URI"

- Uniform Resource Identifier
  - Analogous to the URLs we type into our browsers
- Each part in a triple is identified by a URI



namespace fragment

http://cusip.com + /reg#

(domain name) (path) (id)
http://cusip.com/reg#02209S103

W3C's Semantic Technology Stack Finesses this with the "URI"

- This identifier is truly global
- It means the same thing, regardless of the database or document it is part of

cusip:02209S103 = http://cusip.com/reg#02209S103

- The metadata has been made explicit
- In turn, this yields information that can be "joined" without relying on humans or additional (implicit) knowledge of the metadata

Using URIs (similar to URLs) as identifiers

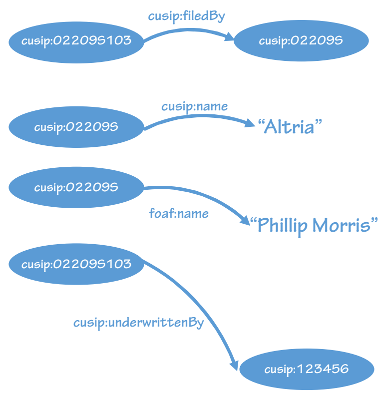
- Gives us truly global unique IDs
- That can be looked up, if needed

$\underbrace{\text{http://isbn10.isbn.org/books\#}}_{\text{Namespace}}\underbrace{\text{1558609172}}_{\text{Fragment}}$

This is the key to "self-assembling structures"

Let's Visualize this Process

- Lets say we the following triples were harvested from completely different sources
- CUSIP: Identifier for a Financial Securities

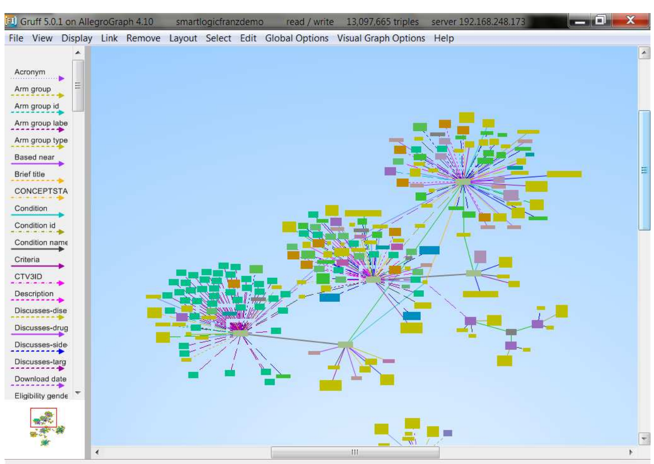


Tinker toys

Accommodates change in place

- A semantic system can evolve in place
- Example: Tasks -> Projects -> Backlogs -> Assignments -> Expenses

Of course these graphs can get more complex than you could represent with Tinker Toys

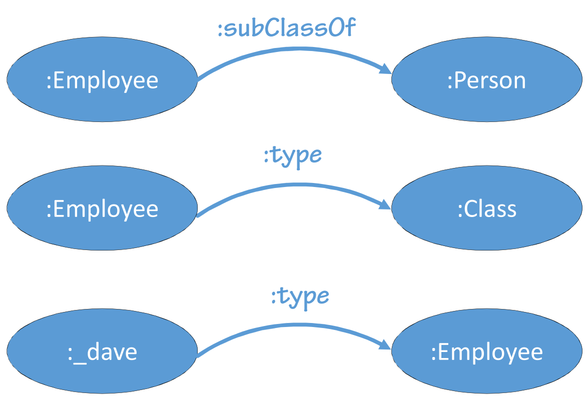


Relation of Metadata to Data

- Metadata is also represented as triples
- It is made explicit
- It can be queried

Including Metadata

- Metadata is the data that defines your current systems



Triples from RDB

- One of the most common sources for triples is from existing relational databases

Making Assertions, Traditionally

- In a traditional system, we make assertions by putting data in tables

Person 1 is named John

PersonId	Name
1	John

Making Assertions, Traditionally

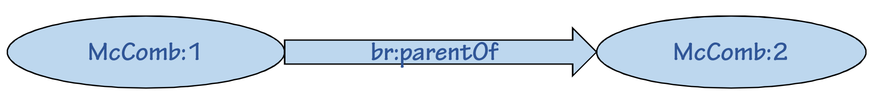
- Including relationships

John is the parent of Dave.

PersonId	Name	ParentOf
1	John	2
2	Dave	3

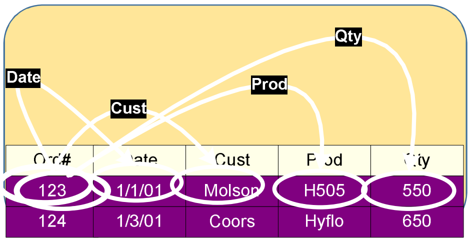
Where do triples come from?

- Triples can come from existing systems

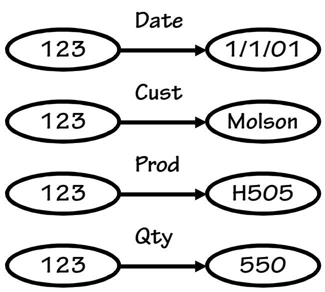


PersonId	Name	ParentOf
1	John	2
2	Dave	3

Where do triples come from?

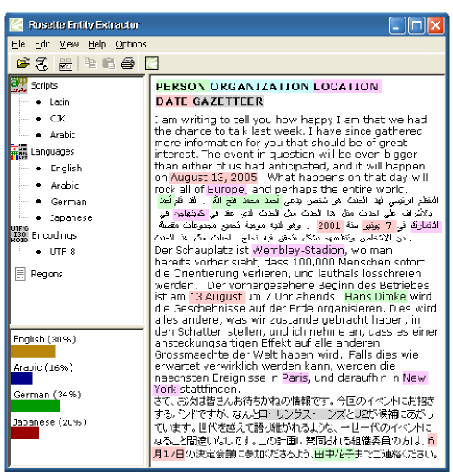


Databases

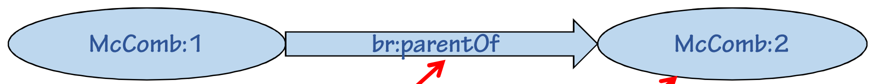


Where do triples come from?

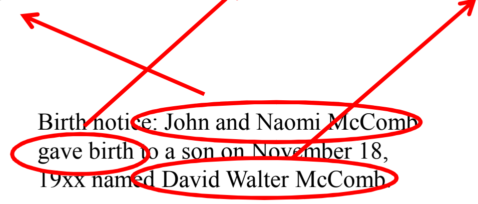
Documents



Where do triples come from?



- Text



- Web pages

81

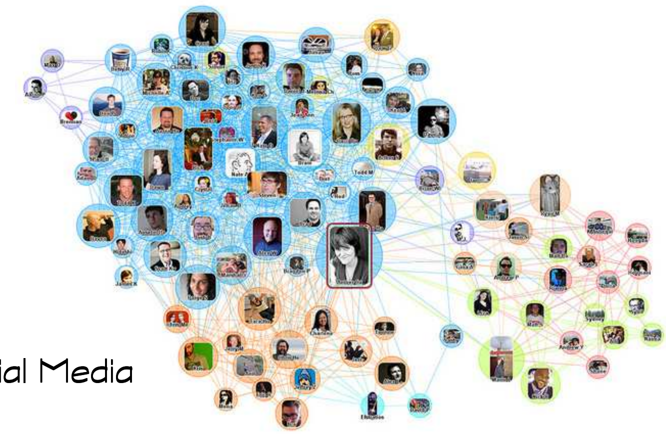
Where do triples come from?

XML and HTML

```
<div class="vcard" style="background-color: F9C; width: 300px; height
  <span class="fn n">
    <span class="honorific-prefix">Mr.</span>
    <span class="given-name">Doug</span>
    <span class="family-name">Mahugh</span>
  </span>
  <br />
  <span class="adr">
    <span class="street-address">One Microsoft Way</span>
    <br />
    <span class="locality">Redmond</span>
    ,
    <abbr class="region">WA</abbr>
    <span class="postal-code">98052</span>
  </span>
  <br />
  <a class="email" href="mailto:dm%61h%75%67h@m%69%63%72
  <br />
  <span class="tel">
    Phone:
    <span class="value">+1-425-882-8080</span>
  </span>
</div>
```

Where do triples come from?

- Social Media

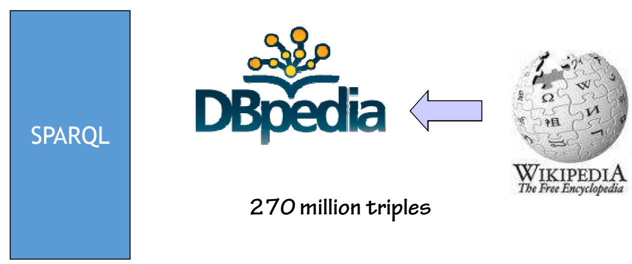


83

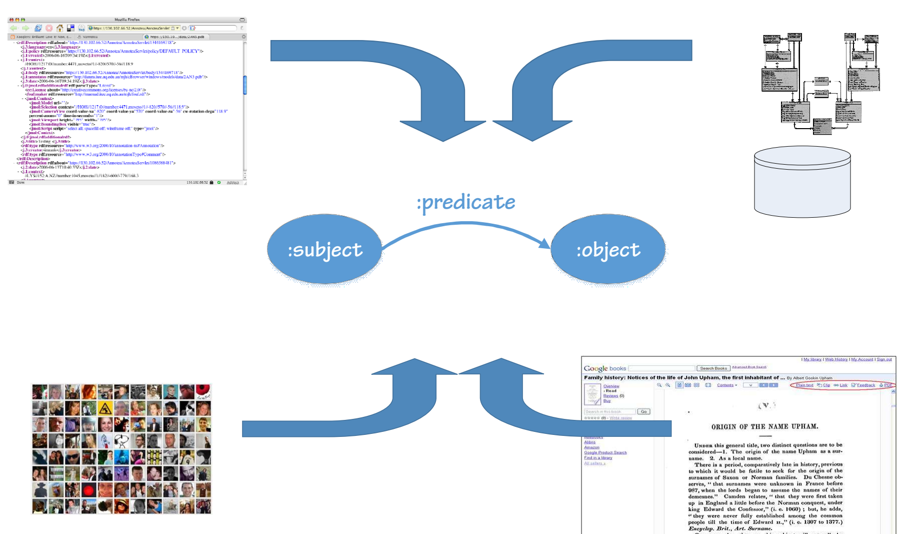
Where do triples come from?

- The Web

DBpedia



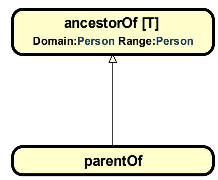
Triple as Common Denominator

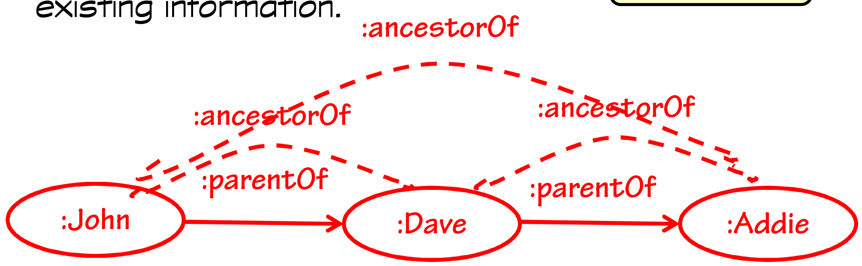


86

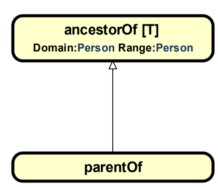
Inferring new information from existing

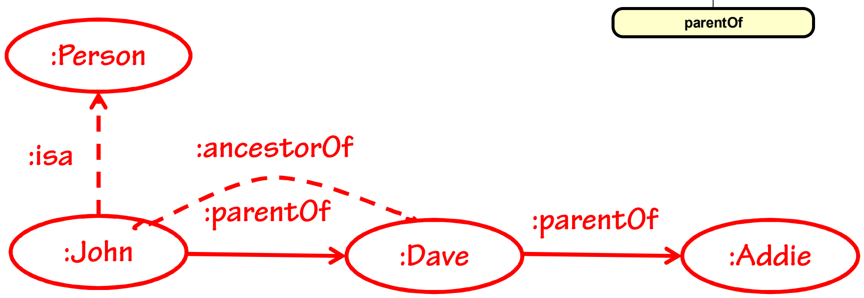
- Ontologies also provide a means for inferring new information from existing information.





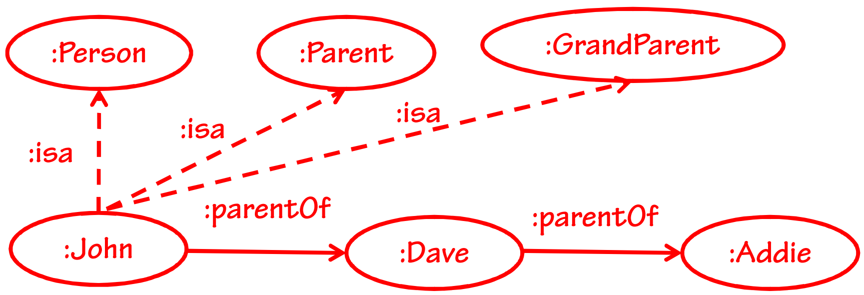
The most important new information we infer is class membership



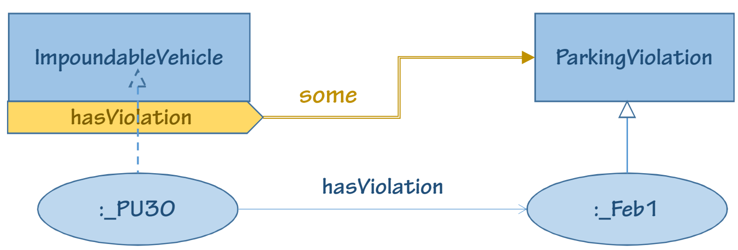


And because class membership is just another triple

- The same instance can be in many classes simultaneously.



Description Logics

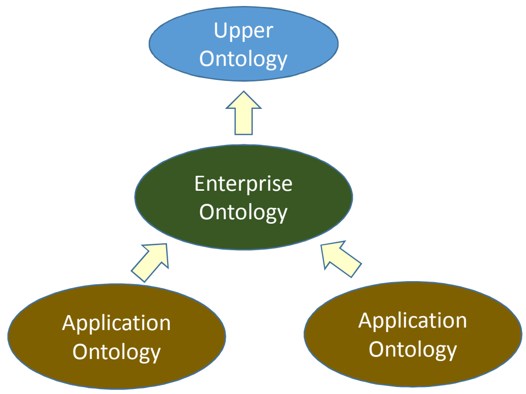


- Description Logics give a way of define the membership criteria for classes.
- This means that computer systems can do a lot of the classification that we currently have humans do.
- And that makes the rules transparent.
(Current approaches bury meaning in code and specifications.)

90

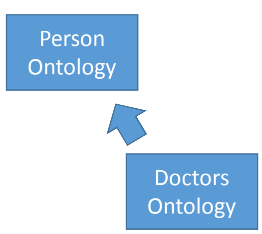
Modularity

- The way that modularity works with ontologies is very cool



Importing Ontologies

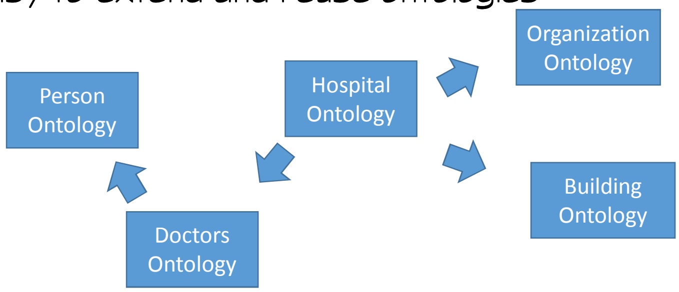
- Because they have no structure (beyond triples), it is very easy to extend and reuse ontologies



A doctors ontology could import a Person ontology and use that as its starting point (people have dates of birth, and addresses and the like)

Importing Ontologies

- Because they have no structure (beyond triples), it is very easy to extend and reuse ontologies



A hospital ontology could not only import and use the doctor ontology, it could create a superclass of doctor, called provider which also might include organizations. It can superclass another class without the other class or ontology being aware.

Versus Relational

- In relational paradigms, there is no real concept of extending a model
- At best, developers copy one schema for a starting point and change it from there

94

Versus Object-Oriented

- In object-oriented approaches, you can extend a common model through inheritance
- But there is no mechanism to create new classes (even superclasses) in descendent models
- Further, with semantic technology, the order in which the ontology extensions are created is not important. You can create the sub-ontology first and unify it with another later

Review: Characteristics of Semantics that Make it Uniquely Suited to Role as EO

Value of a Triplestore / Graph DB

- In general, the main advantage of a Graph DB over a Relational DB lies in its flexibility, and the ability to evolve in place
- We're going to go over three specific examples of how that manifests itself

Global / Resolvable identifiers

- In a relational system, all IDs are contextual; you need to know the DB, the table and the column to use the identifier
- In a triplestore, all identifiers are URIs. They are globally unique and resolvable

Once we assign a URI to a individual or concept, it means exactly the same thing, no matter what database or repository it is in.

What this means is that "joins" are done automatically for you by the system

The relationship of Schema to Data

- In a relational system, a table must exist before you can put data in it
- In a graph DB, you can create data and later associate it with newly created classes

For example, in a triplestore, you could start with people and medical treatments, and later create the idea of a patient, without having to redo any of the data you've already committed

What this means is that you can start simple and add as you need.

Instances can belong to many classes

- In a relational system, a row exists in only one table
- In a graph DB, you can create data and later associate it with newly created classes

For example, in a triplestore, you could have an instance (a URI) that represents a person, who is both a patient and a provider. In a relational system, you create two records and then have to have another way to determine that they are the same.

What this means is a great reduction in redundancy

The Relationship of Applications and Schema

- In a relational system, schemas are built to the needs of applications. They are owned by the application
- In a triplestore, the schema is a shared resource. Many applications cooperate around the same model

Traditionally, every application has its own data model. This is what drives the cost of integration so high. In a semantic system, we share as much modeling as we can, running the integration costs down.

What this means is that data integration is almost a bi-product of building semantically

Our Quest

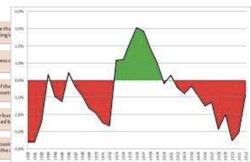
- Find the stability
- The enduring business themes

One aspect

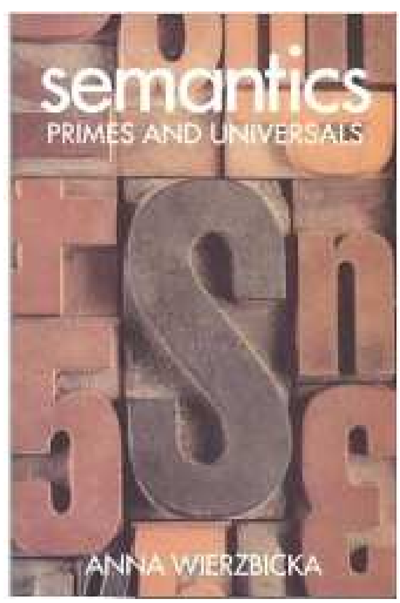
- Things that are generally well agreed upon

Person

Account

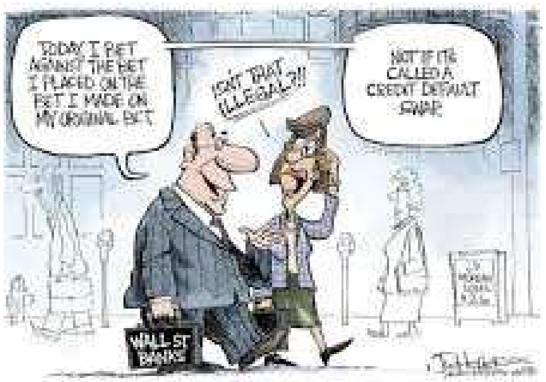


Semantic Primes



Abstract until everyone agrees

Credit Default Swap

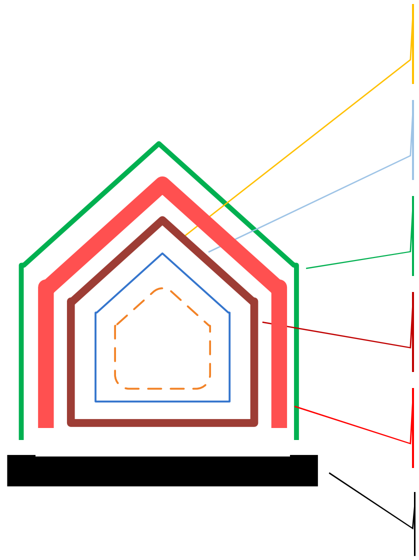


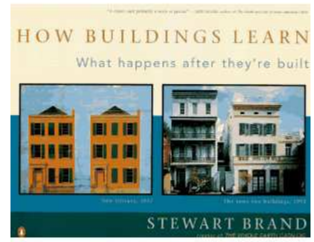
Obligation

Core

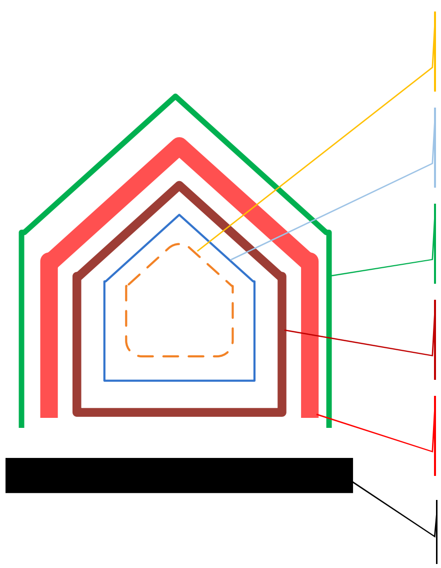
- We're looking for a core that is stable
- Not unchanging
- Just rooted
- In such a way that we can let the things around it change at their natural rate

An Analogy: Pace Layering





An Analogy: Pace Layering



Stuff	Aesthetics
Space	Navigation & Search
Skin	Capability
Systems	Formal Taxonomy
Structure	Enterprise Arch
Site	Enterprise Ontology

Our Position

- A Core Enterprise Ontology:
  - Is Necessary – in order to provide some sort of framework and stability to the rest of your information systems efforts
  - Should be Understandable – the value expands greatly when it is well understood and used
  - Can be Elegant – there are typically a few hundred concepts that cover most information system activity
  - Can be Flexible – the model should be able to evolve in place and be extended by some groups without impacting all

At the end of the EO process

- You will have a model that is 1% as complex as your current models
- And that seems to gain as much as it loses in fidelity in use

Example of how a simple ontology isn't lossy

Getting Started

- Most companies don't want to "boil the ocean" with an Enterprise Data Modeling effort
- They believe (correctly) that this could take 1-2 years, with questionable payoff
- We're going to suggest some techniques for doing this in a much more streamlined fashion
- You could complete this in 3 months
- If you also do it agilely, you could begin using it in 1

Six techniques / methods for getting your EO complete in 90 days

1. Separate your artifacts by purpose
2. Use gist
3. Model the real world
4. Economize expression
5. Postulate the solution, don't extract it
6. Use inference to check for errors

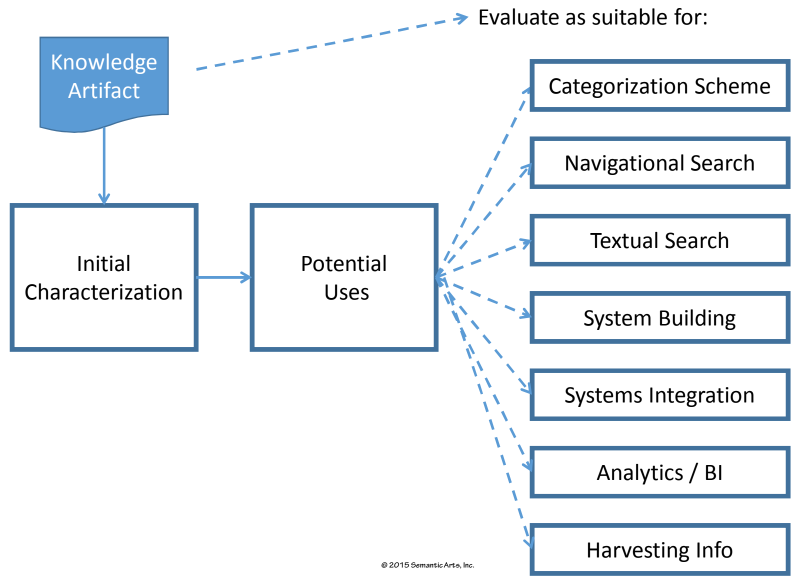
1) separate your artifacts by purpose

- Taxonomy / Ontology Assessment
- Became Knowledge Artifact Assessment

Purpose-Driven

- The assessment of an artifact turned out to be very contextual
- And the main context is "what is the intended purpose for this knowledge artifact?"
- Some characteristics of a taxonomy (for example MECE (Mutually Exclusive/ Completely Exhaustive)) are very important for some purposes,and get in the way for others

For each Knowledge Artifact



We've discovered

- These purposes have wildly different definitions of "goodness"
- An artifact for textual search wants to have many, many synonyms for every term
  - (One of our clients said they had on average 29 synonyms for every term)
- By contrast, having lots of synonyms tends to confuse things during integration and system-building

118

Modularity to the rescue

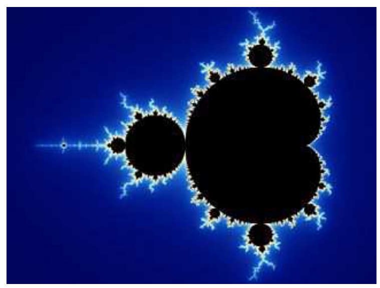
- In the 2nd Act, we mentioned the power of modularity
- Now we can begin to harness that modularity
- Rather than either throw away all the excessive synonyms
  - Or load up on them
  - It's not an either/or question
- Have the core, and modularly extend it with the synonyms
- It you don't need the synonyms; only get the core
  - If you do get them, organize them around the core

What we've found

- Many classes are merely taxonomic differences
  - It doesn't add anything to make them classes
  - And it interferes with the elegance of the model
- Moving them to modules that can be under separate governance speeds development and eases maintenance

Fractal Modeling

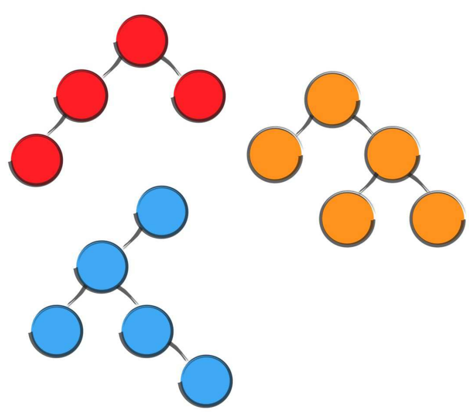
- Move most of your taxonomies out of the class structure
- Separation of governance
- Simplification of the model



121

Move Minor Distinctions to Taxonomies

- We're tempted to put everything we know into our ontologies
- Many distinctions are best "pushed" to taxonomies
- Where mere mortals can debate and rearrange them
- Without destabilizing the ontology



Special Individuals

- Most ontologies have a small number of "special" individuals/instances
- For example, the only semantic distinction between accounts payable and accounts receivable is who "you" are (your firm typically)
- Sooner or later, this becomes a definition based on an instance or several instances
  - (myOnt:_myCompany0001)
- Other special instances include those that participate in definitions that cannot be formally defined practically and have to be accepted
  - ("male" and "female", or "exempt" and "non-exempt", for instance)

Classes and Taxonomic Instances

- Strive to be more like GeoNames than Snomed

	GeoNames	Snomed
Concepts	10 million geographical names	Presumably millions
Classes	19	303,035
Properties	33	152
Taxonomic Categories	645 "feature codes"	In classes

Separation of Concerns

- Meaning (OWL) vs Structure (RDF Shapes)
- If you're trying to answer the question "which properties go on this class" by looking at the ontology, you've collapsed these two ideas
- The ontology should be the province of coining new terms, establishing meaning and providing the rules for inference

RDF Shape Example

```
<ProductRef> {
       rdf:type (spo:ProductReference)
,      gist:identifiedBy @<ProductID>
,      spo:describedBy @<ProductReferenceDescription> ?
,      gist:memberOf  (@<ProductRange> | @<ProductSubRange>
|@<ProductSubSubRange> |@<NewProductRange>) *
,      gist:categorizedBy @< ProductOrComponentType> *
,      gist:categorizedBy @< ProductFunction> *
,      gist:conformsTo @<Standard> *
,      gist:specifiedBy (@<SpecEntry> |@<TabularSpecEntry>)?
}
```

2) Use gist

gist

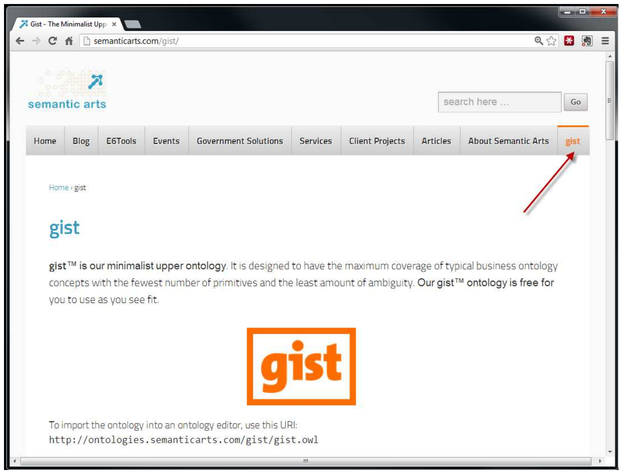
http://semanticarts.com/gist

127

Introducing gist

- An upper enterprise ontology containing a minimal set of concepts required by most businesses.
- Copyright Semantic Arts, Inc.
  Rights to use are conveyed under the
  Creative Commons Attribution-ShareAlike 3.0 license.
- Current version at
  http://ontologies.semanticarts.com/gist/gist.owl
- Consists of a core plus several "subgists."

Get gist from our web site



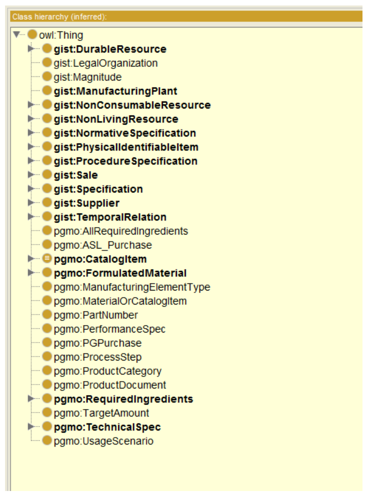
(www.semanticarts.com/gist)

gist Attribution

If you use gist, please include the following attribution:

This work is derived from and/or directly uses concepts from gist, a copyrighted ontology from Semantic Arts, Inc. Rights to use are conveyed under the Creative Commons Attribution-ShareAlike 3.0 license http://creativecommons.org/licenses/by-sa/3.0/legalcode

Early Projects

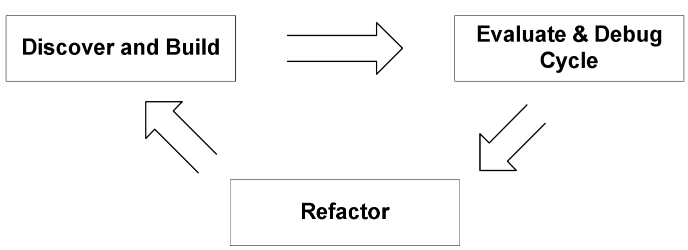


- About half the EO classes were derived from gist

131

Evolution

- We evolved gist and our methodology

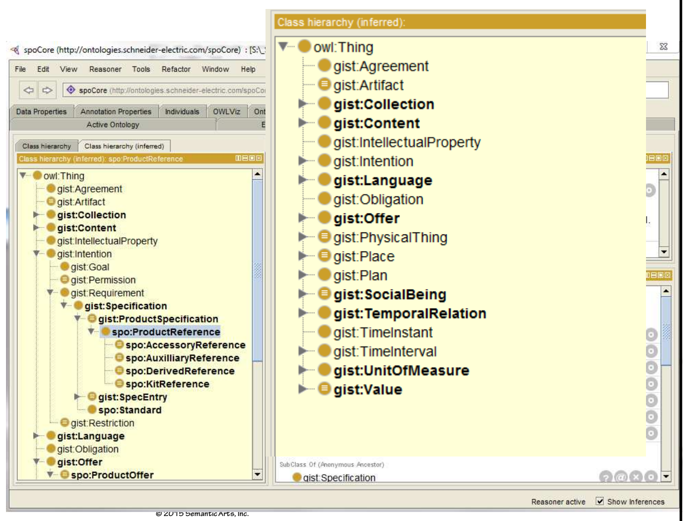


132

Current Projects

gist

- Most classes derived from gist, without even trying



What is gist?

- Fewest concepts
- Broadly agreed on, across industries
- That cover most, of most, enterprises
- Least ambiguity
- Stable, ten years old (used in about a dozen major projects)
- Evolving (we keep refining it)

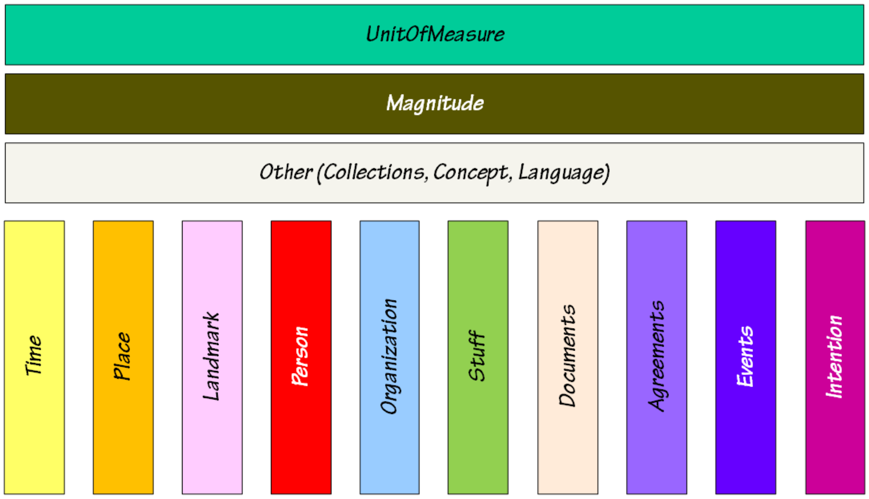
Abstract?

- Many people think that to cover an entire enterprise with a few hundred concepts, they'd have to be pretty abstract
- And some upper ontologies are quite abstract
- One, for instance, has a high-level distinction between "endurants" and "perdurants" (things v. events).
- We think of these as "abstract abstractions"

Concrete Abstractions

- We're trying to work at the level of "concrete abstractions"
- Classes where the members are easily grasped, if slightly abstract
- Person is a concrete abstraction
- We can create an instance of Person, and as we learn more about the person, we may decide (by assertion or inference) that they are also more specific types of persons (Doctors, Brokers, Adults, etc.)

gist – Major Families of Classes



Learning gist

- Manageable number of concepts

Ontology metrics:

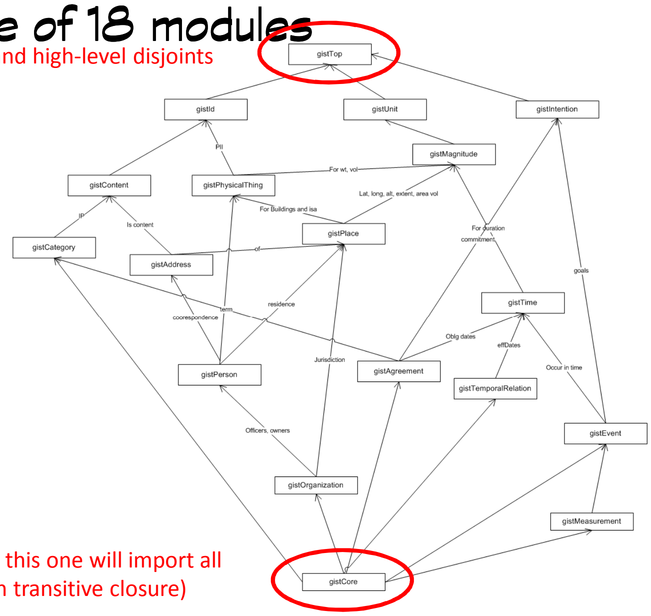
Axiom	1275
Logical axiom count	454
Class count	139
Object property count	108
Data property count	20
Individual count	11
DL expressivity	SROIQ(D)

Modular

- As of 7.x, gist is very modular
- Understanding how the modules fit together adds a bit of conceptual baggage
- However, each module now is so simple as to be almost self-explanatory

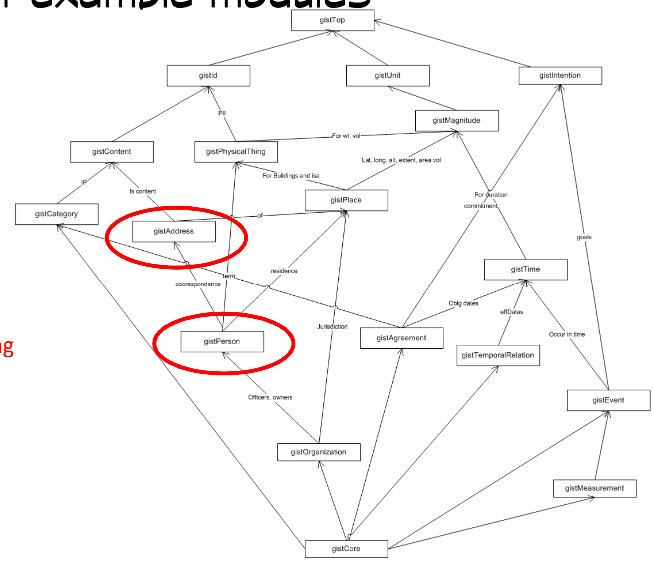
Gist is made of 18 modules

Primitive concepts and high-level disjoints



Opening this one will import all
(through transitive closure)

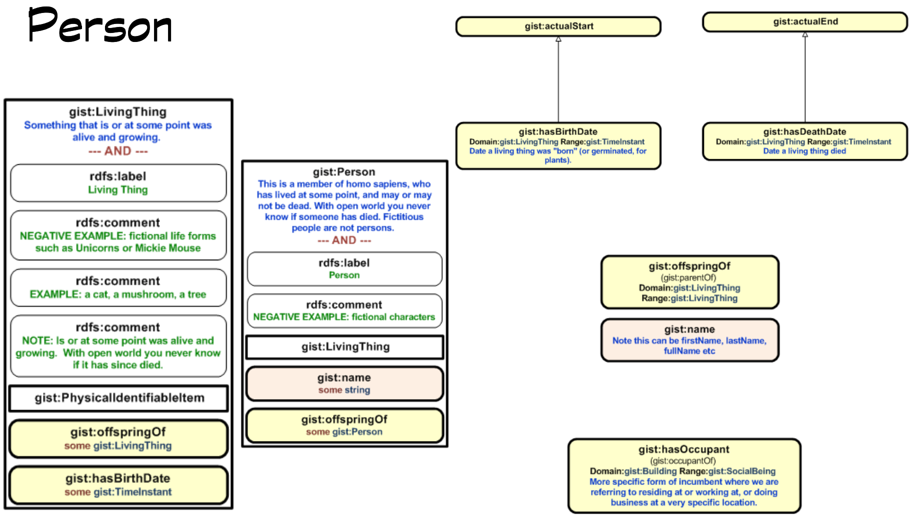
An couple of example modules



Some very useful primitives in an area often confused

Person as (once) Living Thing

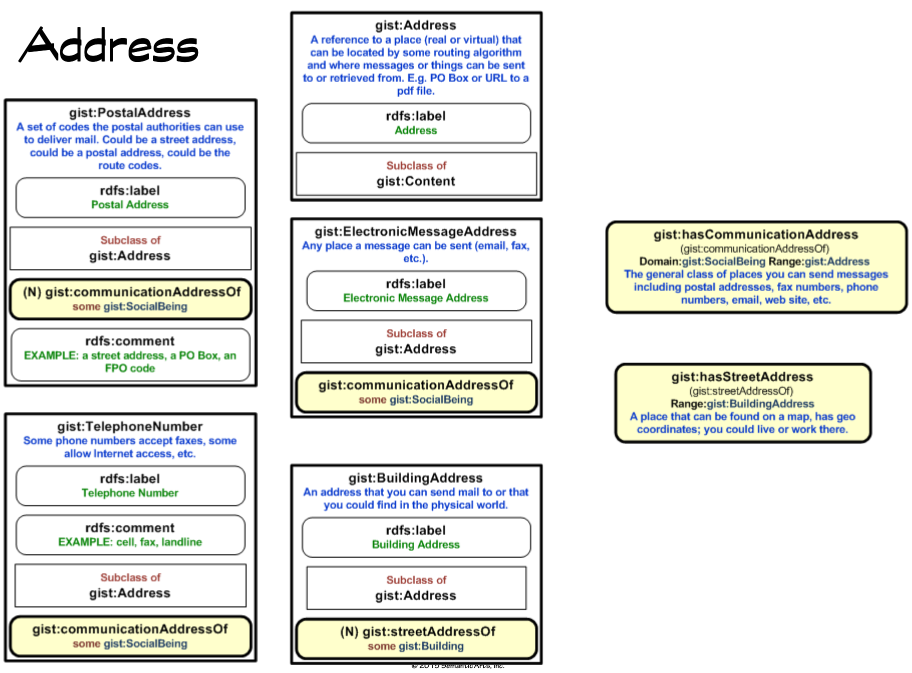
Person



Address

- We treat address as a first-class object (not an attribute of a person or company)
- And the use of that address by a Person or Organization as essentially a communication preference
- This one change makes the chaos of addresses in most enterprises manageable

Address



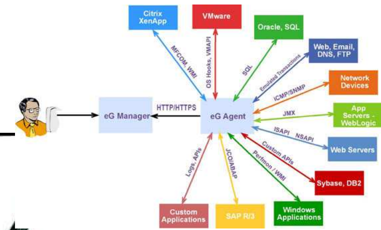
3) Model the real world

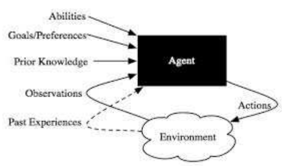
- It's very easy to fall into the trap of modeling the concepts you find in the application
  - (or in people's heads, which often came from an application)
- Many of them are ok
- But the only way to know which is which, is to try to get as close to the real world as you can.
  - It will shine a light on which are contrived
- Some examples: PIMS, Tabulars, Sections, most junction records., legs, and most booleans

Stay away from abstract abstractions

- Person and Agent are both abstractions
- But pretty much everyone (other that foaf) agrees on what a person is
- But agent…
  - In many cases it is Person or Organization
  - But sometimes machine
  - Or program
  - Or animal
- The real meaning is in the property (the agent on "wasBittenBy" is Animal, whereas the agent on "durablePowerOfAttorney" is Person.)

Agents



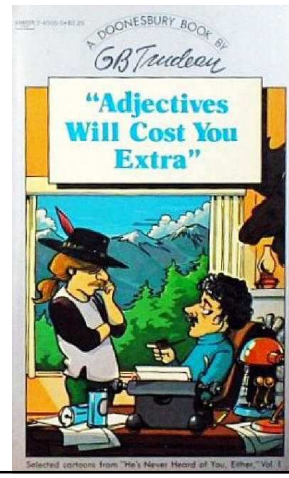


4) Economize expression

- It's tempting to put everything you know plus everything you learn in the ontology
- "you might need it"
- These things clutter up the result
- And confuse the use
- Many are unlikely to be widely agreed upon

In an Enterprise Ontology

- Less is more
- Don't get paid by the pound
- Remember what happened with lines of code?



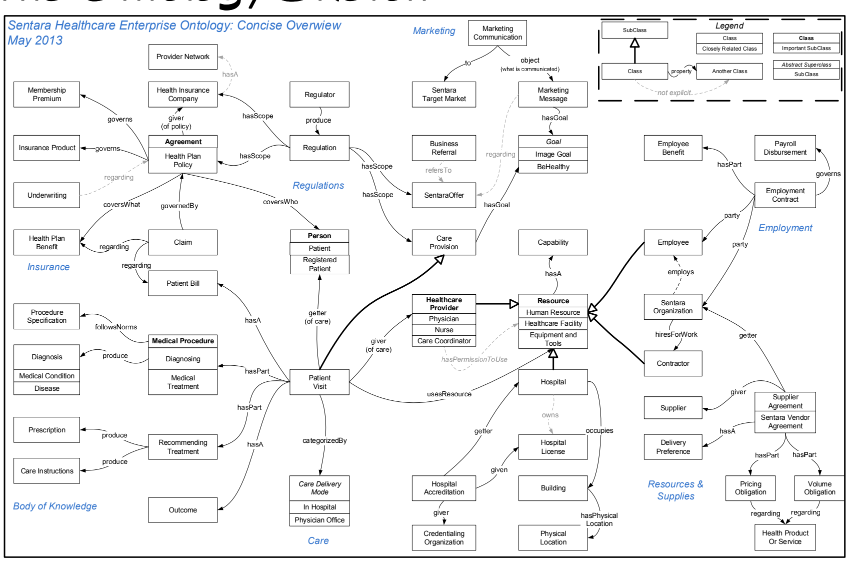
Reducing Cognitive Load

- # of things you must know to be competent with the ontology
- # of things you must be in agreement with in order to commit to the ontology
- # of concepts shared

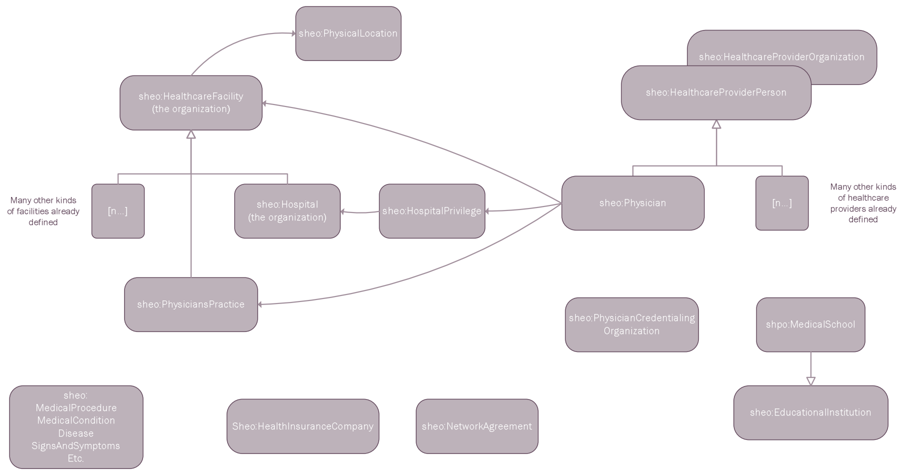
Economizing Properties

- Typical large enterprises have millions of properties (attributes/ columns) in their legacy systems
- This is mostly a product of arbitrary design decisions
- We need to be vigilant
- Properties (with a few very narrow exceptions) cannot be formally defined; we must learn them all
- Our target is to get to a few hundred

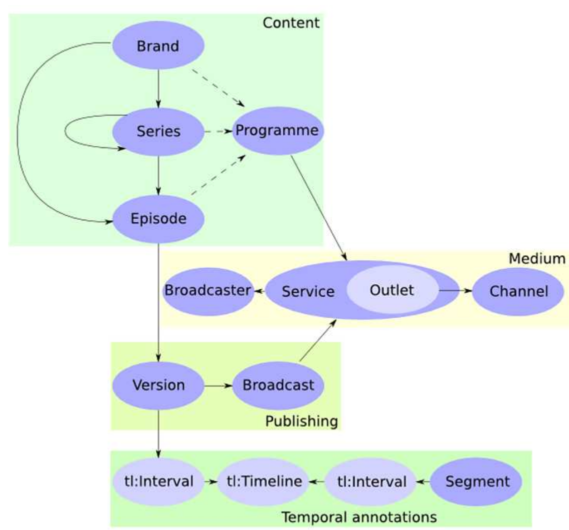
The Ontology Sketch



Overall Shape of the Ontology



BBC Programmes Ontology sketch

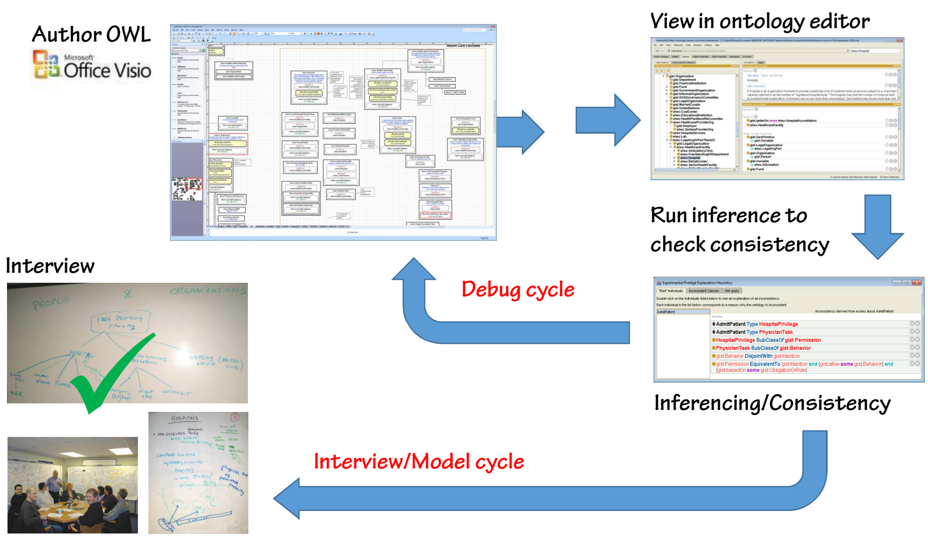


154

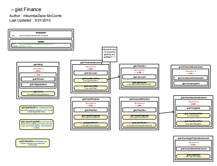
5) Postulate the solution, don't extract it

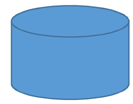
- Over the last several years we have been moving from a "discovery model" to a "postulate model" for ontology development

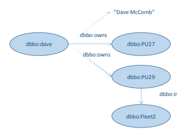
Discovery Methodology



Postulate Methodology







157

Then check existing systems

- Find the correlates
- And look for the things not postulated
- How do they relate?

158

Mapping

- One of the better ways to check for coverage

Physician Data Elements						
Section/ Category	Required Fields	Subject: the subject will be an instance of (rdf:type) the class shown	Predicate	Object: the object will be an instance of (rdf:type) the class shown	Existing Taxo	Notes: when triples are noted, the subjects/object will be instances (rdf:type) the classes shown
Demographic data						
	First Name	sheo:HealthcareProviderPerson (and subclasses)	sheo:firstName	[string]		
	Middle Initial	sheo:HealthcareProviderPerson (and subclasses)	sheo:middleNameOrInitial	[string]		
	Last Name	sheo:HealthcareProviderPerson (and subclasses)	sheo:lastName	[string]		
	Designator/Title	sheo:HealthcareProviderPerson (and subclasses)	sheo:hasProfessionalDesignation	sheo:TitleDesignation [taxo value]	PSDB - Title	
	Alternate First Name	sheo:HealthcareProviderPerson (and subclasses)	sheo:altFirstName	[string]		
	Practicing Status- (Instate practicing physician)	sheo:PracticingStatus	gist:categorizedBy	sheo:PracticingStatusCategory [taxo value]	EPD - Clinician Status? Or Physician Status Type?	preceding triple: **sheo:Physician gist:hasA sheo:PracticingStatus.**
	Date for Inactive Practicing Status	sheo:PracticingStatus	gist:actualEnd	gist:TimeInstant		A sheo:PracticingStatus is a temporal relation, therefore it has a start and end date. (You may only be interested in the en date.)

Many attributes collapsed

- We found 10 attributes for identity (SSN, Provider DB id, etc) are all covered by gist:identifiedBy

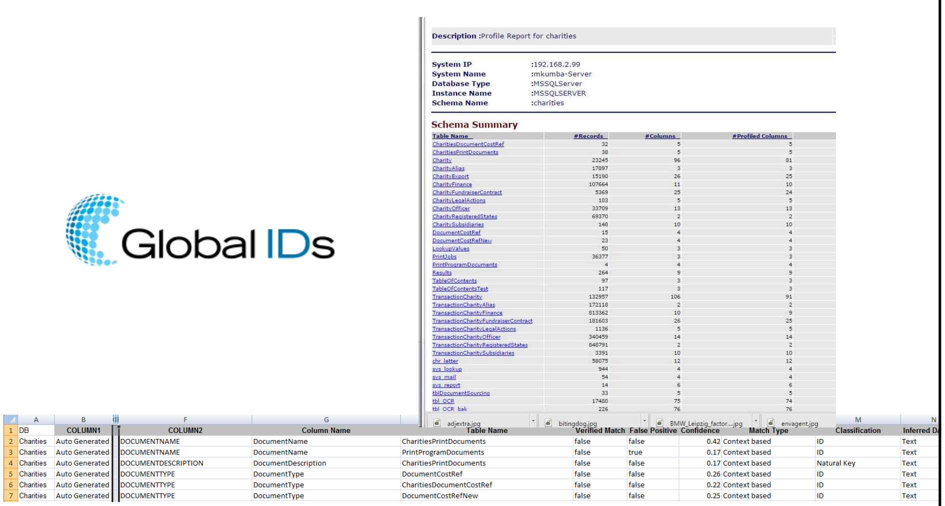
					sheo:GetProofIDgist:allocatedBy [URI for org or app]	
SSN	sheo:HealthcareProviderPerson (and subclasses)	gist:identifiedBy	sheo:SocialSecurityNumber		succeeding triples: **sheo:SocialSecurityNumber gist:uniqueText [string] . sheo:SocialSecurityNumber gist:allocatedBy [URI for org or app]**	100-55-1234
Provider Database ID	sheo:HealthcareProviderPerson (and subclasses)	gist:identifiedBy	sheo:ProviderDatabaseID		succeeding triples: **sheo:ProviderDatabaseID gist:uniqueText [string] . sheo:ProviderDatabaseID gist:allocatedBy [URI for org or app]**	888
Person ID	sheo:HealthcareProviderPerson (and subclasses)	gist:identifiedBy	sheo:PersonID		succeeding triples: **sheo:PersonID gist:uniqueText [string] . sheo:PersonID gist:allocatedBy [URI for org or app]**	150
Medicaid ID	sheo:HealthcareProviderPerson (and subclasses)	gist:identifiedBy	sheo:MedicaidID		succeeding triples: **sheo:MedicaidID gist:uniqueText [string]**	5703333

Mapping added a bit to the ontology

- As you'd imagine, the act of crossing all those t's and dotting all those i's lead to a few extensions to the model
- But not much, and nothing that really changed the shape of the model
- And this is what we would hope: even as we add additional data sources, internal or external, we expect them to be extensions to the existing structure

Profile

Global IDs



Description : Profile Report for charities

System IP :192.168.2.99
System Name :mkumba-Server
Database Type :MSSQLServer
Instance Name :MSSQLSERVER
Schema Name :charities

Schema Summary

Table Name	#Records	#Columns	#Profiled Columns
CharitiesDocumentCostRef	32	5	5
CharitiesPrintDocuments	38	5	5
Charity	23245	96	81
CharityAlias	17897	3	3
CharityExport	15190	26	25
CharityFinance	107664	11	10
CharityFundraiserContract	5369	25	24
CharityLegalActions	103	5	5
CharityOfficer	33709	13	13
CharityRegisteredStates	69370	2	2
CharitySubsidiaries	148	10	10
DocumentCostRef	15	4	4
DocumentCostRefNew	23	4	4
LookupValues	50	3	3
PrintJobs	36377	3	3
PrintProgramDocuments	4	4	4
Results	264	9	9
TableOfContents	97	3	3
TableOfContentsTest	117	3	3
TransactionCharity	132957	106	91
TransactionCharityAlias	172118	2	2
TransactionCharityFinance	813362	10	9
TransactionCharityFundraiserContract	181603	26	25
TransactionCharityLegalActions	1136	5	5
TransactionCharityOfficer	340459	14	14
TransactionCharityRegisteredStates	848791	2	2
TransactionCharitySubsidiaries	3391	10	10
chr_letter	58075	12	12
sys_lookup	944	4	4
sys_mail	54	4	4
sys_report	14	6	6
tblDocumentSourcing	33	5	5
tbl_OCR	17480	75	74
tbl_OCR_bak	226	76	76

adjextra.jpg | bitingdog.jpg | BMW_Leipzig_factor....jpg | envagent.jpg

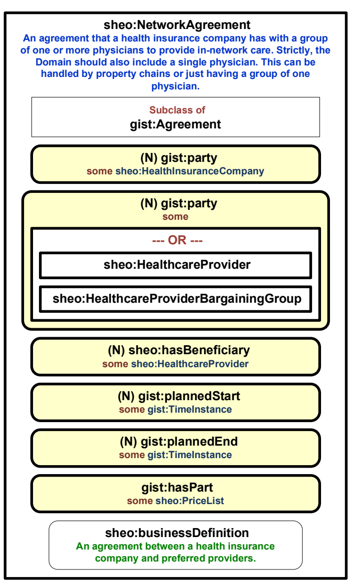
	DB	COLUMN1	COLUMN2	Column Name	Table Name	Verified Match	False Positive	Confidence	Match Type	Classification	Inferred D
2	Charities	Auto Generated	DOCUMENTNAME	DocumentName	CharitiesPrintDocuments	false	false	0.42	Context based	ID	Text
3	Charities	Auto Generated	DOCUMENTNAME	DocumentName	PrintProgramDocuments	false	true	0.17	Context based	ID	Text
4	Charities	Auto Generated	DOCUMENTDESCRIPTION	DocumentDescription	CharitiesPrintDocuments	false	false	0.17	Context based	Natural Key	Text
5	Charities	Auto Generated	DOCUMENTTYPE	DocumentType	DocumentCostRef	false	false	0.26	Context based	ID	Text
6	Charities	Auto Generated	DOCUMENTTYPE	DocumentType	CharitiesDocumentCostRef	false	false	0.22	Context based	ID	Text
7	Charities	Auto Generated	DOCUMENTTYPE	DocumentType	DocumentCostRefNew	false	false	0.25	Context based	ID	Text

162

6) Use inference to check for errors

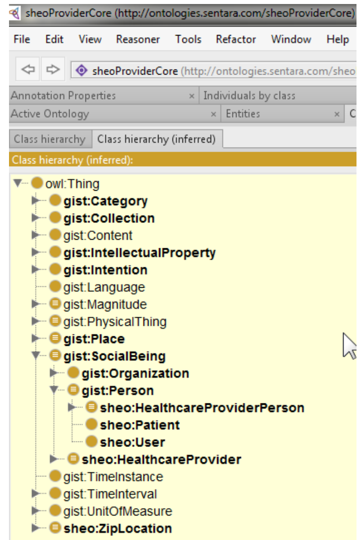
- Use the power of semantics to help find errors
- And hidden similarities

Partial Example - Network Agreement



- Even though the agreement between physicians and insurance companies didn't show up in the data we were provided, we know it exists
- It is an essential bit of information, which initially we will only be aware of for InsCo agreements. But eventually we may become aware of others, and this structure allows us to hold a place for them without incurring any overhead

Check it for logical consistency using Protégé or Top Braid



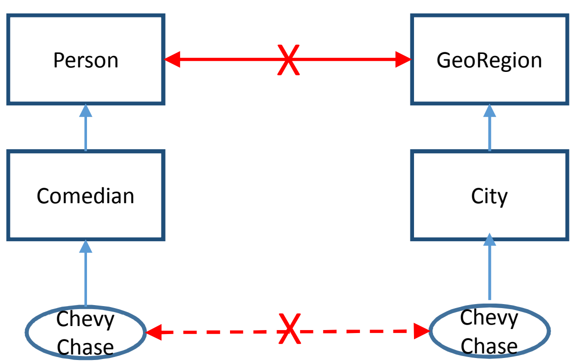
Agile -- Detecting Errors

- The "easy to change" aspect of agile requires a way to detect errors.
- Two of the more effective that we use are
  - High-Level Disjoints
  - ABox Unit Tests

Disjoints

- Most of the errors that a tableaux reasoned will surface stem from disjointness (or negation/ complement) assertions
- No disjointness = no error checking

High-Level Disjoints



ABox Unit Tests

- Check for things that should not arise in the course of using the ontology, and use them for unit testing

```
ASK {
     ?tc rdf:type sa:TimeCharge .
     ?tc gist:actualStart ?t1 .
     ?t1 gist:universalDateTime ?start .
     ?tc gist:actualEnd ?t2 .
     ?t2 gist:universalDateTime ?end  .
     FILTER(?end < ?start)
}
```


Case Study

- From discovery to postulate and test case study

Questions?

- For more

  http://semanticarts.com