THOUGHTS ABOUT AN ARCHITECTURE FOR THE SEMANTIC WEB

William Bricken
August 2004

Caveat

I use a *declarative presentation style*, which may cause some confusion between opinion and fact. Everything below is at least supported by my experience.

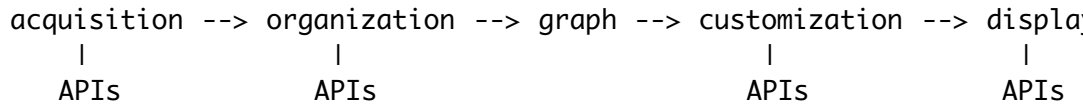
Summary

Main points are indicated by double asterisk ** in the sequent, and are collected here:

-- The graph data-structure is foundational.
-- APIs provide generic flexibility.
-- Knowledge is contextual.
-- Reducing cognitive load is the long term win.
-- Corporate data is the big win.

A Straw-man Architecture

Here is an architecture that supports a general model of information organization (knowledge construction) and can expand opportunistically as dictated by corporate vision and by business opportunities. A simplified picture of this architecture is:



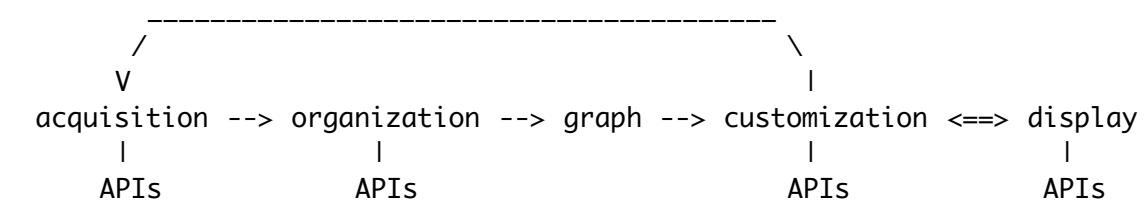
Implicit Feedback

The above architecture presents a linear flow from acquisition to display. There are implicit functional loops, such as

-- customization to acquisition
The user initially customizes acquisition through a search query.

-- display to customization
After seeing the display, the user reorganizes data both locally (customization) and globally (acquisition).

Including these feedback loops yields the following architectural diagram:



*** The graph data-structure is foundational ***

Like the foundation of a building, the central graph data structure must be strong and stable. Reworking a foundation after floors are built is dangerous -- for architecture, resources, and product. Thus the graph supports no APIs. The graph is the correct data structure from which to evolve generic capabilities.

There's a distinction between the abstract graph data structure and its implementation. The graph data structure must be abstracted from its lower-level implementation, and provided with the appropriate set of accessors, modifiers, etc. Efficiency comes from optimizing repetitive transactions in the graph *implementation*. In general, performance optimization comes from customizing a few critical inner loops; optimization of the vast majority of code gains little and risks a lot.

The abstract data structure must support graph optimization techniques, including

-- redundancy removal
-- path optimization
-- unique nodes
-- structure sharing
-- merging nodes
-- node clustering
-- node abstraction
-- implicational structure
-- sparse/dense management

There is, of course, an associated list of content-specific graph optimization techniques.

*** APIs provide generic flexibility ***

Some of the strengths of an API architecture are:

-- opportunistic flexibility
-- customization to different input sources and output displays
-- modularization
-- serendipitous applications
-- partnerships

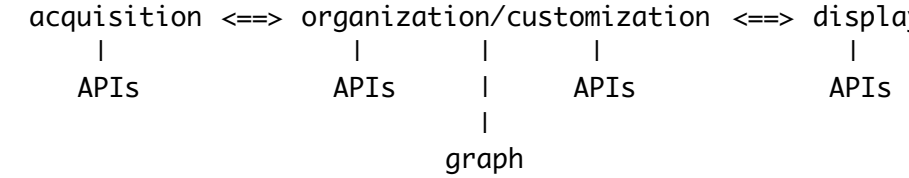
It is critical to incorporate APIs that leverage the work of influential sites (Google, Ebay, Amazon, etc.). These sites will be incorporating significant metadata as they move toward the semantic web, blurring the distinction between acquisition and organization.

Right now, to support the generic API architecture, *organization* of data is a priori to the user experience; it is a "generic conceptual standardization". *Customization* APIs apply only after the generic graph is built, and are limited to user controlled sorting and filtering.

What the Semantic Web Needs (technical rationale not included)

The problem that needs to be solved is the *conversion of information to knowledge*. An list interface (such as Google) can provide necessary condensation of vast quantities of information, however, the goal is not data condensation, it is to present just the right information to meet the user's objective of generating knowledge.

Organization by clustering is not creation of knowledge, at best it is creation of metadata that makes knowledge construction easier. The theme here is that clustering and customization should be viewed as the same process, rather than separated into components external to the user (textual organization) and internal to the user (semantic customization). The main implication for the strawman architecture is that the architectural features *organization* and *customization* should be merged, yielding this (symmetrical) architecture:



Most desirable would be dynamic clustering driven by user objectives. Note that there is no feedback loop from display to acquisition; all transactions

are mediated by the organization/customization module which uses the graph as the central mechanism of structure. The two sets of APIs connected to the organization/customization module still refer to external and internal (to the user) information structuring. Uniting the two will require some research and analysis, outlined next. Multiple APIs for organization/customization refer to multiple perspectives on the same dynamic database. Thus the graph does not necessarily represent pre-structured search information, it represents the multiple requirements of multiple external and internal views.

This architecture suggests a different type of interface, for which the acquisition function is hidden from the user. The user formulates the knowledge objectives, which would include keyword and syntactically structured search as well as custom and goal-oriented guidance. the architecture would acquire information in the background as dictated by semantic needs. The user experience is not one of search-and-organize, it is one of ask-for-what-you-want.

*** Knowledge is Contextual ***

A user is dynamically converting information into knowledge through two main methods: targeted search and browsing. Each makes a fundamentally different use of the tool, but both have the same strict criterion for success: "Do I *understand* the information in the display?" Thus an architecture for the Semantic Web also includes an educational component (understanding knowledge), as well as an organizational component (converting information into knowledge). The *semantic gap* is the difference between presented and desired information.

The central issue is that user semantics -- the meaning that is being sought within the information -- is *unique to each user* and to each user objective. Organization of the external structure of information (both syntactic and metadata) at best presents to the user a generic template of whatever was input during the acquisition step. A centralized graph data structure built from this generic information fails to include what the user is seeking. Generic template information biases all further filtering/customization. Organizing/clustering acquired information first in terms of the user's objectives places a semantic component directly within the central graph data structure.

How can semantic organization be achieved? The Semantic Web is a beginning attempt to present *knowledge* in response to queries. Currently the user must provide explicit knowledge structuring; the task of tool design is to minimize the burden of this specification, and to minimize the display of undesired information. What is most needed is *semantic clustering*, classification based on knowledge and context, rather than on the structure of acquired information.

Some relevant techniques include

-- semantic organization:
clustering and categorization informed by the user's objectives
-- semantic associations:
a graph of relationships between knowledge and objectives
-- content integration:
joined semantic categories
-- inferred structure:
deduced connectivity within information
-- cognitive maps:
a graph of the user's understanding of relevant concepts

Contextual Organization

The objective of semantic query processing is to provide the user with what is *implicitly wanted*, rather than what is explicitly requested. That is, the architecture needs to provide *contextually organized knowledge*, making information easier for a user to understand. Many techniques have been explored to achieve this end, and the ultimate solution will probably involve a hybrid of them all.

Textual clustering makes the argument that classification based on text snippets and statistical term organization can be quite satisfactory, without any predefine grouping, pre-build knowledge base, or pre-processing of all document collections. This approach is solely looking outward, and does not include any notion of user-defined objectives (thus forcing the strawman architecture to separate organization from customization). User input is confined to interaction with the search engine, in the form of the initial search query. One can add filters by semantic fields, such as pricing, using what might be called *keyword semantics*. What is actually needed is a model of the user's objectives.

What is known about semantic customization includes:

-- user goals and motivations are *unique*, degrading the effectiveness of externally grounded information organization
-- user goals are often dynamic, ranging from browsing to realtime learning
-- metadata is not semantic data
-- filtering out important material (false positives) is a far worse error than returning unimportant material (false negatives)
-- taxonomic organization is helpful, but does not scale and requires multiple perspectives
-- users anthropomorphize their machine interactions extensively
-- textual display does not scale
-- relevance is more important than accuracy/correctness

-- visualization tools do not scale
-- semantics is required both during organization and during customization
-- in the case of end-users, organization and customization are the same
-- user goals and understanding are unique
-- information accretes
-- distillation of concepts and themes as clustering

The net result is that *organization and customization are the same thing*. What is needed is

-- background acquisition driven by customization
-- contextual queries
-- strong dynamic filtering based on semantic criteria
-- goal inference and problem solving
-- hierarchical path completion (top-down and bottom-up problem solving)
-- semantic normalization (using metadata for classification)

The software implications of a semantic organization/customization strategy include

-- a programming language that supports contextualization (first-class functions and macros)
-- building customized user models and cognitive maps
-- persistent memory of a user's prior intentions
-- organization based on semantic, goal-oriented requests

As well, our architecture needs to include some standard structural and taxonomic filtering, including

-- shallow knowledge taxonomies
-- concept/argumentation maps
-- Bayesian spam filter-like technology used to detect irrelevant pages

Content Analysis

To generate semantic knowledge, the content of a document needs

-- categorization
-- clustering
-- entity extraction
-- fact extraction
-- summarization
-- indexing

This can be explored by building small contextual taxonomies (words and concepts) for some specialized applications (egs: Mathematica, construction industry's Uniform Building Code, teaching US History, Boeing's CATIA). It is also necessary to build user models, with dynamic update (machine learning)

Why the Semantic Organization Problem Is Not Solved

The Appendix contains a description copied for the Triplehop website. Triplehop has made semantic search its primary differentiator. Yet the search fields and filters that the user fills in are not particularly different that those available to (and used by) other search engines.

There are several simple reasons why semantic search (and contextual programming) has not been fully incorporated into software builds:

-- companies are confusing metadata with semantics
-- user profiling is usually a lower priority development need
-- HCI folks are generally not good programmers,
and thus have communication problems
-- cognitive mapping is subtle
-- in general it is believed that users are groups and not individuals
-- semantics is considered too difficult
-- only the most powerful programming languages allow
contextual functions

This last reason is the most important. Languages that permit context to be incorporated into a computation use *first-class functions* and *dynamic macros*. Almost all widely used languages are not powerful, since they are designed for "average" programming tasks. As well, many object-oriented languages do not permit the dynamic construction of classes, the OO analog of macros. There are well known reasons for avoiding the power of contextual programming, none of which apply to a start-up.

Two examples of contextual programming in current use are the Yahoo storefront (around 20K? stores) and most of the airline reservation industry. As well, contextual design is necessary for parallel processing programs.

Quick Analysis of Competitive Products

These lists are certainly not comprehensive and, as quick overviews, may contain significant errors.

Taxonomy of Services

- Business
  - document management
    - entrieva, 80-20, hyperwave
  - search and organize
    - copernic, northernlight, vivisimo, youramigo, triplehop
  - knowledge
    - thebrain, exsys, venetica, semagix, spotfire
  - visualization
    - anacubis, objectfx, xplane
  - training
    - conceptsystems, banxia, macroVU
- General
  - visual organization of search
    - TouchGraph, KartOO, Groxis
  - visual organization of data
    - Hypertree, MapNet, WebMap,
      netviz, chartworksinc, tableausoftware
  - conceptual relations
    - CIOS, Interspace
- Specialized
  - marketing intelligence
    - intelliseek
  - document/data mining
    - DocMINER, inxight
  - specialist tools
    - Cytoscape, TextArc, Mathematica

Somewhat orthogonal competitive alternatives

-- corporate accounts and cash-from-customization (data mining)

- entrieva, 80-20, hyperwave
- copernic, northernlight, vivisimo, youramigo, triplehop
- thebrain, exsys, venetica, semagix, spotfire
- DocMINER, inxight

-- visual organization of data, expanded API data-types (multimodal)
Hypertree, MapNet, WebMap,
netviz, chartworksinc, tableausoftware
anacubis, objectfx, xplane
TouchGraph, KartOO, Groxis

-- education and building a product experience base (tools)
conceptsystems, banxia, macroVU
Cytoscape, TextArc, Mathematica

-- mental mapping, personalization of knowledge (marketing intelligence)
CIOS, Interspace
intelliseek

Educational Market -- an Irresistible Teacher's Tool

*** Reducing Cognitive Load is the Long Term Win ***

Strategy
provide a service to the teacher that is irresistible
target all teaching levels, especially college courses
seed familiarity rather than build a profit center
high level authoritative certification of educational approach
now is an appropriate time to diversify the product offering

Gotchas
high school teachers do not have time to use tools,
even excellent tools
government schools are not a stable profit center
it is undesirable for teachers to become better teachers

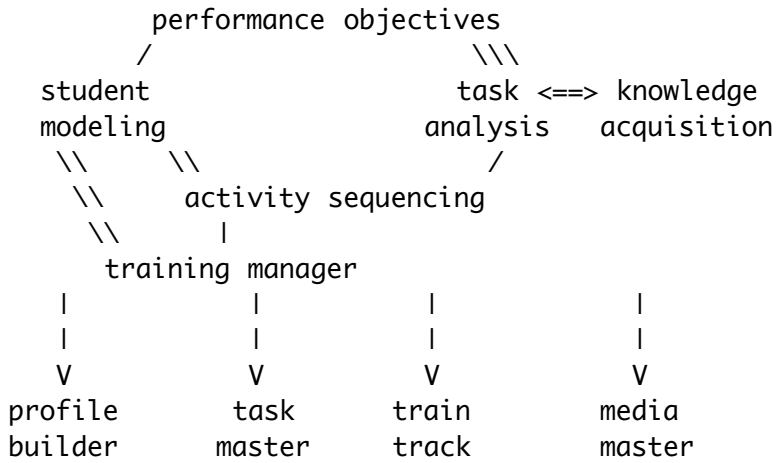
Educational Theory
situated learning
social learning
individualized instruction
non-sequential modular teaching

Educational Software Tool/Teacher's Assistant
Make a map of your subject matter
let students roam the dynamically pre-organized territory
teachers can dynamically monitor student activity with MetaMap

LessonGraphs	use it to organize lesson plans
ProfileBuilder	use personalization for each student
TrainTrack	use dynamic reorganization based on student mastery
TaskMaster	automated updating of the teacher's metamap

Extensions
- knowledge acquisition (MediaMaster)
- student map of learning/school-work
- canned half-year exploratory "lessons"
- student evaluation wizard
- built-in tutor
- an API for macroVU's "Can Computers Think"
- task-specific interface

Instructional Architecture and Tools



Corporate Training Market -- building familiarity

*** Corporate Data is the Big Win ***

APIs to corporate protocols for
- documents
- email
- ORACLE and other databases
- scripting languages in general use
- SQL
- windows-like tree display

in-house staff to take care of special clients
- relations
- marketing and sales
- support
- education
- customization

Corporate Training Market

```
-- corporate training ~$50B market
-- DoD training ~$30-50B
-- use corporate training  to gain corporate data customers
```

APPENDICES

Triplehop

A NY-based *contextual* enterprise search engine.

Primary motivation:

"The relevancy of search results depends on the task at hand and the intended use of the information collected, and thus results change based on context."

Technical description:

"The basic objective of a predictive algorithm for collaborative filtering (CF) is to suggest items to a particular user based on his/her preferences and other users with similar interests. Many algorithms have been proposed for CF, and some works comparing sub-sets of them can be found in the literature; however, more comprehensive comparisons are not available. In this work, a meaningful sample of CF algorithms widely reported in the literature were chosen for analysis; they represent different stages in the evolution of CF, starting from simple user correlations, going through online learning, up to methods which use classification techniques. Our main purpose is to compare these algorithms when applied on multi-valued ratings. Experiments were conducted on three well-known datasets with different characteristics, using two protocols and four evaluation metrics, representing coverage, accuracy, reliability and agreement of predictions with respect to real values. Results from such experiments showed that the memory-based method is a good option because its results are more precise and reliable compared with the other methods. Online Learning methods exhibit a good level of accuracy with low variation, which makes them reliable models. On the other hand, Support Vector Machines generate predictions with acceptable agreement; however, their accuracy depends on the characteristics of the input data. Finally, Dependency Networks did not offer good results when applied on multi-valued rankings. The run experiments confirm that the characteristics of datasets keep being an important factor in the performance of methods."

Evaluation of Relevant Free Java Software

Bootstrapping capabilities using open-source, public domain software provides additional low-effort APIs and development tools. Low-overhead evaluation of available compatible software not only provides growth potential, it also helps to define the capabilities that are generally available to the competitive field. Some examples from sourceforge.net:

JUNG

the Java Universal Networks/Graph API; a Java-based open-source software library designed to support the modeling, analysis, and visualization of data that can be represented as graphs. Its focus is on mathematical and algorithmic graph applications pertaining to the fields of social network analysis, information visualization, knowledge discovery and data mining.

JGraph

the most powerful, lightweight, feature-rich, and thoroughly documented open-source graph component available for Java. It is accompanied by JGraphpad, the first free diagram editor for Java that offers XML, Drag and Drop and much more.

TouchGraph

a set of interfaces for Graph Visualization using spring-layout and focus+context techniques. Current applications include a utility for organizing links, a visual Wiki Browser, and a Google Graph Browser which uses the Google API.

BNJ: Bayesian Network Tools in Java

an open-source suite of software tools for research and development using graphical models of probability.

OpenCyc

the open source version of the Cyc(r) technology, the world's largest and most complete general knowledge base and commonsense reasoning engine. OpenCyc can be used as the basis for a wide variety of intelligent applications.

Algernon-J

a rule-based reasoning engine written in Java. It allows forward and backward chaining across Protege knowledge bases. In addition to traversing the KB, rules can call Java functions and LISP functions (from an embedded LISP interpreter)

jatha

a Java library that implements a large subset of Common LISP, including most of the datatypes (e.g. packages, bignums). The API allows access to LISP from Java. Jatha is useful as a fast, embedded LISP language, or as a standalone LISP.

FreeMind

A mind mapper, and at the same time an easy-to-operate hierarchical editor with strong emphasis on folding. These two are not really two different things, just two different descriptions of a single application. Often used for knowledge and content management

Some Experiences

I ran into illuminating examples of the essential problem of meaning while trying to answer these questions:

1980s

for National Geographic:
How can we find the picture that we want from a database of 14M pictures?

for Collier's Encyclopedia:
How can we find and catagorize multimedia to illustrate every article in the encyclopedia?

for the CIA:
How can we track every publication in the world for intelligence information?

for the Air Force Office of Scientific Research (AFOSR):
How can we answer dynamic queries about strategy by looking at a database?

for Stanford:
How can we discover and correct user errors?

1990s

for the Office of Naval Research (ONR):
How can we be aware of the meaning of data from a diversity of sensors especially in the middle of chaos (war, fire, sinking ship, etc)?

for Autodesk:
How can we make complex CAD design information more intelligible?

for Boeing:
How can we know that each of 2M parts meets specification
and is going to be delivered on time?

for Interval Research:
How can the design of computation and computers be made easier by reducing the hierarchy of languages from specification to actualization?

for BTC:
How can we design efficient semiconductor circuitry with 100M design elements?

The answer to National Geographic (circa twenty years ago!) was to associate metadata with each picture. This was an *academic* reply, since the effort to do so was forbidding. Worse, however, was the forbidding technicality that each picture could be associated with thousands of different metatags. Collier's had the tag structure in the form of articles, and it was even tractable to identify the cross-article tagging with about 50 outgoing references per article. However, multimedia lacked metatags, but even with metadata, there was no assurance that these tags corresponded to articles. The CAI too had a similar problem, but since they were only interested in text, the solution was a semantic search engine that eventually formed the basis of the Yahoo engine. AFOSR's problem also required the association of meaning with stored information, and the solution was an expert system with a semantic inference engine.

The examples from the 1990s, however, changed the playing field, moving the problem from one of *organization of information* to one of *customization of knowledge*. These problems too required semantic information, but in addition, the organized information needed to be cognitively accessible. Solutions to the automated construction of cognitively accessible knowledge are not yet in reach.