Article: Word processing is so last century

Formalizing internal narratives using internal declarations and making them look pretty

Kurt Conrad
conrad@SagebrushGroup.com
http://@SagebrushGroup.com

Kurt starts stuff up. Conversations. Improvisational ensembles. Communities of practice. Nuclear reactors and SGML training programs for the US Department of Energy. XML systems and new organizations for many others.

He's found that getting everyone on the same page before designing and engineering page-production pipelines speeds progress. His latest work involves using bottom-up, multi-perspective decision making techniques to voice both community values and semantic values in order to develop agreements and bots that better-manage complexity and accelerate change.

In this paper, Kurt attempts to describe his personal exploration into using markup to make sense of things by voicing the perspectives of an often-confusing semantic technology researcher and a second persona, the flippant Test subject.

Principle researcher & Test subject The Sagebrush Group

Authoring markup | Knowledge management | Sense making | Semantic formalization | Ontologies | Bottom-up negotiations | Collaborative communications

Abstract:

This presentation describes a journey of replacing WordPerfect with prodoc.dtd, a semantic authoring doctype; and how prodoc evolved to enable computer-assisted sense making, based on markup that formalizes knowledge flow analysis and modeling semantics.

The paper also describes some of the challenges associated with multi-perspective decision making and techniques for negotiating and formalizing dynamic, natural-system ontologies. Lessons learned from WordNet and SUMO integrations are summarized.

The paper concludes by suggesting that building editors and bots that use author-authored markup to digitize and amplify logic systems could, in many organizational settings, contribute to more-intelligent value optimizations and better long-term performance.

Section 1: Executive Summary

Sense making is all about knowing what to do in a given situation, both individually and collectively. Many observe problems with our capacity for collaborative, intelligent behavior. As automated systems (bots) get more involved in decision making, the world must be described more explicitly for intelligent, automated behavior.

This is where natural-system and formalized ontologies come into play.

This paper focuses on using markup to capture the natural languages that individuals use when they make sense of the world — the languages of internal narratives — to start negotiations around formalizing natural system ontologies into automated support systems to help individuals and groups make sense of complexity from multiple perspectives.

Formalizing an individual's personal ontology involves digitizing the way that they think about and organize information. This paper reports lessons learned from experiments developing a personalized XML doctype, prodoc. After being modified, as needed, for over a decade, prodoc has evolved to ease

the authoring of information from many different perspectives/ logical systems/ ontologies.

> “
>
> -- Test subject
>
> What makes for a better bot? One that's fast, cheap, low-risk, easily trained, and thinks like me.
>
> ”

The first section, *What's computer-assisted sense making?* looks at the role of automation in helping see patterns and organizing actionable information to enable the intelligent behavior of individuals, groups, and bots. It starts by relating semantic markup, semantic technologies, and formalized ontologies to human and automated sense making processes.

The path to prodoc contrasts word processing's focus on visual formatting behaviors and storage models with generalized markup, where text files contain start and end tags that describe nested containers that make content easy to process and stylesheets provide visual formatting specifications.

It introduces semantic markup, which involves giving the containers meaningful names, and semantic authoring, which allows authors to easily create new markup and name the data structures, themselves. Technically simple, allowing authors to declare new markup within documents has significant policy and value-proposition impacts.

prodoc in practice describes computer-assisted sense making tools that a test subject developed using custom markup and element/ attribute-based control surfaces. These control surfaces, sometimes augmented with form controls, allow the author to easily change and adjust CSS visual rendering properties. Sheet music, engineering accessible color pallets, and a modeling language are described.

The politics of markup and meaning deals with politics, which is defined as the way that groups make decisions. Complex global publishing lifecycles bring countless perspectives to the negotiation table. When semantic formalization and authoring are viewed from this perspective, semantics get dynamic. People change their minds about what's meaningful and how to communicate it.

A generalized process for values-based decision making is introduced, and its use in stabilizing and formalizing ontologies during markup development is described. Possible roles for semantic authoring in bottom-up data negotiations are considered. The results from mapping personalized markup to authoritative third-parties, WordNet and SUMO, are reported.

The paper concludes by introducing H1, a semantic authoring training system that is available for testing.

Section 2: What's computer-assisted sense making?

Sense making is all about knowing what to do in a given situation, both individually and collectively. Many observe problems with our capacity for collaborative, intelligent behavior.

Computers can assist with communications. Likewise, they can assist with sense making, helping people organize information, visualize information, see patterns, and make decisions. As automated systems (bots) get more involved in decision making, the world must be described more explicitly for intelligent, automated behavior.

Section 2.1: Onto-what?

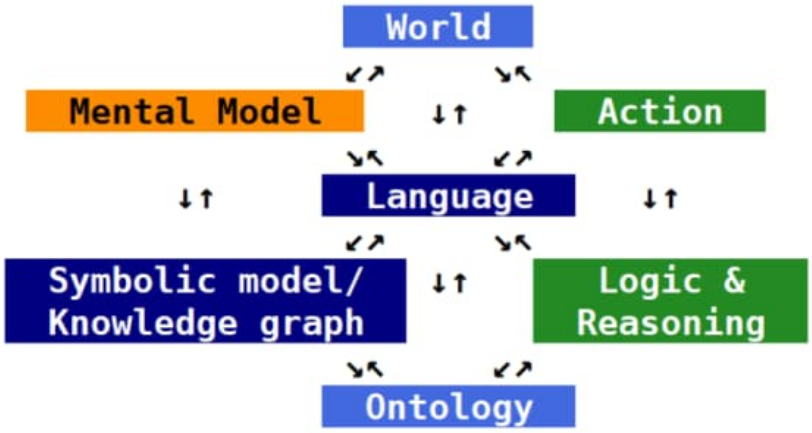
What does it take to explain everything to a computer? This is where the ways that we perceive

existence (natural-system ontologies) and formalized computing ontologies come into play. What's an ontology? Here are a couple of descriptions:

> “
>
> -- W. V. O. Quine
>
> Ontology is the subject that asks the question “What is there?”
>
> The answer can be stated in one word: “Everything.”
>
> ”

Sowa Hexagon

[The Sowa Hexagon relates language to the world and to ontologies]

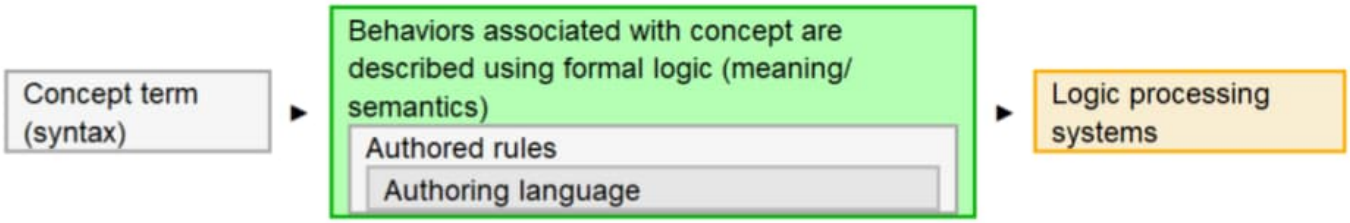


John F. Sowa's Hexagon puts language at the center and emphasizes the importance of mental models and logic in the formalization of an ontology. Formalizing real-world and abstract concepts for our buddies the bots effectively involves not only dealing with “what exists?” as a list of terms, but also by dealing with all of the models and behaviors that Sowa identifies.

Fully-formalized computing ontologies associate terms with formal logic. That's really expensive. Markup, from this perspective, could be considered a semi-formalized ontology.

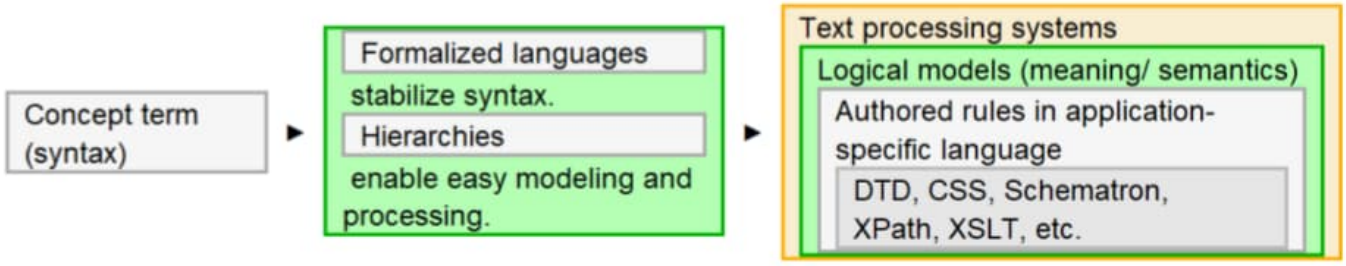
Formalized ontologies

[Formalized ontologies associate behavioral logic directly with terminology]



Semi-formalized, markup-based ontologies

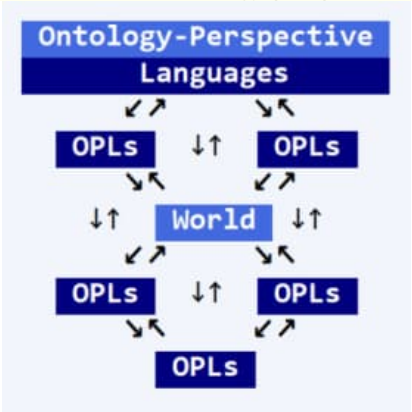
[Markup formalizes the terminology but not behavioral logic]



Section 2.2: Working with multiple languages, perspectives, and ontologies

Conrad corollary

[Languages reflect different ontology-perspectives of the world]



The Conrad corollary to Sowa's hexagon sees the world awash in languages that reflect different ontological perspectives, concepts, and logic systems. These are reflected in terminology and usage. Some are naturally-occurring. Some are engineered. Some are dysfunctional. Modern communication systems allow them to compete, globally, for attention and influence.

attempted to integrate and synthesize multiple ontologies and associated languages:

- Word processing and its visual concept of the document, which focuses on how text is arranged and looks on a page, and the ad-hoc, often-redundant behaviors associated with applying layout and style formatting rules with a mouse.
- The UNIX tradition of little languages and little bots for text processing.
- Semantic markup, SGML, XML, etc., which are based on IBM's hierarchical concept of the document.
- Knowledge management and engineering. Concepts for describing the fractal ways that data is transformed into information and synthesized into actionable knowledge to enable intelligent, intentional behavior. Data's about the details. Information's about organizing. Knowledge's about actionable behaviors.
- Multiple operational ontologies for concepts of semantics and meaning. What “meaning” means for humans, automated, and organizational systems. How semantic technologies and formalized ontologies can be used to engineer knowledge about and understanding of meaning in these different behavioral contexts..
- Policy making and performance. Techniques for helping folks articulate perceived values to rapidly reach agreements around terms, meanings, priorities, and operational details. The vocabularies of the Government Performance and Results Act, activity-based costing, and value

canvasses.

> “
>
> -- Test Subject
>
> Remember, style is something that things fall out of, and relational database tables are OK for data, but didn't and don't work for organizing documents.
>
> ”

Section 2.3: How prodoc helped make sense of big words

The test subject's experiments in computer-assisted sense making involved using markup to

- Combine concepts and techniques to model associations
- Analyze and refine those models to design and debug knowledge flows and computing systems
- Develop ad-hoc and switchable visualizations to identify, highlight, and communicate informative patterns
- Iteratively-refine markup and support systems based on new priorities and learnings
- Negotiate “optimal” balances of operational and management considerations, such as priorities and quality criteria, author effort, development effort, markup flexibility and specificity, cool examples, schedule, lifecycle costs, and performance
- Build agile, continuous delivery pipelines that enable anything, anytime, modifications to structure, layout, look, and feel; producing highly-contextualized PDFs and invalid, funny-looking source documents.

Section 2.4: Lessons learned

Co-locating knowledge and behavior tends to improve performance. Multiple perspectives tend to improve quality (and avoid crashes). The big challenge is how to integrate multiple perspectives and synthesize holistic, intelligent behavior.

Markup brings important capabilities to the table. It helps with the mechanics of getting knowledge to the agents responsible for behavior. It's both human and machine readable, and makes for an excellent negotiation framework.

Knowledge gaps to be identified and resolved through analysis of knowledge flows and language patterns. Plugging knowledge gaps better align behaviors and improve overall performance. Knowledge flows can be engineered to enable specific organizational behaviors and disable others, either through knowledge gaps or cost drivers. That's quite the ticket when associated with behavior prediction markets.

Shared meaning is negotiated. Meaning has multiple dimensions. When multiple agents interact with knowledge, they bring multiple perspectives. Just for starters, there's the plumbing side, how to move knowledge between agents.

Adding the policy and performance perspectives brings, “Why inform them?”“Who benefits?”“What are the costs?” The list of unique stakeholder perspectives can be countless, especially after folks apply new logic systems after changing their minds.

Knowledge architectures relate the bits and pieces of how knowledge enables behavior. Markup can be used in a knowledge architecture not only to enforce top down data quality standards, but enable the bottom-up negotiation of complex systems.

One way to use markup to integrate multiple-perspectives is by capturing the natural languages that individuals use when making sense of the world — the languages of internal narratives — and using those languages to start negotiations around formalizing natural system ontologies into automated support systems (to improve ergonomics and operational performance) and establishing community standards (for management system performance).

Formalizing an individual's personal ontology involves digitizing the way that they think about and organize information. This paper reports lessons learned from experiments developing a personalized XML doctype, prodoc. After being modified, as needed, for over a decade, prodoc has evolved to ease the authoring of information from many different perspectives/ logical systems/ ontologies.

The development of specialized data structures is at the essence of the idea of computer-assisted sense making. Organizing information, both structurally and visually, enables patterns to be identified and logic systems applied to enable behavior. When the situation makes sense, you know what to do.

Bots can do many things to help things make sense. Many experiments involved fine tuning the visual characteristics of authoring interfaces to make authoring as ergonomically-comfortable and time-efficient as possible.

At times, multi-perspective visualizations conflict with each other, potentially damaging signal-to-noise ratios. Switches are useful to help visualize patterns, even shifting layouts and visual style mappings to highlight different aspects, as appropriate. Explicitly-structured markup makes these types of processing and rendering tricks much easier to operationalize.

Section 3: The path to prodoc

prodoc is the synthesis of:

- Some relatively simple things (e.g., word processing)
- Some surprisingly simple things (e.g., helping people figure out what the agree on)
- Some medium complexity things (e.g., markup systems)
- And some somewhat more-complex things (e.g., understanding knowledge flows from multiple perspectives)

> “
>
> -- Test subject
>
> “No, prodoc's not available. It's an ecosystem of polished production systems and abandoned experiments. More importantly, you'd have to think like me. That process would likely hurt both of us.”
>
> ”

Section 3.1: Bots hate word processing's ornery visually-oriented ontology

Let's start with the familiar. Imagine building a spreadsheet with only three columns: bold, *italic*, and underline? How would you make sense of anything?

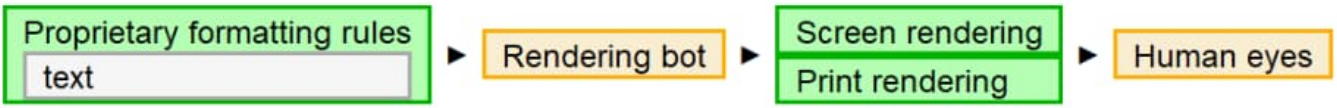
Word processing (WP) is an approach to documents that is firmly planted in the world of paper. It started by managing whitespace and adding a few visual formatting tricks to replace typewriters. Apple dramatically focused attention on defining the document in terms of its visual characteristics when it introduced the Macintosh, with synchronized bitmapped displays and printers.

We're still largely at that stage, where the bulk of the world's documentation is encoded in proprietary formats that concern themselves mostly with visual formatting of information on physical and virtual pages. Even when the files are encoded using marked-up text, the metadata and other metaknowledge artifacts store mostly visual properties.

WP's visual orientation is problematic. It really requires humans to be in the loop to look, see, recognize, interpret, synthesize with other understandings to apply missing implicit knowledge, decide, and act.

Word processing

[Word processing is optimized to drive screens and printers to feed eyes]



The visual metaphors also limit the ability of computers to both mediate human-to-human communications and also to participate in the conversation. Even the XML-based word processing encodings are rats nests of difficult-to-process markup. The noise of visual parameters swamps the signal of individual characters. Doable, but expensive.

And this is the bottom line of why WP is obsolete technology. It's just too expensive. Labor costs are too high, in large part because you can't build happy little bots to automate anything. You have to buy bots or hire someone to build big grumpy bots.

WP provides flexibility to easily tune appearance, but reuse? Global imperatives require fine-grained, context-sensitive documentation to drive intelligent behavior. That means automation. That means containers. Sure, you can move files, but you can't move fine-grained information in and out of narrative documents. WP data is, to quote someone I met at a non-destructive nuclear assay conference, "just too stupid to move."

And this isn't the way that the big dogs roll. The publishing industry started walking away from this technology, decades ago.

Section 3.2: Semantic markup to the rescue

In the 1990s professional publishers were getting cut up worse than the weak kid at a knife fight. Today, news publishers are really bleeding out.

The world of electronic documents was disrupting paper-based publishing. New terms, new technologies, and new processes forced new business logic and strategies. CD-ROMs hit first. Many organizations made significant investments to reengineer their documents to rapidly-release CD-ROMs.

Then the web hit, with its wicked learning. Publishers asked themselves, "How many times would this happen?" "How expensive would conversion costs be for each new technology cycle?" "How many different file formats will we need?" "How many different variations for different delivery devices?"

Fortunately, the web came with the seeds of the solution. The format used for web documents, the HyperText Markup Language (HTML), was based on an international standard, the Standard Generalized Markup Language (SGML). SGML defines rules for declaring, applying, and validating markup languages — a language for defining markup languages — a "meta-language" so to speak.

SGML was designed around the needs of document owners, authors, and publishers. It enables hub and spoke architectures to be implemented with authoritative, single sources of generalized markup that feed various delivery systems (CD-ROMs, web, etc.).

Section 3.2.1: Separating structure and style

Fundamental to generalized markup is the separation of content from style. It leverages lessons learned that go back to John W. Seybold's work at ROCAPPI, where lookup tables were first used to replace complex and error-prone typesetting markup codes with short — easy to type and differentiate — strings of text.

Typesetting lookup table

Easy strings of text	Confusing, error-prone typesetting codes
"** **"	;;; ldasj;fsadlkf dsalkf jdsa;l fjdsl;f dsl;f jdsafldslfkds ld :::
"** **"	;;; a s;dfkjsadfklds fldskaj fdskf ds;lkf sdlkf dslf sadlfk jdsfkj ds :::

In this world, formatting is associated with generalized data structures, making it easy to apply new styles or transforms, as the need arises. For this document, it took a couple of days to:

- Add new data structures to prodoc to make authoring easier to match DocBook requirements.
- Create a new stylesheet, `pdoc2docbook.xsl`, to transform prodoc XML to DocBook XML.

  It stripped all of prodoc's attributes, mapped element names from one markup language to the other, and changed the structure of a few branches.
- Tune the XML source and XSLT files to create valid DocBook XML.

  prodoc structures that couldn't be rendered were converted to images.

Once the pipeline was stable, attention went back to authoring and editing. A handful of whitespace issues required manual cleanup.

Section 3.2.2: Integrating lifecycle perspectives

The focus on document lifecycles drove a number of important features that are missing from contemporary word processing platforms and relational database systems.

Markup language files are text files. That simple fact improves accessibility and lowers lifecycle costs by orders of magnitude. Nearly any computing device and system can interact with a markup-encoded text file. The Extensible Markup Language (XML) dramatically lowered SGML's cost profile, and there are a wealth of XML-based markup technologies.

Documents are naturally hierarchical. Books contain chapters. Chapters contain paragraphs... Markup is organized around addressable trees and nodes and nested containers. These explicit data structures lower level-of-effort for both human and machine processing.

Semantics enter the picture when naming the nodes in the tree. When creating custom data structures to solve real-world authoring and publishing issues, the names are often associated with real-world meanings and organizing concepts. These terms, their meanings, and various encoding decisions take us into the world of semantic technologies and formalized ontologies.

Semantic markup is the phrase given to this approach to document management, where XML is used to define an application-specific markup language, where the names and data structures reflect complex conceptual frameworks and models.

The challenge when using markup for single sources is to focus all authoring on the single, highly-refined source document, so that all derivatives flow from there.

“

-- Test subject

> Hub and spoke, baby, with low cost transforms. Hub and spoke.
>
> ”

Section 3.3: From semantic markup to semantic authoring

> “
>
> -- Test subject
>
> prodoc helps me make sense of the world. Markup brings ergonomic efficiencies to the typing process. Semantic markup brings ergonomic efficiencies to the thinking process, especially when concepts and logic systems can be visualized in different ways.
>
> ”

Most semantic markup systems are organized around the idea of collaborative authoring and publishing. An organization-specific markup language is defined as a doctype with a set of rules that describe the names and relationships of the various containers. Everyone in the group works within those rules, which function as quality standards.

Doctypes are designed to standardize documents and protect publishing systems. The structural rules, which are described using an explicit schema language, typically change very slowly to ensure consistency of work processes. Standardized stylesheets are usually part of the equation.

Semantic authoring inverts this thinking.

Authors are the semantic authorities, they should have all the operational authorizations available to author semantic markup as they deem appropriate, as easily as possible. Every doctype should allow authors to use the internal declaration subset to enable bottom-up formalization and communication of models, in addition to content.

Semantic authoring is about creating documents that help authors make sense of the world. It lets individuals create semantic markup to formalize their conceptual frameworks. Once formalized, these sense making systems are easier to reuse, communicate, and automate.

Reducing the incremental costs of new markup means document-level extensions, making every document a little markup language laboratory *The idea of “little languages” comes from AT&T's development of the UNIX operating system, where little languages and little bots were developed for managing the streams of text associated with managing telecommunication systems in the 1970s*.

Semantic authoring doctypes leverage the Internal Declaration Subset, which is a mechanism for authors to create their own markup declarations. This allows new elements and attributes to be declared within any document. They can even be used within the current document, if the rules of the doctype allow.

prodoc was built on an HTML core (HTML for semantic authoring: hsa.dtd), but comparable semantic authoring extensions could be incorporated into other doctypes. Adding semantic authoring extensions to a DITA subset have been discussed, for example.

Why open Pandora's Box? What chaos would ensue?

Section 3.3.1: Individual impacts

Any number of strategic computing projects have failed because organizations couldn't get folks to use XML editors.

Why would an individual consider this craziness? Everyone's plate is full. This requires learning and ever since Microsoft took the margins out of WP through bounding, investments in this space have withered, for both individuals and organizations. WP is an ignored necessity, driven by cost-minimization strategies.

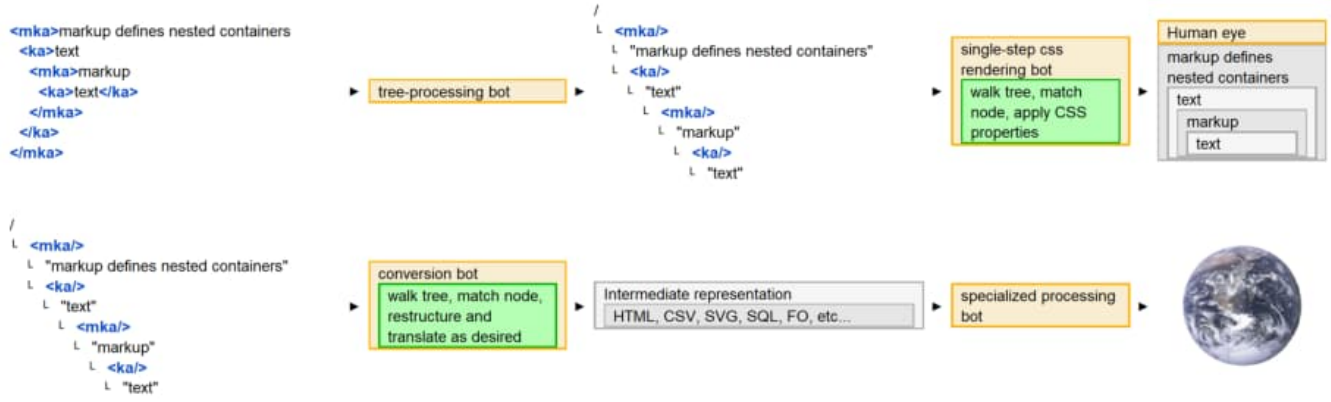
The standard XML consultant's answer to any question is, "That depends." Markup is largely policy neutral. You can apply it to almost any set of value optimizations. Markup languages are adopted, customized, and designed around a purpose. When you use someone else's markup, you are organizing your information around someone else's semantic values and value propositions. That's a cost driver.

Everyone's plate is full because the world is increasingly complex. There are a lot of bots that help, but they're all pretty much built by someone else to help them meet their needs.

What if folks had bots of their own? Little bots that could be put together in an afternoon? Organized around personally-valuable information. Individuals creating their own little languages and bots inverts markup's dominant value propositions.

Generalized markup

[Markup is processed into trees. Those trees can be rendered for display using CSS. They can be converted to other languages to interact with the world.]



When concluding XML courses I challenge students to build their own "bootcamp app":

- An XML fragment of information that has personal meaning.
- Some CSS to make it pretty and a DTD to validate it.
- Extra credit for an XSLT to HTML conversion. With that set of languages you can code the guts of almost any automated solution.
- Pro tip: Add a text processing language to get stuff into markup.

> "
>
> -- Test subject
>
> " AWK is awful good."
>
> "

Reports ranged from ham radio datasets to a DVD collection with a color scheme that would drive a person to drink. The common denominator? Feelings of accomplishment and excitement about the future. Personal markup is compelling. It's the ultimate computer game.

When markup is optimized around individual purpose, it's an enabler, with corresponding changes to brain chemistry. A recent episode of the Public Broadcasting Series, Nova, focused on the

neurochemistry of decision making.

> “
>
> -- Test subject
>
> Agency is powerful stuff. Life is unscripted. Perceptions of control are powerful. Actual, impactful, personally-meaningful operational control? Priceless.
>
> ”

The brain chemistry associated with fear is another important factor. Fearful adults staring at blank screens with their jobs on the line don't learn much. Fearful improvisers tend to get frantic. Open another document template and experiment with markup is like playdough and enables much more creativity.

> “
>
> -- Anat Shenker-Osorio
>
> ...when people are made afraid, their amygdala starts firing and their prefrontal cortex literally is starved of blood. You can’t have both things [fear and logic] going. And so if you’re asking people to sort of be in their rational brain..., then you need to present this as the possibility of creating something good...
>
> ”

> “
>
> -- Primary researcher
>
> Future research hopes to work with neuro-divergent individuals, including those with sensory-integration impairments who have had to create very sophisticated coping mechanisms and supporting ontologies. How much of this knowledge is tacit and inexpressible? How much could be formalized as language and would there be benefits? Stories heal. Would this form of storytelling amplify and add additional dimensions to those benefits?
>
> ”

Giving voice involves two dimensions, the content and the pipeline that communicates that voice. Authoring content that is designed for collaboration and integrates easily with existing computing infrastructures adds social dimensions to individual value propositions.

> “
>
> -- Test subject
>
> Our voices are so valuable that we're the only animal that chokes itself to death. Would average individuals actually use markup to model their realities? Look at what they do with spreadsheets and the various cloud integration toolkits. This is an accessibility/ price/ performance thing.

”

Section 3.3.2: Organizational impacts

“

-- Doris Lessing, author

Humanity’s legacy of stories and storytelling is the most precious we have. All wisdom is in our stories and songs. A story is how we construct our experiences. At the very simplest, it can be: ‘He/she was born, lived, died.’ Probably that is the template of our stories – a beginning, middle, and end. This structure is in our minds.

”

Markup has strategic organizational impacts. That's why global publishers use it.

Markup helps authors make sense of the past, present, and future. It makes it easier to organize information for efficient, increasingly-customized and personalized pipelines. This is human behavior we're trying to influence. We're not just dealing with data.

Authoring meaning through content, structure and style

```
Past meaning      \   Observe    / Future meaning
Established values \     O      /  Future values
Past practice       \    ||    /   Future behavior
Source origins      /    /\    \   Downstream k flows
Old logic          /   Orient   \  Re-contextualized logic
Event & prior K   / Decide & Act \ Learnings
```

Markup is good for top-down communications and alignment. If top-down alignment systems were sufficient, our existing single-source-based communication and publishing systems should be ensuring top-notch performance.

But operational realities in countless natural economies point to the need for fundamental changes in the rules of the symbolic/ market economy. That appears to be becoming increasingly difficult as the mechanisms that influence market behavior appear to be cutting themselves off from any communications that might challenge the current systems for monetizing human behavior.

Through countless decisions, systems evolve and are optimized around specific value propositions. This has happened to markup systems and associated technologies. The ISO:SGML platform, quite simply, establishes a benchmark standard for open systems. XML's refinements have brought countless new voices to the table, but author-authored markup has simply not gotten the same attention as single-sourcing and those value optimizations are reflected in the available technology alternatives through many small details that increase authoring effort in subtle, but impactful ways.

Bottom-up sense making and communications are becoming more important. Enabling individuals to customize markup in an organizational setting can be expected to not only increase reuse and collaboration, but also better-enable multi-perspective systems and decisions.

Formalizing individual value-optimizations as semantic values shares many traits with fundamentally-enabling technologies, like email. It opens up new channels of communication. How many organizations justified their initial investments in email with its impact reducing paper mail. That happened, but it didn't anticipate the completely-new models of communication that were enabled.

> “
>
> -- Primary researcher
>
> The big experiment: Can bottom-up knowledge flows be established quickly-enough to build the new understandings and consensus around the logic needed to rapidly realign global supply chains around strategic physical and operational constraints.
>
> ”

Section 4: Document-level controls to capture and communicate meaning

> “
>
> -- Test subject
>
> Semantic authoring is like car restoration on *Full Custom Garage.* The markup is the frame, and the CSS is the sheet metal. Pound at will. Make the data dance and drive change.
>
> ”

prodoc provides two primary ways to influence the meaning of information: data structures and visualizations.

The basic process for making sense of new data?

1. Do existing structures work? If so, adapt, otherwise...
2. Add markup to give the data structure
3. Pick meaningful names
4. Add visual differentiation
5. Adjust the look and geometry to look for patterns
6. Rinse and repeat until the data becomes actionable
7. Add a switch or something to change the appearance in real time, if it helps

Sometimes the process starts by making the data look pretty before creating custom data structures. The ability to incrementally add structures and style rules to the base platform reduces the incremental cost of small projects. Successful experiments get moved from the laboratory document to prodoc.dtd.

In practice, document-level experimentation has been key to the evolution of prodoc. The full case study document is the actual file that started the development of the **`<music/>`** element.

Some of the design alternatives were driven by knowledge loss. The old markup didn't make sense. Create new markup that makes sense for current thinking. A few data structures only stabilized after multiple documents with partial solutions were created, sometimes across years. A couple of systems only went into production after the best features of 3 or 4 solutions from different perspectives were integrated.

Section 4.1: Author-driven structural changes

> “
>
> -- Kevin Kelly
>
> Fast, cheap, and out of control
>
> ”

Most publishing systems use slow moving schema and stylesheets to ensure consistency of authoring. Semantic authoring is intended to be fast moving. Allowing ideas to be quickly captured and different approaches tested. Creating ecosystems of markup alternatives, driven by personal and small-group imperatives.

> “
>
> -- Test subject
>
> Running off to a DTD file? Too far away, too slow, too impactful. I'm the authority here. Let me extend structures in the document.
>
> ”
>
> “
>
> -- Eric Cartman
>
> Respect my authoritah
>
> ”

This example of enabling SGML Internal Declaration Subset extensions comes from h1.dtd, a training doctype. An updated oXml framework will be released on June 10th, 2023.

Elements are organized into four non-overlapping groups: **%divs;**, **%blocks;**, **%heads;**, and **%spans;**, so that they can be easily extended and combined, at will. This is the example of the definitions for **%divs;**

***%divs;** declarations includes the **%sa.divs;** extension mechanism for use in documents*

```
<!-- ____________________________________________________________________
. . {
. . {         %divs; - infinitely-nesting wrapper elements
. -->
<!-- ....................................................................
. . :                 %h1.divs;         . baseline hierarchical divisions
. -->
 <!ENTITY % h1.divs           " div|
                               h1|
                               hsa|
                               hsg|
                               prodoc" >
<!-- ....................................................................
. . :                 %sa.divs;         . semantic authoring interface
. -->
```

```
 <!ENTITY % sa.divs           "" >
<!-- ..........................................................................
. . :                %divs;           . (roll-up)
. -->
 <!ENTITY % divs             " %h1.divs;
                               %sa.divs;" >
```

Document header, where new elements are added through the ***%sa.divs;*** *interface*

```
<!DOCTYPE div PUBLIC "-//SG//DTD h1//EN" "../h1.dtd"[
<!ENTITY    h1               "h1" >
<!ENTITY  % sa.divs          "" ><!-- e.g., "| ename" -->
<!ENTITY  % sa.blocks        "" >
<!ENTITY  % sa.heads         "" >
<!ENTITY  % sa.spans         "" >
<!ENTITY  % sa.atl           "" >
<!ELEMENT   ename            (#PCDATA) >
<!ATTLIST   ename
              aname          CDATA                                  #IMPLIED >
]>
```

A common reason for adding spans to a document is to customize tables. **<tr/>** contains **%spans;**, and the set of **%spans;** includes **<td/>** and **<th/>**. Adding any element to the set of **%spans;** adds them to table rows.

Section 4.2: Author-driven visual changes

> “
>
> -- Test subject
>
> “Quickly changing the look and feel of text is WP's core value proposition. Pretty, but dumb. And it takes a lot of mousing around to set the properties on all those unrelated objects. Hurts my arm. Let me keep my keys on the keyboard, please. Let me apply style changes to branches and not leaves.”
>
> ”

Since the semantic authoring platform was developed to displace WordPerfect, various markup-based style controls (control surfaces) were implemented.

Elements can function as visual control surfaces. **<p/>** adds whitespace. **<t/>** doesn't. **<frame/>** adds a border. **<block/>** doesn't. Mapping the oXmlRename Element function to Alt-N enables quickly changing how data looks by changing the element name. This frequently happens when tuning inline markup to adjust prominence.

Global attributes provide most of the visual control surfaces, overriding the CSS stylesheet defaults. Some of the simplest attributes are direct pass through. **@borders**, **@padding**, and **@style** accept standard CSS syntax. Many css properties were renamed to simplify authoring: **@bgcolor**, **@p.left**,

`@p.right`, `@face` (typeface).

Some attributes add support for fixed values. `@scale`, for example, sets font-size using arbitrary values. `@sgscale` contains a standardized and generalized set of values based on the square root of the golden ratio. It started as a reference, for quickly bringing up a list of values, before being given an active role.

`@color` and `@bgcolor` are defined with a number of named colors that map to a color library. One of the color pallets is for modeling knowledge flows. Another attribute, `@kstyle`, applies background, foreground, and border color combinations based on the value of the `@k` attribute, which describes the element's role in knowledge flows.

A few controls are more aspirational than functional. `@lineheight` never seems to work. `@max-height` and the pagination settings are waiting for more powerful engines.

A variety of `@sh*` attributes show and hide control panels, table grids, id values, purple numbers, included content, and `<xi:include/>` controls. A couple more activate CPU-crushing focus and hover behaviors.

Section 5: prodoc in practice

One day, the test subject was amused by the total absurdity of keeping project notes and analysis in WordPerfect (WP) while simultaneously writing XSLT specifications using an XML editor. The test subject rebuilt that specification doctype for a project and started modifying it to replace WP for all professional documentation. The customized authoring and publishing platform was named "prodoc."

Technically, prodoc is an **emphasis *emphasis***, centered around `prodoc.dtd`, and a supporting toolkit, with many of the tools based on Oxygen XML Editor and Author software (oXml).

And so started a grand experiment. Could someone who had provided training, support, and engineering to thousands of WP users transition?

The only "active" WP document is business cards that are encoded in tables with offset crop marks.

> "
>
> -- Test subject
>
> Go back to WordPerfect? Are you crazy? Here, I can grab an element by pressing Alt-E. Even copying and pasting is easier. When I go to another editor, none of my compositional idioms are there. Yuck!
>
> "

> "
>
> -- Test subject
>
> I don't know how I could move these documents back to a word processor. Even accepting the loss of structure, the algorithmic visual effects would take too many mouse clicks to be worth the effort, and probably require a separate graphics package.
>
> "

Section 5.1: `@class` — Authored class styles

> “
>
> -- Test Subject
>
> “Look Ma. No stylesheet changes”
>
> ”

An early design principle was, “Hands off the **@class** attribute. That's for user control.” Recently, an approach was implemented that allows authors to define classes and associate specific CSS styles, all within a document.

The **@style** attribute maps to the oXml -oxy-style CSS property. This allows raw CSS to be passed to the CSS processor for the defined element.

Adding a **@styleclass** attribute on an element makes the **@style** value available for reuse. A little XPath handles the indirection from **@class**, through the preceding **@styleclass**, to **@style:**

▽***@class** markup*

[The first <t/> element defines the style for the class. Others reference the defined styles through the @class attribute]

```
<tl docbook="image" id="classMarkup">
  <t>*** <org styleclass="org" style="font-weight:800;color:green"
    >org/@styleclass="org" and @style set style</org> ***</t>
  <t>*** <org class="org">org/@class"org" replicates style</org> ***</t>
  <t>*** <org class="org">again</org> ***</t>
</tl>
```

▽***@class** CSS*

[CSS associates the @style, @styleclass, and @class attributes]

```
                    *[class]
{
  -oxy-style          : oxy_xpath(" let $x:=@class \
                                  return (preceding::*[@styleclass=$x])[1]/@style",
                                evaluate, dynamic-once);
}
```

▽***@class** rendering*

[This figure demonstrates the CSS rendering through reference]

*** org/@styleclass="org" and @style set style ***
*** org/@class"org" replicates style ***
*** again ***

▽Section 5.2: **<awkbuddy/>** — An interactive development environment block

One day, dreaming about the keyboard macro processors, I thought, “AWK! of course, AWK!” The result is **<awkbuddy/>**, which inverts most code/ documentation conventions by putting code fragments into a standard prodoc.

▽***<awkbuddy/>** codeblocks*

[Shows the awk, source, target, and cmd codeblocks within an awkbuddy element]

```
prodoc
└ awkbuddy/@id=""
  └ codeblock/@id="awk"
  └ codeblock/@id="source"
  └ xi:include/codeblock/@id="target"
  └ codeblock/@id="cmd"
```

The code blocks can be arranged, as desired, to co-locate knowledge with decisions. I've tuned mine on eye movements, by wrapping the codeblocks in table structures. The `awkbuddy.cmd` script rips the prodoc into separate files and runs the pipeline. Factoring out the file-processing overhead made AWK a much more convenient tool. More little bots have been built. Much time saved. Much more complexity managed.

Perhaps more importantly, it improves knowledge retention and library accessibility. Simply having the code and a source fragment is usually sufficient to remember the project. On the other hand, if you want to write a novel to explain a few lines of AWK, the tools are at hand.

Custom IDs can be used to run multiple awkbuddies from the same prodoc and prevent collisions in the generated files. File-level collisions across separate awkbuddies (`*.abud`) never proved to be much of an issue. If the default target fragment has been overwritten, just rerun the transform.

Section 5.3: **<bbody/>**, **<branches/>**, **<branch/>** — Hierarchical tables

The first hierarchical table was developed to do stakeholder analysis. The generalized implementation uses a specialized table body element: **<bbody/>**. Changing **@display** switches between the full table and a simplified blocks view, which makes it easier to restructure the branches.

table/@display="table" *(columns displayed for data processing)*

[Column one shows indentation to match the nested branches. Column two, the XPath. Column three, nothing important.]

Tree	XPath	Etc.
└ stakeholders	table/bbody/branch/tr/td	Another
└ internal	table/bbody/branch/branches/branch[1]/tr/td	column
└ group	table/bbody/branch/branches/branch[1]/branches/branch[1]/tr/td	w/
└ individuals	table/bbody/branch/branches/branch[1]/branches/branch[1]/branches/branch[1]/tr/td	dumb
└ external	table/bbody/branch/branches/branch[2]/tr/td	data...

table/@display="block" *(columns hidden for tree processing)*

[The same hierarchical table in block/ list mode]

<bbody/> *markup*

```
<table shindent="show" hs="corner" display="block" shfocus="show">
```

```
    <bbody>
      <branch>
        <tr>
          <th id="th">stakeholders</th>
          <td><xpath>table/bbody/branch/tr/td</xpath></td>
        </tr>
        <branches>
          <branch>
            <tr>
              <th>internal</th>
              <td><xpath>table/bbody/branch/branches/branch[1]/tr/td</xpath></td>
            </tr>
            <branches>
              <branch>
                <tr>
                  <th>group</th>
                  <td><xpath>table/bbody/branch/branches/branch[1]/</xpath><xpath
                </tr>
                <branches>
                  <branch>
                    <tr>
                      <th>individuals</th>
                      <td><xpath>table/bbody/branch/branches/branch[1]/<xpath>bra
                    </tr>
                  </branch>
                </branches>
              </branch>
            </branches>
          </branch>
          <branch>
            <tr>
              <th>external</th>
              <td><xpath>table/bbody/branch/branches/branch[2]/tr/td</xpath></td>
            </tr>
          </branch>
        </branches>
      </branch>
    </bbody>
  </table>
```

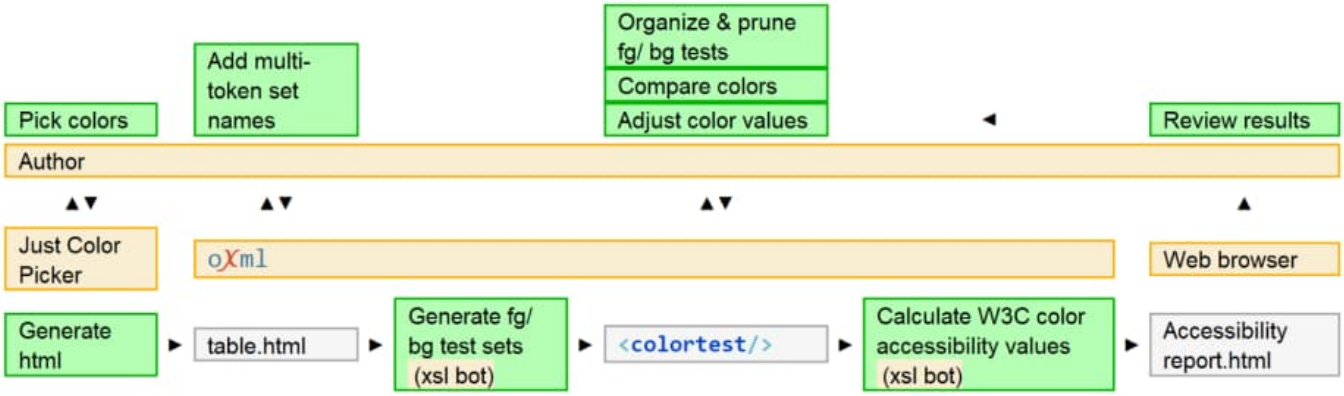
Section 5.4: **<colortest/>** — Automating accessible color negotiation pipelines

Challenge: Use the W3C-published formula for calculating the accessibility of color combinations. Over many years, the system was re-factored. Copying and pasting values into a small XML model was fine for small projects, but I had to seek client approvals on a pallet of 24 colors that were going to be used

for strategic branding.

Color testing and tuning pipeline

[The pipeline enabled efficient tuning of color pallets]



The Just Color Picker tool generated HTML. A new transformation was created to map the HTML output to a new prodoc element, **<colortest/>**. The production accessibility calculator was updated to handle the new schema and missing functionality was added.

Implementation had a mix of performance objectives: large swatches to make small comparisons, built in black, gray, and white samples, ability to instantly see color changes based on editable hex values, demonstrate single-sourcing concepts to client, save me an armload of effort

Section 5.5: **<h/>** — Depth-based headings because big headings are ugly

HTML uses numbered headings, prodoc, on the other hand, uses nested divs with a single, context-sensitive **<h/>** element

The following XPath scales headings based on the maximum depth of the document. 1.272 is the square root of the golden ratio. A recent document needed a gentler scaling factor. The cube root of the golden ratio (1.174) was used, instead.

CSS to scale headings based on document depth

```
h {
font-size : oxy_xpath("concat(
            math:pow(1.272,((max(//h/count(ancestor::*[@display='div'])) - cou
            'em')", evaluate,dynamic-once) ;
}
```

Section 5.6: **<kfam/>** — An element and generalized design language to make sense of knowledge flows from multiple perspectives

> “
>
> -- Test subject

The kfam ontology includes a lot of concepts, but you only really need to focus on three to make sense of knowledge flows: [agent in an orange

box] agent, [artifact in a blue box] artifact, and [behavior in a green box] behavior.

”

The knowledge flow analysis and modeling language — kfam — represents the most extreme example of the fractal nature of this approach to computer-assisted sense making.

The base language, markup, and visualizations associated with kfam have evolved over 30 years of discussions around organizational performance. New concepts and ideas around those concepts interplay with new ways to organize and control the appearance of kfam data. The result is a design language, where kfam concepts are used to name elements, attributes, colors, etc.

One view of sense making is answering the question, “How to make sense of this data? How to organize it without incessantly banging on the keyboard?”. The meaning of kfam from this perspective centers on ergonomics, the behavior of eyes and hands. Operationally, that's tuning the markup, interfaces, and bots to organize data into actionable knowledge.

When multiple datasets need the same treatment, shifting from manual to automated processes makes sense. This type of computer-assisted sense making —making sense of new data sources — would often be considered secondary deliverables, infrastructure that enables behavior, lowers costs, and improves quality.

When kfam is applied to primary activities, the questions shift to, “How do I make sense of this situation. What are we trying to accomplish? Who knows what? What doesn't know enough?”

This type of sense making focuses on making sense of a behavioral domain, the performance objectives, and the specific knowledge requirements that enable intelligent behavior.

`<kfam/>` element is a customized table that evolved to help the Test subject make sense of:

- The ways that knowledge flows through organizations and is acted upon, operationally, planned, and implemented
- The various bits and pieces of XML system architectures At its core, kfam is a language for dealing with the fractal nature of language and behavior.
- Competing, multidimensional value optimizations. The variety of performance objectives and the specific knowledge requirements that enable intelligent behavior within a span of control.

When engineering markup systems, analysis of knowledge flows has impacts on architectural decisions, fine-grained markup decisions, usability, and occasionally how to deal with organizational dynamics.

By being closely associated with primary work products and being rather abstract, the meaning of kfam concepts has been highly-dynamic, making formalization an iterative process. Of all of the systems built on the prodoc platform, kfam has had the most iterations and has the most moving parts. It is the most expansive example of semantic authoring being used to create markup based on a person's conceptual models.

“

-- Test subject

I started working with markup at the same time that Bo introduced me to knowledge management and Joe, values-based decision making. I couldn't untangle those conceptual frameworks with a Lampson crane.

”

Section 5.6.1: kfam conceptual language

kfam's organized around three core concepts: agents, artifacts, and behaviors. Those three concepts are sufficient to describe a knowledge flow.

Agentsact, make decisions, and exhibit behavior. The set of agents comprise individuals, organizations, systems, and automated agents (bots).

Artifacts are physical or conceptual objects.

Knowledge can also be characterized through such distinctions as

- Knowledge artifacts (ka) and meta-knowledge artifacts (mka)
- Data, information, knowledge
- Explicit, implicit, and tacit forms

Behavior comprises actions and decisions. Knowledge enables behavior. Co-location of knowledge and decisions generally improves performance. Behavior goes by many different names in different contexts.

Policy is what you do. Policy is behavior. Written policies are only guidance. The pages don't act. Ideally, they are semantic triggers, triggering the intended behaviors by the acting agents.

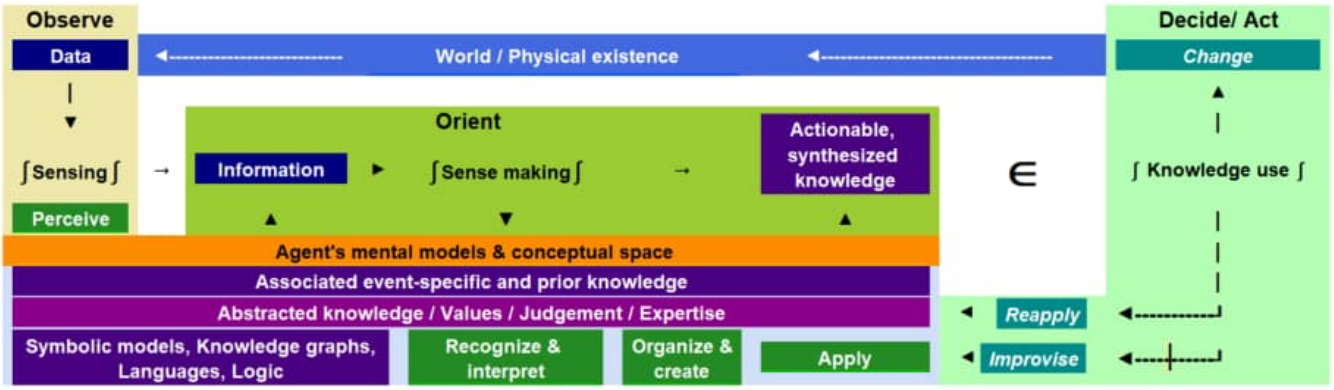
Meaning is behavior. Words have no meaning without action. Knowledge flows and lifecycles comprise such behaviors as knowledge creation, retention, transfer, use, and destruction.

Values are disassociated and abstracted knowledge systems that synthesize pattern recognition and evaluation to quickly apply established logic to operational details with minimal thought. They represent distilled and automated behaviors. Unscripted life requires active management systems; automate the operational details. Novelty focuses active attention, thinking, and — if there isn't too much fear — creativity.

While agent, artifact, and behavior are enough to describe a knowledge flow, additional details help with detailed analysis and engineering. The following diagram relates the kfam “data ∫ information ∫ knowledge ∈ behavior” model with Sowa's concepts and OODA loops, the process that fighter pilots use to Observe, Orient, Decide, and Act.

Knowledge flows associated with knowledge enabling behavior

[Observe relates to kfam sensing behaviors. Orient to sense making. Decide and act to knowledge use.]



Section 5.6.2: kfam markup language

In markup systems, elements are first-class objects. Early experiments with extending and

customizing semantics using attributes proved unworkable due to accessibility issues.

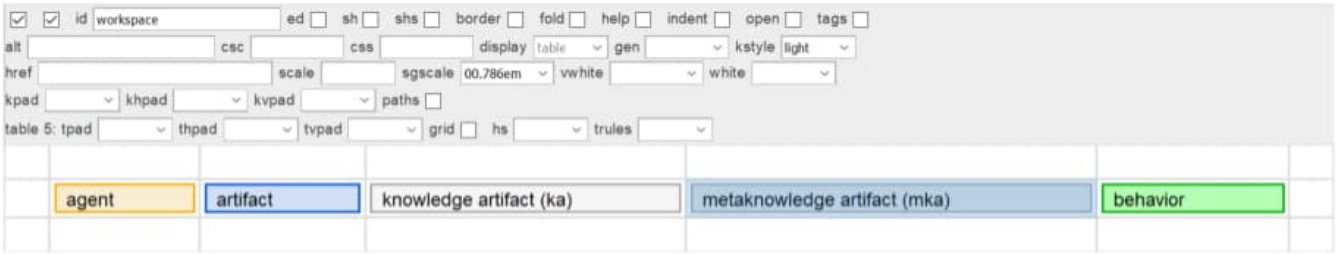
A couple of other design principles: The egg carton principle. How many eggs would you buy if you had to create your own egg cartons, on the spot? Be generous when you build containers. Redundancy, many ways to say the same thing, doesn't hurt.

Formalizing kfam starts by creating **`%spans;`** based on the concepts, e.g., **`<agent/>`**, **`<artifact/>`**, **`<behavior/>`**. Likewise for attributes. **`%semantic-atl;`** includes **`@agent`**, **`@artifact`**, and **`@behavior`**.

Most modeling is done in a specialized table called **`<kfam/>`**, which uses a restricted element set and has a fairly extensive set of controls:

`<kfam/>` workspace

[Four rows of CSS controls adjust table properties. They sit above a kfam table with light formatting]

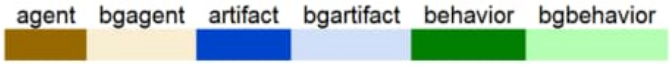


Section 5.6.3: kfam visual language

The kfam visual language starts with colors. Named colors are defined with HSL values and variables in LESS stylesheets. The names are defined for use as attribute values in prodoc.dtd so that they can be applied to **`@color`** and **`@bgcolor`** using picklists. One of the color pallets is for modeling knowledge flows.

Named foreground and background colors

[foreground and background colors for agent, artifact, and behavior]



Another attribute, **`@kstyle`**, applies **`<kfam/>`**visualizations to any element. **`@kstyle`** accepts the values **`(bgcolor| color| custom| dark| full| gray| kflow| light| line| none)`**. It applies background, foreground, and border color combinations based on the value of the **`@k`** attribute, which describes the element's role in knowledge flows. prodoc.dtd sets default **`@k`** attribute values for most spans, which can be overwritten to lie to the author, e.g., **`agent/@k="behavior"`**, would create a secret **`<agent/>`** camouflaged to look like a **`<behavior/>`**.

`@k` → `@kstyle` CSS style mappings

[Setting the @k attribute to "agent" maps any element to the class of agents]

<agent k="agent"> ►

```
                                *[k='agent']
{ -oxy-tags-background-color    : @kfam-agent-bg;
  -oxy-tags-color               : black;
}

*[kstyle='light'][display]      *[k='agent']
{ background-color              : @kfam-agent-bg;
  border-color                  : @kfam-agent-border;
  color                         : black;
}

*[kstyle='light'][display]      *[k]:not(*[k='kx'])
{ border-style                  : solid;
  border-width                  : 2px;
}
```

@kstyle variations applied to a table

[The @kstyle attribute can be changed to get a range of visual effects]

@kstyle="bgcolor"	agent	artifact	ka	mka	behavior
@kstyle="color"	agent	artifact	ka	mka	behavior
@kstyle="dark"	agent	artifact	ka	mka	behavior
@kstyle="full"	agent	artifact	ka	mka	behavior
@kstyle="gray"	agent	artifact	ka	mka	behavior
@kstyle="kflow"	agent	artifact	ka	mka	behavior
@kstyle="light"	agent	artifact	ka	mka	behavior
@kstyle="line"	agent	artifact	ka	mka	behavior

Section 5.6.4: kfam modeling

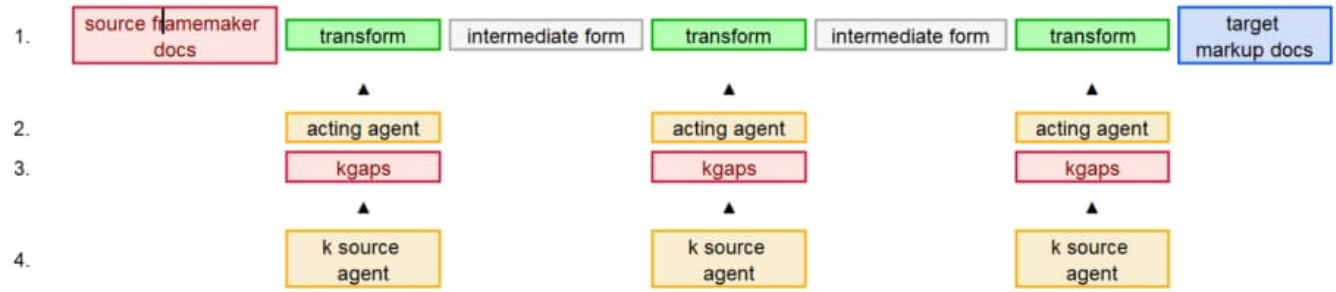
kfam started as a natural language for discussing project activities and how knowledge enables behavior. Some of that ambiguity has remained as the language has evolved into an analytical system, modeling language, and engineering framework. The most complete, published definition of kfam can be found in *Knowledge Flow Modeling and Analysis with Focus on Enabling Actions and Decisions within the Business Process*.

kfam was first used as a modeling language when markup program managers were being blamed for expected IT schedule slippages. Finding the solution was pretty easy.

1. Map out the processing steps, defining each of the transforms and intermediate data forms
2. Identify the responsible agents. Who was doing the work? A person? A bot? Some combination?
3. Any knowledge gaps or other barriers to progress?
4. Where does any missing information need to come from?

Knowledge gap analysis

[Source documents are converted to targets through a series of transformations and intermediate forms. The acting agents have knowledge gaps, and need the assistance of knowledgeable sources]



It turned out that all of the knowledge gaps were pending IT decisions. The problem went away. We also made a couple of refinements to the startup process.

Here's a more-complete set of kfam elements: Any element can join a class by setting the **@k** attribute to the listed value.

@k values and associated colors

[Agent colors are in the red-orange range. Artifacts, blue-purple. Behaviors use greens and yellows]

agent	Acts or has the power to act
bot	Automated agent
gov	Governing agent
ind	Individual agent
issue	*"Every problem's a management failure"* — DOE Management Oversight Risk Tree
mgmt	Managing agent
org	Organizational agent
sys	System agent
artifact	Physical or conceptual object
pa	Physical artifact
ka	Knowledge artifact
aka	Abstract/ disassociated knowledge artifact
data	Item of factual information
info	Organized, meaningful data
k	Synthesis of enabling knowledge artifacts
mka	Metaknowledge about a ka
rel	Relationship/ associative/ associated Metaknowledge
behavior	Actions and decisions
mksense	Sense making / meaning association / informing behaviors
ku	Knowledge utilization event
sense	Sensing, perceiving, data acquisition behaviors
v	Value, expectations, default logic, meaning, worth, enabled future behaviors
kx	Exclude from `<kstyle/>` processing. Override any default value

Section 5.7: **<music/>** — Rationalizing chord/ lyric pairings

When chord symbols lose alignment with the lyrics, they lose their meaning. With WP, the pairings regularly fall out of alignment for countless reasons:

Lyric-chord charts with misaligned text (it gets much worse, especially with proportional fonts)

[Shows the F#m chord above the word "Me" preventing the next chord "G" from aligning with the word "eat"]

```
   D              F#m G       A
In remembrance of Me eat this bread
   G              Em            A
In remembrance of Me drink this wine
   Bm             G  C           A
In remembrance of Me pray for the time
     D         F#m     G   A
When God's own will is done
```

Markup was used to make the chord **<c/>** and lyric **<l/>** relationships more explicit.

***<lyric/>** lines, with lyric fragments **<l/>**, and chord **<c/>**symbols next to each other so they don't get lost*

```
<music>
  <lyric>
    <l>In </l>
    <c>D</c>
    <l>remembrance of </l>
    <c>F#m</c>
    <l>Me </l>
    <c>G</c>
    <l>eat this </l>
    <c>A</c>
    <l>bread</l>
  </lyric>
</music>
```

XPath was used to manage horizontal and vertical offsets.

CSS rendering with fully-automated offsets

[Shows the G chord correctly aligned with the word "eat"]

```
D                F#m  G        A
In remembrance of Me   eat this bread
```

All the CSS gymnastics cause cursor-positioning problems. Adding an edit mode solved the problem:

***music[@view="edit"]** markup*

```
<music view="edit">
  <lyric>
    <l>In </l><c>D</c><l>remembrance of </l><c>F#m</c><l>Me  </l>
    <c>G</c><l>eat this </l><c>A</c><l>bread</l>
```

```
    </lyric>
  </music>
```

▽ ***music[@view="edit"]*** *rendering*

[Shows the chords and words on one line to simplify editing]

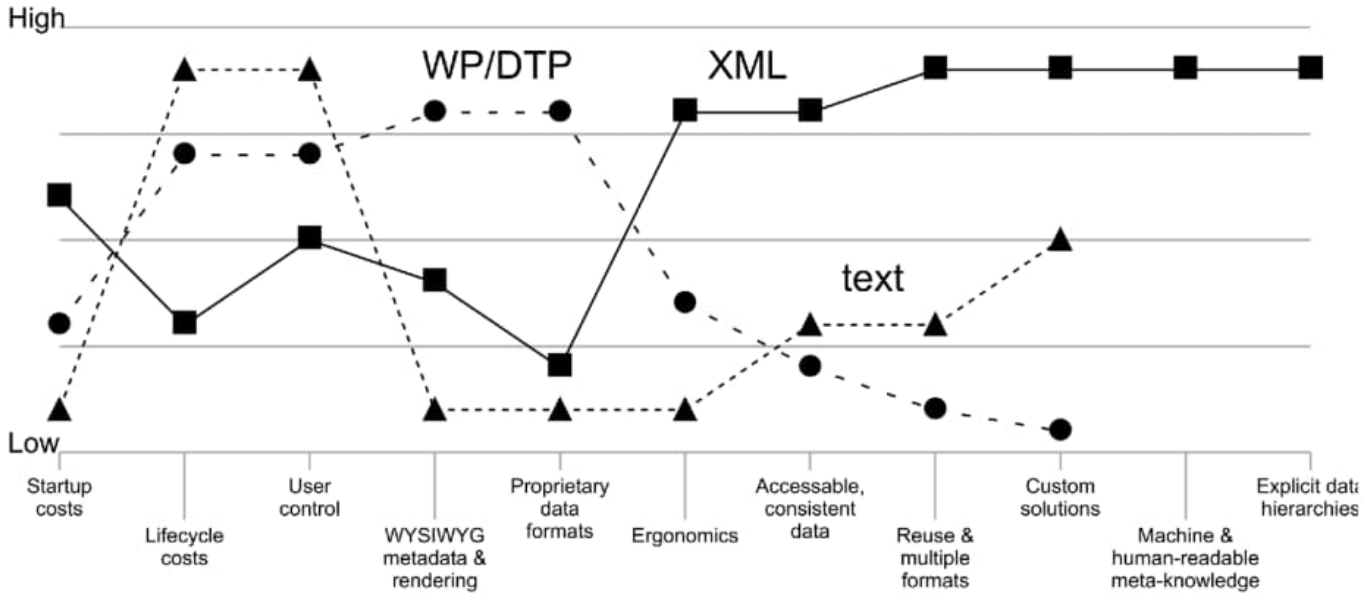
```
In | D remembrance of | F#m Me | G eat this | A bread
```

▽ Section 5.8: **<vcanvas/>** — Visualizing and comparing sets of value optimizations

Value canvases come from the book *Blue Ocean Strategy*, They provide a good way to visualize and compare multiple sets of value propositions. The following SVG rendering was generated from data entered into a **<vcanvas/>** table.

▽ *Rendering of* ***<vcanvas/>*** *datasets*

[Value canvas comparing text, word, and XML value propositions]



▽ Section 5.9: WordNet and SUMO integrations — Associating markup with dictionaries & formal logic

There are different ways to organize bottom-up markup. A technical solution is namespaces, but that just keeps the chaos from bugging the bots.

> “
>
> -- Anonymous source
>
> Namespaces! A linguistic solution without behavioral context (e.g., requirements). So many different implementation models. So much fun.
>
> ”

Another approach is negotiations. As individual markup is shared and encounters similar/ overlapping markup, conversations are used to generalize the concepts and markup. Different concepts get

different names. Individual markup feeds departmental standards. Semantic generalization flows up through layered DTDs, matching the organizational hierarchy. This represents an evolutionary approach to bottom-up data modeling and system design.

Coming at the question from the opposite direction, individual markup could be anchored to a separate, semantic authority. At the first Ontolog face-to-face meeting & workshop in 2003, Adam Pease introduced SUMO, the Suggested Upper Merged Ontology to the group. The aftermath is described in the forward to In Adam's book *Ontology*:

> “
>
> -- Duane Nickull
>
> The result of [Adam's] behavior was quite infectious. Adam, Kurt Conrad, and I ended up in a late night sushi restaurant somewhere near Menlo Park, CA, discussing how to map SUMO concepts to Mandarin, Japanese, and Cantonese, how WordNet can reference SUMO, and why First-Order Logic (FOL) constraints are generally a cool concept to have in advanced computer systems. An upper-level ontology, such as SUMO, is a common, shared conceptualization of a domain.
>
> ”

Section 5.9.1: Approach

The draft implementation uses prodoc's document-level extension mechanism to add global attributes a test document:

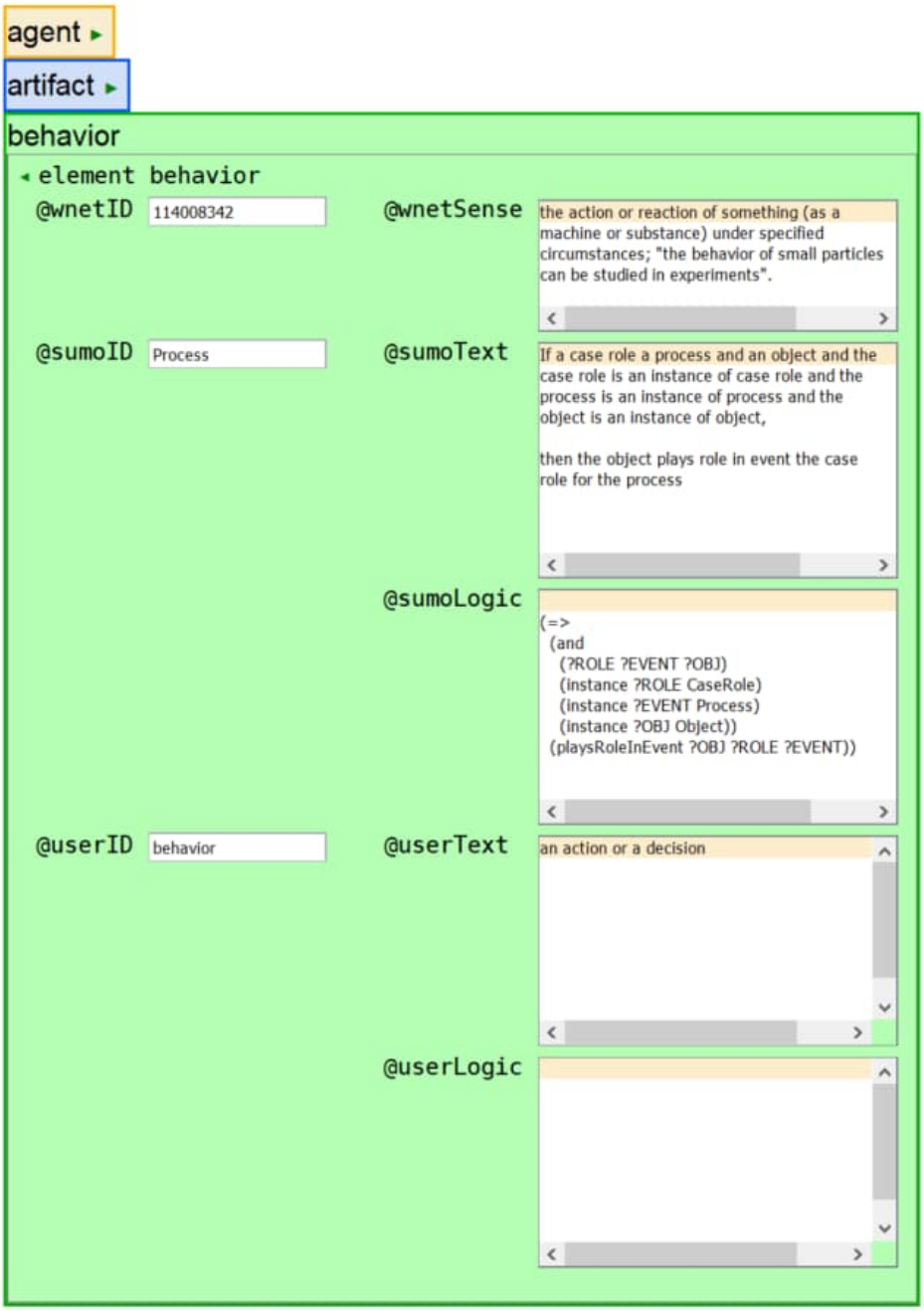
Semantic formalization attribute list

```
<!-- ren to fsem.atl when generalized -->
<!ENTITY  % sa.atl
"
              shsem           (show| hidden)   #IMPLIED
              sumoID          CDATA            #IMPLIED
              sumoLogic       CDATA            #IMPLIED
              sumoText        CDATA            #IMPLIED
              userID          CDATA            #IMPLIED
              userLogic       CDATA            #IMPLIED
              userText        CDATA            #IMPLIED
              wnetID          CDATA            #IMPLIED
              wnetSense       CDATA            #IMPLIED
" >
```

The approach was tested with the concepts **<agent/>**, **<artifact/>**, and **<behavior/>**. The **@shsem** attribute controls the expansion and collapsing of the associated form control that provides access to the WordNet and SUMO attributes set.

Elements mapped to WordNet definitions and SUMO logic

[A behavior element has been expand to show interfaces for WordNet and SUMO connections]



Section 5.9.2: Findings & next steps

> “
>
> -- Roger C. Schank, cognitive scientist
>
> Humans are not ideally set up to understand logic; they are ideally set up to understand stories.
>
> ”

An obvious next step is to connect the form control to a SUMO repository and add a lookup/ query interface. The level of effort appears reasonable.

More thought needs to go into a more challenging issue, authoring fully-formalized semantics. Spelling KIF is one thing. Spelling out formal logic in KIF is another.

Semantic markup can simplify knowledge transfer by making implicit organizing concepts more explicit. This is especially useful during learning, before a new conceptual framework's language and logic have been fully-internalized.

Expressing KIF concepts more explicitly through markup is expected to help it make sense in more behavioral contexts:

- For individuals, faster comprehension and productivity, especially when style variations can be applied to help differentiate concepts
- For the bots, more tool options. XSLT and XPath, by themselves, dramatically expand the software development options.

In his *Standard Upper Ontology Knowledge Interchange Format* document, Adam describes SUO-KIF using BNF syntax:

▽*SUO-KIF definition*

```
upper ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
N | O | P | Q | R | S | T | U | V | W | X | Y | Z
lower ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
n | o | p | q | r | s | t | u | v | w | x | y | z
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
special ::= ! | $ | % | & | * | + | - | . | / | < | = | > | ? |
@ | _ | ~ |
white ::= space | tab | return | linefeed | page
initialchar ::= upper | lower
wordchar ::= upper | lower | digit | - | _ | special
character ::= upper | lower | digit | special | white
word ::= initialchar wordchar*
string ::= "character*"
variable ::= ?word | @word
number ::= [-] digit+ [. digit+] [exponent]
exponent ::= e [-] digit+
term ::= variable | word | string | funterm | number | sentence
relword ::= initialchar wordchar*
funword ::= initialchar wordchar*
funterm ::= (funword term+) | (funword sentence+)
sentence ::= word | equation | inequality |
relsent | logsent | quantsent
equation ::= (= term term)
relsent ::= (relword term+) | (relword sentence+)
logsent ::= (not sentence) |
(and sentence+) |
(or sentence+) |
(=> sentence sentence) |
(<=> sentence sentence)
quantsent ::= (forall (variable+) sentence) |
```

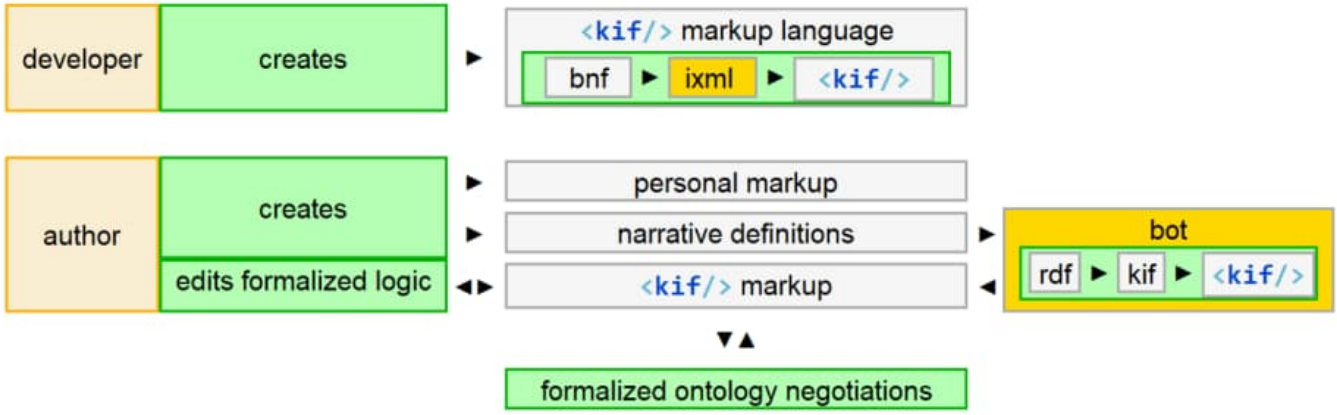
```
(exists (variable+) sentence)
```

ixml was used to transform KIF to **<kif/>**. The most significant finding was that the definition appears to include a couple of unnamed layers that deserve clarification.

Integrations with other automated systems have been explored. OntoInsights focuses on the value propositions associated with storytelling and deep narrative analysis. It generates RDF fragments that can be translated into KIF.

From semi-formalized markup languages to fully-formalized ontologies

[A developer creates the kif markup language. An author creates markup and definitions to feed a bot that generates kif markup, which feeds editorial and negotiation processes]



> “
>
> -- Test subject
>
> Every time I look in one of those fully-formalized thingies, it never agrees with the way I think.
>
> ”

Section 6: Bottom-up negotiations to define shared meanings

> “
>
> -- Paul Strassmann, The Politics of Information Management
>
> Policy is what you do, not what you say or write.
>
> ”

A few more mantras from Strassmann: “Information management is primarily a question of politics and secondarily one of technology.”“Operational decisions take care of today. Management decisions protect the future.”“Without privacy, there is no freedom.”

Negotiations around markup are rarely zero-sum. The ability to handle optional structures usually provides sufficient flexibility to accommodate different stakeholder interests. Even so, as the complexity of the project increases, integrating the multiple perspectives becomes more challenging.

Just as bottom-up markup starts with defining semantic values, a parallel, bottom-up approach to decision making starts with conversations to define common human values.

As the discussion topics progress, the group creates a shared vision for change and looks at the current conflicts from a position of safety. Issues are prioritized, and initiatives defined — quickly and without conflict.

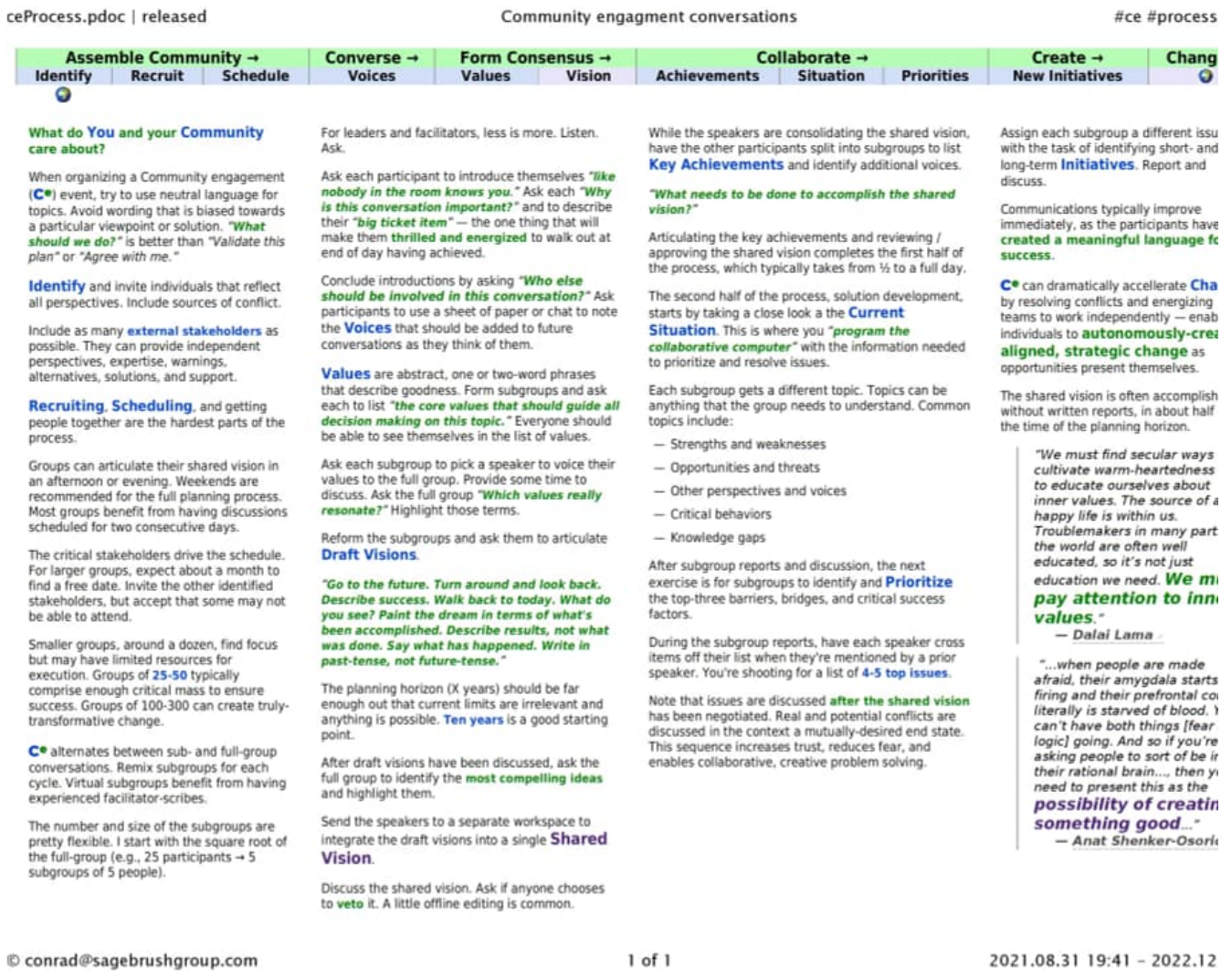
More importantly, the process is energizing. Individuals see their values in the shared vision, and the new initiatives as a way to accomplish personal objectives.

From an ontological perspective, the identification, articulation, and prioritization of abstract values, starts a formalization process. The corresponding conversations create shared meanings to support the new language.

The following training aide describes the process in more detail:

Community engagement process

[Four columns provide guidance on assembling a community, conversing and forming consensus, collaborating, and creating change]



ceProcess.pdoc | released

Community engagment conversations

#ce #process

Assemble Community → | Identify | Recruit | Schedule

Converse → | Voices

Form Consensus → | Values | Vision

Collaborate → | Achievements | Situation | Priorities

Create → | New Initiatives

Chang

What do You and your Community care about?

When organizing a Community engagement (C•) event, try to use neutral language for topics. Avoid wording that is biased towards a particular viewpoint or solution. *"What should we do?"* is better than *"Validate this plan"* or *"Agree with me."*

Identify and invite individuals that reflect all perspectives. Include sources of conflict.

Include as many **external stakeholders** as possible. They can provide independent perspectives, expertise, warnings, alternatives, solutions, and support.

Recruiting, **Scheduling**, and getting people together are the hardest parts of the process.

Groups can articulate their shared vision in an afternoon or evening. Weekends are recommended for the full planning process. Most groups benefit from having discussions scheduled for two consecutive days.

The critical stakeholders drive the schedule. For larger groups, expect about a month to find a free date. Invite the other identified stakeholders, but accept that some may not be able to attend.

Smaller groups, around a dozen, find focus but may have limited resources for execution. Groups of **25-50** typically comprise enough critical mass to ensure success. Groups of 100-300 can create truly-transformative change.

C• alternates between sub- and full-group conversations. Remix subgroups for each cycle. Virtual subgroups benefit from having experienced facilitator-scribes.

The number and size of the subgroups are pretty flexible. I start with the square root of the full-group (e.g., 25 participants → 5 subgroups of 5 people).

For leaders and facilitators, less is more. Listen. Ask.

Ask each participant to introduce themselves ***"like nobody in the room knows you."*** Ask each ***"Why is this conversation important?"*** and to describe their ***"big ticket item"*** — the one thing that will make them **thrilled and energized** to walk out at end of day having achieved.

Conclude introductions by asking ***"Who else should be involved in this conversation?"*** Ask participants to use a sheet of paper or chat to note the **Voices** that should be added to future conversations as they think of them.

Values are abstract, one or two-word phrases that describe goodness. Form subgroups and ask each to list ***"the core values that should guide all decision making on this topic."*** Everyone should be able to see themselves in the list of values.

Ask each subgroup to pick a speaker to voice their values to the full group. Provide some time to discuss. Ask the full group ***"Which values really resonate?"*** Highlight those terms.

Reform the subgroups and ask them to articulate **Draft Visions**.

"Go to the future. Turn around and look back. Describe success. Walk back to today. What do you see? Paint the dream in terms of what's been accomplished. Describe results, not what was done. Say what has happened. Write in past-tense, not future-tense."

The planning horizon (X years) should be far enough out that current limits are irrelevant and anything is possible. **Ten years** is a good starting point.

After draft visions have been discussed, ask the full group to identify the **most compelling ideas** and highlight them.

Send the speakers to a separate workspace to integrate the draft visions into a single **Shared Vision**.

Discuss the shared vision. Ask if anyone chooses to **veto** it. A little offline editing is common.

While the speakers are consolidating the shared vision, have the other participants split into subgroups to list **Key Achievements** and identify additional voices.

"What needs to be done to accomplish the shared vision?"

Articulating the key achievements and reviewing / approving the shared vision completes the first half of the process, which typically takes from ½ to a full day.

The second half of the process, solution development, starts by taking a close look a the **Current Situation**. This is where you ***"program the collaborative computer"*** with the information needed to prioritize and resolve issues.

Each subgroup gets a different topic. Topics can be anything that the group needs to understand. Common topics include:

- Strengths and weaknesses
- Opportunities and threats
- Other perspectives and voices
- Critical behaviors
- Knowledge gaps

After subgroup reports and discussion, the next exercise is for subgroups to identify and **Prioritize** the top-three barriers, bridges, and critical success factors.

During the subgroup reports, have each speaker cross items off their list when they're mentioned by a prior speaker. You're shooting for a list of **4-5 top issues**.

Note that issues are discussed **after the shared vision** has been negotiated. Real and potential conflicts are discussed in the context a mutually-desired end state. This sequence increases trust, reduces fear, and enables collaborative, creative problem solving.

Assign each subgroup a different issu with the task of identifying short- and long-term **Initiatives**. Report and discuss.

Communications typically improve immediately, as the participants have **created a meaningful language fo success**.

C• can dramatically accellerate **Cha** by resolving conflicts and energizing teams to work independently — enab individuals to **autonomously-crea aligned, strategic change** as opportunities present themselves.

The shared vision is often accomplish without written reports, in about half the time of the planning horizon.

> *"We must find secular ways cultivate warm-heartedness to educate ourselves about inner values. The source of a happy life is within us. Troublemakers in many part the world are often well educated, so it's not just education we need.* ***We m pay attention to inn values****."*
> *— Dalai Lama*

> *"...when people are made afraid, their amygdala starts firing and their prefrontal co literally is starved of blood. Y can't have both things [fear logic] going. And so if you're asking people to sort of be i their rational brain..., then y need to present this as the* ***possibility of creatin something good****..."*
> *— Anat Shenker-Osori*

© conrad@sagebrushgroup.com

1 of 1

2021.08.31 19:41 - 2022.12

Section 7: h1.dtd — Build your own prodoc

h1.dtd is based on an HTML subset. It's scheduled for release on June 10th.

h1.dtd summary

```
<!-- ============================================================================
                                                                    dawkIns ....
            dawk Summary

    SPath:    D:\kc\xt\h1\
    SFile:    h1.dtd
    Gen:      2023-05-30 22:12
    GenBy:    D:\kc\xt\bin\dawk.awk
    Target:   D:\kc\xt\bin\scratch\dawk_tgt.txt
    fileType: xml
    chopCol:  79

    Module:   h1.dtd
              HTML 1 Document Type Declarations

    Public Identifier:

              -//SG//DTD h1//EN

    Date:     2021.05.29

    Version:  Released for testing

    Contents:

   175    Declaration constants (parameter entity declarations)
   178
   179      Extensible element constants
   182
   183        %divs; - infinitely-nesting wrapper elements
   186                 %h1.divs;           baseline hierarchical divisions
   194                 %sa.divs;           semantic authoring interface
   198                 %divs;              (roll-up)
   203
   204        %heads; - top of div (metadata) elements
   207                 %h1.heads;          baseline heading containers
   211                 %sa.heads;          semantic authoring interface
   215                 %heads;             (roll-up)
   220
   221        %blocks; - paragraph-level elements
   224                 %h1.blocks;         baseline blocks (email benchmark)
   239                 %sa.blocks;         semantic authoring interface
```

```
243                 %blocks;            (roll-up)
248
249         %spans; - inline elements
252                 %h1.spans;          baseline spans
280                 %sa.spans;          semantic authoring interface
284                 %spans;             (roll-up)
289
290       Attribute constants
293
294         Semantic attribute entities
297                 @alt                alternate (text for image)
301                 @agent              acts or has the power to act
305                 @artifact           physical or conceptual object
309                 @author             agent responsible for writing text
313                 @behavior           an agent's actions and decisions
317                 @class              css classifications
321                 @content            add @name for custom "attribute"
325                 @created            when created
329                 @href               hypetext reference
333                 @id                 unique identifier
337                 @idref              reference to a single identifier
341                 @modified           when modified
345                 @name               add @content for matching value
349                 @src                media source
353                 @status             relative ranking
357                 @tags               #words that classify or categorize
361                 @type               nature or genre
365                 %sem.atl;           (roll-up)
385
386         Enumerated attribute value sets for style attributes
389                 %avs.align;         horizontal text alignment values
394                 %avs.display        display property values
402                 %avs.open.closed;   details open/closed switch values
406                 %avs.valign;        vertical alignment values
412                 %avs.vwhite         vertical whitespace
417                 %avs.whitespace     text whitespace
422
423         Style attribute entities
426                 @align              horizontal text alignment
430                 @bgcolor            background color
434                 @border             (width style color)
438                 @color              text color
442                 @colspan            column span
446                 @display            default display geometry is inline
450                 %a.display.def;     defined default (e.g., "block")
```

```
454                 @height             vertical size
458                 @margin             whitespace ouside border
462                 @open               open/close display & displayblock
466                 @padding            whitespace inside border
470                 @rowspan            row span
474                 @scale              scales font size
478                 @style              inline css
482                 @valign             vertical alignment
486                 @white              text whitespace
490                 @width              horizontal size
494                 @xspace.pre         xml:space="preserve"
499                 %style.atl;         (roll-up)
517
518         %gatl; - extensible global attribute set
531                 %sa.atl;            semantic authoring interface
535                 %gatl;               (roll-up)
541
542       Element content model constants
545                 %div.model;         generalized content model for divs
549                 %fig.model;         content model for figure element
553                 %kitchen.sink;      merged content models
557                 %mixed;             parsable character data and %spans;
561                 %text;              parsable character data content model
565                 %undefined;         text
569
570     Element and attlist declarations
573
574       %h1.divs; declarations
577                 div                 nested, hierarchical division
583                 h1                  H1 wrapper
589                 hsa                 HTML for semantic authoring wrapper
595                 hsg                 HTML for semantic generalization wrap
601                 prodoc              prodoc wrapper
607
608       %h1.heads; declarations
611                 h                   heading
617
618       %h1.blocks; declarations
621                 block               generic block of text
627                 codeblock           block of source code
634                 figure              wrapper bordered content
639                 hr                  horizontal rule, thematic break
644                 ol                  ordered list wrapper
649                 p                   paragraph
654                 qblock              block formatting for quoted text
```

```
 659                t                    text
 665                table                tablular, matrix data
 670                  tr                 table row
 675                tl                   text list
 680                ul                   unordered list
 685
 686      %h1.spans; declarations
 689                a                    hyperlink anchor
 694                agent                one who acts, that which acts
 699                artifact             physical or conceptual object
 704                b                    bold text
 709                behavior             action, decision, patterns
 714                big                  big text
 719                br                   line break
 724                details              @open="closed" hides all but summary
 729                i                    italic text
 734                img                  inline image
 742                meta                 meta-data
 747                name                 designates and references concept
 752                nowrap               non-breaking text
 757                q                    quotation
 762                s                    strikeout
 767                small                small text
 772                source               origin, reference
 777                span                 generic inline, span of text
 782                strong               important text
 787                summary              visible heading for details element
 792                target               goal, result, focus
 797                td                   table data cell
 802                th                   table header cell
 807                u                    underline
 812
 813    Document constants (general entity declarations)
 816                &bull;             • bullet
 820                &circle;           ∘ ring operator
 824                &emdash;           — emdash symbol
 828                &h1;                 "h1" string
 832                &H1;                 "H1" string
 836                &hsa;                "hsa" string
 840                &Hsa;                "Hsa" string
 844                                non-breaking space
1034
1035    END  h1.dtd
```

 # 2023.05.30 22:12 # D:\kc\xt\h1\ # h1.dtd #

```
                                                                    dawkIns .....
  ========================================================================== -->
```