A Hypertextual Augmentation of the Amazon E-Commerce Service (ECS) 4.0

James Creel, Jiajun Lu, Adam Mikeal Adam Mikeal, Colin Speight
Computer Science Department
Texas A&M University
3112 TAMU
College Station, TX USA
{jsc6064, j0l3429, adam, colin}@cs.tamu.edu

ABSTRACT
In this paper, we define an ontology to describe the relationships between merchandise available for purchase on Amazon.com. After gathering raw XML data from their public API, we apply a Python script to meaningfully parse the data according to the ontology as encoded in the RDF Schema language. Our ontology is made as widely comprehensible as possible by extending the Dublin Core Vocabulary. Visualization tools can browse the ontology and the data sets it describes, offering perspectives into Amazon products previously unavailable to the user. The volume and variety of the information we can thus make machine-readable provides evidence that certain promises of the semantic web are feasible, within specific contextual constraints. In particular, Amazon's products become available to a federated knowledge/data-base semantic web through the use of meaningful, well-defined XML structures.

Keywords
Semantic web, ontology, schema, hypertext, RDF, Amazon, ECS, Dublin Core, Python

INTRODUCTION

The semantic web has received much attention recently; depending on your perspective it is either a waste of research effort or a revolutionary concept set to fundamentally alter our society. Marshall and Shipman propose that there are really three distinct viewpoints from which the "semantic web" concept is defined:

1. The first perspective is that of a universal library, or what they call the "Library of Alexandria" concept—all kinds of information from many contexts collected into one resource, to be read and understood by humans in a variety of situations. In many ways, this is Google's vision of the web, and the perspective that guides the development of their services.
2. The second perspective on the meaning of "semantic web" includes the same level of universal content, but the audience is not human readers but machine-based agents, using data encoded in a semantically-aware system to represent universal knowledge. This perspective would provide the framework for the vision outlined by Tim Berners-Lee and others in the original discussions the semantic web [3].
3. Finally, there is the perspective that Marshall and Shipman term a "Federated Knowledge Base/Database"—while maintaining the machine-oriented audience, the context is particular rather than universal, thus establishing a domain-specific framework for semantic communication [8].

Much of the criticism of the semantic web is motivated by the difficulty of processing information in a context free manner, and is directed at the perspective outlined in point (2) above, the universal "Knowledge Navigator" vision of the semantic web. As Marshall and Shipman highlight, however, generic knowledge representation in a context-free environment is a classic problem in artificial intelligence that at present has no robust solution. By imposing a context on the information made available, one enhances the possibility of creating functional, communicating (semantically-aware) systems [8].

By creating an RDF representation of the Amazon product database and item relationships, we are placing our semantic web solution in squarely within the "Federated Knowledge Base" perspective, since we intend for our data to be a machine-parsable representation of the data within a specific domain—namely, the Amazon E-Commerce system.

E-Commerce and Recommendation Systems

While the rapid growth of E-commerce has allowed companies to offer more product options and customization choices to consumers, this often carries the extra burdens of information processing when making product selections. One widely utilized solution to this problem is to offer a recommender system that uses metrics such as an analysis of past buying habits, comparisons between customers and the products that they have purchased, or reviews from

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License.

To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA

other customers to make predictions of future shopping behavior and give a degree of personalization for each customer who uses the site. Schafer, Konstan, and Riedl argue that recommender systems enhance E-commerce sales in three ways:

1. Since visitors to E-Commerce sites frequently look over items without purchasing anything, recommender systems help turn browsers in to shoppers by assisting customers find what they want easily.
2. Improving cross-selling by suggesting additional products to customers to purchase. An example would be suggesting additional items during the checkout process based on items in the shopping cart.
3. Encourage customer loyalty by creating a value-added relationship between the site and the customer. The more a customer uses a recommendation system and teaches it what they want, the more likely they are to return to the web site rather than buy from a competitor [10].

These systems however, suffer from limited applicability – that is (as implied by 3. above) they tend to apply only to product available from a single provider. If product information were represented with widely recognizable semantics, recommendation systems would be able to make sense of products from a variety of sources.

Amazon's Item-to-Item Algorithm

The recommendation system available through Amazon.com is a major marketing tool for the company and is used extensively to tailor and personalize the web site to the habits and tastes of each individual customer. After logging in, a customer may click on a "My Recommendations" link that leads them to an area where they may rate the products that have been recommended and those they have previously purchases, filter their recommendations by product line and subject area, and see how items were recommended to them (in other words, they can see which product they rated that triggered the recommendation). In addition to this personalized area, customers also receive recommended products based on the items that are currently in their shopping cart [7].

The algorithm used by Amazon to generate recommendations matches each of the user's purchased and rated items to other similar items, then compiles all similar items into a recommendation list. Items are considered similar by scanning past sales records and determining which items are frequently bought in pairs by other customers. The more frequently the product pairs occur, the higher the similarity metric that is used to gauge how related two items are [7]. Items that have the highest similarity rating are added to the customer's recommendation list if they can be paired with products that have been purchased or rated by the user. For example, if a customer has purchased or given a high rating to "The Mystery Science Theater 3000 Collection, Volume 1", then the recommendation system will find other items that have been purchased by other customers who bought this DVD set. The more customers who buy the products in pairs, the higher the probability that combination will be used as a recommendation.

The current recommendation system favors customers who not only frequently purchase items from Amazon, but also take the time to rate individual products on the website. Therefore, it would be worthwhile to investigate alternate methods of constructing product recommendations for the casual or new user to the Amazon.com website. It would be particularly interesting to see if the current recommendation algorithm that is used can be improved by incorporating other product relationships that exist in the Amazon database, but are not utilized in the current incarnation of the recommendation system.

Amazon E-Commerce Service

With the introduction of the Amazon E-Commerce Service, it is now possible for developers to access Amazon's extensive and exhaustive database of product information to power their own E-Commerce applications. This service is provided free of charge once a user has registered for an account and permits an unlimited number of queries to the Amazon API. These queries can reveal a wealth of information for every product available in the Amazon catalog, including product data (such as title, pricing, and images) and content generated from and contributed by Amazon customers (such as product reviews, wish lists, and Listmania! lists).

By combining information gleaned from the Listmania! and wish lists with the current recommendation system, it may be possible to not only create a more robust recommendation system that incorporates more human-generated relationships into the system, but also generates detailed product recommendations for novice and casual users.

PROJECT DESCRIPTION

Our project defines an ontology describing the relationships between merchandise available for purchase on Amazon.com, and the nature of that merchandise.

We will begin by gathering the data available through the Amazon API using SOAP XML calls, and apply a Python script to re-encode Amazon's product data under our well-defined schema. Information in this form is easily readable across platforms and is made widely understandable by adhering to the established ontological standards of the Dublin Core Metadata Initiative.

In practice, SOAP calls must be moderated with certain limits to prevent traversals of excessively long duration. In the current implementation, users are required to provide a single product for initial seeding of the system. The system is suitable for long-term data storage, employing a durable MySQL database to store Amazon's raw information for processing. We can thus store vast portions of Amazon's product database and produce RDF encodings of selections of products on demand. In the current implementation this is all the capability required, because the RDF encodings

quickly become too large to browse visually, which is the only semantic capability we currently provide.

The second stage of our project consists of a graphical visualization tool to allow users to interact with this data in a unique way, and perhaps serendipitously discover connections between products that Amazon doesn't make clear. Visualizations are available to the user through a variety of RDF processing tools, including the likes of Welkin and Gravity. The user is presented with a graph-based view of the RDF in which each product in a web of related products appears as a single node, with edges going to related products and to other nodes representing aspects of the product. In both these tools, users can apply filters to observe only certain links. Furthermore, the Amazon web page for each product is available through the product-node URI. Finally, nodes provide links to external resources such as the Dublin Core code.

Technologies involved

The **Resource Description Framework (RDF)** is a language for describing independent resources in a machine-readable format so that independent resources may be associated. RDF provides a common framework by using metadata to represent Web resources and their relationships, thereby allowing a variety of different software applications to process the metadata. RDF uses XML syntax to save and exchange data, which is represented by RDF/XML.

Since the Amazon API returns data describing its products in raw XML format, one requires an ontology to meaningfully parse the data. That is, one must specify the classes of things which may exist and the relationships that may exist between them. Ontology, once solely in the purview of philosophy, has become a popular tool for making sense of electronic data. For instance, there exist ontologies describing Dublin Core metadata [5], the Java programming language [6], and references, such as the ontology published by Advanced Knowledge Technologies [1].

Suitable languages for encoding ontologies describing RDF include RDF Schema, OWL, or DAML+OIL.

RDF Schema (RDFS) language permits the assignment of classes and properties to the objects we describe. Furthermore, one may create a hierarchy of said classes through the `subClassOf` property. Also, properties may be organized hierarchically using the `subPropertyOf` property. The ability to create hierarchical structures make existing RDF ontologies highly extensible. RDFS also supports other sophisticated forms of expression, such as the application of properties to actual triples my means of RDFS reification. The core vocabulary is defined in a namespace conventionally called `rdfs` which string is often used as shorthand when remotely referencing the vocabulary [15].

The **OWL Web Ontology Language** is a Web ontology language designed to allow distributed and separate sources to exchange information and process its content instead of just presenting information to humans [13]. It enhances the ability of RDF to automatically interpret web content by providing additional vocabulary along with formal semantics. For example, RDF is limited to being able to define relationships between no more than 2 identities. However, it is often necessary to go beyond this limitation and describe the relationships between 3 or more identities. For example, Mary borrowed a Book from Jane. Mary, Book and Jane are three identities.

Users may implement one of three sublanguages of OWL, depending upon their needs. OWL Lite is designed for users that need a classification hierarchy and simple constraints. OWL DL keeps all OWL language constructs, but restricts their usage to certain situations. OWL Full is intended for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

DAML+OIL [4] is a language for encoding ontology provided by The Joint United States / European Union ad hoc Agent Markup Language Committee. The committee was created in October 2000 by Jim Hendler of DARPA and Hans-Georg Stork of the European Union Information Society Technologies Programme (IST). Like OWL, the DAML+OIL language extends RDFS Properties in its specification, and supports set based reasoning by including RDFS Properties for disjunction, intersection, complement, and union. These Properties are applied to the class of Classes, as restricted by the RDFS domain. The language also supports the building of classes by restricting their properties. This is achieved with a class of Restrictions, which are characterized by RDFS Properties whose RDFS domain is the class of Restrictions and whose RDFS range is the class of Properties, Classes, or some atomic elements.

These considerations significantly complicate the basic RDFS framework. However, they confer a variety of advantages. Rich languages such as these have addressed issues such as cardinality constraints on properties, specifying that a given is transitive, specifying that a given property is a unique identifier (or key) for instances of a particular class, specifying that two different classes (having different URIrefs) actually represent the same class, specifying that two different instances (having different URIrefs) actually represent the same individual, and specifying constraints on the range or cardinality of a property that depend on the class of resource to which a property is applied. We defer such considerations for future projects.

The Dublin Core Metadata Initiative offers the **Dublin Core** ontology, an RDF representation of their popular metadata standard [6]. This includes the Dublin Core Type Vocabulary, which provides a set of classes, and the Dublin Core Metadata Terms, which provides a set of properties applicable to elements of those classes. This set of vocabularies provides a solid framework which may be extended to form other ontologies, thereby creating a common frame of reference for diverse metadata applications. For example, the Dublin Core `Creator` property can by extended onto other, more specific types of

properties, such as author or artist—both a type of creator in a specific context.

Amazon E-Commerce Service (ECS) exposes Amazon's product data and E-Commerce functionality, allowing developers, web site owners, and merchants to leverage the data and functionality that Amazon uses to power its own E-Commerce business [2]. ECS uses SOAP (Simple Object Access Protocol, an XML-based communication protocol) to share data between a client and the server. We plan to query ECS to gather detailed information about all products available in the Amazon.com product catalog. Also, we plan to use the data provided by ECS to build relationships between different products using the "**Customers Who Bought This DVD Also Bought**" fields as well as Listmania! product listings. Finally, ECS can also provide product images that we will incorporate into our interface. Additional information that we may find useful includes customer reviews, wish lists, and category classifications.

Welkin and **Gravity** are both graph-based RDF visualizers supporting RDF/XML syntax [12]. Both implemented in Java and providing similar (yet slightly different) capabilities, these tools allow you to visualize the graph structure created by the RDF triples in your documents. Welkin is part of the MIT SIMILE project, a collection of technologies and applications related to the semantic web, and additional information can be found at http://simile.mit.edu/welkin/. Gravity, created and maintained by Salzburg Research, can be located at http://semweb.salzburgresearch.at/apps/rdf-gravity/.

SCENARIOS FOR USE

Searching Amazon by Keyword

Suppose a user named Mike wants to buy some books to learn the PHP programming language. However, he doesn't know exactly where to start.

He searches for the term "PHP" on Amazon.com, which returns a huge list of books related to PHP. One advantage to using Amazon.com is the ratings and hints that allow a pick the top sellers or highest rated books from the entire inventory of books Amazon.com carries. While this may help Mike narrow down his choices, he still has to go through each book's web page to find out more detailed information, such as reader reviews.

Since it is time consuming to find the right books, it will be very helpful for Mike to make a decision by providing an overview of the entire collection of books about PHP.

Mike will be able to use our tool and search Amazon's inventory for books about PHP. Information about the books will be pulled from the Amazon ECS and used to build relationships between related. Next, the tool will display a graphical representation of the relationships to Mike, thereby allowing him to see which books are frequently referenced by other items and may be a good selection for the subject. This approach can save Mike some time and allow him to find out which book is the best for his purposes.

Filtering similar items for genre

Mike has acquired an extensive associative representation of results for the search for the term "archery" using our tool, and is browsing this information. It so happens that he was in fact looking for fiction involving archery rather than non-fiction accounts of technique or practices. He can indicate to an assisting agent that he would like to restrict his view to the genre of "historical fiction" or "fantasy", and get a much-simplified subset of the available graph. This preference and its context could be recorded, to facilitate suggestions for navigation during future use.

SYSTEM FUNCTION

Populating the Local Database

In a production environment, we envision our product being used by Amazon to provide product information in RDF format. Within this context, we would have direct access the Amazon backend database of product information. However, since we do not have such access (and to our knowledge, nor does such a representation of Amazon's data exist), we must first construct a database using the Amazon API.

To construct this database, we have written Python scripts to automatically query the API, parse the XML SOAP data, and store the results in a MySQL database. We will provide some initial seed data in the form of an Amazon-specific unique product identification number (ASIN) to begin construction of this data source. For example, we might pick a movie or book title as our starting point, resolve it's ASIN, and pass that number as the starting point for the population script. Subsequent queries would then be sent automatically based on the related titles. Since the metadata for these related titles is limited to the product's ASIN and title, additional API queries are required for each title to capture the full record, including all metadata and its related titles.

While the Amazon.com public API provides unlimited queries to developers, they restrict the request rate to one transaction per second. So, we begin construction of our local database by providing an initial seed, such as a movie or book title, and recording its corresponding metadata. Included within that metadata will be a maximum of five recommended titles, a maximum of ten Listmania! product listings in which that title appears, and a maximum of five usernames gathered from customer reviews. By sending additional searches to the Amazon API, we can gather additional titles to store in the database by resolving reviewer names to their wish lists and the Listmania! names to their title listing. Therefore, from a single initial product query, we can generate a maximum of 305 additional related titles. However, since these titles will only have metadata for the ASIN and title, we must then generate additional queries for each of these 305 (potential) new titles to complete its description in the database.

Being restricted to one query per second, we decided to allow a month of continuous queries for database generation. Construction of this simulation database will take a considerable amount of time, but we felt it was

necessary since the rate restriction on Amazon's API would prevent us from generating RDF directly from API queries.

In reality, we rarely hit 305 related titles on a single ASIN, as there will likely be overlap between the various products and their related items. For example, if product A reports a relationship with products B and C, there is a good chance that at least one of the two products, B and C, will also report a relationship with A (and more likely, both will report such a relationship). Thus, a significant amount of "pruning" will occur during the building of the data source; since we are placing the results into a RDBMS as they are gathered, we will be able to eliminate unnecessary work by determining if the particular ASIN currently in the "queue" already has full metadata.

An interesting side effect of this phenomenon will be the ability to collect statistics on the rate of overlap between products, and then compare these rates across different criteria. Perhaps products categorized as literature have a very high rate of overlap between products, while technical and scientific writings have very little. We believe that interesting conclusions can be drawn from an analysis of these relationships, and the ways in which they change across various genres, subjects, etc.

Currently, this phase is partially implemented using the Python scripting language and a MySQL database. The database schema created to store the product information is available in Figure 1. The script pulls the XML data from the Amazon SOAP server, parsing the result, and storing individual records with full meta-data. Since our database schema is enforcing a strict taxonomy of product and attribute types, the script is also handling the task of creating the types in the auxiliary tables if necessary (as happens the first time the script encounters a product with a particular attribute it has not yet seen). As new attributes are added to the database schema, if they are interesting then the ontology may be updated to represent their semantics.

Formulating relationships between products

Once we have built a sizeable local database to work with, we can begin generating RDF data by querying product metadata stored within the database. While it will be a fraction of the entire body of Amazon product information, we should have sufficient entries in the database to extract a large number of relationships for examination.

In our ontology, there are three possible ways that an individual product entry can be considered related to another. First, any products appearing in the "Related Items" section are considered to be directly related to an entry (recall from the previous discussion of Amazon's API that this section is roughly analogous to the "customers who bought this also bought" section on their website). It is reasonable to expect that users who buy the same products can have similar interests, so this relationship allows a use to serendipitously discover products that they might enjoy.

A second relationship can be uncovered by extracting user names and ratings from the product reviews returned by the API. If a user took the time to post a review on the web site and give a high rating, then we assume that they enjoyed the product and may be a good source for related items. By extracting the user ID from the XML SOAP data, we can query the API for the corresponding user name, then request the titles from the user's wish list, if they have defined one on the website. We consider these relationships to be the "loosest" – that is, there may be likely be a less obvious relationship between the products since people tend to create widely diverse lists of "wish" items, from books to movies to electronics to home and garden equipment. As a result, these relationships also have to potential to be the most interesting,

Finally, two products may be considered related if both appear on the same Listmania! product listing. These associations are particularly valuable since the lists are defined by users and usually have an overarching theme that binds all the products on the list, such as "My Favorite Movies (The Ones I Own) " or "Why Zombie Flicks are Better than Real People v 1.0." This relationship can be generated by extracting the list number from the XML SOAP data, then issuing a query to the API for each list. Additional queries will be needed for each title returned. These relationships would be considered fairly "tight", as the lists do tend to follow common themes or rules that would group products together (although it will be difficult, if not impossible for a computer to discover the metrics used by the human author of the list).

Since there are three different categories of relationships between products, each representing a different type of relationship, we have decided to represent the relationship type by assigning to each a different weight, depending on the closeness of the relationships the type is describing. Additionally, if a single product appears in more than one relationship type from another single product, the weights of those relationships will be summed to create the total weight of the "closeness" of the two items. So if product A appears on product B's "Related Items" list as well as two of B's Listmania! lists, the total weight of the relationship between product A and B would be (related items weight) + (Listmania! weight) + (Listmania! weight).

For instance, if items on the same Listmania! product listing carry a weight of five and items on a "Related Items" list carry a weight of seven, then the cumulative weight of the aforementioned example relationship between A and B is seventeen. This total weight allows us to give a degree of relevancy between products, ultimately indicating to the end user which items are the most closely related to their target product, and which ones exist on the periphery. A possible implementation for a client that uses this data might allow the user to assign their own weights to the three different types of relationships, thereby affecting which products appear most tightly linked.

Now, since RDF descriptions store information only as triples (subject, property, object), it is impossible to encode relationship links including their types and weights using only one RDF triple. In order to associate such a variety of information with a single link, one may employ the technique of "blank nodes." Here, one node (product) is

linked by a RDF triple to another node (the blank node) that represents the link itself. This blank node in turn is linked by a RDF triple to the destination product node, and is also linked in other RDF triples to element nodes representing the type of link and the weight of the link. Though this approach offers the advantage of complete versatility and expressivity, it has the disadvantage of indirectly associating the products that we would like to associate directly. That is, two edge traversals are required to get from the source to destination product in the semantic graph. This representation could be taken advantage of using RDF visualization tools that were specially designed to recognize these indirect links. However, Welkin and Gravity have no capability to recognize the significance of these indirect links, so we offer the capability to produce RDF descriptions that have untyped, unweighted, direct product links stored in solitary triples. This naive solution simply permits attractive and intuitive visualizations in these generic tools.

Representing the ontology

Fundamental to our project is the ability to transmit this collected data in a semantically aware format, such as the W3C's Resource Description Framework. Since RDF only provides a framework for describing data, and doesn't specify what the data means, an ontology is required. We have chosen RDF Schema as the language for the encoding our ontology.

The XML returned by calls to Amazon's SOAP gives various item attribute tags within a given product description. One such attribute is ProductGroup, a broad characterization of the nature of the product (Book, Kitchen, etc.). These ProductGroups may be correlated with the other item attributes that may appear in the description. For instance, every product description may include the item attribute "Feature" or "PriceDescription," but only Books may include an Author. An investigation of large quantities of Amazon product data has revealed the general pattern of such item attributes within ProductGroups, motivating a taxonomy of ProductGroups. This hierarchy is given in figure 2. The leaf nodes of the tree in figure 2 are each named after an observed Amazon ProductGroup. This taxonomy must however be updated by hand whenever changes are observed in Amazon's database schema.

Item attributes tend to refer to aspects of products that are captured by the Dublin Core Vocabulary. For example, the Author of a Book is a sort of Creator, which is a Dublin Core property. Thus, we define Author as a subproperty of Creator, producing a hierarchy of properties alongside the hierarchy of classes. The property hierarchy is given in figure 3.

The applicability of particular properties to particular classes is enforced by the RDF Schema "domain" property. Thus, the Author property in our ontology is a subproperty of the Creator property and has as its domain the class of Books. Therefore, one may not apply the Author property to Kitchen products. Applicable properties are given below the class name in Figure 2.

Generation of RDF descriptions

Using the database created in Phase 1, the next step is to generate the RDF files that hold the collected data formatted according to our ontology. Since we store the data in an intermediate stage (the RDBMS), the creation of the RDF files can happen on a cached, post-processed version of the data. A script written in Python, preferred for its robust XML support and ease of string manipulation, then generates the RDF.

Once the RDF has been generated, it will be necessary to confirm its validity, both syntactically and taxonomically. Tools such as the W3C's RDF Validation Service [17] provide excellent syntactical validation, but do little to aid the author in identifying taxonomic errors. Successful ingestion of our RDF into a third-party RDF processing tools provides a practical evaluation of our ontology and the data sets to which it gives semantics. In particular, validation of the ontology was greatly facilitated by the RDFS Explorer [9].

Visualization of RDF descriptions

The differences between the two visualizers focused mainly on the mechanism used to spatially arrange the nodes on the screen. Welkin seemed to naively cluster locally-defined nodes while pushing all the externally-referenced nodes to the outer regions of the visualization (see figure 4). Gravity, on the other hand, tended to group associated nodes spatially on the screen; for example, nodes that contained links to only one other node would appear close to one another, and nodes that connected to more than one other node were arranged somewhere between the two (see figure 5). Ideally, a visualizer with knowledge of our ontology would use the weighting property inside each Link representation to determine the spatial arrangement of the nodes; links with a higher weight, and hence a closer relationship, would appear spatially closer in the visualization.

Additionally, the two tools were most diverse in their treatment of RDF properties. Welkin only creates vertices in the rendered graph for each RDF node (both locally and externally described), but not for the property values that are RDF literals. Gravity, on the other hand, draws vertices for both RDF nodes *and* literals, creating a much denser graphical representation of the RDF data set.

CONCUSION AND FUTURE WORK

We have seen how an enormous wealth of product data can be made comprehensible to a wide enough audience of computer programs to fulfill the promises of the federated knowledge/data-base semantic web. The data is made comprehensible by adhering to the Dublin Core standard, but we are able to express more specific information because our ontology extends the basic Dublin Core Vocabulary. Our work encourages the development of generic agents or recommendation systems for a knowledge/data-base semantic web by expanding the set of knowledge available to these entities. In addition, the ontology may be of interest to sellers of Amazon products or others involved in Amazon product metadata applications. In particular, the RDF descriptions we

generate from Amazon product data could be of immediate use to any entity that understands the Dublin Core Vocabulary.

Of future interest to us are applications that take advantage of our additional semantic detail. There are several applications feasible in the near term, including search algorithms that can identify the complex RDF structures that we find interesting. The primary example of interesting structures in the semantic graph is the indirect links via blank nodes which represent weighted, typed links between products. We obviously desire to filter according to link types and weights or combinations thereof. Welkin and Gravity are not presently equipped to handle such analysis.

In addition, other structures exist in these semantic graphs, which may be exploited. Such structures may represent analogy. The simplest such case is when two RDF triples are analogous if their subjects, properties, and objects are correspondingly typed to sibling nodes or the same nodes in the class and property hierarchies. However, one may envision the identification of more complex analogies arising from more complex associations between subgraphs consisting of more than one triple. For example, one might choose to associate books written by the same author within a particular period of time.

Also, when drawing the nodes of the semantic graph representing our RDF descriptions, custom visualization tools could take advantage of the freely downloadable images of Amazon products. Thus, the nodes, which serve as surrogates for Amazon's product pages and in turn for the products themselves, would be made more visually meaningful to the user.

The graph structures our RDF describes can be interpreted as polyarchic, by supposing certain nodes to be roots of certain trees. Portions of data could be translated into mSpace [11] data type, permitting yet another visualization approach.

In conclusion, our RDF descriptions and the ontology they employ present interesting semantic and visual opportunities. At the present time, our data is comprehensible to Dublin Core-enabled software, and amenable to processing by various RDF visualizers. However, much could be done to improve the utilization of our RDF descriptions. More semantically-aware interpretations of our data could reveal a wealth of relationships that are currently invisible to Amazon.com shoppers and administrators.

REFERENCES

1. The AKT Reference Ontology. http://www.aktors.org/publications/ontology/.
2. Amazon.com, http://www.amazon.com/.
3. Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. *Scientific American*, May 17 2001, 34-43.
4. DAML+OIL. Joint US/EU ad hoc Agent Markup Language Committee. http://www.daml.org/committee/.
5. Dublin Core Ontology. http://dublincore.org/2003/03/24/dces.
6. Java Programming Language Ontology. http://simile.mit.edu/2004/09/ontologies/java.
7. Linden, G., Smith, B., York, J. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering in *IEEE Internet Computing*, pp. 76-79.
8. Marshall, C. and Shipman, F. Which Semantic Web? *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, ACM 2003, 57-66.
9. RDFS Explorer. http://xml.mfd-consult.dk/ws/2003/01/rdfs/.
10. Schafer, J., Konstan, J., and Riedl, J. 1999. Recommender Systems in E-Commerce in *Proceedings of the 1st ACM Conference on Electronic Commerce*, pp. 158-166.
11. schraefel, m. c., Gibbons, N., Harris, S. Applying mSpace Interfaces to the Semantic Web. University of Southampton, 2003. http://eprints.ecs.soton.ac.uk/8639/.
12. Semantic Interoperability of Metadata and Information in Unlike Environments. http://simile.mit.edu/welkin/.
13. W3C OWL Specification, http://www.w3.org/TR/owl-features/. http://www.w3.org/2002/07/owl.
14. W3C RDF Specification, http://www.w3c.org/RDF/. http://www.w3.org/1999/02/22-rdf-syntax-ns.
15. W3C RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/.
16. Woolridge, M. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.
17. W3C RDF Validation Service. http://www.w3c.org/RDF/validator.

Figure 1: Database Schema

item				
field	**type**	**key**	**default**	**extra**
id	int(10) unsigned	PRI	NULL	auto_increment
asin	char(10)	UNI		
type_id	smallint(5) unsigned		0	
collected_meta	enum(‘Y’, ‘N’)		N	
collected_links	enum(‘Y’, ‘N’)		N	

item_types				
field	**type**	**key**	**default**	**extra**
id	smallint(10) unsigned	PRI	NULL	auto_increment
name	varchar(50)			

item_attributes				
field	**type**	**key**	**default**	**extra**
id	int(10) unsigned	PRI	NULL	auto_increment
name	varchar(50)	MUL		
data_type	enum(‘string’, ‘number’. ‘date’)		string	
type_id	smallint(5) unsigned		0	

item_data				
field	**type**	**key**	**default**	**extra**
item_id	int(10) unsigned	PRI	0	
att_id	smallint(5) unsigned	PRI	0	
value	varchar(200)	PRI		

item_relationships				
field	**type**	**key**	**default**	**extra**
initial_item	int(10) unsigned	PRI	0	
related_item	int(10) unsigned	PRI	0	
type	enum(‘related’, ‘listmania’, ‘wishlist’)	PRI	related	
total	smallint(5) unsigned		1	

Figure 2: Ontology Class Hierarchy

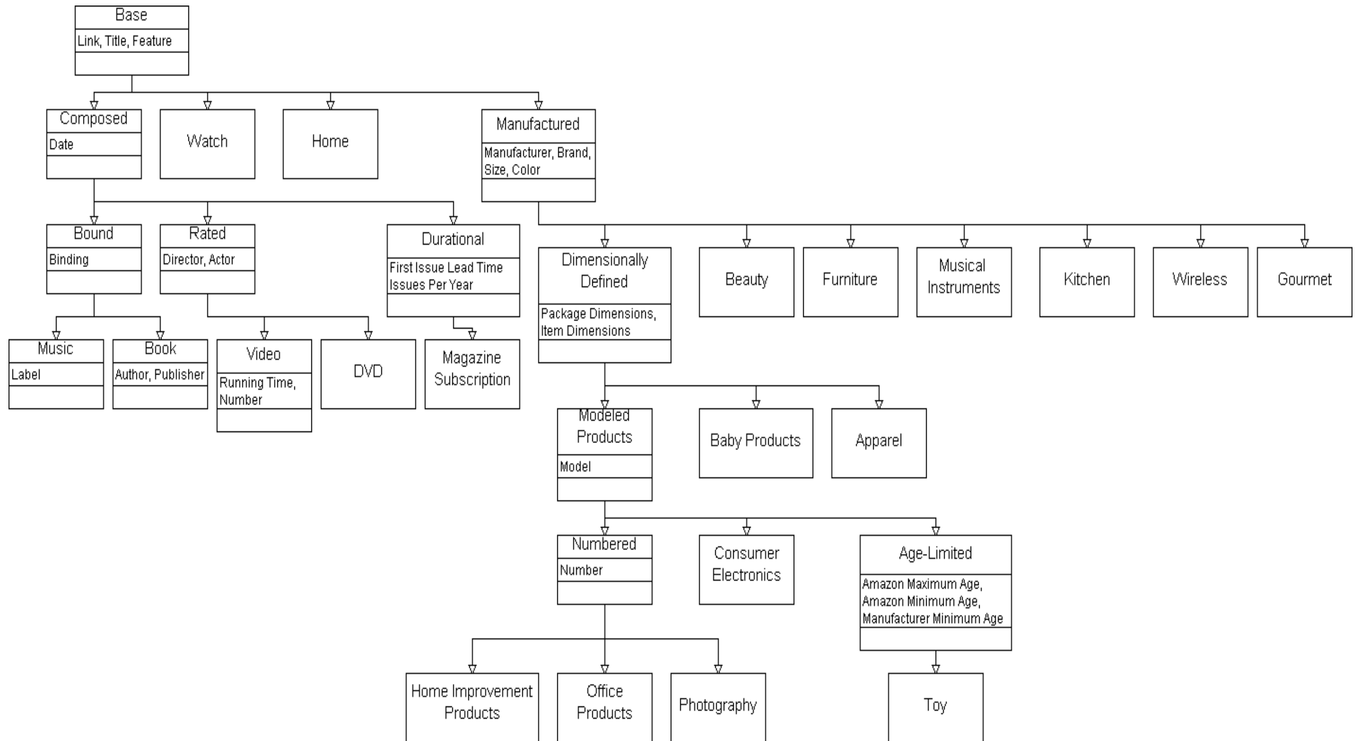

Figure 3: Ontology Property Hierarchy

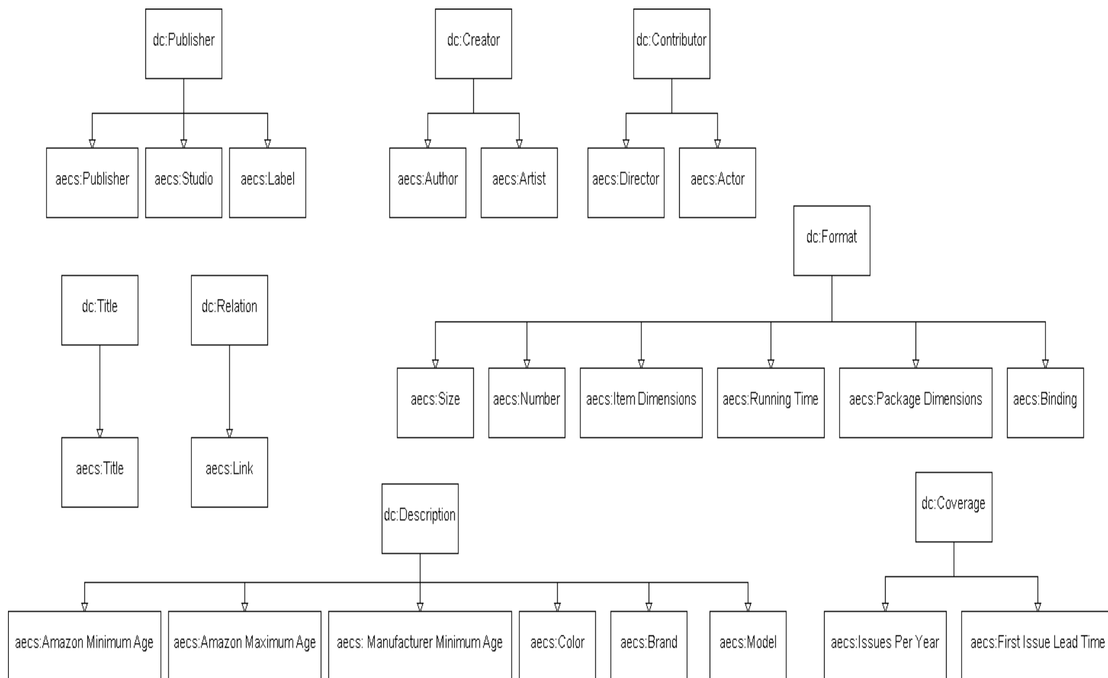

Figure 4: Sample Welkin RDF Visualization

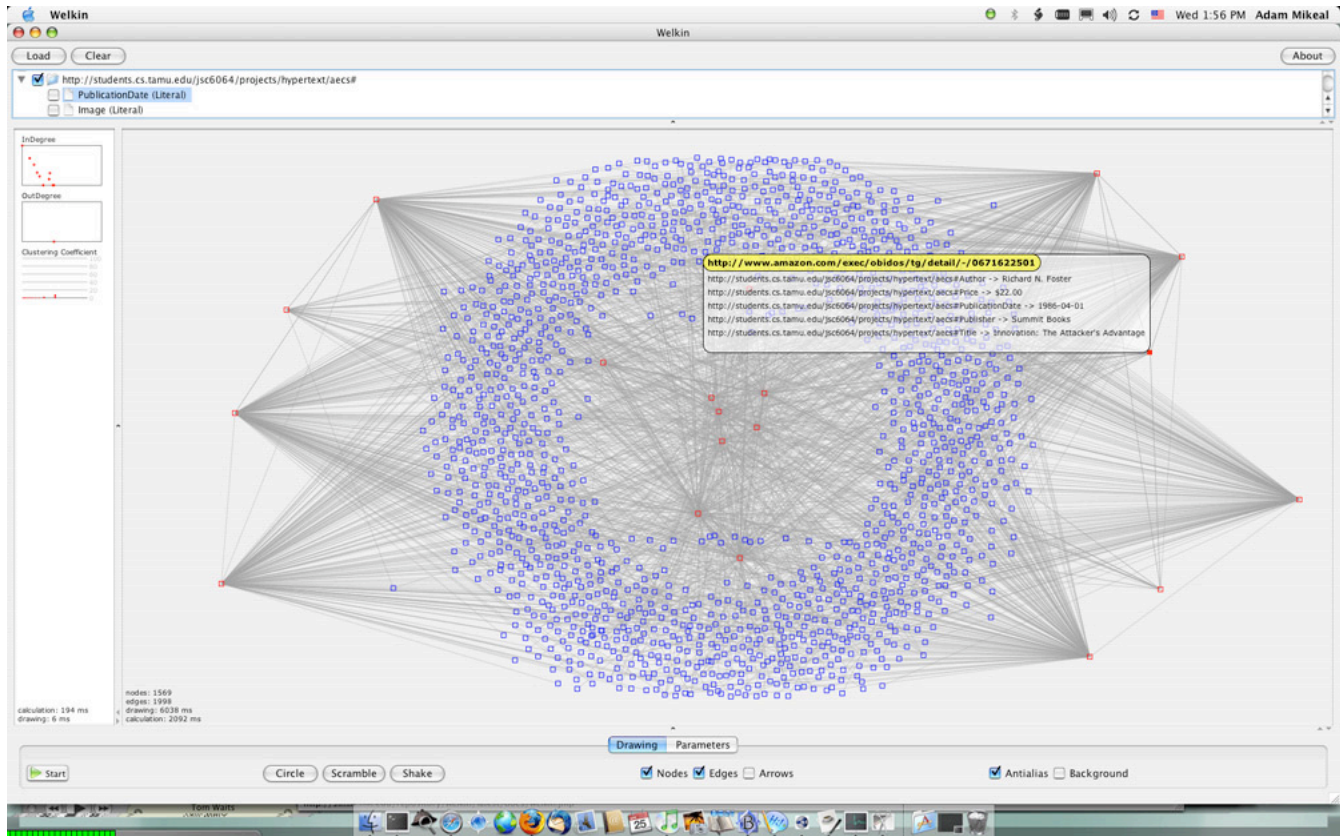

Figure 5: Sample Gravity RDF Visualization