GrassRoots: Socially-Driven Web Sites for the Masses

Frank Uyeda, Diwaker Gupta, Amin Vahdat, George Varghese
U.C. San Diego
9500 Gilman Drive
San Diego, CA 92093-0404
{fuyeda, dgupta, vahdat, varghese}@cs.ucsd.edu

ABSTRACT

Large, socially-driven Web 2.0 sites such as Facebook and Youtube have seen significant growth in popularity [5, 10]. However, strong demand also exists for socially-driven web sites specialized to companies and knowledge domains. Unfortunately, existing tools for building such sites only provide low-level functionality to address recurring search and organization patterns. Further, they require expertise at many levels of the software stack.

Therefore, we propose GrassRoots, a declarative language for modeling socially-driven websites and a compiler to automatically generate the code at several layers of the software stack. We provide abstractions for modeling data and relationships, search, page composition, and navigation. Most notably, we propose a graph-based data model that allows designers to both filter and rank search results using structural and value-based primitives. In this paper, we describe the GrassRoots language and show how popular socially-driven websites can be specified using it. We also describe the `GR` compiler that generates web sites based on GrassRoots specifications.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous; D.2.3 [**Software Engineering**]: Coding Tools and Techniques

General Terms

Design, Languages

Keywords

Web 2.0, Declarative Specification, Code Generation

1. INTRODUCTION

Facebook, YouTube, and Flickr have become key web-based communities fielding millions of users [5, 10]. Besides well-known sites, there is significant interest in *specialized* socially-driven sites. USA Today reports [17] that hundreds of companies have already deployed internal social networks to support operations. For example, a small company may want to host training videos online for employees to view and comment on. Sites like YouTube do not provide adequate customizability or sufficient control for such a company's videos. Recognizing this demand, Cisco and IBM have been developing corporate social networking tools [16, 6].

Besides companies, there is a long tail of communities with specialized knowledge domains that need custom web sites. Examples include insurance (e.g., Wellpoint.com) and education (e.g., RateMyProfessor.com). What is common across all of these specialized sites is the ability for users to submit content and to search for the content contributed by others. We label this class of web sites as *Search & Submit* and note that they expand beyond traditional online social networks. We focus on providing tools and abstractions to ease the development of these socially-driven sites.

Today, both companies and communities with shared interests must piece together their specialized web site using one of many web-development frameworks along with significant customized code. Customized code is necessary because standard web development tools (e.g., Ruby on Rails [9]) do not sufficiently address all the unique aspects of each specialized socially-driven website.

To address the problem of creating such sites, we propose GrassRoots, an abstract language to specify a wide range of socially-driven web sites and the `GR` compiler that generates the web site and configures the backing storage from the GrassRoots specification. We believe that GrassRoots's abstractions also enable other aspects of the application life cycle, such as automatic scaling.

In contrast to popular web-development frameworks, like Ruby on Rails, GrassRoots provides specialized abstractions for searching and ranking results. *We believe that the differentiating feature among many competing socially-driven web sites is the way data is organized and searched.* Thus, abstractions for structuring data and searching are crucial. GrassRoots allows developers to easily experiment by making small changes to the specification that can then be compiled to a working web site. For example, the developer of a cookbook web site might initially allow searching for recipes by category and recipe name. Later, users might request the ability to use ingredients as tags. GrassRoots can implement this new feature by changing a few lines in the site specification and recompiling. By contrast, a Ruby on Rails implementation would require changes to the database, application logic, and presentation layer.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WOSN'09, August 17, 2009, Barcelona, Spain.
Copyright 2009 ACM 978-1-60558-445-4/09/08 ...$10.00.

To demonstrate the power of our abstractions we have specified simplified versions of Flickr.com and Digg using our GrassRoots language (Section 3). We omit complete specifications for lack of space (however, a reference to the complete specification is provided). We do show in Section 3.5 how to add user tagging (a Facebook feature) to a Flickr implementation by adding 3 lines to the specification.

The organization of the paper is as follows. Section 2 motivates our abstractions by surveying various sites and looking at Flickr in detail. Section 3 describes details of the GrassRoots language using Flickr as an example. Section 4 describes the implementation of our `GR` compiler, as well as our experiences specifying other websites. Section 5 describes related work, and Section 6 provides our conclusions.

2. MODELING SITES

Our abstractions are motivated by the observation that Search & Submit websites utilize a relatively small number of search paradigms and organizational structures. Table 1 lists several popular Search & Submit websites with their predominant organizational structures and search methodologies. Flickr and YouTube appear distinct because Flickr hosts photos while Youtube hosts videos, but fundamentally both allow users to search for content based on keywords and tags. Similarly, Digg.com and Craigslist both organize data according to a hierarchy, but Digg.com stores URLs while Craigslist stores listings.

Conversely, sites can have the same content type but utilize different organizational structures that yield different search paradigms. For example, Del.icio.us and Digg.com are both social bookmarking sites that host URLs. However, Del.icio.us organizes URLs using tags, while Digg.com uses a hierarchy and a specialized ranking algorithm. Therefore we claim that a good abstraction for Search & Submit sites should separate the type of data in the site from the site's organizational structure. To further motivate our model, we now describe Flickr.com.

2.1 Example: Flickr.com

Flickr.com is a socially-driven website that allows users to organize photos by associating them with textual tags. Each tag appears as a link directing the user to a page of other photos with that tag. Flickr also allows users to organize their photos into groupings called "sets". A set may contain many photos, and photos may belong to any number of sets.

Flickr users can join groups or form friend relationships with other users. Friend relationships are used as a basis for granting additional access rights to photos, or to easily tracking the new photos posted by friend. Further, groups allow members to share photos with one another through the group's photo pool — a collection of photos contributed by the group's members. As with sets, photos can belong to many different pools and pools can contain many photos.

Abstractly, the data objects in Flickr are Users, Groups, Photos, Sets, Pools, Tags, and Comments. Additionally, relationships exist between instances of these objects. For example, a User may own many Photos, and each Photo has a single User who owns it. These objects and relationships can be modelled as nodes in a graph where special constraints exist on the allowed edges.

While we can think of the data model as a general graph, by imposing more structure on the graph we can provide

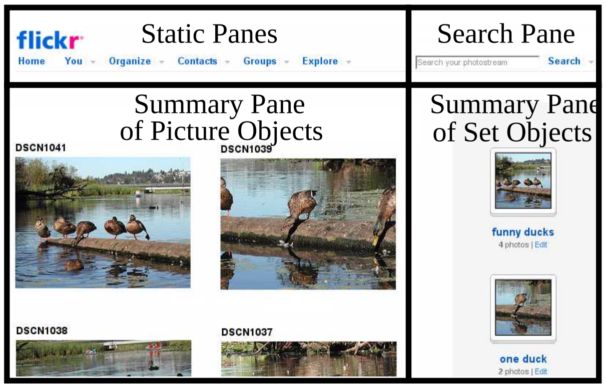

Figure 1: A page is composed of one or more panes. We show a Flickr page overlaid with logical panes.

more powerful search abstractions. We model Flickr predominantly using *bipartite* graphs. Some of these bipartite graphs are constrained to be one-to-many (e.g., Users and the Photo they own) while others are many-to-many (e.g., Users and Groups). However, since the friend relationship between Users is asymmetrical, a *directed* graph structure is used, consisting only of User nodes. Further, in sites like Craigslist and Digg, we specify the graph as a tree, allowing GrassRoots to provide search over specified subtrees.

We have found that five graph structures are very common across Search & Submit sites: *tree, undirected graph, directed graph, bipartite one-to-many, bipartite many-to-many.* The third column of Table 1 lists the graph types seen in various web sites. Specifying these structures allows for specific search operators (shown in Table 2).

Users interact with Flickr by navigating through web pages, each of which is divided into regions which either display data or receive user input. We call these regions *panes.* Figure 1 shows an example of a Flickr page divided into panes. Clicking on objects within each pane may invoke an action, such as navigating to a new page. We describe panes in detail in Section 3.3.

3. SPECIFICATION LANGUAGE

Summarizing the last section, we model a site as various types of data objects organized into a graph with constraints declared on the edges allowed between sets of objects. Users navigate between pages, each of which consists of one or more panes. Navigation invokes searches which populate the panes of the subsequent page.

This section describes the GrassRoots language based on this model, and shows how GrassRoots can be used to quickly specify a Search & Submit website. We use Flickr.com as a running example and show how GrassRoots's abstractions concisely capture the majority of its function and structure.

Sites are described in GrassRoots's specification language and then submitted to the `GR` compiler. From the specification, `GR` produces a database schema and dynamic webpages implementing it. This process is visualized in Figure 2.

Underlying our model are four areas of abstraction (data models, searches, page layouts, and actions) that we now describe, along with examples from our Flickr specification.

Site	Objects	Data Relationships	Search
Flickr	Images	Bipartite Graph, Undirected Graph	Keywords, Tags, Comparison (geo-tag)
Youtube	Video	Bipartite Graph, Directed Graph	Keyword, Tags
Last.fm	Audio	Bipartite Graph, Tree	Tags, Structural
Del.icio.us	URLs	Bipartite Graph	Tags
Digg	URLs	Tree	Taxonomy, Keyword
Craigslist	Listings (Image & Text)	Tree	Taxonomy, Keyword, Comparison
Wikipedia	Articles (Text)	Directed graph	Keyword, Structural
Facebook	User Profile (Image & Text)	Undirected graph, Bipartite graph	Structural, Tags

Table 1: Different but Similar: While various sites have different details, they can all be specified using the same abstractions. In GrassRoots, we differentiate sites by the types of data objects, the structure of the data, and by the predominant types of search used to navigate the site.

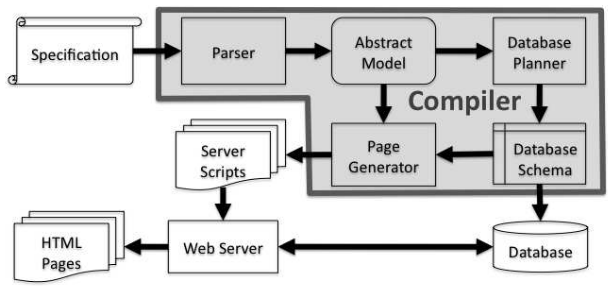

Figure 2: `GR` takes a specification as input and produces a database schema and server-side scripts that implement the site's pages.

3.1 Data Types & Relationships

While data modeling (e.g., UML) is well studied, we include a data model in our language in order to facilitate our new search abstractions. Unlike SQL, we separate the declaration of data types and relationships. This allows developers to interchange arbitrary types of content independent of the site's structure.

GrassRoots defines and supports several simple data types, such as `TEXT`, `IMAGE`, `VIDEO`, `LOCATION`, and `URL`. Simple data types can be grouped and aggregated through the complex data types: `LIST` and `COMPOSITE`. A List, analogous to a Vector in C++, is a collection of instances of a single type, while a `COMPOSITE`, analogous to a struct, is a grouping of one or more named member-types that can be individually accessed. User and group types are special composites (that can be overridden as needed) that include a unique ID and a password hash.

GrassRoots allows common functionality (such as tagging, comments, and versioning) to be be expressed as attributes of data types. For example, the `taggable` and `commentable` attributes, implicitly include the necessary storage and structure to implement these behaviors with no additional effort from the developer. We allow any data type to be declared as a tag or a comment for another type. Examples of this are user names tagging pictures in Facebook, and video comments on YouTube videos. `GR` automatically allocates metadata fields that can be used in search (such as owner and submit time) for each declared type.

As mentioned in Section 2.1, all relationships within GrassRoots are abstracted as graphs. We specify these relationship types as `tree`(t), `graph`(t), `direct_graph`(t), `one_to_many`(t_1, t_2), and `many_to_many`(t_1, t_2), where t is the data type used as the nodes in the graph. Names are given to structures so they can be referenced from search queries. Additionally, edges within structures can be augmented with *integer weights* or *text labels* by adding the `[weighted]` or `[labeled]` attributes to the structure declaration (e.g. `graph[labeled](USER)` ...) While some of these constructs exist in XQuery [18] and ER diagrams [13], none of these existing tools completely capture all our needs.

3.2 Search

The major abstraction provided by GrassRoots is search, an abstraction that aims to encompass such diverse search paradigms as those found in Facebook, Digg, and Youtube. The primary objective of search is to return a set of results in ranked order. This task can be decomposed into two components. First a subset of items matching some filter criteria must be selected. Second, these items must be sorted according to some relevance metric. We now describe filtering and ranking in turn with special emphasis on a set of enhanced operators. These operators leverage the declared structure of the data to make both filtering and ranking more natural and intuitive.

3.2.1 Filtering Results

Searches in GrassRoots begin by specifying the data type of the result along with the cardinality of the result set. Searches returning a single value utilize the `LOOKUP` keyword, while searches returning arbitrarily many results use the `SELECT` keyword. The syntax of the `LOOKUP` keyword allows developers to fetch an object with a given UID, a random object, or the item returned by a function, such as the `min()`, `max()`, or `parent()` functions.

The `SELECT` search permits two types of operators that can be used separately or in tandem. The structural operators filter objects using graph-oriented functions. The allowed operators are determined by the relationships specified in the data model. These operators are listed in Table 2. In addition, structural operators can be nested such that the resulting list from one is used as the input to another. The second class of `SELECT` operators are value filters. We enumerate GrassRoots's search filters in Table 3.

In addition to these simple query types, developers can express more complex filter logic by combining queries using `AND`, `OR`, and `NOT` operators. The syntax of a `SELECT` is:

```
SELECT <type> FROM <structural filter>
WHERE [NOT] <value filter>
  [{AND|OR} [NOT] <value filter>]*
ORDER {ASCENDING|DESCENDING} [BY <ranking function>]
```

Operator	Description
`neighbor`(n)	Find items adjacent to item n.
`children`(n)	Find items which are children of item n in a tree structure.
`parent`(n)	Find the item which is the parent of item n in a tree structure.
`descendants`(n)	Find all items for which item n is an ancestor in a tree structure.

Table 2: Structural search operators.

Search	Description
`matches` x	Find items which have value x.
`contains` x	Find `TEXT` items containing value x.
`less_than` x `greater_than` x	Find items with values less than or greater than x.
`between` x, y	Find items between x and y.
`proximity` o, r	Find items within radius r of point o.
`tagged by` x	Find items associated with tag x.

Table 3: Sample filter search operators.

3.2.2 *Ranking Results*

GrassRoots allows search results to be ordered in four different ways. Results can be ordered *canonically* based on the value of the object or the value of one of its meta fields, such as submit time. Next, objects can be ordered based on *node properties* within the structural relationship, such as degree (number of edges) in a graph, or a node's Page Rank [11]. Third, objects can be ordered by *path properties*, such as shortest-path distance. This mode is currently seen in Facebook, where the results of a user search are ordered by the distance between each result and the querying user. Finally, objects can also be ordered based on a weighted sum of any number of the previous ranking methods. As sample of ranking functions in GrassRoots are shown in Table 4.

Thus in summary, the developer specifies a search filter based on the content's value and structure and also a ranking function to order the search results. Examples of search methods are shown in Figure 3, which can be more clearly understood after we discuss page specifications.

3.3 Pages & Panes

The developer specifies a page as a list of panes, search queries to populate each pane, and a list of arguments to be passed into the page. Panes can be of two types: display panes and input panes. An example of a page divided into panes can be seen in Figure 1. Display panes allow various pages to display different objects, but with the same behavior. Within display panes, a *Detail Pane* displays a single object in depth (e.g. playing a video in YouTube). A *Summary Pane* displays abbreviated information about many objects (e.g. search results). Additionally, *Static Panes* contain static content and is useful for creating page headers. Input panes generate a form and submit button. Input panes can either be *Submit Panes* (to upload new content), *Update Panes* (to modify existing objects), and *Search Panes* (to input terms for search queries).

Sample Flickr page specifications are listed in Figure 3. The *tag_result* page contains a single Summary pane that is populated by a `SELECT` query leverages the `TAGGED BY` filter.

Ranking Function	Description
<field>	Rank canonically by a particular field's value.
<structure>`.degree()`	Sort by a node's edge count in the structure.
<structure>`.pagerank()`	Sort by the node's pagerank within the structure.
<structure>`.distanceTo(`n`)`	Sort by path weight from the item to node n.
<structure>`.hopsTo(`n`)`	Sort by hop count from the item to node n.

Table 4: Sample search ranking operators.

```
PAGE tag_result( TEXT t ) {
  Summary(Picture) results_pane :
    SELECT Picture WHERE TAGGED BY t
    ORDER ASCENDING BY _submit_time;
}
PAGE user_profile( User u ) {
  Detail(USER) owner: LOOKUP User u;
  Summary(USER) friend_list(10, overflow) :
    SELECT USER FROM Friends.neighbor(u);
  Summary(Picture) owned_pics(25, overflow) :
    SELECT Picture WHERE _owner MATCHES u;
  "add picture" -> linkto add_pic();
}
```

Figure 3: Sample Flickr pages. Each page is defined by input arguments along with pane references and search queries. Search results are used to populate the Detail and Summary Panes.

This filter is enabled by the `taggable` attribute given to the Picture type in the data model. While not shown, Flickr's geo-tagging feature can be specified by adding `taggable by` and a `LOCATION` type to Picture's attributes. Searches could then use the `LOCATION` tag or the `proximity` operator from Table 3.

The *user_profile* page takes a User object as input and has four panes: a Detail pane to show the user's profile, a Summary pane that is parameterized to list up to 10 friends and "overflow" remaining results to additional pages, a Summary pane listing up to 25 of the user's pictures, and a Static pane that contains a link to the *add_pic* page.

3.4 Actions & Navigation

Actions are used to both update application state and navigate among pages. Actions are bound to the data fields within display panes or to the `on_submit` keyword in input panes. The keyword `THIS` refers to the object currently being displayed. In the case of summary panes displaying many objects, action links are created for each object and the action parameters are set relative to the object they are displayed with. Navigation is specified by the `linkto` action together with the destination page and optional arguments.

Although we omit details, a formal model of computation would represent the system as a state machine of (display-state, database-state) pairs. The system "executes" when user input occurs, changing the database state and possibly passing input parameters to the next page (display state).

Figure 4 gives a sample of our Flickr pane specifications. The Picture Summary pane displays one or more pictures

```
Summary Pane : Picture {
  pic -> linkto pic_detail(this);
  _owner -> linkto user_profile(_owner);
}
Detail Pane : Picture {
  pic;
  _tag -> linkto tag_result(_tag);
  _owner -> linkto user_profile(_owner);
}
Search Pane : find_picture {
  TEXT t;
  on_submit -> linkto search_result(t);
}
Submit Pane : Picture {
  require pic;
  optional pic_title;
  on_submit -> linkto user_profile(_user);
}
```

Figure 4: Sample of Flickr's pane specifications.

```
COMPOSITE Picture {...} (taggable by USER);
Summary Pane : Picture { ...
  _tag<USER> -> linkto user_profile(_tag<USER>);
}
Detail Pane : Picture { ...
  _tag<USER> -> linkto user_profile(_tag<USER>);
}
```

Figure 5: The changes required to implement tagging by user identifier in our Flickr specification are shown on lines 1, 3 & 6. Other types of data, such as location, could be used for tagging by changing "USER" to the desired data type.

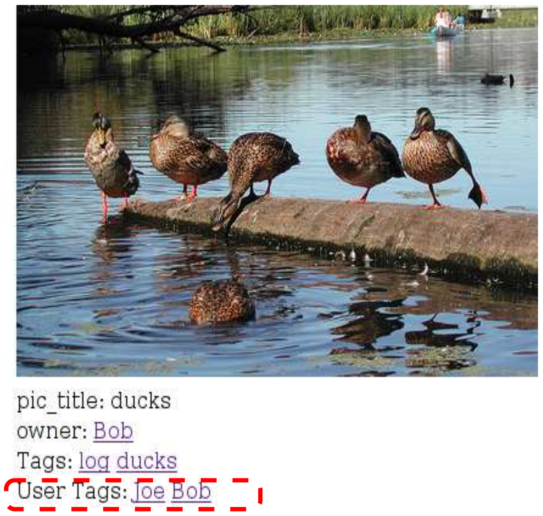
pic_title: ducks
owner: Bob
Tags: log ducks
User Tags: Joe Bob

Figure 6: The pic_detail page output *after* user tagging is added to the specification, as shown in Figure 5. The added user tags are shown in a highlighted box.

along with their owners. Clicking on either of these items navigates to the specified page; for example, clicking on the owner links to the user profile page specified in Figure 3.

The Picture Detail pane is similar to the Picture Summary pane except it also displays a single picture and its tags. The *find_picture* Search pane allows the user to enter a piece of text which is passed to the *search_result* page. Finally, the last pane allows the user to submit a picture; after the submission, the user is navigated to their own profile page.

3.5 Adding a Facebook feature to Flickr

A powerful aspect of abstractions is the ability to easily add or modify parts of the modelled site. While most sites use text tags to categorize objects, Facebook uses user names to tag objects. Adding this kind of functionality to Flickr would require additional database tables and a non-trivial amount of page logic. However, within GrassRoots, data types and behaviors have been separated. This allows developers to specify tags, comments, or other structural relationships using any simple or complex data types. In Figure 5, we show how the *addition of three lines in the basic Flickr specification introduces tagging with users* to the site.

Figure 6 shows the output of a Detail Picture pane produced by our GR compiler after adding user tagging. By default, the compiler lays out panes sequentially on a page within HTML div elements. The developer can then use CSS to customize the page presentation, thus enabling Figure 6 to have the look-and-feel of Figure 1.

4. IMPLEMENTATION & EXPERIENCE

The GR compiler's logical components are depicted in Figure 2. Our current implementation converts GrassRoots specification files into SQL schemas and PHP server scripts. PHP can be replaced with a language of choice such as Ruby or Python.

GR is currently written in Java and consists of about 15,000 lines of code. Our specification parser leverages the JLex [7] lexical analyzer and the CUP [3] parser generator for Java. The parser converts the site specification into an object model which is used by both the database planner and page generator components. As an example of GrassRoots's capability, the core pages and features of Flickr were specified in 180 lines. This includes user profiles, groups, photo sets, tagging, and comments. After compilation, this specification resulted in a prototype consisting of 2,000 lines of PHP, and schemas for the 11 required database tables. We have also concisely specified versions of several other sites in Table 1 including Digg and YouTube in 150 lines and 205 lines, respectively. The specifications can be found in [1]

We are also working with learning theorists to deploy a music recommendation web site that automatically tags uploaded MP3s with features, such as genre (e.g., Classical) and mood (e.g., Calm), as well as weights for how well each tag applies. Users can search based on many tags (e.g. Calm and Classical). Results are ordered based on a song's weighted relevance to all search tags. We model this site using a weighted bipartite graph that connects tags to songs. Searches select neighbors of tags ordered by the sum of edge weights. Existing tagging packages for frameworks such as Rails do not support relevance weights or multiple concurrent tags. We plan to make the client-side interface more interactive by adding AJAX support for reloading specific panes. This can be done by adding actions to specify which pane to reload and the parameters of the new search.

5. RELATED WORK

We now survey related work in the web development space. Prior work focuses on tools to develop *general purpose* websites with search abstractions limited to relational queries. By contrast, our GrassRoots provides higher level graph-based search abstractions by focusing on a *specialized* subset: search-driven, social websites.

Web Frameworks: Web scripting languages (e.g., Ruby) have their own frameworks (e.g., Rails) [9] for adding specific search features such as tagging. However, these libraries limit developers to a limited set of search components, which can only be customized by extensive code modifications. Second, frameworks such as Rails provide only one layer for web application development; the developer still has to be familiar with SQL, HTML, CSS, Ruby, etc., each of which has a different programming model. While languages have incorporated Object Relational Mappers (ORMs) such as ActiveRecord in Rails, designers must still *manually* design their own database schemas

Online Services: Ning [8], Blist [2] and DabbleDB [4] are online services that allow users to create database-driven applications. All online platforms are closed source and require hosting on their servers. Blist and DabbleDB are intended for forms and reports. Ning is a platform for building social networks but lacks the power of graph-based search operations as we provide.

Web Application Builders: Several systems such as WebML [12] and Hilda [19] build *general purpose* database-driven websites based on declarative models. However, neither WebML nor Hilda provide structural search and ranking abstractions tailored to socially-driven web sites.

Graph Databases: Graph databases and query languages [15, 14] provide an extensive set of graph operators. Future versions of GrassRoots could benefit by incorporating a fuller set of the query operators provided by graph databases.

6. CONCLUSION

Beyond high profile sites such as YouTube and Facebook, there is a long tail of specialized socially-driven web sites. Such communities need better tools so that they can focus on content organization and search. We have introduced GrassRoots, a declarative language and compiler which presents a graph-based data model abstraction to web application developers. Most notably, our search abstraction incorporates primitives from both relational and graph query languages in filtering and ranking results.

For deployment, our GrassRoots system must be integrated with tools for page design and layouts such as Frontpage, and tools for scalable hosting such as Amazon EC2. GrassRoots also needs constructs to specify access control and security policies. The separation of application logic from implementation in GrassRoots also provides the opportunity for automatic optimization for performance and scaling.

We envision a future in which experts within a domain can prototype a Search & Submit Web 2.0 site in a few days using a concise GrassRoots specification, modify the specification as they gain experience with users, use existing tools and services to produce a production quality site, and use the `GR` compiler to automatically scale as the site grows in popularity. In doing so we hope that GrassRoots will be a catalyst for fostering online communities and democratizing the creation of socially-driven web sites.

7. REFERENCES

[1] http://www.flare.ucsd.edu/grassroots-specs/.

[2] blist. http://www.blist.com/.

[3] Cup. http://www2.cs.tum.edu/projects/cup/.

[4] Dabbledb. http://dabbledb.com/.

[5] Facebook. http://www.facebook.com/press/info.php?statistics.

[6] Ibm center for social software. http://www.research.ibm.com/social/.

[7] Jlex. http://www.cs.princeton.edu/ appel/modern/java/JLex/.

[8] Ning. http://www.ning.com.

[9] Ruby on rails. http://www.rubyonrails.org/.

[10] Youtube draws 5 billion video views. http://ir.comscore.com/releasedetail.cfm?ReleaseID=333578.

[11] S. Brin, , L. Page, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, 1999.

[12] S. Ceri, , P. Fraternali, and A. Bongio. Web modeling language (webml): a modeling language for designing web sites. *Comput. Netw.*, 33(1-6), 2000.

[13] P. P.-S. Chen. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*, 1(1), 1976.

[14] S. Flesca and S. Greco. Partially ordered regular languages for graph queries. *J. Comput. Syst. Sci.*, 70(1), 2005.

[15] R. H. Güting. Graphdb: Modeling and querying graphs in databases. In *VLDB '94*.

[16] B. Stone. Social networking's next phase. *New York Times*, March 3, 2007.

[17] J. Swartz. Social networking sites help companies boost productivity. *USA Today*, October 7, 2008.

[18] W3C. *XQuery 1.0: An XML Query Language*, January 23, 2007.

[19] F. Yang, , J. Shanmugasundaram, M. Riedewald, and J. Gehrke. Hilda: A high-level language for data-drivenweb applications. In *ICDE '06*.