Enabling Enterprise Mashups over Unstructured Text Feeds with InfoSphere MashupHub and SystemT

David E. Simmen, Frederick Reiss, Yunyao Li, Suresh Thalamati
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
simmen@us.ibm.com, frreiss@us.ibm.com, yunyaoli@us.ibm.com, tsuresh@us.ibm.com

ABSTRACT
Enterprise mashup scenarios often involve feeds derived from data created primarily for eye consumption, such as email, news, calendars, blogs, and web feeds. These data sources can test the capabilities of current data mashup products, as the attributes needed to perform join, aggregation, and other operations are often buried within unstructured feed text. Information extraction technology is a key enabler in such scenarios, using *annotators* to convert unstructured text into structured information that can facilitate mashup operations.

Our demo presents the integration of SystemT, an information extraction system from IBM Research, with IBM's InfoSphere MashupHub. We show how to build domain-specific annotators with SystemT's declarative rule language, AQL, and how to use these annotators to combine structured and unstructured information in an enterprise mashup.

Categories and Subject Descriptors
H.m [**Information Systems**]: Miscellaneous

General Terms
Design

Keywords
Information integration, Text analytics, Feeds, Mashups

1. INTRODUCTION
Increasingly large numbers of specialized applications are developed by enterprise business users in response to situational business needs [1]. Such applications often require access to information derived by combining data in corporate databases, content management systems, and other IT managed repositories, with data from the desktop, Web, and other sources typically outside IT control. Web 2.0 inspired enterprise mashup technologies, like IBM's InfoSphere MashupHub (MashupHub) [2], have been developed to meet the data integration requirements of such applications. MashupHub, which originated from the Damia research project at IBM Almaden [3], provides visual facilities for quickly and easily creating data mashups that filter, join, aggregate, and otherwise transform feeds published from a wide variety of sources, into new feeds that can be consumed by AJAX, and other types of web applications.

SIGMOD'09, June 29–July 2, 2009, Providence, RI, USA.
ACM 978-1-60558-551-2/09/06.

An important class of enterprise mashup scenarios involves feeds derived from data created primarily for eye consumption, such as email, calendars, blogs, wikis, and web feeds. Such feeds often contain the data needed to perform mashup operations buried within swaths of unstructured element and attribute text. Information extraction technology can be a critical enabler in such scenarios, providing various types of text annotators for discovering entities, relationships, and other attributes that can be exploited by mashup operations.

Current mashup applications make use of information extraction technology by exploiting text annotation services [4] via web services. These services *annotate* unstructured text to produce structured data that a mashup can directly employ. However, even ignoring the obvious scalability concerns, there are significant drawbacks to relying exclusively on external annotation services. One drawback is that the set of generic annotators provided by such services are not necessarily tuned to work well in mashup environments. For example, a feed can join with more sources if it is annotated with more specific attributes such as street (e.g. "650 Harry Road"), city (e.g. "San Jose"), and state (e.g."CA"), versus more general ones such as location (e.g. "650 Harry Road, San Jose, CA"). Writing annotators that work with high specificity and low noise requires careful tuning of extraction rules. Moreover, annotators tuned for feeds must deal intelligently with markup. This requirement might mean ignoring html tags or exploiting XML element and attribute data (perhaps of parent or sibling nodes) in order to achieve greater precision and recall.

Another drawback is that the set of annotators provided are fixed and hence cannot be extended with new annotators that target a particular installation, feed source, or mashup application. For example, a semiconductor company may need to extract information about FPGA users' performance requirements from articles in the technical press, a task that no prebuilt library is likely to accomplish. Even if a remote text annotation service supports customized annotators and dictionaries, it would be hard to share such customization efforts. The reasons are two-fold: first, users of such web services are unlikely to share the same scenario or data sources; second, companies need to protect their intellectual property and unlikely to have their customized annotators and dictionaries stored at a third party.

In this demonstration, we present extensible text annotation capabilities recently added to MashupHub. These capabilities include (1) a set of pre-packaged annotators tuned to work with high precision and recall on important types of enterprise feed sources such as email, instant messages, and news, and (2) the capacity to build, test, and deploy custom, application-specific

annotators. Our solution exploits SystemT information extraction technology developed at IBM Research [5]. SystemT employs a novel algebraic approach to information extraction, wherein annotators are expressed in a high-level declarative language called AQL. The annotators are then compiled into efficient extraction strategies using cost-based optimization techniques. We have integrated the SystemT runtime with the MashupHub server so that text annotators can be executed efficiently over feed text during mashup execution. This approach allows our solution to scale in terms of text size, numbers of requests, and annotator complexity. More importantly, new annotators, expressed in terms of AQL rules, can be created using the SystemT Development Environment [6][7] and uploaded to the MashupHub server, and can be shared by the existing community of MashupHub users.

In the next section of this proposal, we provide a high-level overview of the architecture of our demonstration system. Then, in the final section, we outline the demo scenario that we will use to show the effectiveness of our approach.

2. SYSTEM OVERVIEW

Figure 1 illustrates the overall architecture of the system that we will demonstrate. This architecture incorporates several elements that have never been demonstrated at an academic conference:

- The SystemT Development Environment
- The AQL declarative rule language
- The integration of SystemT's information extraction runtime into InfoSphere MashupHub.

MashupHub: The central component of our architecture is InfoSphere MashupHub (MH), a system for building community-based mashups. MashupHub has a web-based client-server architecture: the MH application server exposes its services to the client via REST APIs.; and the browser-based client exposes the server capabilities to business users through an intuitive GUI, as shown in the screenshot in Figure 2.

The server provides services for managing feeds and for creating and executing feed mashups, presentation widgets, and other application assets. One key component of the server concerning this demonstration is the Damia component. Damia is responsible for compiling and executing data mashups. It exposes a collection of set-oriented operators for filtering, joining, grouping, transforming, sorting, and otherwise manipulating a generic feed data model [3]. Damia also provides a set of functions and expressions for performing fine-grained intra-operator manipulations of string, date, numeric, and other data types.

SystemT: The second major component of our architecture is the SystemT Development Environment, which supports the iterative process of constructing and refining rules for information extraction. The rules are specified in a declarative language called AQL (Annotation Query Language) [7]. The Development Environment provides facilities for visualizing the results of executing rules over a corpus of representative documents. Once a developer is satisfied with the results that her rules produce on these documents, she can *publish* her annotator. Publishing an annotator is a two-step process. First, the AQL rules are fed into the *Optimizer* [5], which uses cost-based optimization to compile

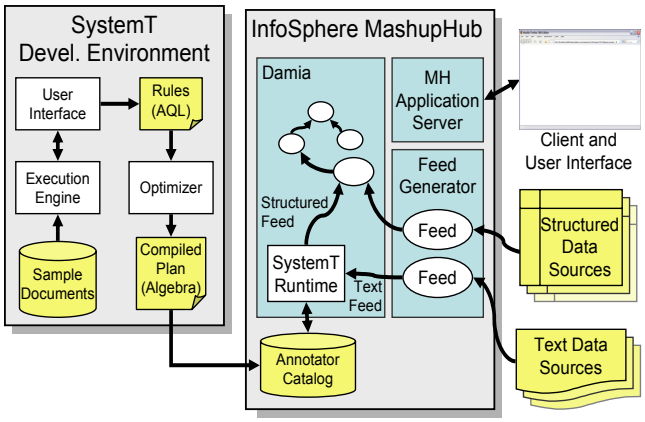

Figure 1: System architecture for our demo.

them into a graph of algebra operator. Then the *Runtime Environment* instantiates the corresponding physical operators.

Integration of MashupHub and SystemT: A new SystemT runtime component has been added to the MashupHub server. It is responsible for managing and executing a collection of compiled SystemT annotators. Two primary APIs are exposed to other server components: an API for annotating feed text, used by the Damia engine; and an administrative API for uploading new annotators, used by the MashupHub client.

3. DEMONSTRATION SCENARIO

Our demonstration focuses on Human Resources Associates (HRA), an imaginary Silicon Valley recruiting firm. HRA maintains a database of top technical talent in Silicon Valley. The firm monitors feeds of information about job openings in the area and matches candidates with openings. When a company hires one of HRA's candidates, HRA receives a commission.

For the purposes of our demo, we focus on one source of job postings: the online bulletin board site Craigslist.com. Craigslist is a popular forum for recruiting among startup companies and government agencies in Silicon Valley. Like most other sources of information about job openings, Craigslist's RSS feed consists entirely of unstructured text.

Currently, HRA's employees need to read every posting on Craigslist by hand, manually querying the firm's database to find potential candidates. This process is expensive and time-consuming, and there is considerable competition among recruiters to be the first to reach the top candidates for a position.

In our demonstration, we will use SystemT and MashupHub to build a "recruiter's dashboard" that speeds the process of matching job postings with candidates. The core of this dashboard application is a mashup that joins the Craigslist.com RSS feed with HRA's database of job candidates. Our demo will use a suite of domain-specific annotators developed with SystemT to extract relevant information from the feed, such as the area of expertise required and the company's size and location. We will show the process of building these annotators in the SystemT Development Environment, compiling the annotators, and loading them into the MashupHub environment, as illustrated in Figure 2.

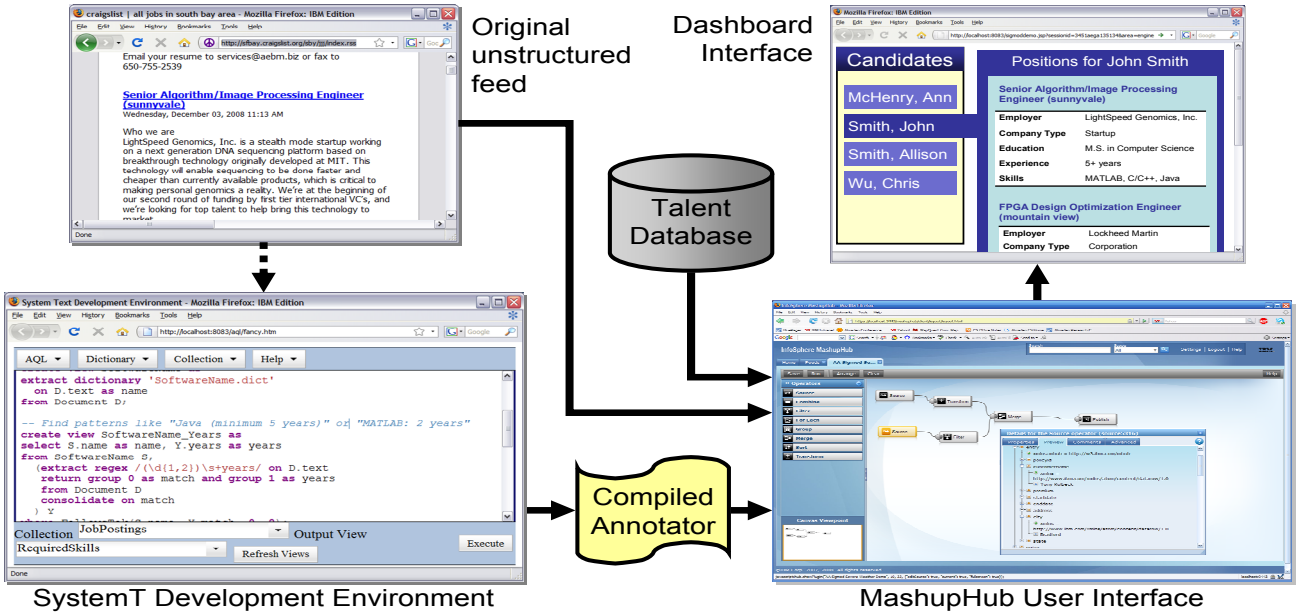

Figure 2: Illustration of our demonstration scenario

The overall application will be constructed as a graph of operators in MashupHub. First, the mashup will poll the RSS feed and invoke the SystemT Runtime to extract information from each entry in the feed. Then the mashup will join this structured information with HRA's database of candidates, finding the candidates that best match each open position. Finally, the system will format the structured information into a report that shows which candidates most closely match each position and provides a summary of the important information about each job posting.

3.1 Demonstration Outline

Our demonstration is comprised of two parts.

In the first part of the demo, we will give a high-level overview of the demonstration scenario and show how the combination of SystemT's information extraction and MashupHub's information integration can solve a realistic business problem.

We will start by showing the unstructured Craigslist RSS feed and the HRA candidate database. After explaining the difficulties of using such a mixture of structured and unstructured data, we will explain how we can use SystemT and MashupHub to construct a dashboard application that combines these two sources of business data.

We will then open up a pre-built MashupHub mashup and explain the different stages of the mashup: Extracting information from the RSS feed, joining this information with the database, and then formatting the result of this join into a useful dashboard. Finally, we will show the dashboard application in action, displaying potential pairings of job openings with candidates and summarizing the important facts about each job opening.

The second part of the demo will delve into the low-level process of constructing domain-specific annotators with SystemT and incorporating the resulting structured information into a mashup.

We will use the SystemT Development Environment to build an annotator that extracts additional types of information from the Craigslist feed, going beyond the types that the first part of the demo extracted. We will show how SystemT and its declarative rule language AQL help facilitate the process of building rules, testing them on a corpus of example feed entries, and refining the rules to make them more accurate. We will then compile the resulting annotator, import the new annotator into MashupHub, and extend the mashup from the first part of the demo to use the additional extracted information. Finally, we will show how using the additional structured information improves the accuracy of the mapping from candidates to jobs and enriches the summary view of job postings.

4. REFERENCES

[1] A. Jhingran, "Enterprise Information Mashups: Integrating Information, Simply", VLDB 2006: 3-4.

[2] IBM Infosphere MashupHub, http://www-01.ibm.com/software/data/info20/how-it-works.html

[3] Simmen, D., Altinel, M., Markl, V., Padmanaban S., Singh, A. Damia: Data Mashups for Intranet Applications. *Sigmod 2008*

[4] Calais, http://www.opencalais.com

[5] Reiss, F., Raghavan, S., Krishnamurthy, R.., Zhu, H.,Vaithyanathan, S.: An Algebraic Approach to Rule-Based Information Extraction. ICDE 2008

[6] SystemT, http://www.alphaworks.ibm.com/tech/systemt/

[7] R. Krishnamurthy et al., "SystemT: A System for Declarative Information Extraction", to appear, SIGMOD Record.