Damia – A Data Mashup Fabric for Intranet Applications

Mehmet Altinel, Paul Brown, Susan Cline, Rajesh Kartha, Eric Louie, Volker Markl, Louis Mau, Yip-Hing Ng, David Simmen, Ashutosh Singh

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA

{maltinel, pbrown1, clines, kartha, ewlouie, marklv, louismau, yipng, simmen, asingh}@us.ibm.com

ABSTRACT

Damia is a lightweight enterprise data integration service where line of business users can create and catalog high value data feeds for consumption by situational applications. Damia is inspired by the Web 2.0 mashup phenomenon. It consists of (1) a browser-based user-interface that allows for the specification of data mashups as data flow graphs using a set of operators, (2) a server with an execution engine, as well as (3) APIs for searching, debugging, executing and managing mashups. Damia offers a framework and functionality for dynamic entity resolution, streaming and other higher value features particularly important in the enterprise domain. Damia is currently in perpetual beta in the IBM Intranet.

In this demonstration, we showcase the creation and execution of several enterprise data mashups, thereby illustrating the architecture and features of the overall Damia system.

1. INTRODUCTION

A mashup is a web application that combines content from two or more applications to create a new application [9]. Situational applications are enterprise web applications built on-the-fly to solve a specific business problem [1]. They are often developed without involvement of the IT department and operate outside of its control. They combine data from a variety of enterprise sources such as SAP or Office applications, back-end databases, and content management systems. Any distinction between mashups and situational applications will become progressively blurred as situational applications augment enterprise data with data outside the firewall. In effect, situational applications are *enterprise mashups*.

Enterprise mashups present a data problem, as they can access, filter, join, and aggregate data from multiple sources; however, these data machinations are typically done in the application mixed with business and presentation logic. In Damia, we aim to develop an enterprise-oriented data mashup platform on which such applications can be built quickly, by enabling a clean separation between the data machination logic and the business logic. In this demonstration proposal, we present the key aspects of the Damia mashup platform which facilitates the specification and execution of enterprise data mashups in IBM's corporate intranet.

VLDB '07, September 23-28, 2007, Vienna, Austria.

To our knowledge, the only other similar service is Yahoo Pipes [12]. Pipes allows for the specification of a data flow graph to combine data feeds, which can be RSS or Atom or RDF. Pipes focuses on merging feeds or on enhancing existing feeds by transforming them via webservice calls (e.g., language translation, location extraction).

Damia goes beyond Yahoo Pipes in several ways: (1) Damia has a principled data model of tuples of sequences of XML, which is more general than Yahoo Pipes. (2) Damia's focus on enterprise data allows for ingestion of a larger set of data sources such as Notes, Excel, XML, as well as data from emerging information marketplaces like StrikeIron [10]. (3) Damia's data model allows for generic joins of web data sources.

2. The Damia System

This section provides an overview of Damia system by briefly describing main components, which are depicted in Figure 1.

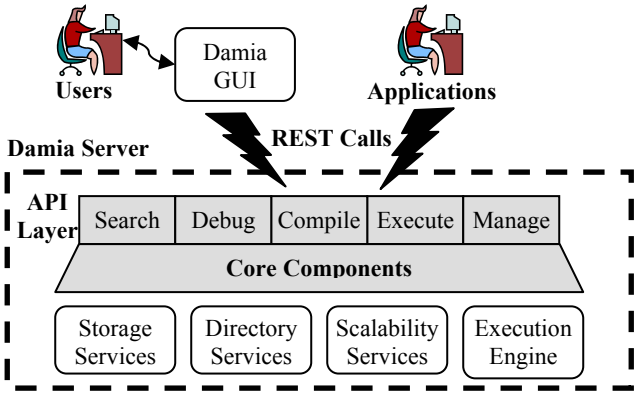

Figure 1: Architecture of the Damia server

2.1 User interface

Situational applications are typically created by departmental users with little programming knowledge; consequently visualization of data mashup operations is critical for any mashup platform. Considering this key aspect, we developed a browser-based user interface that allows Damia users to perform major operations easily and intuitively. The GUI provides facilities for composing new data mashups, searching data sources or existing mashups, and managing stored mashups. The mashup composition follows programming-by-example model which makes the development process more natural and less error prone.

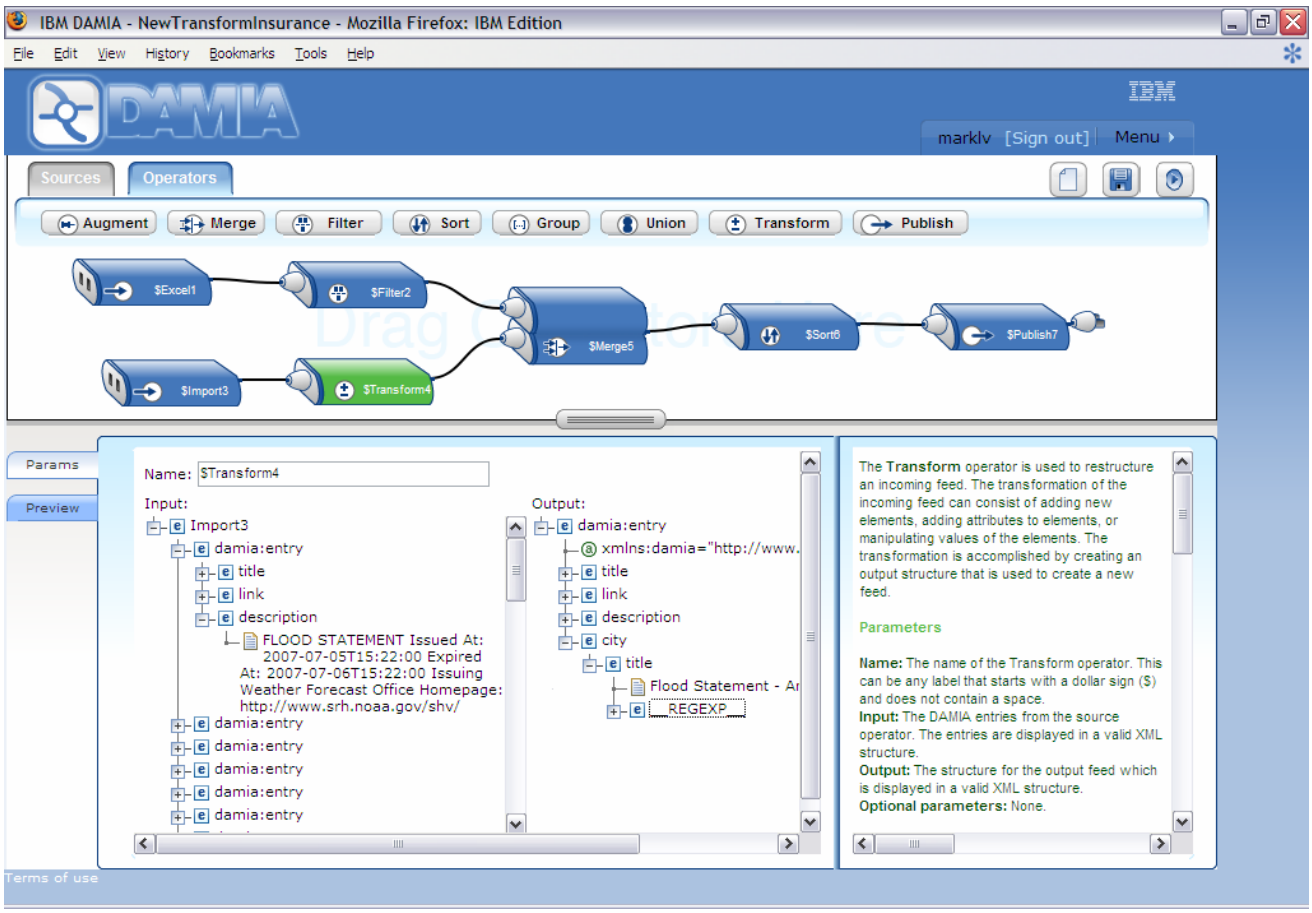

Figure 2: The Damia Mashup Editor

Figure 2 shows a snapshot of the Damia mashup editor, which was implemented with the dojo toolkit [4]. It communicates with the server through a set of REST API interfaces, as illustrated in Figure 1. The GUI allows users to drag and drop boxes, representing Damia operators, onto a canvas and to connect them with edges representing the flow of data between those operators. Users can use a preview feature to see the result of the data flow at any point in the process. The Damia editor interacts with a meta-data repository in order to suggest options for creating "touch points" for joins, grouping, and other operations that require that data be first put in a standard representation. Once the data flow is completed, the result is serialized as an XML document and delivered to the server for further processing.

2.2 Execution Engine

Figure 3 provides a high-level view of Damia execution engine. The Damia engine contains four layers: (1) At the core, Damia relies on PHP environment [8] as its runtime. PHP was an attractive choice for our runtime engine as it has abundant features for accessing web data and for processing XML data. (2) Primitive Damia operators are implemented as PHP classes. They can be wired to each other to create a data flow graph. The primitives provide the expressive power to map most of XQuery to Damia. (4) Feed-Friendly Operators, which are implemented using the primitive operators, provide an XML feed abstraction. An XML feed is as an XML document with a repeating element. XML feeds generalize Atom and RSS feeds, an Atom feed for instance is an XML feed with <entry> as repeating element. Operators can import, filter, transform, augment, sort, and merge XML feeds. (4) Higher-value features are implemented as extensions modules within the engine. These features will be discussed in more detail in section 4.

Damia Operators and Data Model

The Damia operators produce and consume tuples of sequences. A sequence is an ordered set of items wherein an item can be an atomic value or an XML node. The Damia data model provides a closed data model which allows new data mashups to be composed with existing ones. The data model is simpler analog to the one used in XQuery (for example, we do not support node identity). On the other hand, Damia goes beyond most XQuery implementations by supporting sophisticated operators such as group by, probabilistic joins, annotation, etc.

In general, there are three classes of Damia operators: ingestion, augmentation, and publication. Ingestion operators are sources in the Damia data flow graph, in that they bring data feeds into the system. Damia comes with ingestion operators for REST and SOAP Web Services, Excel Spreadsheets, Lotus Notes databases, as well as screen scraping for HTML pages. The overall system of ingestion operators is extensible, as any data source can be ingested into Damia by providing simply a SOAP or REST wrapper. In the terms of the Damia data model, an ingestion operator takes a data object and translates it into a tuple of sequences of XML data.

Augmentation operators are internal nodes of the data flow graph. Damia is extensible in that user defined operators can be written in PHP and can be plugged into the engine or can be made

available as web services. Damia provides operators to extract information from sequences (Extract), to filter tuples (Filter), to iterate over items in a sequence (Iterate), to construct a new sequence from other sequences (Construct), as well as operators to join (Fuse), sort (Sort), aggregate (Group), and perform other sophisticated operations over the sequence data.

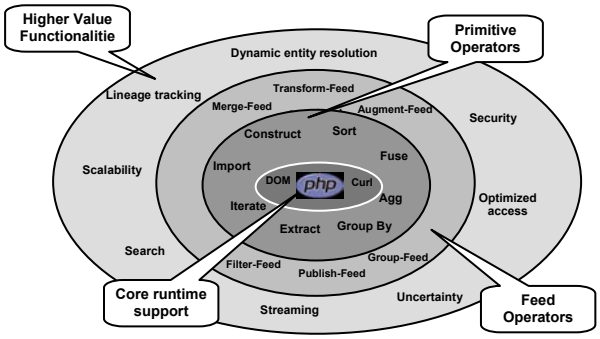

Figure 3: Damia Features

Publication operators are the sinks of the data flow graph. They transform the result of the mashup into common output formats like JSON, HTML, and various XML formats (e.g. Atom, RSS). Damia publication operators can be extended to produce other formats.

2.3 Storage Services

This component handles the storage and retrieval of data feeds created by the Damia community. In addition to publishing data feeds created via Damia data flows, a user can publish resources like Excel Spreadsheets or XML documents to the Damia system, to make them consumable by mashups. It is possible to share published resources with others or to keep them private.

The Damia mashup repository stores both the graphical mashup specification serialized as XML as well as the PHP code that the mashup specification was compiled into. Users execute stored mashups by calling a REST URL provided by the Damia system.

2.4 Directory Services

The Damia system manages metadata for resources and mashups. One aspect of the directory services is to provide basic search functionality on mashup names, descriptions, and creators. However, we envision Damia to be an ecosystem with a community of users growing around it. Damia therefore also provides features for community tagging and rating of mashups.

2.5 Scalability Services

The Damia system is currently implemented over a LAMP stack. There is a wide variety of libraries available in the LAMP ecosystem to offer scalability solutions. We exploited several PHP Pear packages [7] for caching and indexing of mashup resources. Mashup resources are cached in two ways: (1) when they are accessed during execution, (2) periodically the system pro-actively crawls specified sources in anticipation of future requests. The system keeps track of resource usage statistics to improve cache performance.

Damia employs a local XML database to store and index cached resources. Currently DB2/XML [3] is used for this purpose.

3. DEMONSTRATION OVERVIEW

We plan to show unique features of the Damia platform through a demonstration scenarios that combines live data from an RSS feed with a data from an Excel spreadsheet.

In this scenario, an insurance agent wants to know which of his home-owner insurance customers are at risk because of a storm.

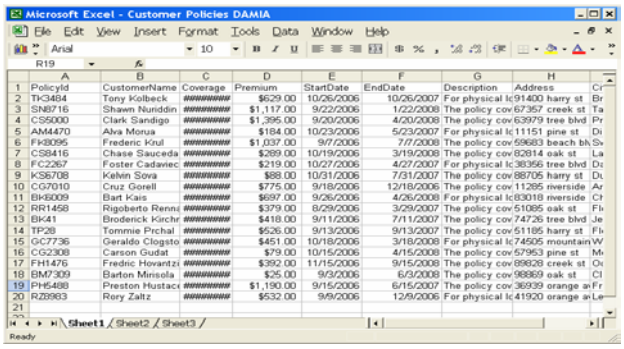

Figure 4: Spreadsheet with policy holders

The insurance agent ingests a spreadsheet (Figure 4) with his insurance policies into Damia, and merges it with a severe weather alert RSS feed from the National Weather Service. In this way, the insurance agent will see the insurance policies of all people affected by a severe weather warning. In the mashup, the weather RSS feed needs to be transformed first to extract city names. In addition, the weather alert RSS feed needs to be filtered for cities in Texas. The results are then sorted by city before being returned as an atom feed which then is displayed in a feed reader or application wiki. In this demonstration, we will show how the mashup is incrementally specified in the GUI using Damia operators (see Figure 2). We will also illustrate the GUI preview function facilitating debugging of a mashup during its specification. We will show how this mashup can be specified as a streaming mashup listening to the severe weather feed, so that automatic e-mail notifications are sent to the agent whenever one of his policies is affected by a severe weather event. Finally, we will run this mashup live and show the resulting policies at risk.

4. Further Research Challenges

This section summarizes advanced features that we are currently exploring as enhancements to Damia:

Ingestion of "Enterprise Data": Mashup applications available on the Web today commonly focus on the consumption of URL addressable resources. However, enterprise data sources are typically exposed in many other forms such as office documents, email, pure databases, etc. So, enterprise mashup platforms must provide enterprise-oriented tools and mechanisms to tap into non-URL addressable sources, and make them available for the developers. In Damia, we developed a set of predefined wrappers for this purpose. Furthermore, we consider utilizing existing page-scraping tools (e.g. Kapow [5] or Lixto [6]) to turn web pages (inside and outside of the firewall) into ready-to-use data sources in the Damia server.

Another unique characteristic of creating enterprise mashups is to cope with a multitude of authentication models used to access enterprise data sources. Although we utilize an LDAP-based single sign-on system in the current prototype, this is actually an open research issue, and we are investigating new techniques to embrace this heterogeneity.

Dynamic Entity Resolution: By definition, the quality of mashup applications is directly related to how they can help to bring data together from multiple sources. Therefore, the level of sophistication in matching different entities is a key distinguishing feature for any data mashup platform. This standardization problem is usually addressed in two dimensions: (1) identification of semantically known entities, and (2) resolving differences in representations. We observe that more and more standardization services are becoming available on the Internet for most commonly used data types[1]. A similar trend is also noticeable within enterprise systems with widespread adaptation of master data management products. This observation is taken into account in the design of dynamic entity resolution module, thereby enabling to exploit growing number of standardization services with little effort.

During mashup design, the Damia system displays applicable standardization services to users and expects them to choose the right ones for the mashup[2]. At runtime, the system can find out possible touch points between entities and can generate the right set of transformation functions to perform the join. For the cases where the Damia server cannot detect the entities, there are facilities for users to introduce the entities into the system and select/register suitable standardization services for entity matching. Our flexible design allows utilizing existing tools and platforms (e.g. ClearForest [2], UIMA [11]) for the detection of known entities in input sources. Once the entity matching task is completed, its steps are remembered to help future users when they try to use same sources in their mashups. Such a "folksonomy-based" approach is aimed at taking the advantage of collective power of Damia community to deliver a promising framework for large-scale data integration.

Streaming*:* RSS and Atom feeds are a norm on the Internet. Huge amount of information today is available via feed interfaces. Taking this fact into account, we designed the Damia system to understand and consume feeds effortlessly. Feeds are inherently "push-based", i.e. their content is dynamically updated at the source without any notification. To be able to cope with this streaming aspect, the Damia system includes mechanisms to detect and process the changes in the feeds, and to deliver notifications to its applications. In enterprise domains, there are many useful mashup scenarios where this feature is extremely valuable, particularly for reporting and dashboarding applications.

Search: Mashup applications are typically created in large numbers since they tightly focus on a specific situation concerning a small group of users. Hence, a common problem is how to find a right mashup application or a data source in mashup corpus when they are needed. An effective search mechanism for mashups requires the understanding of not only the properties of input sources but also how they are processed in data flow. We performed an initial investigation on this problem, and developed a preliminary search mechanism. We believe this is an important research area, and are working on enhancing the current solution.

Lineage: Mashups on the Web usually do not provide any metrics on data quality as this issue is not considered central. However, this is not always true for enterprise mashups applications, as sometimes important business decisions may be made based on their outcome. In such cases, when an enterprise data mashup is composed with other mashups, it becomes very critical to be able to provide lineage information for its users to assess its data quality. Again, this is an active research area, and we only implemented basic mechanisms to address this issue.

Uncertainty: It is not always the case that all operations return exact results in mashups. There may be situations where uncertain results are inevitable: (1) Data sources may return probabilistic (or ranked) data. Typical example is when a search engine result is fed into the mashup. (2) Entity matching may not yield exact results in some cases. Hence, the result of joins may become probabilistic. When an uncertainty is introduced in the system, it has to be understood and modeled in the data flow. Like the lineage problem, this aspect is mostly ignored in Web mashups, but it may be crucial in some class of enterprise mashups. We implemented a set of operators (join, sort, aggregate, etc.) which can deal with probabilistic aspects in the data flow. We are actively exploring new probabilistic models, improving the existing operators and adding new ones.

[1] Geocoding service for address types is the prime example. Many Internet companies including Yahoo and Google provide this service.

[2] Programming-by-example enables us to perform this task.

5. CONCLUSIONS

Proliferation Web 2.0 technologies and ever increasing number of available Web data services gave rise to phenomenal growth of mashup applications on the Internet. We anticipate that enterprise systems will greatly benefit from this new trend by means of enterprise-oriented mashup development platforms. We are laying the foundations of such a platform in the Damia project.

In this paper, we presented our initial Damia prototype, which includes a mashup-oriented data flow engine enriched with novel, enterprise-specific advanced functionalities. Most notable features include a folksonomy-based standardization service, a intuitive GUI front-end, a hosted, scalable mashup server architecture, unique ingestion facilities and utilities for mashup search and management. Going forward, we aim to use this prototype as a playground to explore many interesting research directions outlined in this paper.

6. ACKNOWLEDGMENTS

We thank Garret Hourihan, Ken Coar and Kathy Saunders for helping in building the system. We also thank Anant Jhingran, Hamid Pirahesh, and many colleagues in IBM for inspirational discussions. We furthermore thank all IBMers who have used and continue to use Damia to build situational applications.

7. REFERENCES

[1] A. Jhingran, "Enterprise Information Mashups: Integrating Information, Simply", VLDB 2006: 3-4.
[2] Clearforest Inc., http://www.clearforest.com/
[3] DB2 pureXML™ technology, http://www-306.ibm.com/software/data/db2/xml/
[4] Dojo, the Javascript toolkit, http://dojotoolkit.org/
[5] Kapow Technologies, http://www.kapowtech.com/
[6] Lixto Software Gmbh, http://www.lixto.com/
[7] PEAR - PHP Extension and Application Repository, http://pear.php.net/
[8] PHP: Hypertext Preprocessor, http://www.php.net/
[9] Programmable Web, http://www.programmableweb.com/
[10] Strikeiron Inc., http://www.strikeiron.com/
[11] Unstructured Information Management Architecture (UIMA), IBM Research, www.research.ibm.com/UIMA/.
[12] Yahoo Pipes, http://pipes.yahoo.com/pipes/