

TEXT ANALYTICS AND DATA ACCESS AS SERVICES

A case study in transforming a legacy client-server text analytics workbench and framework to SOA

E. Michael Maximilien, Ying Chen, Ana Lelescu, James Rhodes, Jeffrey Kreulen, and Scott Spangler

IBM Almaden Research Center

650 Harry Road, San Jose, CA, 95120, USA

{maxim, yingchen, lelescu, jjrhodes}@us.ibm.com, {kreulen, spangles}@almaden.ibm.com

Keywords: SOA, Web services, Text Analytics, Text Mining, Software Engineering, Case Study

Abstract: As business information is made available via the intranet and Internet, there is a growing need to quickly analyze the resulting mountain of information to infer business insights. For instance, analyzing a company's patent database against another's to find the patents that are cross-licensable. IBM Research's Business Insight Workbench (BIW) is a text mining and analytics tool that allows end-users to explore, understand, and analyze business information in order to come up with such insight. However, the first incarnation of BIW used a thick-client architecture with a database back-end. While very successful, the architecture caused limitations in the tool's flexibility, scalability, and deployment. In this paper we discuss our initial experiences in converting BIW into a modern Service-Oriented Architecture. We also provide some insights into our design choices and also outline some lessons learned.

1 INTRODUCTION

The Web has transitioned into a collaborative platform for casual end users and increasingly for businesses and organizations. Examples of this shift to human collaborations are the wide and rapid popularity of social networking platforms such as MySpace.com (Hempel and Lehman, 2005), various socially-centered news Web sites such as slashdot.org and digg.com, collaborative knowledge platforms such as Wikipedia, and the myriad of individual blogs currently available on wide ranging topics¹.

Businesses and organizations have also experienced this shift toward more collaboration using the Web. They are increasingly making their employees participate in the collaborative Web and engage their customers using the same platform. For the more internet-centric companies, the results of their customers' collaborations become a key asset to their business. For example, eBay's reputation system (which is entirely customer-driven) is arguably eBay's

most important asset.

One downside of this far-reaching, free-form, human collaboration and participation, is that information is scattered in many different databases. Further, the information is at time highly unstructured which makes gathering knowledge and insights from it an increasingly difficult problem. For example, in a large organization, how can one identify the people with the skills and knowledge on particular technologies, tools, or techniques? In other words, how do you identify experts when the information about people, their expertise, and passions, is increasingly in the forms of wikis, blogs, and other social collaborative tools and platforms?

IBM's Business Insight Workbench (BIW) is a tool designed to address this problem and help answer such questions. Using various mining techniques, data ware-housing capabilities, and some proprietary algorithms, the BIW tool has been widely deployed to help gain insights into vast amount of unstructured and structured information (Cody et al., 2002; Spangler et al., 2003). Using BIW a client gathers and explores the data sets in question. Using searches and operations, the user analyzes the results, which leads to an understanding and then some insights. For ex-

¹As of 15 July 2006, Technorati.com listed more than 80 top-level blogging categories with more than 50,000 new blogs listed per-day and tracking more than 50 million blogs in tota.

ample, using the BIW tool on a company's internal blogs and project databases we can identify employees with certain expertise, e.g., Java™, UML, and so on. Such information is useful when forming new teams for new projects, as well as for cross-fertilizing and balancing the workforce.

While the tool has been widely deployed and successful, it has some important drawbacks. In particular the workbench is comprehensive, which also makes it difficult to use and to integrate into a company's business processes. Second, since the workbench is essentially a thick-client application, it requires much of the resources from the machine where it is running, making scalable deployment problematic.

In this paper we present the architecture and lessons learned from our experiences in transforming the BIW tool from a thick-stand-alone client into a modularized, Web application-centric, and domain-specific, service-oriented architecture (SOA). The new platform is a comprehensive information service platform named Business Information Services on a Network or BISON.

1.1 Organization

The rest of this paper is organized as follows. In the next section we discuss additional motivation for the BISON project; in particular we take a look at some of the key use cases that have driven our approach. Section 3 lists some related work in the field of both Service-Oriented Computing and information data mining. Section 4 contrasts the previous architecture with the new modularized, service-centric architecture. Section 5 shows a demonstration of one AJAX-based Web application solution built using the BISON services. Section 6 highlights the lessons learned in moving to our new architecture (both technical and business lessons). Section 7 concludes with a discussion of future work and directions for additional research.

2 USE CASES AND REQUIREMENTS

Generally the BISON project provides a services framework to enable a domain expert to gain insights into huge amount of unstructured information. Since the research space is vast, we give two use cases (which we have implemented) that cover two important domains and problem classes that are well suited for the BISON technology.

2.1 Root-Cause Analysis Use Case

Root Cause Analysis is a particular analysis scenario using BISON capabilities applied to a data set of problem tickets for some product or set of products. In this scenario the user first begins with a query that captures (for the most part) the kind of problem that they wish to find the root cause of. This is assumed to return a set of problem tickets that match the associated query.

Next the problem tickets are categorized using standard text clustering techniques (Cody et al., 2002; Spangler et al., 2003). This then produces a categorization which the user can browse and edit until it reflects the users view of how the query result should be partitioned. Next the user selects those categories that best match the original problem, discarding those categories that are spurious or irrelevant matches.

The selected problems are then compared to a randomly selected background population of the documents to detect any statistically significant differences in either the structured or unstructured data fields. Any such correlations are brought to the user's attention along with 'typical' examples which should help to indicate the 'root cause' of the original problem.

2.2 Expertise Locator Use Case

Fast and dynamic team building capability is critical to the success of a business consulting service organization. To acquire such ability, one key component is to be able to quickly identify appropriate consulting team with appropriate expertise. Most business consulting organizations maintain significant amount of information in the form of texts in knowledge bases which contain information on which consultants might have been working on different types of projects. Such information, if mined properly, can give insights into the expertise of the consultants.

We list the high-level steps for identifying people's expertise using the BISON tools:

1. Search for documents containing expertise keywords (e.g., 'six sigma')
2. Automatically generate an expertise taxonomy on the query result set.
3. If there are meaningful structured fields such as the authors of the documents that can be used to create experts and expertise linkage, then:
 - (a) classify the result set by using such structured fields;
 - (b) otherwise, a name-annotator is run to extract names out of the documents;

- (c) the extracted names are considered as a structured field; and
 - (d) the documents are then classified using the structured name field.
4. Use co-occurrence analysis (Cody et al., 2002) to generate a co-table that compares the refined document taxonomy and the classification of names or authors.
 5. Find significantly related document categories to people.
 6. Plot the relationship using network graph analysis to see who are related to which document categories; hence indicating a certain level of expertise.
 7. Identify people with similar expertise by examining highly related people names using network graph analysis.

2.3 Requirements

In addition to enable the creation of rich text analytics Web applications for various domains, such as the ones listed above, we also wanted to address various shortcomings with the current architecture. In particular we wanted the BISON services framework to facilitate the following requirements:

1. **Scalability.** This implies scalability in all parts of the systems. Since the BISON users can be varied and numbered, it is important that the system be able to scale to a great number of simultaneous users. While much of the BISON core components are involved in compute and data intensive tasks, it is critical that individual user interface components perform within Web application acceptable responsiveness time.
2. **Fine- and coarse-grained services.** The BIW tool divided its operations into three main categories: explore, understand, and analyze. While such a high-level categorization works for communicating with experts and users of the BIW tool, as we transform the tool to SOA, it became apparent as we brain-stormed on the different services to expose, that some services would be composed of simpler more fine-grained ones. Therefore, a clear goal was to provide different levels of services which could be combined and integrated into business processes and provide transparent value, as well as enable the creation of end-user Web applications (as discussed above).
3. **Achieve reusable components.** While the previous tool was successfully applied in wide-ranging domains, it was clear to us that there are many

common components across the different solutions. Therefore, another important requirement is to make sure to modularize the components of different solutions from the start, thereby encouraging reuse and sharing. This includes the views, controllers (actions), and data models for each solution.

4. **Flexible data sources.** Since a primary goal of BISON is to enable the gathering and discovery of insights from corporate and public data repositories, we wanted to be very flexible in the types of data sources we support. This means that our new architecture should have provisions to ingest, equally well, text and XML data files, as well as relational databases.
5. **Flexible deployment.** Since a BISON solution Web application can either be hosted or deployed at the customer's site. It's important that our new BISON framework allows for flexible and easy component deployments.
6. **Reuse of previous code.** An important criterion driving our architectural, design, and implementation decisions is the goal of reusing and leveraging the good aspects of the previously successful BIW tool and framework.

3 RELATED WORKS

We could not find much previous works in the literature dealing with comprehensive SOA tool or framework for business insights such as ours. However, our work builds on various standard SOA ideas and data mining techniques (Agrawal, 1999). There have also been various efforts in exposing data and knowledge as services, which at their core, are similar to our overall goals with BISON.

We divide the related works into: (1) database, data mining, and information as services; and (2) general approaches to architect and design service-based systems and solutions.

3.1 Databases, Data Mining, and Information as Services

(Kumar et al., 2006) outline and motivate the importance of achieving distributed data-mining, as well as the use of standard Internet protocols, and SOA for achieving scalable solutions. Our approach fits quite well into this overall vision. (Cheung et al., 2006) describe an architecture and framework for data mining using the Business Process Execution Language

(BPEL) (Curbera et al., 2002). Our approach currently differs from theirs in that we have a simple workflow mechanisms based on top level tasks (or steps), which themselves aggregate calls to various Web services invocations. However, since we expose various levels of Web services (fine- and coarse-grained) we could also make use languages such as BPEL to help create composed services out of the simpler ones.

(Guedes et al., 2006) propose a SOA-based architecture for distributed data mining called *Anteater*. The *Anteater* architecture, as ours, distribute the functions of data servers and mining servers to different nodes in the network and use Web services as interfaces. The architecture uses a distributed mining algorithms which can operate in parallel in the distributed mining servers.

Salesforce.com's AppExchange platform² as well as Amazon.com's Alexa³ and ECS⁴ Web services are examples of information and database as services. Since these Web services expose information from databases as a series of Web services, there is a relationship to our services and also generally point to the trend of making data available as services on the Web. However, we should note that our approach and services focus on creating general data-mining Web services which can take many data sources and help expose the content of these sources to help create business intelligence solutions.

3.2 Design and Architecture of Service-Based Systems and Solutions

Another set of related works are in the approaches to convert existing tools or frameworks into service-oriented architectures. These include processes, techniques, and methods.

An example of a comprehensive method for architecting SOA-based systems is the Service-Oriented Method Architecture (SOMA) from IBM (Arsanjani, 2005). Our resulting SOA design and architecture decisions did not result from the SOMA approach; instead, we used a more agile technique of creating a cross-section of the system (or a *spike* (Beck and Andres, 2005)) by implementing two comprehensive solutions and abstracting the various services from the different layers that we anticipated and had to create.

Other approaches in building SOA typically take a model-driven approach to the design. That is, they

²<http://www.salesforce.com/appexchange>

³<http://aws.amazon.com/awis>

⁴<http://aws.amazon.com/>

envision the abstract model of the different components and determine the components' remote interfaces, and then they decide which component can be exposed as Web services. Examples of this approach are the Sonic SOA Workbench (Sonic, 2006), Exaltec's b+ J2EE-SOA Application Generator (Exaltec, 2006), and IBM's SOA Solutions workbench.

4 ARCHITECTURE

As mentioned before, the previous BIW architecture was implemented as a thick Swing Java client application connected to a database back-end. While this allowed for a rich user interface, there are many problems with this approach.

1. A thick-client architecture means that all of the text analytics engine and algorithms run on the client. Since these algorithms are compute-intensive the client has to absorb all of the costs (CPU and memory).
2. There is no easy way to integrate parts of the client functionality into some external business process. This is especially useful when the results of the analysis are repeatable and just need to be run on newer data.
3. The current application aggregates all of the capabilities of the BIW framework. This makes the user interface comprehensive but also difficult to use.
4. The current architecture was designed with a single user in mind. There are no facilities to accommodate multiple and simultaneous users, nor are there any means for preventing access to some functions at a user-level.
5. The current architecture does not have facilities for monitoring and for metering user activities. Metering usage is important if we want to eventually use the resulting solutions in a pay-as-you-use business model.

To address the issues listed above and also with the implicit goals of reusing as much of the previous code-base and functionality as possible, we have created a new SOA-based architecture for the new incarnation of BIW or the BISON project.

4.1 Overview of Architecture

Our architecture is principally based on the layered architecture style documented in (Bass et al., 1998). A well known advantage of layered architectures is the

decoupling of subsystems. There are also other well-known characteristics; for instance, each horizontal layer:

1. depends on layers below it;
2. exposes its own set of service interfaces (local or remote);
3. can be developed separately; and
4. can be deployed independently.

Additionally, one can add vertical layers for common components that span across layers. For our case, we decided to divide our architecture into five horizontal layers and one common vertical layer. Figure 1 illustrates our architecture.

Each horizontal layer can be independently deployed and comprise of a remote and local service API. The APIs are identical, except that the remote API is a Web service with exposed WSDL ⁵. At deployment time, clients of each layer (which can also be another layer) configures their use of the API to bind locally or remotely. This allows clients to be flexibly deployed and take advantage of a faster local API if it is deployed on the same node on the network. Our vertical layer takes care of administration, metering, security, logging, and other cross-cutting concerns. In the following sections we discuss each layer in details.

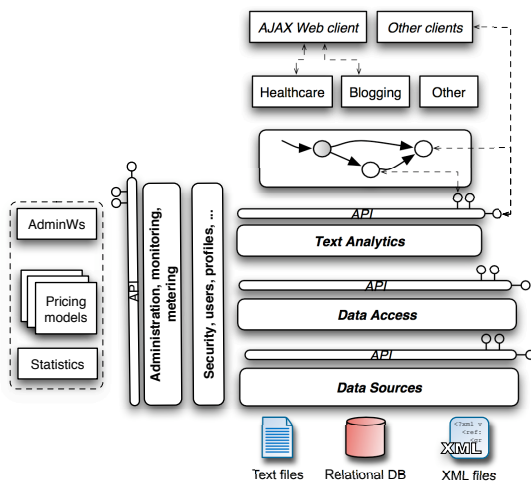


Figure 1: BISON layered SOA.

4.2 Common Vertical Layer

As illustrated in Figure 1, our layered architecture comprises of independently deployable horizontal layers and one vertical layer. The vertical layer is

⁵<http://www.w3.org/TR/wsd1>

meant for common components. In particular our vertical layer comprises the following key features:

1. *Administration*. These include user management functions, including authorization, authentication, user profiles, and general user management facilities, e.g., add and remove users.
2. *Security*. Designed to manage secure access to data sources and text mining and analysis objects. For instance, any intermediary and final objects are associated with a user and can only be accessed by that user.
3. *Metering*. Keeps information about user accesses to various parts of the system. The metering information is important for costing; for instance, for keeping track of service accesses and to appropriately charging users.
4. *Statistics*. Aggregates metering information into useful statistics needed for the cost models, e.g., average service method access for any service, average query length and results, as well as many other statistics.

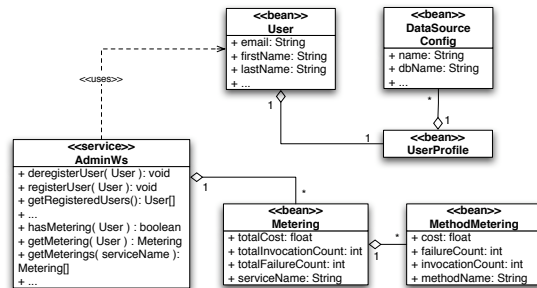


Figure 2: BISON Metering component design.

Figure 2 shows a high-level overview of the metering and user information components. The *AdminWs* is the main Web service exposing administrative access and functions to the horizontal layers. This Web service returns metering information for any Web service on any layer for a user.

4.3 Domain and Client Layers

The domain and client layer contains components and services to ease the creation of domain-specific solutions. In particular we have created a simple workflow-based approach to easily add solutions to our BISON framework. For instance, we have a series of simple steps for common part of any solutions that can then be combined to create full solutions. For example in the use cases discussed in Section 2, the query, taxonomy editing, and correlation analysis

steps are common components across the two different use cases.

To enable the creation of richer client based on AJAX we have created common components to help move data from the AJAX server and the browser and to help cache text analytics data, e.g., categories, taxonomy, co-occurrence data, and so on. This common AJAX caching scheme allows client code to focus on how to best display data to users and elicit user input rather than managing communications to the Web services and caching of data. Section 5 is an example of a complete solution built using the domain and client layers.

4.4 Text Analytics Services Layer

The text analytics services (TAS) layer exposes services to enable the creation of client and domain text mining and analytics steps. For instance, co-occurrence table analysis assumes the creation of a taxonomy on a datastore by running a particular clustering analysis algorithm. All of these functions are provided as part of TAS and are exposed via independent WSDL. Each operation for each service takes a User object to allow for security and metering.

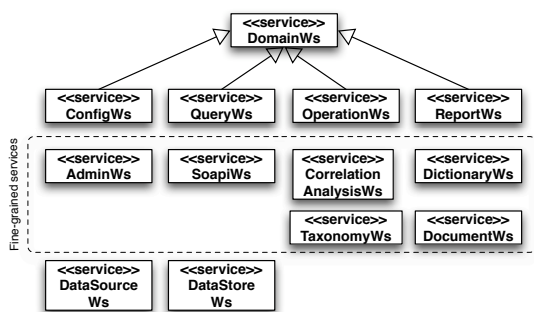


Figure 3: Text Analytics Services (TAS) architecture.

The TAS can be divided into two primary groups of services. Figure 3 illustrates this decomposition. First, a series of fine-grained low-level text mining and analysis services; for instance, we have services such as:

1. *TaxonomyWs* exposes operations to create, access attributes, and modify (edit) taxonomies. A taxonomy is used by all TAS services so this service is key to using TAS.
2. *CorrelationAnalysisWs* exposes operations to perform correlation analysis and access co-occurrence data.
3. *DictionaryWs* and *DocumentWs* enable adding custom dictionary terms to a corpus of data as well as access the documents of that corpus

4. *SoapWs* is a generic service that gives access to other parts of the TAS which do not fit in the other services.

The second set of services exposed by the TAS composes the fine-grained services. These coarse-grained services are used directly in the creation of the steps for the client and domain layers. Examples services are *ConfigWs*, *QueryWs*, and *ReportWs*, which are used to respectively configure data sources, to query datastore, and to generate reports.

4.5 Data Access/Sources Services Layers

The final layer in the BISON stack is the one dealing with extracting, loading, and transforming (ETL) data sources into a form that can be used for text analytics. The Data Source Services and Data Access Services (DSS/DAS) operate on either text files or relational databases. The DSS abstracts the data into a common set of operations and query interface. The DAS expose the resulting data sources as a series of Web services, which can be used by TAS to enable the text mining and analysis services describe in Section 4.4.

An important design point of the DSS/DAS layers is to provide a uniform interface to different data sources. In particular they give a generic query interface to both file based data and relational databases and provide secure access to the data. Finally, another aspect of the DAS is the creation of common data warehousing facilities (independent of data sources). That is, after ETL the data sources are loaded into a star-schema in the DSS/DAS to allow different types of OLAP (Codd et al., 1998)⁶ operations and queries to be performed. These OLAP capabilities are important features needed to implement the TAS.

5 DEMONSTRATION

As an initial demonstration of our architecture we now briefly discuss the details of how we implemented the Root Cause Analysis (RCA) solution discussed in Section 2.1.

5.1 Root Cause Analysis Solution

We implemented the RCA solution using the BISON services discussed in Sections 4.4 and 4.5. Figure 4 shows the first screen presented to a user who is configured to use the RCA solution and has decided to

⁶<http://en.wikipedia.org/wiki/OLAP>

start doing so. Each of the steps on the left hand side (also implemented as tabs) associates to some domain or combination of fine-grained service invocations. For instance, in the Query step results in accessing a datastore, creation of an intuitive taxonomy cluster on its documents, and a search for the query phrase over the taxonomy (all using the TAS).

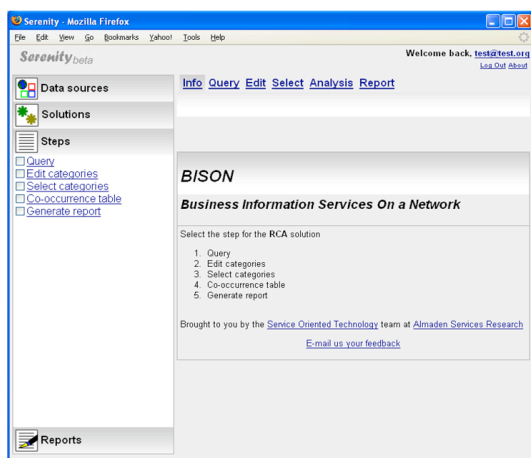


Figure 4: RCA Web application screen shot.

To make the UI more responsive we created an AJAX-based client layer which allows us to invoke some of the text analytics services asynchronously from the user’s browser. Figure 5 shows the taxonomy editing step which showcases the value of this AJAX UI. The categories and documents (which can be very large) are paged to the client for fast access by loading remaining results as needed.

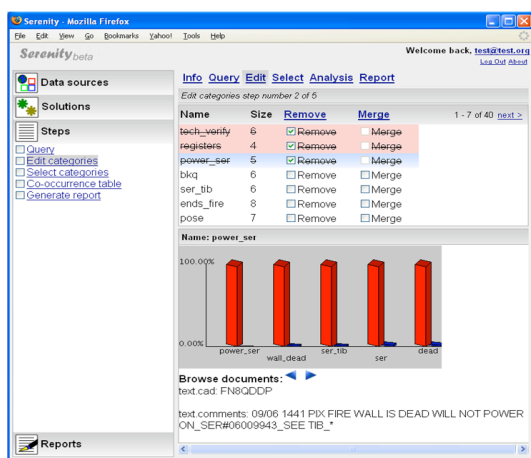


Figure 5: RCA taxonomy editing screen shot.

6 LESSONS LEARNED

We can divide the initial lessons learned from the BISON project into three broad categories: (1) service architecture and design, (2) API design, and (3) dealing with legacy issues. For each category, we give some specifics and generalizations of what we have learned.

As we started the BISON project, we debated a fair amount on how much of the previous code-base we should reuse and how to tackle the move to SOA. We have taken a use-case driven, grass-root, bottoms-up, and iterative approach. Essentially, by taking one important use-case, we have dissected the previous code base and exposed an initial API. This API was then remoted (making necessary adjustments, e.g., to allow serialization of large objects) and then we added the layers on top to build a Web application realizing the use-case. After two or three iterations of the same use case, we saw some patterns in the API and Web applications. We refactored (Fowler et al., 1999) the resulting code and design and evolved the architecture into the layered version illustrated in Figure 1.

Another important lesson we have learned was in deciding how fine- or coarse-grained the APIs should be. We found that this question answered itself as we iterated and implemented more use cases. For instance, after implementing a complete first use-case, we had refactored the code enough that we it became trivial to add new coarser-grained services as they could be built with the primitive, fine-grained services.

SOA is generally seen as a means for breathing new life into legacy systems. However, this can be problematic if the legacy systems have inherent shortcomings or are difficult to abstract as APIs. This was partly our case; nonetheless, instead of throwing away completely our old code-base, by following the iterative approach (discussed above) and abstracted legacy objects into interfaces and using some strategic patterns, such as Factory and Façade (Gamma et al., 1995) we were able to gradually reuse the legacy code while having the capability of fully replacing it in the future. We should also note that the layered architectural-style tremendously helped in that tasks as it gave some module-level components that formed the boundaries for our APIs.

We were able to create a modern, AJAX-based Web application, text analytics solution using our exposed Web services. This initial set of results validates our architecture and also helps validate our evolutionary, use-case-driven architectural approach.

7 FUTURE DIRECTIONS

The primary direction for future research is in validating and refining the architecture with other use cases. Since the original BIW tool was deployed in a wide-ranging set of domains, from health-care and drug discovery to blogs and patent databases, we ideally would like to be able to add one example per domain which would helpfully validate our APIs and services. Such exercises would also allow us to refactor and better structure the different API layers. We would expect more abstractions of common interfaces and also further aggregation and specialization of the domain-specific portions. In the end, we expect to evolve our APIs into common and domain-specific text mining ‘languages’.

Another direction of our research is in improving our pricing models. Currently our metering infrastructure captures some basic usage data. This enabled us to create an initial pricing model to charge customers of BISON on frequency of usage, i.e., frequency of API calls. However, this pricing model is limited, in particular, for most domain applications of text analytics and text mining, the end-user actions are very iterative and repetitive. This means that a model that measures frequency of usage may not accurately reflect the value the end-user got from the services. Instead of simple frequency usage measurements we need to have means for measuring usage patterns as well as meter the results from the service calls.

In addition, coarser-grained pricing models, such as a subscription pay-as-you go or a per-user-session pay-as-you-use model may be more appropriate for many of our use cases. To that end, we have made the pricing model component flexible and pluggable. This will allow us to experiment with various models for different clients and domains.

Finally, another important direction is in the user interfaces and user experience aspects of the entire BISON stack. In particular we want to create an initial set of reusable UI views that we can use to assemble new solutions. Also, since in general, as mentioned before, the process of mining for insights is very iterative and repetitive, it would be beneficial if the end-users are able to collaborate and provide feedback to the tool; for example, enabling the ability to tag particular documents in a result set or the ability to rate documents, entities, and other aspects of a BISON application. By aggregating the feedback we could improve the experience of users of common data sources, e.g., enterprise users mining enterprise data. The BISON engine could also use this aggregated feedback to improve some of the clustering and mining operations.

REFERENCES

- Agrawal, R. (1999). Data Mining: Crossing the Chasm. In *Proceedings of 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99)*, San Diego, CA.
- Arsanjani, A. (2005). Service-Oriented Modeling and Architecture. Technical report, IBM Global Services.
- Bass, L., Clements, P., and Kazman, R. (1998). *Software Architecture in Practice*. Addison-Wesley, Boston, MA.
- Beck, K. and Andres, C. (2005). *eXtreme Programming Explained: Embrace Change, 2nd Edition*. Addison-Wesley, Boston, MA.
- Cheung, W., Zhang, X., Wong, H., Liu, J., Luo, Z., and Tong, F. (2006). Service-Oriented Distributed Data Mining. *IEEE Internet Computing*, 4(10):44–54.
- Codd, E. F., Codd, S. B., and Salley, C. T. (1998). Providing OLAP to User-Analysts: An IT Mandate. Technical report, E.F. Codd Associates.
- Cody, W., Kreulen, J. T., Krishna, V., and Spangler, W. S. (2002). The Integration of Business Intelligence and Knowledge Management. *IBM Systems Journal*, 4(41).
- Curbera, F., Golland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S. (2002). Business Process Execution Language for Web Services, Version 1.0. www-128.ibm.com/developerworks/library/specification/ws-bpel/.
- Exaltec (2006). Exaltec’s b+ J2EE-SOA Application Generator. www.exaltec.com/appgenerator.html.
- Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA.
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.
- Guedes, D., Meira, W. J., and Ferreira, R. (2006). Anteaater: A Service-Oriented Architecture for High-Performance Data Mining. *IEEE Internet Computing*, 4(10):36–43.
- Hempel, J. and Lehman, P. (2005). The MySpace Generation. Technical report, Business Week Online.
- Kumar, A., Kantardzic, M., and Madden, S. (2006). Distributed Data Mining: Frameworks and Implementations. *IEEE Internet Computing*, 4(10):15–18.
- Sonic (2006). Sonic SOA Workbench. www.sonicsoftware.com/products/sonic_workbench.
- Spangler, W. S., Cody, W., Kreulen, J. T., and Krishna, V. (2003). Generating and Browsing Multiple Taxonomies over a Document Collection. *Journal of Management and Information Systems*, 4(19):191–212.