Extensible Data Management in the Middle-Tier

Brian F. Cooper[1,2], Neal Sample[1,2], Michael J. Franklin[1,3], Joshua Olshansky[1] and Moshe Shadmon[1]

[1]RightOrder Inc.
3850 N. First St.
San Jose, CA 95134 USA

[2]Department of Computer Science
Stanford University
Stanford, CA 94305 USA

[3]Computer Science Division
University of California
Berkeley, CA 94720 USA

{cooperb,nsample}@db.stanford.edu, franklin@cs.berkeley.edu, {josho,moshes}@rightorder.com

Abstract

Current data management solutions are largely optimized for intra-enterprise, client-server applications. They depend on predictability, predefined structure, and universal administrative control, and cannot easily cope with change and lack of structure. However, modern e-commerce applications are dynamic, unpredictable, organic, and decentralized, and require adaptability. eXtensible Data Management (XDM) is a new approach that enables rapid development and deployment of networked, data-intensive services by providing semantically-rich, high-performance middle-tier data management, and allows heterogeneous data from different sources to be accessed in a uniform manner. Here, we discuss how middle tier extensible data management can benefit an enterprise, and present technical details and examples from the Index Fabric, an XDM engine we have implemented.

1. Introduction

The multi-tier architecture for distributed enterprise applications is used to provide adaptability, interoperability, and fast time to market for enterprise computing. The benefits are well known, and the multi-tier approach is quickly becoming a standard, especially for e-commerce applications [8,3]. In order to support enterprise applications, this model must be efficient, in terms of both high performance query processing, and efficient development and deployment. Unfortunately, the realization of many of the benefits of the multi-tier architecture is hindered by traditional data management technologies. These technologies suffer from two primary drawbacks: (1) they were designed and optimized for intra-enterprise, client-server applications, with clear, rigid requirements and (2) they are not deployed in a manner consistent with multi-tier, standards-based distributed architectures [1,10]. A new data management architecture is needed.

We propose an approach to this problem, called eXtensible Data Management (XDM). The goal of XDM is to provide high performance management of data that is irregularly structured, or whose structure may change over time. The system must be able to integrate new data into the database, even if the new data has a different schema or structure than the existing data. Users should be able to efficiently access all data, regardless of structure. Moreover, it should be possible over time to go beyond merely "efficient" operations to provide highly optimized access to data along frequently used paths. At the same time, because the data is irregularly structured, the data management system should provide users with assistance in formulating queries. In other words, the data management system should present a self-describing view of the data that can be queried in a robust, flexible way that is resilient to irregularity in the data.

To maximize the benefits of XDM, it is deployed using standardized, modular components in the middle-tier. This will allow the data management system to best meet the needs of applications that are themselves deployed with modular components and distributed using middle-tier application servers. Application designers have been able to utilize application servers that offer clustering, fail-over and flexibility, and we hope to extend the same benefits to XDM.

We have developed an XDM system based on a novel technology, called the Index Fabric. The Index Fabric has several important advantages over existing technology. First, it does not require a pre-existing schema for the data. Instead, the data management system is self-describing, so that the schema can be used in a descriptive manner to aid in formulating queries, rather than in a prescriptive manner to restrict the form of the data. Second, the system is dynamic, supporting the introduction of new data types and relationships. These changes do not require down-time as the system is reconfigured, and can be undertaken without interfering with existing access patterns. Third, the system is efficient and highly scalable, providing order of magnitude performance improvements over traditional systems, even as the size and complexity of the data grows. In previous work [5], we have examined the core technology of the Index Fabric, specifically in the context of managing semistructured data such as XML. Here, we examine how to deploy and exploit this technology in a multi-tier e-commerce architecture.

In this paper, we discuss how extensible, middle-tier data management can address the twin challenges of flexibility and efficiency for today's e-commerce applications. Specifically, we make several contributions:

- We present an architecture for deploying eXtensible Data Management in the middle tier of an e-commerce application.
- We discuss implementing XDM with the Index Fabric, an engine that supports schema flexibility and robust, flexible queries, while providing high performance.
- We illustrate the challenges and solutions of extensibility using a case study.

This paper is organized as follows. Section 2 introduces the case study that we use as a running example. Section 3 outlines the benefits of deploying extensible services in the middle tier. In Section 4, we discuss the technical details of the Index Fabric, an XDM engine that provides a unique combination of flexibility and efficiency. Section 5 will revisit the case study to show how XDM can be applied. In Section 6 we examine traditional solutions, and in Section 7 we discuss our conclusions.

2. Case study

We will use a hypothetical case study of "Acme Industrial Parts." This case study illustrates the challenges faced when a large diverse enterprise tries to leverage data management for e-commerce, and how an XDM system based on the Index Fabric can help.

Acme is a well-known name in industrial parts, and has several different selling venues. Acme sells its parts directly to large customers, as well as through retailers such as hardware stores. Some orders are placed via Acme's sales force, using a custom built sales application, while some are placed through a web portal. Acme needs a data management system to manage its products and provide information about the products to each of these venues, as well as to manage inventory and other internal product information.

Acme faces a number of challenges in trying to use traditional systems to manage its sales and inventory:

Each application has different data needs. Although some of the needs overlap, some are quite different. It is possible to create a global schema that serves all of the business needs and processes, but such a schema is likely to be complex and unwieldy. In contrast, a lowest common denominator schema can be defined containing only the elements needed by all applications, but this schema would not serve any application particularly well.

Acme can choose to build separate back-ends customized for each application. However, this limits the ability to keep all of the back-ends consistent (e.g. new products must be manually loaded in each of the databases according to the local schema). Moreover, these custom back-ends need to be integrated with the enterprise-wide inventory system.

New services are hard to deploy. For example, to add a "custom built industrial parts" service, Acme must create a new database to support that application. This requires glue code to be manually written to integrate that system with others such as the inventory system. If Acme decides not to create a new database, then it must be able to extract appropriate information for the new service from existing databases, even if none of the existing databases are a close fit with the new service.

The system must be efficient and scalable. Acme is always creating more products, producing more information related to its products, and acquiring competitors that have their own product databases. Acme cannot use a solution that is flexible if the solution does not scale.

Clearly, Acme needs a new kind of data management solution. This solution must be able to support multiple different access patterns over the same data. It should be able to store relevant information for each product, without requiring that every product record fit a uniform, global schema. The system should also support evolution of the database to provide new services in the future, and do all these things with high performance.

3. Data management in the middle-tier

There are many longstanding and well-understood arguments for moving applications to the middle tier [8,3]. The traditional client-server architecture is being rapidly replaced by the superior *n*-tiered alternative. The middle tier is an appropriate place for XDM, since there are unique benefits to deploying data services in the middle tier that cannot be realized in any other deployment. These benefits are realized compared to alternate architectures: either placing XDM at the "front-end" (application layer or above, see Figure 1) or at the "back-end" (the database.)

Integration: Middle tier data management allows us to integrate multiple schemas and sources to provide a powerful interface for all front-end applications.

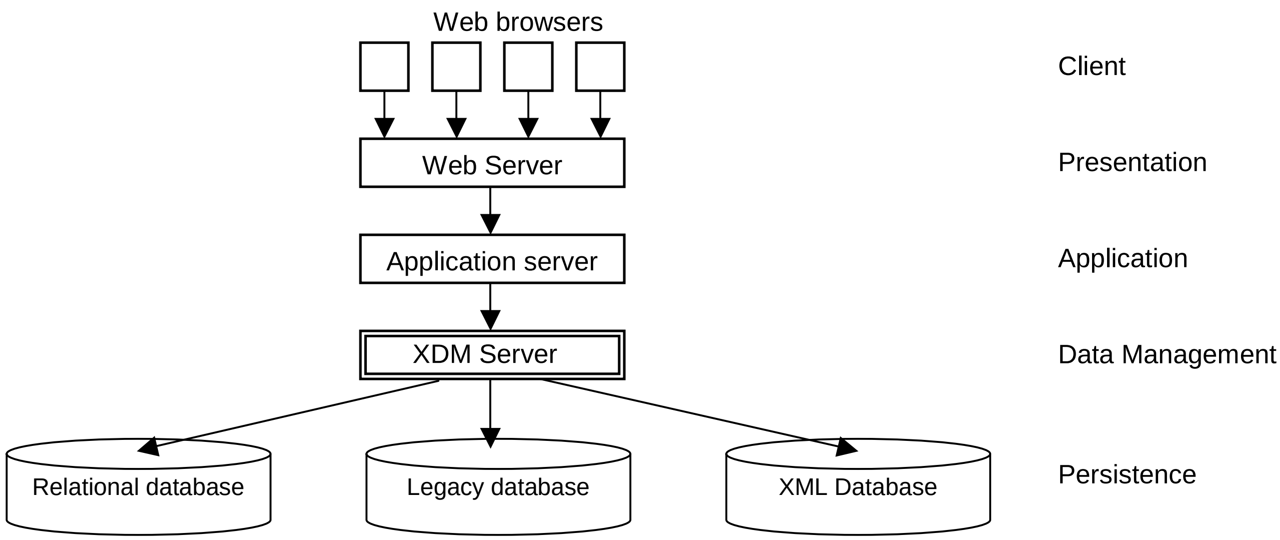

Figure 1. Multi-tiered application architecture with eXtensible Data Management layer.

Deploying XDM in the middle tier avoids the need to reimplement integration in every front-end application. This allows Acme to bring together databases that may have been developed separately for each warehouse or each business unit.

Redeployment: Business components can be relocated between clients and servers without impacting the architecture of either. This means Acme can upgrade its XDM infrastructure without recompiling and redeploying front-end applications or application server. It also allows redeployment to happen despite heterogeneity of back-end architectures. Moreover, since the deployment is centralized, the management is centralized as well, and thus more efficient.

Clean separation: Data management processes can be isolated from arbitrary restrictions imposed by specific implementations. The front-end architecture is often dictated by the needs and capabilities of the various Acme user groups, and may not be appropriate for data management tasks. The back-end architecture may be dictated by the requirements of the database system or by Acme's large infrastructure investment, and may not be adaptable to the needs of XDM. Clean separation also makes it easier to deploy new applications and databases into the system, since they merely have to understand the relevant APIs in order to interoperate.

Load balancing and work distribution: Acme can enhance scalability by providing parallel clusters in the middle tier to run XDM services. Managing data at the back-end makes load balancing quite difficult, as there is no global external view of incoming requests (and knowledge of peers may be equally limited). Front-end developers should be able to focus on business logic and presentation, and not have to worry about the mechanics of load balancing.

Figure 1 shows the XDM server conceptually in a new middle tier layer. In the actual implementation, the XDM may be encapsulated in a server in its own layer, or may be integrated with the application server (e.g. as the persistence mechanism of a J2EE Enterprise Java Beans layer) The latter option is useful to eliminate network communication latency between the application server and separate XDM server.

4. The Index Fabric

We have argued that an appropriate architecture for eXtensible Data Management is to provide extensibility via a middle tier component. However, there must be a core technology to provide these services that has the required flexibility and scalability. We have implemented the Index Fabric, a data management engine that is a good substrate for XDM capabilities. The Index Fabric represents a new approach to data management that offers both flexibility and high performance. The key components of the Index Fabric are:

- The data representation, which is self-describing and provides the flexibility.
- The indexing structure, which provides the efficiency and scalability.

4.1 Index structure

The indexing system is a novel, multilayered index based on Patricia tries [12]. Patricia tries are like normal tries, except they only index the differences between keys instead of whole keys. This means that the index scales well, since it grows slowly as new keys are inserted. Moreover, the size of the index depends only on the number of indexed keys, not on the length of keys. This means that the Patricia tries can manage large numbers of long (and thus complex) keys and remain compact. The Index Fabric applies successive layers of Patricia tries to provide balanced, efficient access even if the trie is highly unbalanced. This novel extension to a basic Patricia trie index minimizes the disk I/O's needed to perform a search. In our implementation, a single index disk I/O is needed for any search, even if the database contains a billion items. Updates can be handled by first performing a search (using the multilayer structure) to find the appropriate portion of the index, and then updating the affected blocks.

We focus here on applying the Index Fabric to the problem of eXtensible Data Management for e-commerce. For more details about the data structure and algorithms, see [5,6].

4.2 Self-describing data

Keys indexed by the system have embedded semantic hints that describe the nature of the managed data. This feature is necessary to support irregular, non-uniform and dynamic schemas, since XDM system can reflect the actual structure of the data, without having to translate it into a single, uniform schema. It is also necessary to support robust, exploratory search, since the self-describing elements can help the user in formulating queries by revealing what types of information are available in the database. The indexed keys with embedded hints are mapped to data items in the database; however, the data items themselves remain stored in their native form.

The Index Fabric embeds semantic hints by representing data as *designated strings*. A designator is a special character or string of characters that has semantic meaning. The combination of designators and the matching semantic concepts (found in the designator dictionary; see Section 4.3) makes the data self-describing. For example, Acme might assign the designator **T** to "item type," **D** to "dimensions" and **P** to "price." Then, a particular item such as a drill can be represented as the keys "**T** Drill [242]", "**D** 11in x 5 in x 7 in [242]", "**P** $64 [242]"; this encodes that item 242 is a drill, with dimensions of 11in x 5 in x 7 in, and which costs $64. (The object ID 242 may be an XML document number, a rowid in a relational database, an OID in an object oriented database, or any appropriate data pointer.) A different item may be encoded as "**T** Hammer [165]", "**C** Red [165]", "**P** $12"; this is a red hammer that costs $12 (if **C** is "Color"). These items may have come from the same data source, such as the Acme inventory database. Alternately, the second item may have its own schema because it originally resided in a different database, such as the inventory database of a company Acme acquired. By encoding both records using the same metaphor (designated keys), schema flexibility is possible, since the data engine only has to manage designated strings and does not require a uniform schema or data format. Moreover, the designators assist searches, since the data engine itself can indicate to users that some items have color information, and other items have dimension information. The system does not have to maintain explicit NULLs to indicate that a record does not have an attribute.

Note that the task of choosing appropriate self-describing semantic hints is not trivial. We are not trying to solve the problem of automatically extracting semantic information from source data or a full ontological markup language. Instead, we hope to provide the basic building blocks, in the form of designators, for managing and searching information with high efficiency. We also assume that a mechanism exists for dealing with varying formats for the data itself. For example, one price may be represented as "$5" while another is represented as "5 US$". A data cleaning step or wildcards in the query language are traditionally used to manage such discrepancies, and can be applied here as well.

Within the same XDM system, Acme can manage sales information. Figure 2 shows a portion of the index for invoices; this illustrates how designated records are represented in the Patricia trie structure. (For clarity, the multilayer structure is omitted in the figure.) Designators in this example are strings constructed from individual semantic concepts, and the following concepts are used in the figure:

`invoice` = **I**	`address` = **A**	`phone` = **P**
`buyer` = **B**	`seller` = **S**	`count` = **C**
`name` = **N**	`item` = **T**	

To search the index in Figure 2, the system constructs a search key based on the user query. Thus, to search for invoices where ABC Corp. is the buyer, the system would search for "**IBN** ABC Corp." This is done by starting at the root (the node labeled "0"), and following the appropriate edges (labeled "**I**", "**B**", etc.) until the correct data record is reached.

4.3 Dynamic structure

The data engine is able to manage new types and structures of data over time. This is important to support seamless integration of new data sources, which may have schemas different from the existing database. Moreover, it is possible to enhance the existing data by adding new tags. For example, Acme may decide to add a new

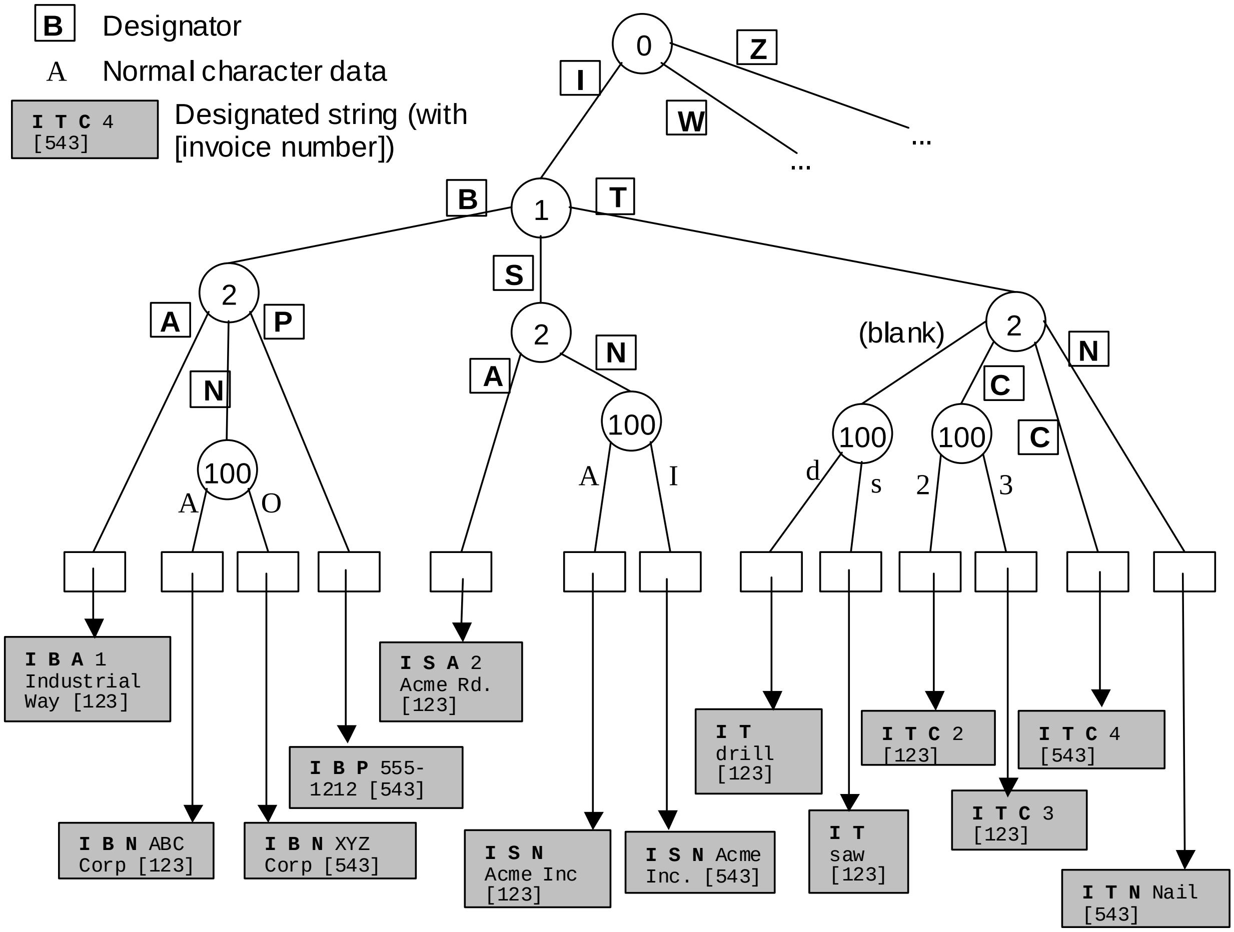

Figure 2. Patricia trie index for invoices.

attribute, "weight," to its product records. This is done simply by creating a new designator for the new attribute, and then inserting keys encoded with the new designator.

The Index Fabric tracks dynamic tags via a *designator dictionary*, which allows designators to be mapped to semantic concepts. In the case of Acme, the dictionary keeps the relationships (**T** → type, **D** → dimensions, **P** → price, …). Because new mappings can be added to the dictionary at any time, new designators can be created to support new semantic concepts. Adding new concepts is also efficient, requiring only the addition of a new mapping to the dictionary and then new designated strings to the data system. These tasks do not disrupt existing data access paths or applications. Moreover, the designator dictionary provides additional flexibility, such as the ability to update mappings (e.g. change "price" to "cost") or map multiple concepts to the same designator (e.g. map both "price" and "cost" to **P**). This flexibility is an advantage of using designators to represent semantic information.

4.4 First-class relationships

The XDM system manages important relationships between data items as first class objects. This means that relationships are explicitly materialized, and managed in the same way that data items are managed. This allows us to efficiently deal with complex relationships, since these relationships do not have to be reconstructed at query time (e.g. using joins in a conventional tuple-oriented representation). Moreover, if the relationships are first-class objects, they are managed as self-describing items. This supports query formulation, since the user or application can browse the XDM system to determine what relationships exist between the data items. For example, after reaching the node labeled "1" in Figure 2, the user can see what types of objects (in this case, buyer **B**, seller **S**, and itemlist **T**) are related to invoices.

The Index Fabric manages important relationships by representing them as designated strings, just like normal data elements. For example, to represent the fact that an item "**T** drill bit [789]" is to be used with another item "**T** drill [988]," the database administrator can direct the system to materialize the key "**T** drill [988] **T** drill bit [789]." This key is treated the same as any other designated string, and thus is a first-class object. To search for bits for a particular drill, we can search for keys prefixed by "**T** drill [988] **T** drill bit." Similarly, a search for keys prefixed by "**T** drill [988]" returns everything related to that drill item, either returning the designators describing related item types, or returning the related items themselves (depending on the user requirements.)

4.5 Efficiency and scalability

The Index Fabric provides highly efficient and scalable data management, which allows the system to support large numbers of designated keys with high performance. The system is scalable in terms of: the amount of data managed (because of the small index size), the complexity of the data (because of the support for long, designated strings), and the number of access paths through the data (because of the ability to manage multiple paths in a single, compact index). Newly integrated data can be queried efficiently along "raw access paths" that follow the structure of the data. Over time, as the importance of certain alternative access paths becomes apparent, a data administrator can add "refined access paths" that provide optimized access along these paths. (See [5] for a detailed discussion of raw and refined paths). Many access paths can be optimized in this way because new access paths are represented simply as keys in the index, and keys can be added to the index cheaply due to the compression provided by Patricia tries.

The multilayer Patricia trie index provides fast lookups and updates, even if the database is large. Moreover, the length of keys does not impact efficiency. In previous work [5] we have compared using a popular commercial relational database with and without the addition of a middle tier Index Fabric XDM system. Adding the Index Fabric increased the performance of the system by an order of magnitude or more. Moreover, as the complexity of the data and queries grew, the performance gain provided by the Index Fabric also grew, demonstrating that the flexibility of the Index Fabric XDM does not come at the cost of performance.

In the example of Acme, a scalable and efficient Index Fabric means that new services can be deployed, and an ever-increasing product line managed, without fear of bogging down the whole system. This frees the company to invent new ways to serve customers, without having to worry about whether the underlying data management system is up to the task.

5. Acme revisited

By deploying XDM in the middle tier, Acme can support rapid development of applications. Moreover, using the Index Fabric as the basis for XDM provides a high performance, scalable solution. We have implemented the Product Directory, an application that companies such as Acme can use to manage direct sales. The Product Directory application leverages the extensibility of the Index Fabric XDM layer to provide several key features; in this section, we focus on two such features: 1) the ability to integrate new product data into an existing directory, and 2) the ability to browse the evolving structure of the database.

From time to time, Acme must add new product records to the directory. These records may not match the schema of the existing products, such as when Acme acquires a company and wants to integrate its existing product database. Integration is done using the "publish catalog" function of the Product Directory. Figure 3 shows a screenshot of the interface to the application. On the left of the screen ("My Catalog") is the schema of the new data to be integrated. On the right of the screen is the external view of the current integrated database ("eMarket").

A user can create a mapping between the attributes in the new data and attributes that already exist in external view. For example, the "speed" attribute of the new product may match the existing "speed" attribute, but the "wt." attribute may have to be mapped to an existing "weight" attribute. This mapping leverages the designator structure within the Index Fabric: each attribute is mapped to a designator, and multiple attributes can map to the same designator. As a result, once records are added to the index, they are searchable by existing applications, because the attribute names are represented internally using designators that already work for the existing applications. If the new products have attributes that do not match any existing attributes, then the "Add new attribute" function can be used. This efficiently changes the schema of the underlying database, because new designators can be added without rebuilding the whole index.

The search function is similarly flexible. Users can navigate the relationship structure of the database directly to find the products they need. This navigation follows the relationship structure that exists explicitly in the Index Fabric. The screenshot in Figure 4 illustrates a form of exploratory search, where the application gives feedback about which attributes are available and which may be relevant to the current search. The user has performed a basic search (for example, on a keyword) and the system has returned a large result set. Now the user can focus the search, based on the attributes that exist for the products in that result set. The "Include Attribute" column allows

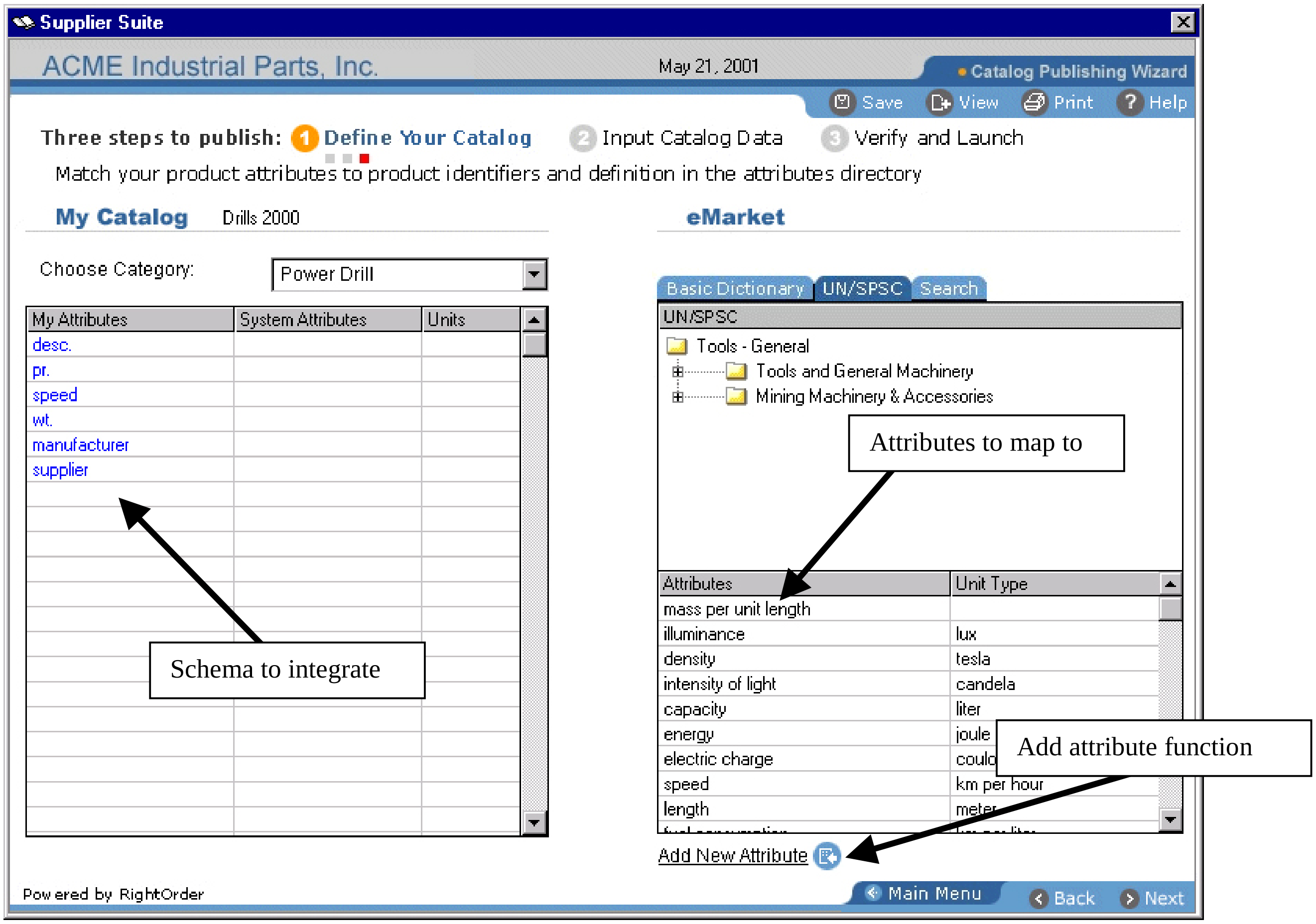

Figure 3. The publish function of the Product Directory.

the user to decide which attributes to search on, chosen from the list in the "Name" column. "Relevancy" indicates the proportion of the result set that has the named attribute. For example, 100 percent of the returned products have a "supplier name" attribute, while only 40 percent have a "ram" attribute. The relevancy is provided from the designator information in the index, and is used to reorganize the user interface (in this case by sorting the attributes). This illustrates how the self-describing nature of the Index Fabric allows the application to dynamically adapt to the user's current needs.

The flexibility of the Index Fabric means that this application can be built easily. The efficiency and scalability of the Index Fabric means that the application performs well, even for large product databases.

6. Existing solutions

Other approaches have been suggested for providing adaptable data management over heterogeneous sources. One possibility is to use a wrapper/mediator architecture [2,7,11,15]. In this approach, a common data schema is defined, and a wrapper is created for each data source to translate between the common schema and the source's native interface. A mediator handles the task of accepting queries, forwarding them to the appropriate wrappers, and collating the results. Unfortunately, significant work must be expended to create a wrapper for every new source that enters the system. Moreover, it is often necessary to reconfigure or rewrite the wrappers and mediator to handle new services. Thus, while mediators can integrate heterogeneous sources, they are too inflexible for dynamic environments.

Another possibility is to provide unstructured access to the data [13,14]. This solution is similar to a web-search engine: searches are formulated as key words, and the data engine performs full-text searches over the databases [4,9]. This solution overcomes the problem of heterogeneity of structure. Moreover, the search cannot take advantage of semantic information present in the structure of the data. Thus, a great many results can be returned that are not relevant to the search, and must be filtered by the user or end application. At the same time,

Figure 4. Exploratory search interface.

the underlying sources may not provide full-text search capability, eliminating this as a feasible option.

7. Conclusion

E-commerce applications are placing ever greater demands on enterprise data management infrastructure. eXtensible Data Management provides features key to rapid development and deployment of new services, while maintaining and evolving existing applications. These features include the ability to integrate multiple sources, manage dynamic and irregular schemas, provide assistance in formulating queries, and manage complex relationships, all while providing high performance and encapsulating data management. At the same time, we have argued that these data services should exist in the middle tier, to provide such benefits as source security, efficient redeployment, and enhanced modularity. An implementation of XDM, such as our Index Fabric, can provide immense benefits, both in the short term and over time, to developers of e-commerce applications.

References

[1] R. Agrawal, A. Somani and Y. Xu. Storage and querying of e-commerce data. In *Proceedings VLDB*, September 2001.

[2] Y. Arens, C. Chee, C. Hsu and C. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. In Journal of Intelligent and Cooperative Information Systems, Vol. 2, June 1993.

[3] C. Berg. The state of Java application middleware, part 1. JavaWorld, March 1999.

[4] W. Cohen. A web-based information system that reasons with structured collections of text. In *Proceedings of Autonomous Agents AA-98* (1998), 400-407.

[5] B. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon. A fast index for semistructured data. In *Proceedings VLDB,* September 2001.

[6] Brian Cooper and Moshe Shadmon. The Index Fabric: Technical Overview. Technical Report, 2000. Available at http://www.rightorder.com/technology/overview.pdf.

[7] H. Garcia-Molina et al. The TSIMMIS approach to mediation: data models and languages. Journal of Intelligent Information Systems, 8:117--132, 1997.

[8] L. Haas et al. Optimizing Queries across Diverse Data Sources. In *Proceedings VLDB,* September 1997.

[9] A. Howe, and D. Dreilinger. SavvySearch: A MetaSearch Engine that Learns Which Search Engines to Query. AI Magazine, 18(2), 1997.

[10] A. Jhingran. Moving up the food chain: supporting e-commerce applications on databases. SIGMOD Record, 29(4): 50-54, December 2000.

[11] W. Kent. Solving domain mismatch and schema mismatch problems with an object-oriented database programming language. In *Proceedings VLDB,* September 1991.

[12] Donald Knuth. *The Art of Computer Programming, Vol. III, Sorting and Searching, Third Edition.* Addison Wesley, Reading, MA, 1998.

[13] A. Salminen and F.W. Tompa. Pat expressions: an algebra for text search. Acta Linguista Hungarica 41, pages 277-306, 1994.

[14] VanRijsbergen, C. J. Information Retrieval. London: Butterworths, 1979.

[15] G. Wiederhold and M. Genesereth. The conceptual basis for mediation services. IEEE Intelligent Systems, pages 38-47, September/October 1997.