Combining a Publish and Subscribe Collaboration Architecture with XQuery Approaches

M. Brian Blake
Department of Computer Science
Georgetown University
Washington, DC, USA
blakeb@cs.georgetown.edu

David H. Fado and Gregory A. Mack
Advanced Systems and Concepts
Science Applications International Corp. (SAIC)
Arlington VA, USA
{David.H.Fado, Gregory.A.Mack}@saic.com

ABSTRACT
Markup languages, representations, schemas, and tools have significantly increased the ability for organizations to share their information. Languages such as the Extensible Markup Language (XML) provide a vehicle for organizations to represent information in a common, machine-interpretable format. Furthermore languages, such as the Document Type Definition Language (DTD) and XML Schema Definition Language (XSD) allow organizations to share the schema and structure of their data. Though these approaches facilitate the collaboration and integration of inter-organizational information, the reality is that the schema languages are reasonably difficult to learn, and automated schema integration (without semantics or ontology mappings) is currently an open problem. We introduce an architecture to facilitate organizational collaboration. In this paper, we introduce such an architecture that combines the *push* features of the publish/subscribe protocol with distributed registry capabilities. In addition, a Java-based, service-oriented implementation entitled *Sharx* is described and evaluated.

Categories and Subject Descriptors
D.2.11 [**Software**]: Software Engineering: Software Architectures – *domain-specific architectures.*

General Terms
Design, Experimentation, Standardization, Languages

Keywords
Distributed and heterogeneous information management, management of semi-structured data

1. INTRODUCTION
Web data can be defined as a form of data marked-up for Web use, sometimes also called *semi-structured* data. XML is a markup language that provides a universal format for organizations to represent their underlying web data. Objects in XML data are specified using *elements.* A list of elements and their *attributes* represent the fundamental structure for defining an XML schema. A major problem occurs when two organizations develop their schemas independently and produce different XML schemas for essentially the same data or capability. In the future when these organizations need to merge their capabilities, there is a requirement for *schema integration.* Although there are modeling approaches that facilitate integration [17], the idea of schema integration without context information or *semantics* is an open problem. In fact, automated tools for schema integration, when semantics are not present, are practically nonexistent, and human-intervention and organizational collaboration are required.

Instead of addressing the problem of schema integration first, we believe the initial step should be creating a seamless organizational collaboration environment which will facilitate web data sharing. Although registries [9] have typically facilitated data sharing, they lack the capability of real-time distribution of data. We introduce an architecture that extends standard registry capabilities with an integrated messaging infrastructure. Furthermore, the architecture supports a suite of graphical services that allow users to create, store, and activate XML query instructions on the web data prior to distribution. This architecture can serve as a first enabling step towards a *fully-automated* collaboration architecture, perhaps employing ubiquitous, agent-supported technologies. In the next section, the usage of the architecture is discussed in an applied setting. Also, the necessary modes of operation are discussed. In Section 3, the related works are discussed. In Section 4, the design of the new architecture is described in detail. The subsequent sections discuss the SHARX implementation and its operations. The final section summarizes our conclusions.

2. INTELLIGENCE SHARING SCENARIO
The major innovation of the proposed publish and subscribe collaboration architecture is the ability for the implemented system to stand-alone. This approach is similar to peer-to-peer (P2P) frameworks [1] with respect to the ease of operational use. The sponsoring organization provides the web-accessible machine, but by the nature of the publish/subscribe paradigm, the distribution architecture should operate independently. This approach also extends the P2P approach by enabling machine-to-machine capabilities. General machine-to-machine interfaces can enable the delivery of distributed information directly to local software services.

2.1 A Domain-Specific Scenario
Under the sponsorship of the Air Force Research Lab (AFRL), new infrastructures, that support information sharing and advanced analytical collaboration, are being created to support intelligence analysts. Many companies and universities are

creating intelligence tools containing analysis information and complex models. These tools publish assets in some form of structured mark-up language. A major challenge in this domain is making these assorted capabilities available on a *single analyst desktop*.

As an example, intelligence analysts typically issue or receive urgent requests for information. These requests spur the formation of a team of perhaps a dozen relevant experts who must construct accurate responses in a short period of time. Due to the nature of this domain, the composition of these communities of interest will evolve as intelligence needs change and the cognitive support tools employed will vary according to the shifting group membership. Such a dynamic environment cannot rely on a strict schema applied to all analysts without undermining the capability to bring in new users, new points of views, and new tools. One key to facilitating collaboration in a dynamic environment is the capability of comparing schemata without merging them, thereby exposing different perspectives among members of the community of interest. The knowledge of varying perspectives accelerates team formation by exposing potential points of miscommunication.

Currently, the team formation and processes are relatively ad-hoc. Using the proposed architecture, an analyst will employ a principled approach using his/her desktop to create and participate in an on-demand, virtual *community of interest*. The proposed architecture supports advanced collaboration by providing data sharing as a first step so that the differences of the schema and the subsequent merging activities can be included as part of the community of interest definition.

2.2 Steps for Deployment

In order for this collaboration architecture to be adopted, there is a critical requirement that deployment steps be straight-forward and effortless. Therefore, an essential part of specifying the architecture is also specifying deployment steps. Any implemented system must comply with the deployment steps listed below.

1. *A sponsoring organization establishes a web-accessible server machine with adequate processing capabilities.*
2. *The collaboration environment is created on the machine.*
3. *Through automated means, the main interface of the system is immediately available.*
4. *Automated processes pre-configure client components and add local web links to enable providers and requesters to download the components.*
5. *With exception to server back-up and maintenance, the application should run independently. The sponsor administers the server and is also a user of the system (i.e. provider and/or requester).*

3. RELATED WORK

Although there are projects that address XML schema integration [17][14], these projects do not focus on creating distributed infrastructures that enable inter-organizational information collaboration. This area, as described in Section 1, is generally divided into two technologies, registry infrastructures and messaging frameworks. There are several registry applications that help organizations store and share their information represented in XML, WSDL, SOAP, etc. Specifications such as Universal Description, Discovery, and Integration (UDDI) and OASIS ebXML, describe registry frameworks for storing the aforementioned representations. There are many applications that implement these specifications, jUDDI [11], XML.org , to name a few. These specifications and tools, however, do not support the distribution of the underlying instance information.

There are other messaging approaches that support the distribution of web data. Applications such as JAX-RPC [10] and WebSphere (formerly XML MQSeries) [27][28], support the transport of web data however collaboration of schema information is not supported, but used only for validation. In some cases, schema information is used, but only for validation. These applications tend to be tightly coupled, and they do not support an asynchronous publish and subscribe protocol [3].

There are other technologies that address both areas of advertisement and distribution, but not with the completeness of the approach introduced here. JAX-R [9] is a XML registry that can be combined with JAX-RPC to enable both the storage and distribution information. This framework does not support an asynchronous environment. In addition, publish and subscribe features are limited to the schema information and not to the instance data. AT&T's Web Services Connect [26] is an architecture for integrating data across organizational boundaries but, according to recent specifications, it requires human-driven customization. KnowNow[13] has an architecture that supports the publish and subscribe features, however it is limited to information presented on web pages. Using embedded scripting and a shared server, organizations can receive the web-based data changes in real-time. KnowNow, however does not focus on the storage features. No other research project, technology, or application meets all the requirements of storage, asynchronous messaging, querying, and customization.

Another related area to address schema integration is the work of the semantic web [12][5]. However, this line of research mostly supports semantic matching technologies using ontological approaches. Describing semantic web approaches in detail is not in the scope of this paper. However, it is important to note that the architecture introduced in this paper is intended to act as an infrastructure to support future semantic web approaches.

4. A PUBLISH/SUBSCRIBE COLLABORATION ARCHITECTURE

In this section, the general design and anticipated operations of the new architecture are discussed in detail.

4.1 Architectural Design

The publish/subscribe collaboration architecture can be discussed in terms of server-side modules and client-side modules as shown in Figure 1. The server contains the core functionality of the system for configuration. A provider organization can register with the system by providing a schema or a specific type of web-based information using the *graphical user interface service.* Also using the graphical user interface, a new requester organization can subscribe to the instance data (based on pre-registered schemas) as it is populated on the server. All registration information is stored in the *user account information* and *markup*

schemas and query instructions partitions of the server-side *data repository*. The requester can choose to receive complete data files or a subset of the web-based data. In choosing a subset, the *publication and subscribe configuration services* capture and store the *query instructions* for that requester in the data repository.

During regular operations, web-based information is distributed in real-time. This distribution of information requires interface modules on both the client-side and server-side. On the client-side, there are two modules that package and unpackage the instance data for transmitting to and receiving from the server. There is a human-driven module, *human user interface service*, and a module that enables machine-to-machine interface, *application interface service*. Also enabling distribution is a server-side distribution service, *machine-to-machine interface service*. At operations time, provider organizations will transmit instance data regularly and the *distribution services* will propagate the subscribed data to requesters while recording the transactions in the data repository (*transactions and instance information*). At times, *parsing and query services* can be used to process the distribution.

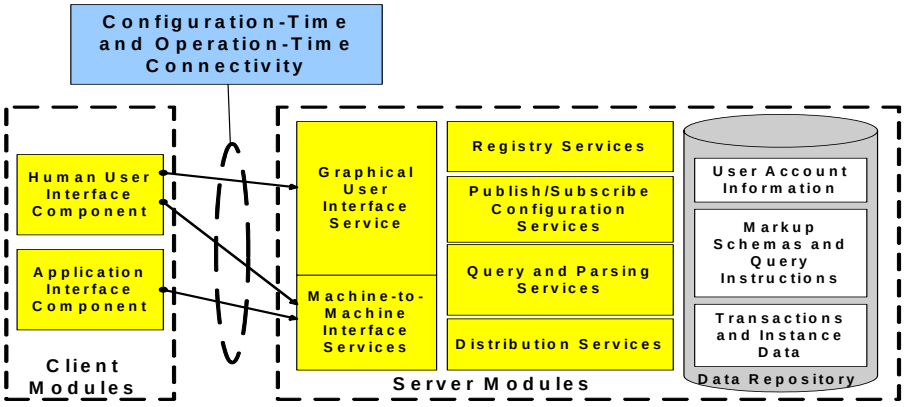


Figure 1. High-Level Publish/Subscribe Collaboration Architecture.

5. SHARX: AN IMPLEMENTATION

In order to validate the publish and subscribe collaboration architecture, a prototype was created. The prototype is a service-oriented, component-based system that leverages technologies such as Java-based web services, XML querying software, a relational database, and web server packaging techniques. The implementation is entitled *Sharx* (i.e. derived from ***SHAR****e* ***X****ML*). In this section, the Sharx implementation and its operational usage is discussed in great detail.

5.1 Sharx Component-Based Implementation

In order to facilitate future evolution, Sharx was implemented as a set of autonomous components. For the purpose of this paper, we discuss the five major high-level components, although it should be noted that several of the components can be decomposed further. The five major components are the *graphical user interface component*, the *distributed transmission component*, the *client proxy component*, the *collaboration processing component*, and the *data management component*. The Sharx system diagram is shown in Figure 2.

In general, the Sharx architecture is a database-centered architecture. The system control instructions are stored in the database along with the user, schema, and instance information. The data management component manages the data requests for all of the other remaining components. All information and requests within the server *and* among Sharx clients are transmitted as pre-established *communication objects*. The use of pre-established objects promotes the creation of clean component interfaces. The data management component is incorporated into each of components. By routing all information requests through the data management component as communication objects, the system is easily evolved to any new registry or relational database in the future. Since each component communicates to the database for control *and* information requests, the architecture can easily evolve at run-time when the addition of new components is required. The following subsections discuss each of the Sharx components and their underlying technologies in detail.

5.1.1 Data Management Component and Database

As discussed in the previous section, the data management component is the only component that connects directly to the data repository. Other components embed the data management component and utilize the data communication functions by passing communication objects. The data management component is implemented in the Java programming language and consists of a data repository with the MySQL relational database and the JAXR registry. In the initial version, all information was stored in MySQL, although software was created to extend the JAXR application. In future work, the user and system control information will be stored in a relational database while the schema information will be stored in the registry.

The Sharx database is separated into three distinct but integrated areas. The three areas are *System Control Information*, *User Configuration Information*, and *Transactional Information* as shown in Figure 3.

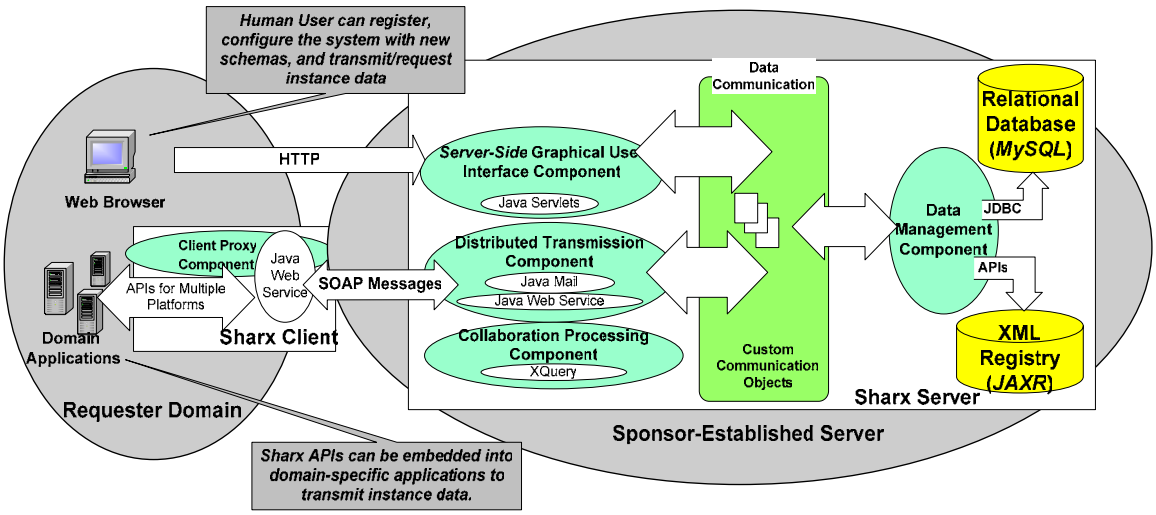


Figure 2. Sharx System Diagram.

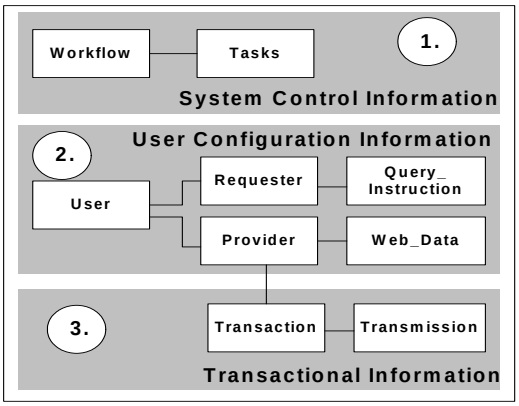


Figure 3. Sharx High-level Database Structure.

System Control Information consists of entities that describe the workflow-based operation of the components when a new request is received. Every organization that connects to the Sharx system is specified as a user in User Configuration Information. As users register their data schemas and frequency of delivery, they are further specified as providers. As users subscribe for existing data, they are specified as requesters. Requesters can have parsing and querying instructions as specified in the query_instruction entity. Finally, records of incoming requests are stored in the Transaction Information

5.1.2 Graphical User Interface Component

Sharx has a server-side graphical user interface where users can register to the system and submit their web data. The user interface was implemented using Java Servlets. A screenshot of the servlet-generated web page is illustrated in Figure 4.

5.1.3 Collaboration Processing Component

The collaboration processing component contains two underlying components, a *matching component* and an embedded XML *query component*. The matching component is implemented in the Java programming language. The AT&T XML-QL software [29] is embedded into a query component to perform the query on the incoming instance data. Although XQuery is the current approach for XML query, XML-QL was chosen for this initial prototype because of availability [2]. In later experiments the query building capability will be expanded to XQuery-based implementations. The collaboration processing component uses both the matching and query components to produce a list of distribution instructions later used by the distributed transmission component when sending the resulting information to requesters.

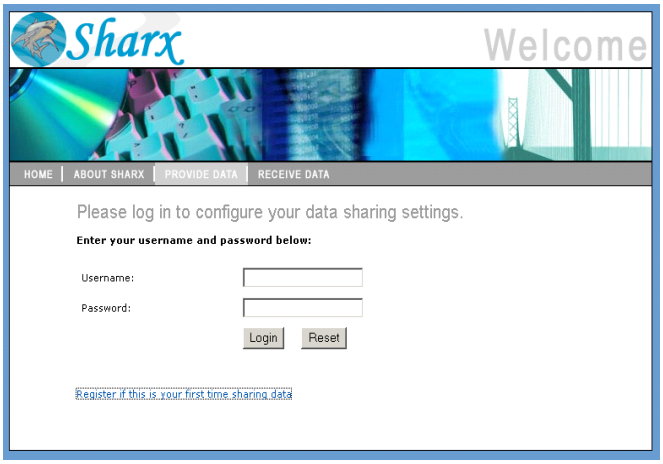


Figure 4. Sharx Main Web Page.

5.1.4 Distributed Transmission and Client Proxy Components

The distributed transmission and the client proxy components meditate all communication between the Sharx server and clients. Both server and client machines must contain a compliant web server, such as Apache. The distributed transmission and client proxy components are *2-way* web services. These components are registered on the web server and can both send and receive communication objects embedded in SOAP messages. These components are Java-based web services implemented with JAX-RPC [10] software included in the Java Web Services Developer Pack.

5.2 Sharx System Operations

The Sharx system contains automated capabilities to assure the ease of deployment. All of the Sharx components are packaged together as one module in a Simple Web ARchive (WAR) file [21]. The Tomcat web server can automatically unpackage war files and install the Java software upon restart. The Sharx deployment procedure capitalizes on this automated feature. A new sponsor can deploy the Sharx system by simply placing the Sharx.war file in a specified location and restarting their web server. An additional automated script captures the IP address of the server and pre-configures the client proxy components. The deployment steps are summarized in Figure 5.

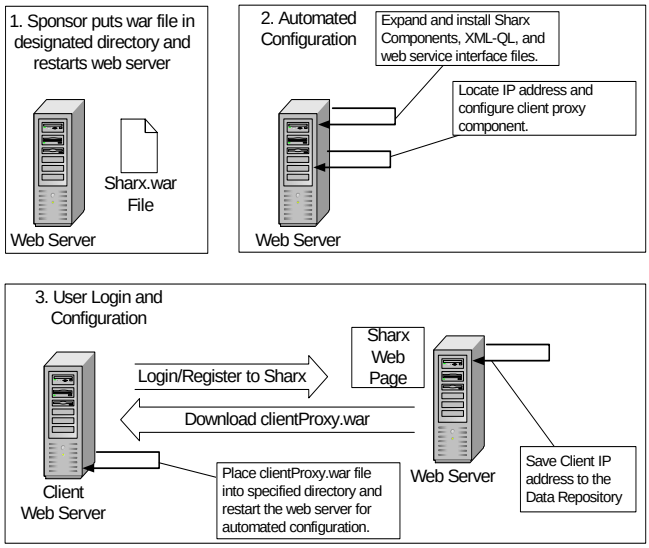


Figure 5. System Deployment Steps.

5.3 Building Query Instructions

Although Sharx embeds the XML-QL software as a component, custom software was created (i.e. collaboration processing component) that displays the XML schema so that a requester can develop query instructions. When a user logs into Sharx and subscribes for an XML file, the user is given the opportunity to subscribe to the instance data as it is received or to request a subset of the data. Implemented in the collaboration processing component is a hierarchical graphical display of the XML schema. In initial development, the collaboration processing component focuses on XSD. A sample XSD file is represented in text and graphical form in Figure 6. In addition, sample instance data is shown. The XSD represents information about a list of books and the underlying characters in those books. Hypothetically, another organization may be interested in new books that contain only specific characters. In such cases, the other organization may only need the book title and author and may not require the additional character information such as friend-of, since, and qualification.

Also in Figure 6, the web page resulting from the Sharx collaboration processing component is displayed. In this web page, a user subscribes to a specific subset of data included in the original XML file. The user can select specific elements using the checkboxes to the left of each element. In lower portion of Figure 6, the *book*, *title*, *author*, *character*, and *name* elements are chosen. Using the textbox to the right of the name element, the figure shows that the resulting XML file will contain books with the character, *Snoopy*. It is not in the scope of this paper to describe the XML query language in great detail, but the specification can be found at [29]. The Sharx collaboration processing component generates an XML query file based on the XML-QL specification as shown in Figure 7. The resulting XML instance data is also shown in Figure 6.

6. DISCUSSION

Probably the most closely related architecture work in the area of assisting human collaboration for schema integration was the blackboard system introduced by Ram et al. [18]. Ram introduces a black-board system for integrating relational schemas. This work is related by nature of the support for human collaboration. The Sharx approach relies on technologies that push information to the stakeholders, where as the blackboard approach relies mostly on human-driven collaboration. By addressing schemas for web data, Sharx addresses a different domain than that of relational schemas. Other approaches use publish/subscribe architectures for collaboration on mobile devices [4][6]. These approaches concentrate on smaller lookup queries but do not address large-scale collaboration on web data.

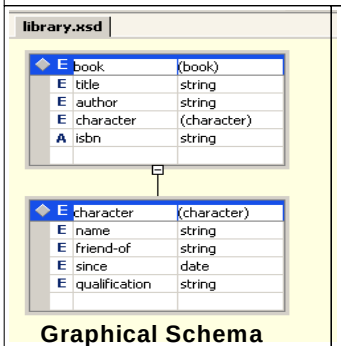
The authors understand that the contributions of the investigations in this paper are applied by nature. Moreover, the major innovation of this work is the introduction of a new collaboration architecture that capitalizes on recent paradigms and technologies. We believe this architecture represents a uniquely innovative combination of approaches to assist in the human collaboration of web-based data. By implementing the Sharx operational prototype consisting of current leading technologies, the approach was validated.

7. ACKNOWLEDGMENTS

A significant portion of the SHARX implementation was developed by undergraduate students of the spring 2004 offering of COSC 346 Software Engineering II at Georgetown University, Washington DC. The students are Ian Costello, Jeffrey Hole, Steve Perlow, and Rana P. Kashyap. AT&T's XML-QL application was used to implement a portion of the XML querying capability. Micardo Johns of AEG Capital, Washington DC helped to motivate the problem of data sharing in the domain of student finance. We also acknowledge conversations with Dr. Leo Obrst and Dr. Len Seligman.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="friend-of" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
              <xs:element name="since" type="xs:date"/>
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XSD Schema

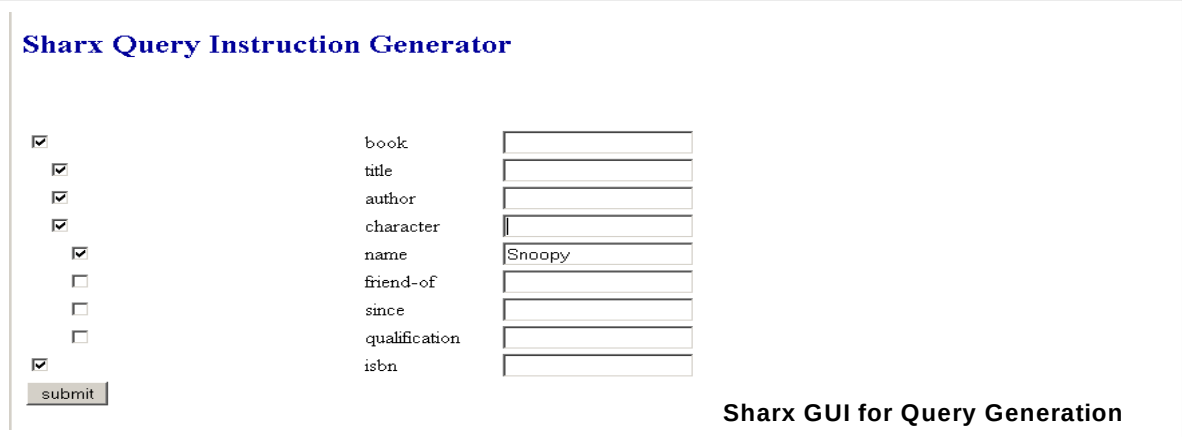


Graphical Schema Display (.Net)

```
<?xml version="1.0" encoding="utf-8"?>

<book isbn="0836217462">
    <title>Being a Dog Is a Full-Time Job</title>
    <author>Charles M. Schulz</author>
    <character>
        <name>Snoopy</name>
        <friend-of>Peppermint Patty</friend-of>
        <since>1950-10-04</since>
        <qualification> extroverted beagle </qualification>
    </character>
    <character>
        <name>Peppermint Patty</name>
        <since>1966-08-22</since>
        <qualification>bold and tomboyish<qualification>
    </character>
</book>
```

XML Instance Data



Sharx GUI for Query Generation

```
function query($db) {
construct    <book>
                  <title>$title</>
                  <author>$author</>
                  <character>
                       <name> $name </>
                  </character>
             </book>

where        <book>
                  <title>$title</>
                  <author>$author</>
                  <character>
                       <name> "Snoopy" </>
                  </character>
             </book>
             </> IN source($db)
}
```

Resulting XML-QL Query Files

```
<?xml version="1.0" encoding="utf-8"?>
<book isbn="0836217462">
     <title>Being a Dog Is a Full-Time Job</title>
     <author>Charles M. Schulz</author>
     <character>
          <name>Snoopy</name>
     </character>
</book>
```

Resulting XML Instance Data

Figure 6. XML Sample, Sharx Query Instruction Generation Page, XML-QL Query File, and Resulting Instance Data.

8. REFERENCES

[1] Aberer, J. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, Trento, Italy pp. 179–194, 2001.

[2] Boag,S., Chamberlin, D., Fernandez, M., Florescu, D., Robbie, J., Simeon, J., and Stefanescu, M. XQuery 1.0 : An XML Query Language (XQL). Technical Report, W3C, April 2002. Available from http://www.w3c.org/TR/xquery

[3] Busi, N. and Zavattaro, G. Publish/Subscribe vs. Shared Dataspace Coordination Infrastructure: Is it Just a Matter of Taste, In *Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001)*, Boston, Massachusetts, 2001.

[4] Cugola, G. and Nitto, E.D. Using a Publish/Subscribe Middleware to Support Mobile Computing. In *Proceedings of the Workshop on Middleware for Mobile Computing* (Hiedelburg, Germany), Nov. 2001

[5] Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I. The Semantic Web: the Roles of XML and RDF. *IEEE Internet Computing*, *4* , 5 (Sept-Oct 2000) 63-73

[6] Fenkam, P., Kirda, E., Dustdar, S., Gall, H., Reif, G. Evaluation of a publish/subscribe system for collaborative working. In *Proceedings of the 11th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2002)*, June, IEEE Computer Society Press.

[7] Fensel, D., Hendler, J.A., Lieberman, H. and Wahlster, W. *Spinning the Semantic Web*. MIT Press, Boston, MA, 2002.

[8] GoXML™ Registry and Messaging, Xenos Corporation (2004): http://www.xenos.com/solutions/prod_goxml_registry.asp

[9] Java API for XML Registries (JAXR) (2004): http://java.sun.com/xml/jaxr/index.jsp

[10] JAX-RPC (2004): http://java.sun.com/xml/jaxrpc/index.jsp

[11] jUDDI, Apache Project (2004): http://ws.apache.org/juddi/

[12] Klein, M. Intepreting XML documents via an RDF schema ontology. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications* (September 2002) 889-893

[13] KnowNow (2004): http://www.knownow.com/

[14] Losio, B.F., Salgado, A.C., and Luciano, R.G. Conceptual Modeling of XML Schemas. In *Proceedings of the 5th ACM International Workshop on Web Information and Data Management* (New Orleans, LA November 2003) 102-105

[15] OASIS ebXML Registry Specification (2004): http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf

[16] OWL-S (formerly DARPA Agent Markup Language for Services (DAML-S)) (2004): http://www.daml.org/services/owl-s/

[17] Passi, K., Lane, L., Madria, S.K., Sakamuri, B.C., Mohania, M.K., and Bhowmick, S.S. A Model for XML Schema Integration. In *Proceedings of the 3rd International Conference on E-Commerce and Web Technologies (EC-Web 2002)* (Aix-en-Provence, France, September 2-6, 2002) Springer-Verlag, Berlin 193-202

[18] Ram, S. and Ramesh, V. A blackboard-based cooperative system for schema integration. *IEEE Expert*, *10* , 3 (June 1995) 56 – 62

[19] Sharx (2004) : http://www.cs.georgetown.edu/~sharx/index.html

[20] Simple Object Access Protocol (SOAP) (2004): http://www.w3.org/TR/soap12-part0/

[21] Simple Web Archive File (2004): http://access1.sun.com/techarticles/simple.WAR.html

[22] The Sharx Prototype (2004) : http://www.cs.georgetown.edu/~sharx

[23] Universal Description, Discovery, and Integration (UDDI) (2004) http://www.uddi.org/

[24] Web Ontology Language (2004) : http://www.w3.org/TR/owl-features/

[25] Web Services Description Language(WSDL) (2004): http://www.w3.org/TR/wsdl

[26] WebServices Connect, AT&T (2004): http://www.att.com/abs/serviceguide/docs/wcs_sg.doc

[27] Websphere MQ Family (2004): http://www-306.ibm.com/software/integration/mqfamily/

[28] Wolfson, D. and Tong, M. DB2 MQ Functions: Using MQSeries and XML Extender from DB2 Applications. *IBM DeveloperWorks, 2004*, *http://www-106.ibm.com/developerworks/db2/library/techarticle/wolfson/0201wolfson.html*

[29] XMLQL, AT&T (2004): http://www.research.att.com/sw/tools/xmlql