A Distributed Event Stream Processing Framework for Materialized Views over Heterogeneous Data Sources

Mahesh B. Chaudhari[#*1]
[#]School of Computing, Informatics, and Decision Systems Engineering
Arizona State University,
Tempe, AZ 85287-8809, USA
[1]mahesh.chaudhari@asu.edu

Supervised by Suzanne W. Dietrich[*2]
*Division of Mathematical and Natural Sciences
Arizona State University,
Phoenix, AZ 85069-7100, USA
[2]dietrich@asu.edu

ABSTRACT

Data-driven applications are becoming increasingly complex with support for processing events and data streams in a loosely-coupled distributed environment, providing integrated access to heterogeneous structured data sources such as relational databases and XML data. This paper provides the foundation for defining a framework for materialized views over heterogeneous data sources in an event stream processing environment. A prototype using commercial off-the-shelf components illustrates a “proof of concept” of the framework for investigating the research challenges in the incremental view maintenance of materialized views. Specifically, the paper explores LINQ as a materialized view definition language for defining views over both relational and XML structured data sources while respecting the native format of the data sources to take advantage of the established technology.

1. INTRODUCTION

Database applications are becoming increasingly dynamic, requiring the monitoring of streaming data and events in a distributed environment. In the past, research was individually focused on continuous queries over streams [1, 12] and distributed event processing [22]. In stream processing applications, data are received continuously and ordered by their arriving timestamps [4]. The streaming data, which generally arrives at a very high rate, is processed using continuous queries with filtering capabilities and access to the persistent data sources. In contrast to streams, a primitive event is defined as an atomic and instantaneous occurrence of interest at a given time [5]. Composite events detect complex and meaningful relationships among the multiple occurrences of primitive events. In event processing applications, active rules define the behavior of the application in response to the occurrence of the events.

Although event processing and stream processing are similar in the aspect of consuming the generated information, each paradigm is designed for different functional purposes. Stream processing involves the execution of continuous queries over a large volume of data generated at a high rate to extract meaningful information. Event processing typically deals with a lower volume of data or primitive events to establish correlations that form composite events. Thus, stream processing can serve as a processing step before event processing by reducing the volume of relevant data for the event handler. There are many dynamic applications, in domains such as financial stock market monitoring or criminal justice, which have a need to define events over streaming data with access to heterogeneous data sources. Thus, the integration of steams and events is now receiving attention in the research community [2, 9, 15, 18].

VLDB 2010 PhD Workshop, September 13, 2010, Singapore

The framework proposed in this paper uses event stream processing as a fundamental paradigm. The distributed nature of the framework with support for access to heterogeneous data sources originated from an active database perspective to use events and rules to build enterprise applications [29]. The application of the proposed framework to heterogeneous data sources results in a system that may be considered a dataspace support platform [11] where data sources co-exist and the framework provides the developer with an environment that supports the challenges of building their application over disparate and distributed data sources. Dataspaces differ from data integration approaches [16] that semantically integrate the data sources using a common mediated schema. This research is novel in that it utilizes event stream processing within the framework of the dataspace system.

The objective of this paper is to describe a framework for investigating the incremental maintenance of materialized views over structured data sources such as relational databases and XML data for distributed event stream processing. Section II provides an overview of a prototype environment built using the Coral8 Event Stream Processor, SQL Server 2008, Oracle 11g Server, XML documents and the C# programming language with .NET framework 3.5. This work is in its initial stages and most of the information in this paper is in its exploratory phase. Hence Section III describes important research challenges and possible research directions in solving the problems. The paper concludes with a brief discussion of future research directions and expected novel contributions of this loosely-coupled distributed event stream processing framework.

2. DISTRIBUTED EVENT STREAM PROCESSING ENVIRONMENT

The concept of a Distributed Event Stream Processing Agent (DEPA) was first envisioned in [28] but not implemented. This paper illustrates a proof of concept implementation of a framework built using a DEPA as a fundamental building block, establishing a foundation for research challenges involving the incremental maintenance of materialized views over heterogeneous distributed data sources within this framework. Specifically, a DEPA can subscribe to different event or data streams. DEPAs can handle data streams from different data sources, including application event generators, the output from continuous queries over sensor data, and streams of incremental changes from databases, such as the Oracle Streams [23] or Change Data Capture feature for SQL Server 2008 [21] for monitoring database log files. The event or data streams can be either in relational format or in a structured XML format. Processing these events or data streams often requires access to persistent data sources such as relational databases and XML data.

The concept of streams has typically been associated with real-time applications having strict low-latency requirements [27]. Such real-time stream processing applications rarely require storing the incoming streaming data for historical reasoning. On the other hand, passive systems continuously poll for conditions of interest, adding overhead and latency. An active system can reduce this additional overhead and latency by incorporating event processing capabilities. In an event stream processing environment, primitive or composite event definitions relying on streaming data may tend to span for a longer duration of time, thus adding latency to the overall system. To satisfy such event definitions along with continuous queries with longer windows may require storage of the in-coming streaming data for later processing. Thus, an event stream processing active system performs better than a passive polling-based system; however, they do not tend to have strict latency requirements as purely stream processing systems.

The main focus of this research is to explore the use of materialized views over heterogeneous structured data sources in such a distributed event stream processing environment as shown in Figure 1. A materialized view is a query whose results are computed when defined and stored in the database with appropriate indices [14]. Upon reference to the view, the materialized results are retrieved instead of recomputed. Each DEPA maintains its own set of resources, such as data and event streams, active rules, relational and structured XML data sources to which the DEPA subscribes. Even though DEPAs communicate with each other to exchange information, it is crucial to define materialized views for each individual DEPA for efficient stream processing, composite event detection, and the execution of active rules. The vision is that each DEPA will have a specific responsibility within the distributed system, which will increase the probability of common subexpressions on which to define materialized views for improving performance. Providing materialized views closer to the specific responsibilities is expected to reduce the overall latency of the event stream processing system [27].

The DEPA prototype uses Coral8 as the event stream processor. Coral8 has its own Continuous Computation Language (CCL) for defining different data streams and event streams [8]. CCL has SQL-like syntax and the streams can be either in relational or XML format. Along with stream definitions, CCL also provides the capability of defining continuous queries and composite events with temporal and windowing capabilities over the streams with access to external persistent data sources, such as relational databases and XML documents.

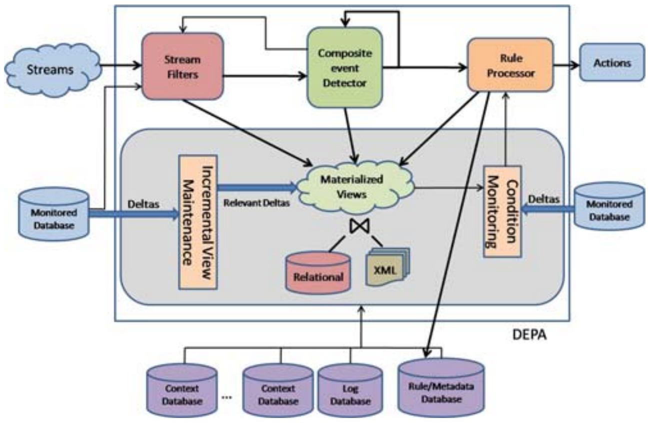

Figure 1. DEPA Architecture

In this research project, the Language INtegrated Query (LINQ) [3, 20] is being explored as the materialized view definition language for the DEPAs. LINQ is a declarative, strongly-typed query language and has syntax that is SQL-like with from, where, and select clauses. An advantage of LINQ is that the same language can query a collection of: *objects* in memory or an object-oriented database, *tuples* in a relational table, or *elements* in an XML document. Thus LINQ provides a unifying paradigm to query heterogeneous data sources declaratively. This scenario is likely for a DEPA, since the event and stream processing often requires access to heterogeneous structured data sources at the same time. Figure 2 illustrates a scenario for a LINQ query that can process streaming data and events along with access to one relational and one XML data source. However, there are research challenges involved in using LINQ as a common platform to access heterogeneous data sources.

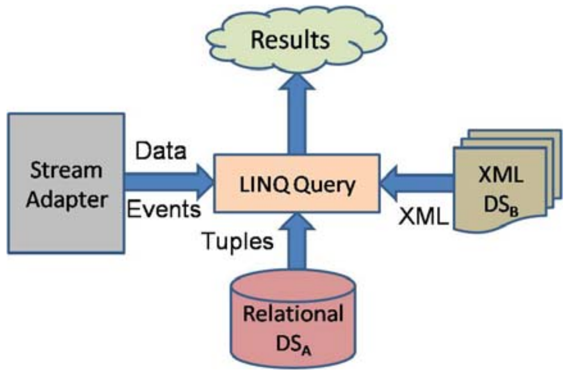

Figure 2. Processing Streams using LINQ over heterogeneous data sources

LINQ provides a layer of abstraction over the different data sources. A LINQ query is represented internally as an expression tree. The LINQ framework then uses a data-source specific LINQ provider to automatically generate an equivalent query over that data source. Also, the LINQ framework is fully extensible. There are numerous third party LINQ providers available to query different types of data sources, such as LINQ to Amazon, LINQ to MySQL, Oracle, and PostgreSql [3]. At present, the default

LINQ provider supports for one persistent data source and the remaining data sources have to be in-memory. However, for DEPAs, it is necessary to access more than one persistent data source through a single query. This research will use the extensible features of the LINQ framework for optimization by writing a LINQ provider that will access multiple persistent data sources in the DEPA environment.

As a motivating example, consider a sample scenario in criminal justice. Assume that the photo radar speeding tickets are streamed over an XML stream (SpeedingTickets). The Motor Vehicle Division (MVD) stores vehicle records in an XML file (VehicleInfo) and the driver license information in a relational database (LicenseInfo). The personal information (PersonalInfo) is stored in an associated state-level relational database. In order to process each speeding ticket, data is required from both the XML file as well as the relational tables so that the speeding tickets can be mailed directly to the vehicle owner's home address. The following LINQ query uses the speedingPlateNum from the SpeedingTickets stream to process each violation:

```
var VehicleDetailInfo =
  from v in VehicleInfo.Vehicle
  from l in DBCon.LicenseInfo
  where l.LicenseNum.Equals(v.Owner.DriverLicense)
         &&
         speedingPlateNum.Equals(v.Info.PlateNum)
  from p in DBCon.PersonalInfo
  where p.SSN.Equals(l.SSN)
  select new {/* project needed fields */};
```

LINQ's unifying paradigm also provides a candidate materialized view definition language for the event stream processing framework. In this inherently distributed and heterogeneous environment, the use of materialized views, especially within a DEPA, is an optimization technique that should be investigated. For example, a materialized view that relates a vehicle with the license and the personal information of its registered owner would likely provide reuse opportunities. This materialized view combines both relational and XML data sources as shown in Figure 3. The data structure used to store the materialized view is part of the proposed research.

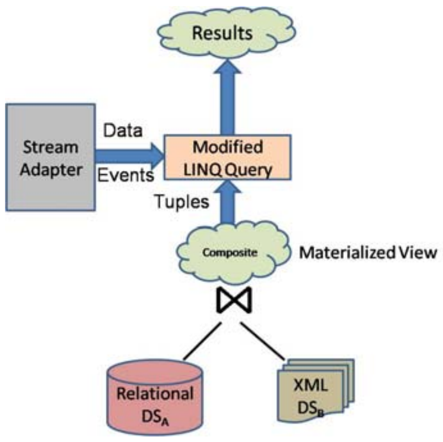

Figure 3. Processing Streams using LINQ and Materialized Views over heterogeneous data sources

Once the materialized view is registered with the system, the original LINQ query can be rewritten to query the materialized view instead of the original data sources [14]. The modified LINQ query that accesses the materialized view ViewPersonVehicle is shown below:

```
var VehicleDetailInfo =
   from mv in DBCon.ViewPersonVehicle
   where speedingPlateNum.Equals(
          mv.VehicleInfoPlateNum)
   select new {/* project needed fields */};
```

The original LINQ query and its corresponding version using the materialized view were implemented in the distributed event stream processing prototype using the C# with .NET Framework 3.5, Coral8 Server 5.6.2, SQL Server 2008, and XML files stored on a hard drive. As expected, query execution in the DEPA environment using the materialized view shows significant performance improvement.

3. RESEARCH CHALLENGES

One aspect of this research is to explore the feature of providing materialized views local to the DEPAs for processing events and data streams. There is a well-known trade-off with the use of materialized views and their maintenance. The materialized view must be updated when the data on which it depends has changed. Rather than recomputing the entire view, an incremental strategy is preferred to modify the materialized view based on the changes to the underlying data sources [10, 13]. An important future research direction is the investigation of incremental view maintenance within this distributed event stream processing framework.

Prior research on incremental view maintenance for heterogeneous data sources has converted the different data sources to one format, either the XML data was converted into relational data [26] or the relational data was converted into XML data [30]. Once all the sources were in one format, then the materialized views were defined and established incremental view maintenance approaches were used. However, these techniques require the overhead of converting one source into another. Additional mapping information is also required to reconstruct the results from the converted format to the original source format.

The use of LINQ as a materialized view definition language opens new alternatives to explore. LINQ orchestrates subqueries over the underlying data sources, respecting the technology of each data store. One question to investigate is whether this orchestration of queries has advantages over the cost of conversions. Currently, LINQ allows access to only one persistent relational database and other data sources have to be in-memory. Hence, this research will explore the extensibility of the LINQ framework by investigating the implementation of a new LINQ provider that can handle multiple persistent databases and in-memory data sources through a single query.

Another challenge is the examination of incremental view maintenance in this loosely-coupled distributed event stream processing framework. The changes or deltas to the underlying data sources can be modeled as streams in a DEPA. For example, Oracle has Oracle Streams and SQL Server 2008 has Change Data Capture to identify the changes occurring in the relational database. For XML files, there are change detection algorithms to capture the changes occurring in the XML documents [17, 31]. Future research will focus on using these deltas in their native format as streams to incrementally update the materialized views.

The perspective of introducing agents for processing distributed events and streams with access to heterogeneous structured data sources introduces several research challenges:

1. *Dependency analysis across different filtering queries to identify common subexpressions as potential candidates for materialized partial joins*

An incremental approach to the evaluation of the resulting language that integrates streams, events, and persistent data is essential. In the DEPAs, the continuous queries, LINQ queries, SQL queries, and composite event definitions are the different query expressions accessing heterogeneous data sources. By exploring multiple query optimization over these expressions, common subexpressions can be detected as the potential candidates for materialized views. The candidate materialized views can represent partial joins across relational and XML data sources.

Multiple query optimization and detecting common subexpressions for Select-Project-Join (SPJ) queries have been explored extensively for SQL queries [7, 24, 25, 32]. Multiple query optimization reduces memory consumption and improves performance when processing multiple queries. It is a well-known fact that identifying common subexpressions is a NP-hard problem and hence it can be addressed by using a heuristic approach [7, 24]. The proposed research is going to explore multiple query optimization over SQL queries, LINQ queries, continuous queries and filter conditions in composite event definitions.

For DEPAs, the continuous queries and event definitions are defined using CCL. The Antlr based parser for CCL will provide access to different subsections of the continuous queries and event definitions. LINQ query is translated into an expression tree that can be traversed to access its different sections. Similarly, the SQL queries can be represented as directed acyclic graphs (DAG) [25, 32] to have access to its different parts. The proposed research will analyze these different tree representations to develop a heuristics-based algorithm to detect common subexpressions. The algorithm will require metadata-level access to all the data sources. One of the assumptions for the project is that each DEPA is designed to perform its own specific task within the distributed framework. This assumption is expected to increase the likelihood of identifying common subexpressions. Thus, each DEPA maintains its own metadata repository over all the subscribed persistent data sources and the different query expressions.

Since the DEPAs are autonomic computing agents, registering/unregistering data sources at run-time, the metadata repository dynamically updates itself at run-time. The metadata repository consists of two components. The persistent component stores the access information and the corresponding run-time classes provide dynamic access to the metadata. Along with different query expressions, this metadata repository also provides access to the already defined materialized views in the system to avoid detection of duplicate candidates. The metadata repository can be accessed at different access-levels and one DEPA can share its metadata repository information with other DEPAs. In order to facilitate such behavior, the metadata repository is exposed in terms of services based on a Service-Oriented Architecture (SOA). The metadata repository is stored in an Oracle 11g database and the metadata services are implemented using the Windows Communication Foundation (WCF) [19]. Details regarding the metadata repository and its services are available in [6].

2. *Techniques for selectively materializing the partial joins over relational as well as XML data sources*

The dependency analysis of the various query expressions will identify common subexpressions as potential candidates for materialized views. One research challenge will be the design of a heuristic algorithm that will selectively choose partial joins over heterogeneous data sources that will be beneficial to materialize, using a cost-based approach in a distributed environment.

The proposed design for addressing research challenges 1 and 2 is shown in Figure 4. This research is going to investigate LINQ as the potential materialized view definition language. Once the candidates are chosen, corresponding materialized views will be defined in the system using LINQ and the original query expressions will be rewritten to use the materialized views instead. The metadata repository for all the registered data sources will provide the necessary metadata-level information for the query optimizer.

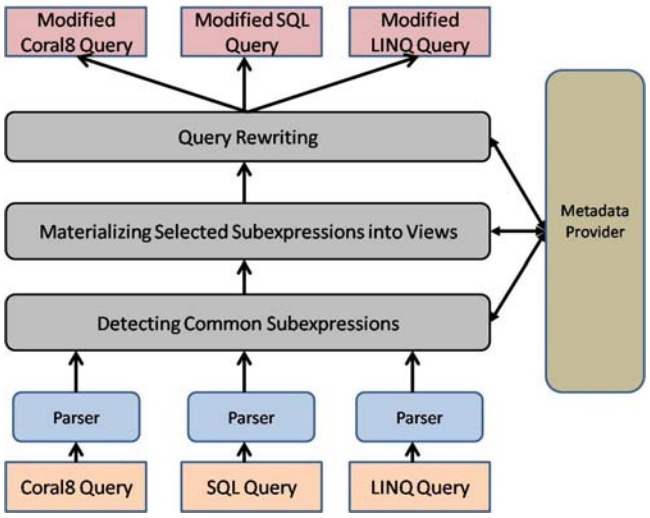

Figure 4. Multiple Query Optimizer

3. *Incremental Evaluation and Materialized Views for Integrating Streams, Events, and Persistent Data*

Capturing of deltas or changes to the original data sources is important in incremental view maintenance of the materialized partial joins. Deltas can be captured for Oracle Server using Oracle Streams or for SQL Server 2008 using the Change Data Capture feature. For capturing changes occurring in XML documents, there are various diff algorithms to generate deltas. There are expected challenges for this proposed research with respect to using native deltas to incrementally update the materialized views over heterogeneous data sources as shown in Figure 5.

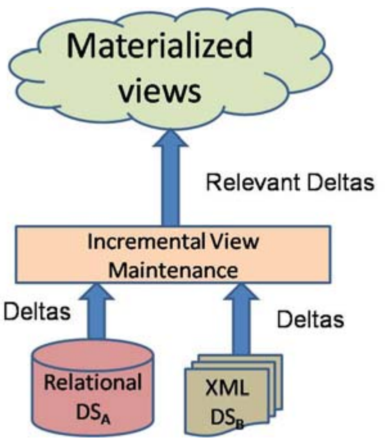

Figure 5. Using Deltas in Native format for Incremental View Maintenance

4. DISCUSSION

This paper has discussed the research challenges for the incremental maintenance of materialized views in a distributed event stream processing framework. The materialized views are defined locally to the DEPAs that process the events and streams to avoid frequent trips to the distributed data sources. A unique aspect of this research is the use of LINQ (and its extensible framework) as the materialized view definition language, which supports querying the data and their deltas in their native formats. The research challenges include the identification of common subexpressions across a suite of related but disparate languages and the selection of which common subexpressions should be materialized as views to improve the performance of the distributed system. These materialized views will then be incrementally updated to reflect the changes in the underlying data sources, which can be either in a relational or XML format. The development of the incremental evaluation techniques for the materialized views over heterogeneous data sources is expected to improve the performance of today's increasingly complex enterprise applications that require support for handling events and streaming data.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0915325.

6. REFERENCES

[1] Babu, S., and Widom, J., Continuous queries over data streams, *SIGMOD Rec.*, vol. 30, 2001, pp. 109-120.

[2] Barga, R. S., Goldstein, J., Ali, M. H., and Hong, M., Consistent streaming through time: A vision for event stream processing, *in CIDR*, 2007, pp. 363-374.

[3] Calvert, C., and Kulkarni, D., Essential LINQ. Boston, MA: Addison-Wesley, 2009.

[4] Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., and Zdonik, S., Monitoring streams: A new class of data management applications, *in Proc of VLDB*, 2002, pp. 215-226.

[5] Chakravarthy, S., and Mishra, D., Snoop: an expressive event specification language for active databases, *Data Knowl. Eng.*, vol. 14, 1994, pp. 1-26.

[6] Chaudhari, M. B., and Dietrich, S. W., Metadata Services for Distributed Event Stream Processing Agents, *in the 19th International Conference on Software Engineering and Data Engineering (SEDE2010)*, San Francisco, June 16-18, 2010, pp. 307-312.

[7] Chen, F. F. and Dunham, M. H. 1998. Common Subexpression Processing in Multiple-Query Processing. *IEEE Trans. on Knowl. and Data Eng.* 10, 3 (May. 1998), 493-499.

[8] Coral8 Engine. (2009) Aleri Inc. [Online]. Available: http://www.aleri.com/products/aleri-cep/coral8-engine, accessed on March 10, 2010.

[9] Demers, A. J., Gehrke, J., Panda, B., Riedewald, M., Sharma, V., and White, W. M., Cayuga: A general purpose event monitoring system, *in CIDR*, 2007, pp. 412-422.

[10] Dietrich, S. W., Maintenance of recursive views, in Encyclopedia of Database Systems. L. Liu and M. T. Özsu, Eds. Springer Verlag, 2009.

[11] Franklin, M., Halevy, A., and Maier, D., From databases to dataspaces: a new abstraction for information management, *SIGMOD Rec.* vol. 34, 4, 2005, pp.27-33.

[12] Golab, L., and Özsu, M. T., Issues in data stream management, *SIGMOD Rec.*, vol. 32, 2003, pp. 5-14.

[13] Gupta, A., and Mumick, I. S., Maintenance of materialized views: Problems, techniques, and applications, in [14], 1999, pp. 145-157.

[14] Gupta, A., and Mumick, I. S., Materialized Views :Techniques, Implementations, and Applications. Cambridge, Mass.: MIT Press, pp. 589, 1999.

[15] Jiang, Q., Adaikkalavan, R., and Chakravarthy, S., MavEStream: Synergistic integration of stream and event processing, *in Proc of ICDT '07*, 2007, pp. 29.

[16] Lenzerini, M., Data integration: a theoretical perspective. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Madison, Wisconsin, June 03 - 05, 2002), PODS '02, ACM, New York, NY, 233-246.

[17] Leonardi, E., and Bhowmick, S. S., Detecting changes on unordered XML documents using relational databases: A schema-conscious approach, *in Proc of CIKM '05*, 2005, pp. 509-516.

[18] Li, X., and Agrawal, G., Efficient evaluation of XQuery over streaming data, *in Proc of VLDB '05*, 2005, pp. 265-276.

[19] Löwy, J., *Programming WCF Services*, Sebastopol, CA: O'Reilly Media Inc., 2009.

[20] Microsoft Corporation. (2009) LINQ: .NET Language-Integrated Query. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb308959.aspx, accessed on March 17, 2010.

[21] Microsoft Corporation. (2009) Tracking Data Changes: SQL Server 2008 Books Online. [Online]. Available:

http://msdn.microsoft.com/en-us/library/bb933994.aspx, accessed on March 17, 2010.

[22] Mühl, G., Fiege, L., and Pietzuch, P., Distributed Event-Based Systems. Berlin; New York: Springer-Verlag, pp. 384, 2006.

[23] Oracle Corporation. (2009) Oracle Streams - Features Overview. [Online]. Available: http://oracle.com/technology/products/dataint/htdocs/streams_fo.html, accessed on March 23, 2010.

[24] Park, J., and Segev, A., Using Common Subexpressions to Optimize Multiple Queries. In *Proceedings of the Fourth international Conference on Data Engineering* (February 01 - 05, 1988), IEEE Computer Society, Washington, DC, 311-319.

[25] Roy, P., Seshadri, S., Sudarshan, S., and Bhobe, S., Efficient and extensible algorithms for multi query optimization, *SIGMOD Rec.* 29, 2 (Jun. 2000), 249-260.

[26] Shanmugasundaram, J., Kiernan, J., and Shekita, E. J., Querying XML views of relational data, *in Proc of VLDB '01*, 2001, pp. 261-270.

[27] Stonebraker, M., Çetintemel, U., and Zdonik, S., The 8 requirements of real-time stream processing, *SIGMOD Rec.* 34, 4 (Dec. 2005), 42-47.

[28] Urban, S., Dietrich, S., and Chen, Y., An XML framework for integrating continuous queries, composite event detection, and database condition monitoring for multiple data streams, *in Proc of Daghstul Seminar on Event Processing*, 2007, pp. 1-5.

[29] Urban, S. D., Dietrich, S. W., Na, Y., Jin, Y., Sundermier, A., and Saxena, A., The IRules Project - Using Active Rules for the Integration of Distributed Software Components, In *Proceedings of the IFIP Tc2/Wg2.6 Ninth Working Conference on Database Semantics: Semantic Issues in E-Commerce Systems* (April 25 - 28, 2001), R. Meersman, K. Aberer, and T. S. Dillon, Eds. IFIP Conference Proceedings, vol. 239. Kluwer B.V., Deventer, The Netherlands, 255-275.

[30] Wang, L., Rundensteiner, E. A., and Mani, M., Updating XML views published over relational databases: towards the existence of a correct update mapping, *Data Knowl. Eng.*, vol. 58, 2006, pp. 263-298.

[31] Wang, Y., DeWitt, D. J., and Cai, J. Y., X-diff: An effective change detection algorithm for XML documents, *in Proc of ICDE'03*, 2003, pp. 519-530.

[32] Zhou, J., Larson, P., Freytag, J., and Lehner, W., Efficient exploitation of similar subexpressions for query processing, In *Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data* (Beijing, China, June 11 - 14, 2007), SIGMOD '07, ACM, New York, NY, 533-544.