SMILE – Distributed Middleware for Event Stream Processing

Rob Strom, Chitra Dorai, Gerry Buttner and Ying Li
IBM TJ Watson Research Center
19 Skyline Drive
Hawthorne, New York, 10532
{robstrom,dorai,gbuttner,yingli}@us.ibm.com

ABSTRACT

In this paper, we describe the SMILE (**S**mart **MI**ddleware, **L**ight **E**nds) system which is one of the earliest systems built in the area of distributed event stream processing. SMILE unites the "publish-subscribe" model of messaging middleware with the "continuous query" model of database systems. In SMILE, information producers, which may be sensors, applications, or databases, generate streams of events, such as RFID data, news items or stock trades; consumers specify stateful subscriptions to derived views, such as "individuals trading top 5 total volume of stock Y within x minutes before a major news story about the same company Y"; the SMILE system constructs and deploys a dataflow network of computations which process events from producers, compute derived views and deliver continuous and timely updates of subscribed views to consumers. Research challenges addressed by SMILE include: rigorously defining the semantics for correct state delivery; implementing this semantics in the presence of failure; distributing computations optimally over a network of multiple servers; scheduling and controlling message flow in this network; smoothly integrating this technology with other components, e.g., databases, services, user interfaces. SMILE is applicable to any "sense and respond" scenario that requires monitoring and processing of events as they are generated. Applications span multiple industries, e.g., monitoring financial opportunities and detecting potential fraud in financial industry, systems management and alerts in data centers, inventory management in RFID applications, and business performance management. In this paper, we describe the SMILE system, its architecture, services supported, and implementation details, and its use in multiple application scenarios.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Continuous Queries; Continuously Updated Views

General Terms

Design, Experimentation, Theory.

Keywords

Distributed Systems, Publish-Subscribe systems, Continuous Queries, Streaming Systems, Event Processing, Fault-tolerance.

IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.
ACM 978-1-59593-638-7/07/0004.

1. INTRODUCTION

Conventional messaging middleware supports simple queuing, routing, and filtering of events delivered from producers to consumers. The SMILE system (**S**mart **MI**ddleware, **L**ight **E**nds) extends this capability to allow multiple consumers to write declarative stateful queries over one or more event streams from producers, to receive results that are continuously updated as new events are received.

1.1 Stateful queries

SMILE generalizes the "publish-subscribe" model of messaging systems, and extends it with the "continuous query" model of database systems. In the publish-subscribe paradigm, there is a world-wide network of producers and consumers. Producers deliver events to *streams* or *topics*; consumers request data of interest. Producers and consumers are mutually anonymous, and consumers need not be aware either of producers or of the requests by other consumers. The middleware figures out how to generate an efficient delivery plan to get the information from producers to consumers. Whereas in conventional publish-subscribe messaging, consumers can request filtered subsets of streams, in SMILE, the consumers can additionally request views defined via *stateful* functions of stream histories. A stateful function is one where the processing of each event in a stream may depend upon previous events from that stream, or from a different stream. Stateful functions include joins, aggregations, and top-k. In SMILE, consumer views are specified using SQL-like continuous queries. Sample queries include, "tell me the average trading price of the 10 highest-volume trading issues", or "alert me whenever two ATM transactions from the same customer appear at different ATMs within an excessively short time window". Stateful queries are deployed over a long time period, and their results are constantly refreshed to reflect new input events.

1.2 Deployment

As part of deployment, the SMILE system takes a collection of stream definitions and queries, and performs the following steps:

- Converts the queries representing stateful subscriptions into *incremental* transforms. Incremental transforms compute *changes* to the result of a query operator in response to events indicating *changes* to one of the inputs to the operator.
- Organizes and optimizes these transforms into a dataflow graph or *delivery plan.*
- Optimally assigns the transforms of the dataflow graph to specific servers, using information about server topology,

message rates, expected computational costs of transforms, and capacities of servers and links. There may be large numbers of producers and consumers, distributed over a wide, possibly internet scale network of servers.

- Generates auxiliary logic components to handle inter-server communication, failure detection and recovery.
- Deploys the generated components to the servers.

1.3 Benefits

The approach of SMILE provides significant benefits in consumability for developers: producers need simply provide declarative schema definitions for each streams; consumers need simply provide declarative, SQL-like queries of desired state; the complexity of how and where to perform the computation and information dissemination is left to the system, and producers and consumers need no other coordination beyond awareness of the stream schema definitions.

The SMILE project and the resulting system implementation have contributed to distributed stream processing research with key innovations in the following areas: (1) a mathematically rigorous safety and liveness service guarantee for event stream processing called "eventual correctness", a counterpart to the ACID guarantee of databases, but more practical for highly scaled and widely distributed systems; (2) mechanisms for achieving these guarantees in the presence of server and link failures, using checkpoint-replay fault-tolerance algorithms that exploit the monotonicity and determinism of the SMILE transforms; and (3) algorithms and heuristics for placing computational components onto servers in order to minimize end-to-end delay. The quality of service guarantee for accuracy of state computation and the ability to deliver intermediate results distinguish SMILE from all other systems in the field, commercial or academic. This distinction is critical and makes SMILE uniquely useful in various time-critical applications.

2. DEMO

In our demonstration of the SMILE system, we will show a number of applications from different industries, in which data is captured at one or more sources, and different subscribers show interest in receiving either (a) a continuously updated dashboard of information derived from the data history, or (b) a collection of notifications (alerts) of critical, actionable conditions derived from the data history.

For each scenario, we will show the definitions of the input streams, a number of different subscriber views, and the results shown to subscribers. We also will demonstrate the consumability of SMILE in terms of query language, the deployment facility, and the distribution and failure scenarios.

2.1 Stock Trading Scenario

In this application, different streams represent offers to trade particular stocks: each event contains an offer for so many shares of a stock of a named issue, to buy or sell at a named price. These "buy" and "sell" streams are aggregated in different ways for different views (e.g., total volume grouped by issue, or by issue-price pair). The aggregated queries are then joined: for example, the total buy volume by issue-price is joined to the total sell volume by issue-price. Another query then computes the difference between the total buy and sell volumes and selects those issue-price pairs whose difference exceeds a certain threshold.

2.2 Fraud-Detection Scenario

In this scenario, events represent ATM cardholders issuing transactions against accounts from different ATM machines.

There is a large volume of transactions. The subscription is for detecting the subset of these that indicate a possible fraudulent use of an ATM card: in this case, two uses of the same account within x seconds from sites far enough away that a car traveling at 60 miles an hour could not have reached both sites within the x seconds.

2.3 RFID for Product Management Scenario

In this scenario, the events are the sensing of tagged objects and the registration of these objects with RFID numbers. Other events represent shipments of groups of objects, including expected destinations and expected arrival times. Various usual and unusual views are requested including: duplicate RFIDs being registered to two different items; inventory for each destination of the items expected but not delivered; objects expected and overdue; objects received at locations other than where intended.

3. ACKNOWLEDGMENTS

Our thanks to multiple people who have contributed to the SMILE project: O. Damani, R. Ginis, A. Khorlin, J. Li and Y. Zhao.

4. REFERENCES

[1] O. Damani and R. Strom, "Smart Middleware and Light Ends for Simplifying Data Integration", IEEE International Conference on Information Reuse and Integration, 2006.

[2] R. Strom, "Technical Challenges and Future Directions for Event-Based Systems", Panel Presentation, 4th International Workshop on Distributed Event-Based Systems, June 2005.

[3] R. Strom, "Fault-Tolerance in the SMILE Relational Subscription System", Proceedings of the International Workshop on Distributed Event-Based Systems, May 2004.

[4] R. Ginis and R. Strom, "An Autonomic Messaging Middleware with Stateful Stream Transformation", International Conference on Autonomic Computing, May 2004.

[5] Y. Jin, and R. Strom, "Relational Subscription Middleware for Internet-Scale Publish-Subscribe", Proc. SIGMOD Workshop on Distributed Event-Based Systems, June 2003.