Push Technology Personalizatiom Through Event Correlation

Asaf Adi, David Botzer, Opher Etzion, Tali Yatzkar-Haham
IBM Research Laboratory in Haifa
{(Adi / Botzer / Opher / Tali)@il.ibm.com}

Abstract

"Push Technology" stands for the ability to transfer information as a reaction to event occurrence. This demonstration proposal describes Amit, a middleware framework that resolves a major problem in this area: the gap that exists between events that are reported by various channels, and the actual cases in which the user needs to react to, hereby called; *reactive situations*. These situations are composition of events or other situations (for example, "when atleast four events of the same type occurred") or content filtering on events (for example, "only events that relate to IBM stocks") or both ("when atleast four purchases of more than 50,000 shares have been performed on IBM stocks in a single week"). This paper describes the generic application development tool, the middleware architecture and framework, and describes the demo.

1. The problem

Reactive applications are those that include components that respond to the detection of events by triggering alerts or other actions (active databases is an example of it). The importance of reactive applications has increased in the recent years with the emergent of e-commerce applications (stock market, business opportunities, sale alerts), as well as system management applications, command and control applications, and customer relationship management applications. Many tools in different areas have been built to detect events, and to couple their detection with appropriate actions. These tools exist in products that implement active databases, event management systems, the "publish/subscribe" protocol, real-time systems and similar products.

Most current tools enable the application to respond to a **single event**. A major problem in many reactive applications is the gap between the events that are supplied by the event source, and the situations to which the clients are required to react, which can be (possibly complex) predicates on the event history. In order to bridge this gap in contemporary systems, the client must monitor all the relevant events, and apply an ad hoc decision process in order to decide if the conditions for reactions have been met.

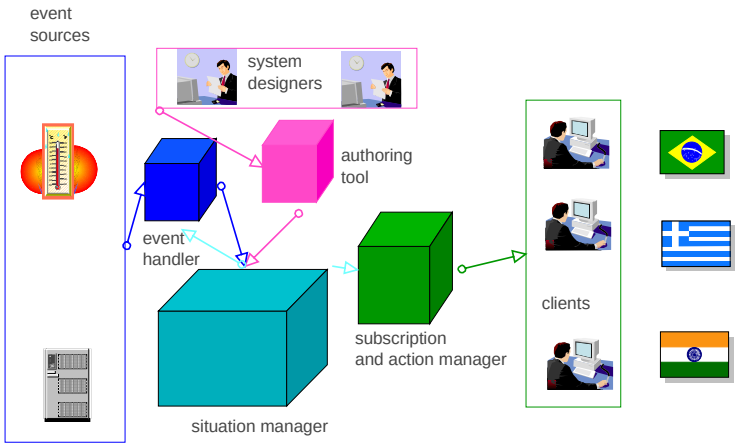
Some examples of situations that need to be handled are:

- The client wishes to activate an automatic "buy or sell" program if, for any stock that belongs to a predefined list of stocks that are traded in two stock markets, there is a difference of more than 5 percent between the values of the same stock in distinct stock-markets, where the time difference of the reported values is less than 5 minutes ("arbitrage").
- The customer relationship manager wishes to receive an alert if a request was reassigned by different agents at least three times.
- A groupware user wishes to start a session when there are 10 members of the group logged in to the groupware server.

In most current implementations, the clients need to store and process all the events. For example, in the arbitrage case, the client has to subscribe to quotes in different stock markets, accumulate the events, correlate them and decide when to operate the "buy or sell" program (in the second case). This may be impossible in some cases, such as "thin" clients without significant storage and processing capabilities. Even if it is possible, the solution that requires a client to process single events may result in a substantial overhead (ad-hoc programming efforts, communication traffic is significantly increased, redundant storage). The problem is intensified due to the many-to-many relationships that exist between the event sources, and the target clients. For example: many stock traders may subscribe to the information services of multiple stock markets.

The goal of the active middleware framework is to personalize push technology through event correlation and enable each client to detect customized situations without the need to be aware of the occurrence of the basic events, or their source.

Figure 1: The active middleware architecture



2. The Architecture

Figure 1 illustrates the implementation's architecture. The architecture consists of the following components:

2.1. Event sources:

The "push" style of event reporting is typical for many information notification applications.
An example is a stock market reporting application.

2.2. Event handler:

This component consists of two sub-components:

Event adapters: programs that convert the reported events to a standard format;

Event base: a data store (implemented on top of a DBMS or a file system) that stores the event instances that are reported by the event sources.

2.3. The authoring tool:

The authoring tool is the system designer's vehicle to define metadata for situations and actions' definitions. All the definitions are phrased as XML propositions, while the meta-meta-data is defined as DTD. The metadata resides in a data store.

2.4. The situation manager:

This is the middleware engine. Its goal is to detect the desired situations.

The situation manager receives two types of input:

- The metadata, which is a collection of parsed XML propositions that guide the situation manager.
- The event instances that are being submitted from the sources using the event adapters.

The situation manager employs composition operators and content filtering on the basic events, and detect situations. Each detected situation is detected as an event, a feature that enables the definition of nested situations. The architecture may vary from a totally centralized solution of having a single situation manager, to a totally distributed solution, in which each subscriber has its own situation manager. In other cases there are multiple instances of the situation manager, that are either subject base, or peers that are aimed at improving the scalability. Each enterprise can choose its own architecture.

2.5. The Subscription and Action controller

This component uses the metadata definitions to decide what to do when the situation is detected. This information has two components:

- Who are the subscribers to this situation?
- What action should be taken for each subscriber (e.g., real-time alert notification, Email message, putting a message on a message queue, triggering a software module)?

2.6. Subscribers:

The clients that subscribe to the information or action. The action can be performed at the client's site, or at the middleware's site.

3. The Actual Demo

The Demo will show the following items:

- Definition of metadata (events and situations) using XML editing GUI (new applications can be written on-the-fly).
- Run-time reaction to simulated events file.
- Run-time trace of situation detection.
- Post-mortem graphical representation of the situation detection

The Demo will show a variety of applications. Examples are:

- E-brokerage application of personalized subscription to situations related to the stock market.
- System management application of personalized subscription to problems and other events.
- Reactive coordination application such as the 2 Phase Commit protocol.

References

[1] S. Chakravarthy & D. Mishra - Snoop: an expressive event specification language for active databases. Data & Knowledge Engineering, 13(3), Oct 1994.

[2] C. Collet, T. Coupaye, T. Svenson - NAOS - Efficient and modular reactive capabilities in an object-oriented database system. In Proceedings. VLDB'94.

[3] S. Gatziu, K. Dittrich - Detecting composite events in active database systems using Petri

Nets. Proceedings IEEE RIDE'94.

[4] N.H. Gehani, H.V. Jagadish, O. Shmueli - Composite event model specification in active databases: model and implementation. Proceedings VLDB'92.

[5] G. Kappel, S. Rausch-Schott, W. Retschitzegger - A Tour on the TriGS active database system - architecture and implementation. Proceedings ACM SAC'98.

[6] Nebula white paper:
http://www.linmor.com/library/white_pa/nms_wp.html

[7] Nervecenter white paper:
http://www.seagatesoftware.com/nervecenter/

[8] K. R. Sheers - HP OpenView Event Correlation Services. HP Journal. Oct 1996.

[9] S. Yemini et al.- High Speed and Robust Event Correlation. IEEE Communications Magazine, May 1996.

[10] D. Zimmer, R. Unland, A. Meckenstock - A General model for event specification in active database management systems. In Proceeding 5th DOOD, 1997.