

Adaptive Object-Oriented Filtering Framework for Event Management Applications

Ehab Al-Shaer

School of Computer Science, DePaul University, Chicago, IL 60604

Mohamed Fayad

Department of Computer Science, University Of Nevada, Reno, NV 89557

Hussein Abdel-Wahab and Kurt Maly

Computer Science Department, Old Dominion University, Norfolk, VA 23529-0162

Event filtering is an essential element in event management applications. In event management environments, the filtering mechanisms are employed to track the events generated from applications at run-time and perform the corresponding appropriate actions. Several key applications domains, such as system and network management, distributed system toolkits, communication protocols and active databases, utilize event filtering for various management purposes. The goal of this paper is to describe the object-oriented design and implementation of an adaptive event filtering framework which can be integrated and reused efficiently to develop event management applications for various domain environments. In our approach, the event filtering framework captures the common components and design patterns of event management in different domains. The major contribution of this work is to provide a flexible event filtering framework that can be efficiently adapted to different domain-specific requirements and with minimal development effort. In this paper, we also present examples of using the event filtering framework for developing event management applications in different domains.

Categories and Subject Descriptors: Application Frameworks [**Software Engineering**]: Software Development

General Terms: Event Management, Application Frameworks

Additional Key Words and Phrases: framework, filtering, design patterns, monitoring

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its data appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2000 ACM 00360-0300/00/0300es

and active databases. In enterprise-wide event management (such as large-scale monitoring), a large volume of events can be generated by the applications at run-time. However, users may be interested in only a small portion of such events or detecting correlations between events in the system. Therefore, event filtering is required to detect specified events, reduce the volume of event notifications and efficiently deliver events to the corresponding users or perform the appropriate actions. This eliminates the unnecessary network traffic and unnecessary processing by end-point management applications and thereby increases the aggregate system performance significantly.

Although event filtering is used in several application domains, every domain has its own specific goals and requirements that impact the design and the implementation of event filtering mechanism [Al-Shaer 1997]. For these reasons, an object-oriented filtering framework is significantly important in order to improve the reusability and extensibility of the filtering in various event management application domains. In this paper, we describe the object-oriented event filtering framework that encompasses the common components and design patterns required by event management applications. This filtering framework improves the reliability and performance of event management applications while minimizing the development effort and cost.

This paper is organized as follows: Section 2 describes the components and the design features of the event filtering framework; Section 3 presents examples of using the event filtering framework for developing efficient management applications; and Section 4 presents our concluding remarks and future work.

2. EVENT FILTERING FRAMEWORK COMPONENTS

The Object-oriented application framework is a reusable, semi-complete application that can be utilized to produce a custom applications [Johnson and Foote 1988]. In this section, we describe the components of the event filtering framework that support the basic infrastructure and services required in several application domains. Developers in different domains can integrate, reuse and extend the framework components to develop domain-custom event management applications.

2.1 Event Definition Constructor Component

Event management applications require the users to specify the events format prior to the filtering process. The Event Definition Constructor (EDC) component is a set of related objects that provide basic interfaces to define events. In order to emphasize the design modularity, this component is divided into several classes that construct *event attributes*, *event operators (basic and advanced)*, *primitive events* and *composite events* which represent the basic elements of event definition [Al-Shaer 1997] (called hot spots [Schmid 1997]). Developers can directly reuse the services provided by the EDC interfaces via object composition or they may customize this component by developing different alternative specifications of the event elements. For example, developers can define different event operators or composite events models. This feature is important to make the event filtering framework extensible and adaptable to various application domains that have variations of event elements definitions.

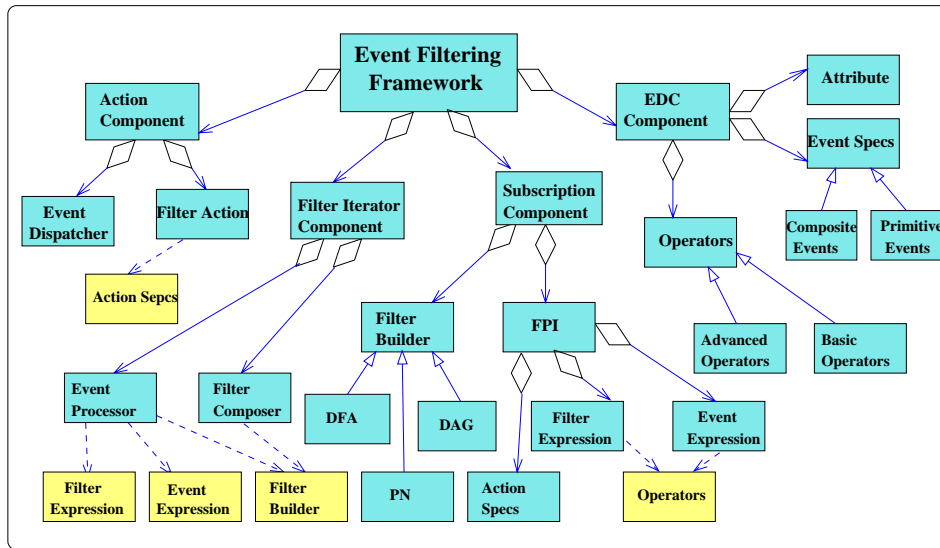


Fig. 1. The Event Filtering Framework Classes

2.2 Subscription Component

The subscription component enables the event management users to specify their management/filtering demands represented as *filter programs*. The filter program defines the target event(s) to be detected and the associated actions. A typical filter program consists of a set of predicates joined together using special operators called *filter expression operators* that describe the required event. The subscription component uses the *filter programming interface (FPI)* class in order to validate and interpret the filter programs defined by the users (interpreter pattern). The second task of the subscription component is to build the filter itself by constructing the filter internal representation which is a connected graph representation conveying all information peculiar to this filter. The subscription component uses the *filter builder* class to achieve this task. Therefore, the subscription component uses these two classes to separate the filter construction from its actual representation (builder pattern) which provides a broad adaptation in the filtering framework. For example, a filter represented in any programming interface can be constructed using many different filter internal representation such as a Directed Acyclic Graph (DAG), Petri Nets (PN) or Deterministic Finite Automata (DFA) and vice versa. This is a significant feature for event management systems since applications domains use various models of filter programs (i.e., programming interface) and number of different internal representations such as DAG and PN depending on the domain requirements. More discussion and examples of such variability are presented in [Al-Shaer 1997].

2.3 Filter Iterator Component

This component is also called a filter/event processor component. The main task of this component is to operate the filter internal representation constructed by the subscription component as described above. In other words, it represents a set of algorithms used to access and manipulate the elements of the filter internals (e.g., DAG, PN or DFA) without exposing its underlying representation (iterator pattern) [Gamma et al. 1995]. Developers can reuse the iterator algorithms in the *filter composer* class to insert, delete and modify filter programs in the internal representation. The filter iterator component also has the *event processor* class which inspects incoming events from the observed system and determine if an event is *detected* or *rejected*. The filter composer permits the developers to customize the framework to adopt different alternatives filter internal representations such as DAG and PN. In addition, various filtering optimization techniques can be applied using the event processor class independently from the filter internal representation itself. This makes the filtering framework adaptable to many alternative design issues in event management applications [Al-Shaer 1997].

2.4 Action Component

Whenever an interesting event is discovered, the filter iterator component instantaneously notifies the action component which consequently performs the action(s) specified in the filter program. The action component classes use the information provided by the subscription component (i.e., filter constructor) to identify the corresponding action of a specific filter. The action component classes can be easily customized to perform general or specialized actions related to the domain environment. For example, an action can be the invoking of methods, executing programs or dissemination events to corresponding management applications. In [Al-Shaer et al. 1997b], we presented management action examples for supporting fault recovery in distributed systems. The action component also provides an events dispatching mechanism, via the *event dispatcher* class, based on input/output (I/O) functions or timers routines.

3. EVENT FILTERING FRAMEWORK APPLICATIONS

Event filtering can be used to manage systems in real-time by tracking and classifying applications events. In this section, we briefly describe the use of the event filtering framework in two applications examples.

Electronic mail (or EMail) is a very commonly used application and people like to have an efficient EMail management tools for their mail events. The event filtering framework can be easily incorporated with existing EMail systems to provide a dynamic control for incoming mails. For example, users can instruct the Email event management system which incorporates the filtering framework to discriminate between incoming mail messages based on EMail event attributes such as **Sender**, **Subject**, **Status** or even specific words in the message text. Based on the user interest expressed in the action component, the EMail messages can be, for example, ignored, forwarded to other devices (such as digital pagers or printers), categorized/sorted based on their priority or even disseminated to groups or users. Similarly, users can trigger actions based on a correlation of two or more EMail

events.

Another more complex applications is from management of distributed systems. The event filtering framework is used for developing HiFi¹ distributed event monitoring system described in [Al-Shaer et al. 1997b] and [Al-Shaer et al. 1997a]. In a distributed system environment, events are generated concurrently and could be distributed in various locations in the applications environment which complicates the monitoring tasks and the management decisions. The event filtering framework described in this paper represents the core element of HiFi monitoring system which is used for monitoring and correlating events in large-scale distributed systems. HiFi users can add, delete and modify their management demands (i.e., events and filters) dynamically using the EDC and the subscription components. They also can specify corrective management actions, such as fault recovery and application steering procedures, to be carried out during the application run-time automatically (see examples in [Al-Shaer et al. 1997b]). In HiFi, different agent classes (i.e., local agents and domain agents which are used for tracking local and global events, respectively) may use different *internal representations* of filters, filter *operators* and filter *iterator* implementations. The event filtering framework was effectively used to develop a customized monitoring (or filtering) agents based on their management roles. For example, domain agents use PN as the filter internal representation in order to record the event history and detect event correlation, however, local agents can sufficiently use the DAG representation since they are stateless agents.

Although the event management requirements and implementation are different in these two applications, the event filtering framework is efficiently used to develop both examples with minimal development effort. Similarly, the event filtering framework can be easily used to develop new customized event management applications based on the requirements of the application domains such as security management, fault management, information and news dissemination, quality and safety assurance in large-scale organization.

4. CONCLUSION

Event filtering mechanisms are widely used in several applications domains (e.g., network and system management and distributed systems toolkits) in order to support an efficient event management applications. In such environment, filtering is necessary to track and control application events, which is required in the management process, while reducing the event traffic and the event processing overhead imposed on the end-point management applications. The significance and the broad deployment of the event filtering in several application domains is the primary motivation for developing an Object-Oriented filtering framework that encompasses the common components and design patterns required by event management applications. The framework enables developing customized filtering mechanisms that possess different alternative specifications based on the domain requirements and characteristics. This is obtained by facilitating the reuse of the code, design patterns and the fine-grain design modularity of the framework components which produce an adaptive filtering framework for various event management application

¹See <http://www.cs.odu.edu/~ehab/Projects/HiFi>.

domains. In this paper, we describe the component architecture and interaction of the event filtering framework which was used to develop HiFi event monitoring system for large-scale distributed applications [Al-Shaer et al. 1997b; Al-Shaer et al. 1997a].

REFERENCES

- AL-SHAER, E. 1997. Event filtering framework: Key criteria and design trade-offs. In *IEEE Int. Conference on Computer Software and Application Proceedings* (August 1997), pp. 84–89.
- AL-SHAER, E., ABDEL-WAHAB, H., AND MALY, K. 1997a. Hierarchical filtering-based monitoring architecture for large-scale distributed systems. In *Int. Conference on Parallel and Distributed Computing Systems Proceedings* (October 1997), pp. 422–427.
- AL-SHAER, E., ABDEL-WAHAB, H., AND MALY, K. 1997b. A scalable monitoring architecture for managing distributed multimedia systems. In *IFIP/IEEE Int. Conference on Managing Multimedia Networks and Services Proceedings* (July 1997).
- GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns: Elements of Reusable Software Architecture*. Addison-Wesley Publishing Company, Reading, Massachusetts.
- JOHNSON, R. AND FOOTE, B. 1988. Designing reusable classes. *Journal of Object-Oriented Programming* 1, 2 (June), 22–35.
- SCHMID, H. A. 1997. Systematic framework design by generalization. *Communication of ACM* 40, 10 (October), 48–51.