Programmable Agents for Active Distributed Monitoring

Ehab S. Al-Shaer

Multimedia Networking Research Laboratory,
School of Computer Science, Telecommunications and Information Systems,
DePaul University,
Chicago, IL 60604
Ehab@cs.depaul.edu
http://www.cs.depaul.edu/Ehab

Abstract. The successful deployment of next-generation distributed systems is significantly dependent on the efficient management support that improves the performance and reliability of these applications at run-time. This paper motivates and describes a programmable agents approach for active monitoring as an important attribute for supporting scalable, highly-responsive and non-intrusive management architecture. Active monitoring enables defining re-configurable and self-directed management tasks that can be modified automatically at run-time in order to track the system behavior. Based on observed events and users' monitoring demands, monitoring agents can dynamically customize their assigned tasks and initiate the appropriate monitoring actions. This avoids activating unnecessary monitoring tasks and provides a dynamic monitoring operations. The presented system, which is referred to as HiFi, supports a comprehensive environment including code instrumentation, user subscription, event filtering and action service. The paper also shows monitoring examples that illustrates the application and the effectiveness of active monitoring in managing large-scale distributed systems.

1 Introduction

The next-generation distributed systems are large-scale, resource intensive and more complex. With the increasing demands of deploying large-scale distributed (LSD) systems, an efficient on-line monitoring has become an essential service for improving the performance and reliability of such complex applications. Examples of LSD systems include large-scale collaborative distance learning, video teleconferencing,distributed interactive simulation, and reliable multi-point applications. In an LSD environment, large numbers of events are generated by system components during their execution and interaction with external objects (e.g. users or processes). These events must be monitored to accurately determine the run-time behavior of an LSD system and to obtain status information that is required for management operations such as steering applications or performing a corrective action. However, the manner in which events are generated

R. Stadler and B. Stiller (Eds.): DSOM'99, LNCS 1700, pp. 19-32, 1999.

in an LSD system is complex and represents a number of challenges for an on-line monitoring system. Correlated events are generated concurrently and can occur from multiple locations distributed throughout the environment. This makes monitoring an intricate task and complicates the management decision process. Furthermore, the large number of entities and the geographical distribution inherent with LSD systems increases the challenge of addressing important issues, such as performance bottlenecks, scalability, and application perturbation.

HiFi is an attempt to deliver an active monitoring architecture that explicitly addresses the challenges and requirements associated with managing large-scale distributed systems. HiFi active monitoring approach supports dynamic and automatic customization for management operations as a response to changes in LSD systems behavior [16, 12]. This is achieved through programmable monitoring agents that re-direct their monitoring activities on-the-fly upon users' requests or based on the information (events) collected during the monitoring operations. For instance, instead of monitoring all events and processes in the system, the agents monitor a subset of events/processes and the monitoring activities expand based on the information of the generated events. Therefore, active monitoring reduces the monitoring space significantly and offers a scalable management architecture. The monitoring intrusiveness is also minimized because this architecture enables initiating few monitoring tasks (targets) at the proper time. In addition, HiFi active monitoring enables the agents react spontaneously (e.g., corrective actions) which improves the management operations response time compared with human-in-the-loop model [13].

A number of monitoring approaches and systems for monitoring distributed systems have been proposed (e.g., [5, 9–11, 13–15]). Although some of these systems provide mechanisms for modifying the monitoring requests dynamically, these mechanisms are manual and insufficient to support a programmable or self-directed monitoring tasks (actions) as described in this paper. In addition, they do not support a scalable and fine grain event filtering mechanism which is significant for monitoring "large-scale" distributed systems such as Internet-based applications. This paper is organized as follows: Section 2 explains the monitoring model and language specifications; Section 3 gives an overview of HiFi monitoring architecture and process; Section 4 describes our active monitoring approach and its impact on the management process; Section 5 presents an application example of using HiFi monitoring system for steering distributed reliable multicast protocols; and Section 6 presents the summary and concluding remarks.

2 Monitoring Model

In order to present a complete abstraction of the active monitoring architecture, our work must include modeling the *application behavior*, the *monitoring demands*, and the *monitoring mechanism* with considering the design objectives presented in Section 1. The program behavior can be expressed in a set of events revealed by the application during execution. In our monitoring model,

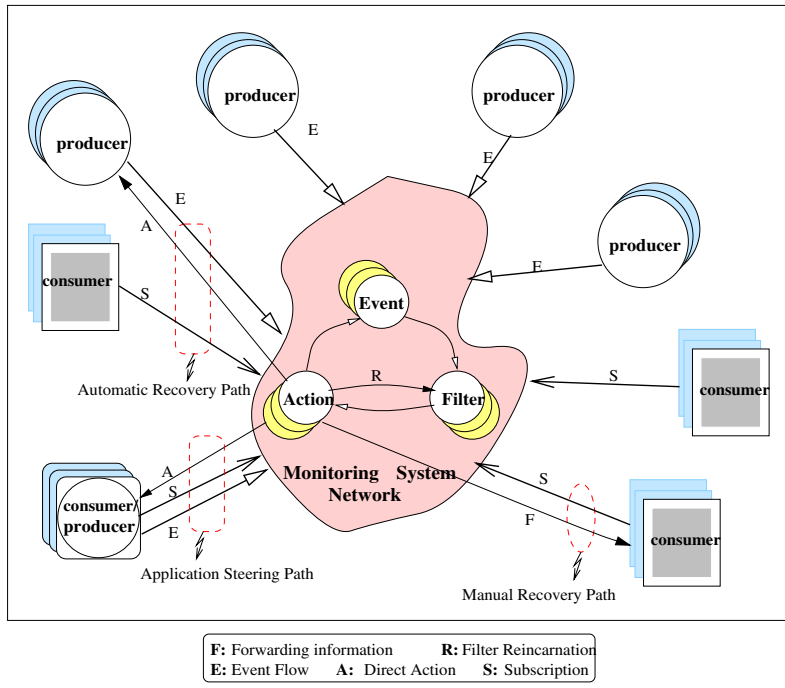

Fig. 1. Monitoring Model

we call the monitored programs *event producers* which continuously emit *events* that express the execution status. An *event* is a significant occurrence in a system that is represented by a *notification message* which typically contains event characteristics such as event type and event source. We classify two types of events used in our model: *primitive events* which are based on a single notification message, and *composite events* which depend on more than one notification message. In the monitoring language, the event format (notification) is a variable sequence of *event attributes* determined by the user but it has a fixed header used in the monitoring process. An event attribute is a predicate that contains the *attribute name* which typically represents a variable in the producer (i.e, program) and a value. The event format also determines the type of event signaling: *Immediate* to forward the generated event immediately, or *Delayed* to allow buffering or batching events in the producer before sending them. Table 1 shows the High-level Event Specification Language (HESL) in BNF. This event abstraction enables consumers (1) to specify any arbitrary event format in a declarative way, and (2) to construct a complex (multi-level) abstraction of a program behavior using composite events. In addition, the event abstraction enables the consumers/users to assign values to the event attributes and does not require specifying attribute type (e.g., `int, float or string`). This provides a simpler interface than CORBA IDL [9]. We call the monitoring objects (e.g.,

```
<Event> ::= EVENT = <Event_Body>.
<Event_Body> ::= <Prim_Event> | <Comp_Event>
<Prim_Event> ::= {<Fix_Att> ; <Var_Att>} <Event_Name>
<Comp_Event> ::= (<Prim_Event> <Event_Op> <Comp_Event> ) |
                 (<Prim_Event> <Event_Op> <Prim_Event> )
<Fix_Att> ::= ModuleName = <String>,
              FuncName = <String>, <Report_Mode>
<Report_Mode> ::= Immediate | Delayed
<Var_Att> ::= <Predicate> , <Var_Att> | <Predicate>
<Predicate> ::= <Att_Name> <Relation> <Value>
<Event_Op> ::= ∧ | ∨ | ∼
<Relation> ::= < | > | = | ≠ | ≤ | ≥
<Value> ::= <Number> | <String>
<Event_Name> ::= <Att_Name> ::= <String>
```

Table 1. High-level Event Specification Language (HESL)

human or software programs) *event consumers* since they receive and present the forwarded monitoring information. The consumers specify their monitoring demands via sending a *filter* program via the *subscription process* which configures the monitoring system accordingly (see Figure 1). The monitoring operation is an *event-demand-driven* model. In other words, the producer behavior is observed based on the event generated (event-based) and on the monitoring requests (subscription-based). Therefore, as illustrated in Figure 1, events received in the monitoring system are classified based on exiting filters. If an event is detected, the action specified in the filter is performed such as forwarding the monitoring information to the corresponding consumers. The filter and action specification is described in Section 4.

3 Overview of HiFi Monitoring Architecture

HiFi employs a hierarchical event filtering-based monitoring architecture to distribute the monitoring load in application environment. Based on a user's monitoring requests, the monitoring system determines the appropriate agent or set of agents within the hierarchy to be tasked with inspection and evaluation of application events. The monitoring system uses fine grain decomposition and allocation mechanisms to ensure that filtering tasks are efficiently distributed among the monitoring agents and prevent events propagation in the network. Since our focus in this paper is on the programmable monitoring environment in HiFi, we give a brief overview of the HiFi system and refer to [2–4] for more details.

Hierarchical Monitoring Agents: In HiFi monitoring system, the task of detecting primitive and composite events is distributed among dedicated monitoring programs called *monitoring agents* (MA). MA is an application-level monitoring program that runs independently of other applications in the system and

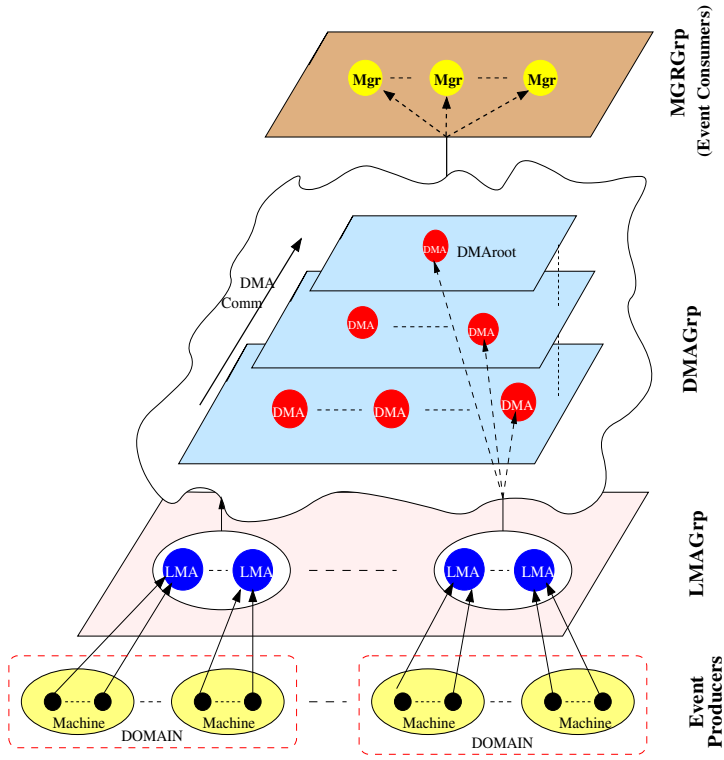

Fig. 2. Hierarchical Filtering-based Monitoring Architecture

that communicates with the outside world (including producers and consumers) via message-passing. HiFi has two types of MAs: *local monitoring agents (LMA)*, and *domain monitoring agents (DMA)* (see Figure 2). The former is responsible of detecting primitive events generated by local applications in the same machine while the latter is responsible of detecting composite events which are beyond the LMA scope of knowledge. One or more producer entities (i.e., processes) are connected to a local LMA in the same machine. Every group of LMAs related to one domain (geographical or logical domain) is attached to one DMA. These DMAs are also connected to higher DMAs to form a hierarchical structure for exchanging the monitoring information. Because of the different roles of LMA and DMA, LMAs use Direct Acyclic Graph (DAG) [6], however, DMAs use Petri Nets (PN) in order to record and track the event history [7].

Subscription Process: The monitoring process starts by a consumer sending a *filter program* that describes the monitoring request to the local MA. The filter is validated and decomposed into subfilters (e.g. F1,..,Fn) using the *decomposition algorithms* in such a manner that each one represents a primitive event [3]. Then, each decomposed subfilter is assigned to one or more LMAs using the

```
<Filter> ::= FILTER = <Filter_Body>
<Filter_Body> ::= [<Event_Expr>]; [<Filter_Expr>]; [<Actions>];
                  <Filter_Name>.
<Event_Expr> ::= ( <Event_Name> <Event_Op> <Event_Expr> )
                 | <Event_Name>
<Filter_Expr> ::= ( <Predicate> <Filter_Op> <Filter_Expr> )
                  | <Predicate> | TRUE
<Predicate> ::= ( <Pred_Att> <Relation> <Pred_Att> )
                | ( <Pred_Att> <Relation> <Value> )
<Pred_Att> ::= <Event_Name>.<Att_Name>
<Filter_Op> ::= <Event_Op>
<Actions> ::= <Action> ; <Actions>
<Filter_Name> ::= <Program_Name> ::= <String>
```

Table 2. High-level Filter Specification Language (HFSL)

allocation algorithms based on the event sources and application distribution. Decomposition and Allocation algorithms are described in [3]. The monitoring system also determines the proper DMAs for evaluating the event and the filter expression of the filter program. When MAs receive delegated monitoring tasks (subfilters) [8], they configure themselves accordingly by inserting this subfilter in the filtering internal representation which is a direct acyclic graph (DAG) for LMAs and Petri Nets (PN) for DMAs [4]. This architecture alleviates any performance bottlenecks or scalability problems by distributing the monitoring load among MAs and limiting the events' propagation to the originating sources [3].

4 Techniques for Active Monitoring

The main goal of active monitoring is to offer dynamically customizable monitoring tasks. This provides a flexible management infrastructure that scales very well with number of producers and causes a minimal overhead in the application environment. In addition, active monitoring, in HiFi, reduces the monitoring latency by supporting an automatic monitoring customization performed by MAs without the users involvement. In this section, we present several techniques developed in HiFi in order to support a programmable agents environment for active monitoring. We also describe the impact of these techniques on improving scalability and performance of HiFi.

4.1 Filter-based Programmable Monitoring Agents

Users (or event consumers) describe their monitoring demands via programs called *filters* submitted to the monitoring system at run-time. A filter is a set of *predicates* where each predicate is defined as a boolean-valued expression that returns *true* or *false*. Predicates may be joined by *logical operators* (such as `AND` and `OR`) to form an expression. In our model, the filter consists of three major components: the *event expression* which specifies the relation between the interesting event, *filter expression* which specifies the attributes value or the

relation between the attributes of different events, and the *action* to be performed if both event and filters expressions are *true*. Table 2 shows the High-level Filter Specification Language (HFSL) in BNF.

Consumers may add, modify or delete filters on-the-fly through the subscription service component [3]. When a consumer performs filter subscription, the monitoring agents re-configure itself accordingly by updating their internal filtering representation [4]. HiFi uses the *subscription protocol*, described in [2], to maintain state consistency and synchronize the monitoring agents. The filter-based programming abstraction enables the consumers to describe the expression relation not only between the events but also between the attributes of different events as well. This improves the expressive power of the monitoring language, and permits fine-grain filtering based on regular expressions.

Example (1): Assume agents have been configured through filter to detect any `AudioWarning` or `VidWarning` events generated from Audio and Video processes, respectively. Consumers can re-program or re-configure the MAs at run-time to detect only the *event correlation* between these two events such that they are both generated by the same machines via sending the following filter.

FILTER= $[(AudioWarning \wedge VidWarning)]$;
$[(AudioWarning.Machine{=}VidWarning.Machine)]$;
[FORWARD]; `Warnings_Correlation_Filter`.

After decomposing and allocating this filter, the DMA will only forward *Audio* and *Video* warning events that are generated by the same machine. Consumers can also delete (deactivate) or modify (re-activate) existing filters using *DEL* and *MOD* in the action part (see Table 3). In addition, the programming environment permits consumers to overload the attributes values in the events in order to create a different event instance in the filter program.

4.2 Event Incarnation

Actions in the monitoring model can be simply *executing a program* (local or remote) or *forwarding* the detected event to the corresponding consumers which are both necessary for automatic fault recovery and application steering, respectively. In order to improve the dynamism and the expressive power of the monitoring system, the model provides more complex actions: a *new event* or a *new filter*. In HiFi, generating new events as an action is called event incarnation. This feature improves the expressive power, performance and usability of the active monitoring system as follows:

- The event-filter-action programming model (see Figure 1) enables the consumer to activate a sequence of monitoring operations (filters and actions) automatically and without the human involvement. In addition, generating new events (as an action) may trigger other filters that are necessary to track

```
<Action> ::= <Exec> | <Event_Name> | <Filter_Rinc> | <Filter_Registers> | FORWARD
<Exec> ::= <Path Name> <Program Name>
<Path Name> ::= <String> / <Path Name> | <String> /
<Filter_Register> ::= <Identifier> = <Event_Name>.<Att_Name>
<Filter_Registers> ::= <Filter_Register> | <Filter_Register>;<Filter_Registers>;
<Filter_Reinc> ::= ADD <Filter_Name>; <Filter_Reinc> | ADD <Filter_Name> |
                   DEL <Filter_Name>; <Filter_Reinc> | DEL <Filter_Name> |
                   MOD <New Filter>; <Filter_Reinc> | MOD <New Filter> ;
<New Filter> ::= <Filter_Name>.EX= <Event_Expr> | <Filter_Name>.FX= <Filter_Expr> |
                   <Filter_Name>.EX= <Event_Expr>; <Filter_Name>.FX= <Filter_Expr>
```

Table 3. High-level Action Specification Language (HASL)

the observed problem. For example, a failure that occurs in a producer (process) as a result of abnormal close of a communication connection (primitive event) may involve two actions: *failure recovery* and *sending* a new event in order to diagnose this failure further (e.g., checking memory usage). Based on this event, new actions (e.g., recovery procedures) could be performed.

- A new event could be an "aggregate" event that summarizes the information of multiple (composite). This reduces event traffic and avoid event report implosion in the monitoring environment. For example, an aggregate event may convey the *average drop rate* of a set of receivers. Section 5 shows how this aggregate event is useful for monitoring reliable multicasting.
- Performing an action such as executing a program may change the state of a running program. Therefore, sending an event that reveals the state change to the monitoring system is important to allow re-observing the behavior, and attaining stability in automatic application steering.

4.3 Filter Incarnation

In addition to the user (manual) re-configuration via dynamic users subscription (described in Section 3), HiFi active monitoring also supports programmable agents that re-configure themselves automatically based on events occurrences. An action can be a filter manipulation (typically, adding a filter, deleting a filter, and modifying a filter). For example, another new filter can be activated in the monitoring environment as a result of detecting an event. We call this *filter incarnation* (see Figure 1) because a filter may add, delete or modify a new filter in the system. Filter incarnation is defined in Table 3 of the monitoring language. Adding a new filter means activating a pre-defined filter that has not been submitted to the system. This is specified in the monitoring language using a special reserved word (`ADD`) with the pre-defined filter name. On the other hand, deletion or modification must be performed on an existing filter for which consumers subscribed. This is specified using the reserved words, `MOD` and `DEL`, with an active filter name. When modifying an active filter, consumers must specify which parts to modify: event expression (`EX`), filter expression (`FX`), or both. This can be designated by appending the filter name as a prefix to `EX`

and/or `FX`. The resulting EX and/or FX are the effective filter parts after the subscription is completed.

Filter incarnation enables users to define "general" monitoring tasks that can be automatically customized by agents at run-time to diagnose specific system behavior such as failures or performance bottlenecks. This also avoids overwhelming the system by a large number of "static" (hardwired) monitoring tasks to observe a large number of system activities. Therefore, consumers can monitor subset of events and request modifying (changing or expanding) the monitoring scope to include other events and processes whenever certain event patterns are detected. In the following, we describe various applications of filter incarnation in active monitoring.

Adding/Deleting Filters for controlling monitor timing: Consumers can specify start and end times for any given monitoring activity based on events. In other words, consumers can specify to start/stop a trace activity when a certain event (primitive or composite) is detected. This minimizes the monitoring overhead and produces concise traces.

Example (2): Assume a consumer wants to monitor the *drop rate* in the "receiving" events (*RecvEvent*) of *bar* program only when the *transmission rate* in the "transmission" event (*TransEvent*) of *foo* program drops below a certain threshold (*STHRESHOLD*). In this case, the consumer can specify a filter (*MonitorSender*) that monitors the "transmission" events of *foo* and triggers another filter (*MonitorReceiver*) that monitors *bar* "receiving" events, if the transmission rate drops below the threshold. The filters example is shown below.

FILTER= $[(TransEvent)]$;
$[(TransEvent.ModuleName =$ "foo" $\wedge\, TransEvent.transrate < STHRESHOLD)]$;
$[ADD\ \ MonitorReceiver]$; `MonintorSender`.

FILTER= $[(RecvEvent)]$;
$[(RecvEvent.ModuleName =$ "bar" $\wedge\, RecvEvent.droprate > RTHRESHOLD)]$;
$[FORWARD]$; `MonintorReceiver`.

The *MonitorReceiver* filter monitors "receiving" events (*RecvEvent*) from *foo* and forwards events to consumers if the *drop rate* exceeds the threshold. Similarly, the *MonitorReceiver* filter can be deactivated (Deleted) based on changes in the *drop rate* in TransEvent. This permits activating/deactivating *MonitorReceiver* filter automatically and at the proper time which minimizes the monitoring perturbation in the application environment.

Modifying Filters Specifications: Usually, monitoring tasks are static and defined prior to any monitoring operation. However, using filter incarnation, the filter specifications can be determined during the monitoring process itself based on event patterns and information. For this purpose, the HASL provides

a set of virtual registers called *filter registers* that consumers can use for loading/restoring variables in/from the monitoring agents. These registers are used by MAs to store attribute values of received events. Consumers can simply assign the attribute value of an event used in EX or FX to a filter register and vice versa for this purpose.

Example (3): For example, consumers may want to generate an event trace (or history) for processes that have produced at least one security warning event (`WarningEvent`). In this case, the module name is not known to the monitoring system at trace specification time. Therefore, the monitoring system must determine the module name during the monitoring operations. This is achieved by using filer registers to save and restore the event information. In the following, the `DynamicErrorTrace` is the filter specification for this example. Notice that *ThisMod* is a filter register that restores the module name, *ModuleName*, after the occurrence of `WarningEvent` of type *SECURITY*. Then, the filer incarnation is used to modify the expression of *TraceProcess* filter so that the modified filter executes the new trace specifications.

FILTER= $[WarningEvent]$;
$[(WarningEvent.ModuleName =$ "ANY" $\wedge\ WarningEvent.Type =$ "SECURITY")];
$[ThisMod = WarningEvent.ModuleName;$
$MOD\ TraceProcess.FX = (AnyEvent.ModuleName = ThisMod)]$; `DynamicErrorTrace`.

FILTER= $[AnyEvent]$;
$[(AnyEvent.ModuleName =$ "ANY")];
[FORWARD]; `TraceProcess`.

The *TraceProcess* is a "generic" filter that monitors and forwards all events from any module to the corresponding consumers[1]. However, this general filter/task is modified by *DynamicErrorTrace* filter to perform a special customized monitoring operation. Therefore, this technique enables activating/deactivating the appropriate monitoring operations (or filters) at the right time (event), and thereby relieving the system environment from the overhead of launching multiple filters or monitoring requests simultaneously. It also reduces the monitoring latency since the monitoring agent can react spontaneously without the consumers intervention. Moreover, the filter incarnation feature provides an extendible programming environment utilizing the power of the recursive event-filter-action model as shown in Section 5.

5 Active Monitoring of Distributed Multi-point Applications

The HiFi monitoring system is used in a number of applications such as application steering, fault recovery and debugging of distributed multimedia systems.

[1] "ANY" is a language keyword that means any string value.

In this section, we present an example of using HiFi for monitoring and steering Reliable Multicast Server (RMS) [1]. One of the known problems in reliable multicasting, is the effect of slow members (e.g., machines) in group communication. A machine is described as a slow machine if its receiving rate is "much" less than the other members in the group. In this case, a slow machine could typically slow down the communication of the entire group because the sender transmission rate, in RMS, eventually adapts to the rate of the slowest receiver. Developing a solution for slow members in multicast groups is beyond the focus of this paper. However, the effective use of HiFi active monitoring is presented in the *dynamic discovery* of slow members (machines) during a multicast session and the *automatic feedback* to the RMS senders which make the proper steering management decision accordingly. The criteria of slow members is defined based on the user specifications. For example, the user (or manager) may define a slow member whose performance is below a certain threshold. In our example below, the RMS sender acts as a manager and sends the threshold information. Figure 3 shows the event (HESL) and the filter (HFSL) specifications used to discover slow members in multicast groups. Each RMS receiver is instrumented (using HiFi) to send *McastRec* event that contains the machine name, the domain name, the group name, total bytes received (KBrec), and number of NACKs scheduled (NackSch) [2]. Because of NACK suppression mechanism [1], the number of *NackSch* gives more accurate estimation of the *drop rate* than number of Nacks sent. The `McastRec` event is sent periodically based on time limit or maximum number of bytes received. And the RMS senders send `McastSend` to indicate two things: the transmission rate (*TransRate*), and the drop rate threshold (*threshold*) for receivers in the group. The *TransRate* is first checked by `MonMcastSender` filters to determine if the "slow members" monitoring activity should be started. If *TransRate* is below the STHRESHOLD, then the `Slow_Memebers` filter is modified to use the *GrpName* stored in the filter register as described in Section 4. This also activates the `Slow_Memebers` filter that compares the NackSch in McastRec and threshold in McastSend to identify slow members. However, because the threshold value is dynamic and may be determined from the overall performance of the participants, another filter (`Update_Threshold`) is used to provide a feedback on the overall drop rate average to senders which consequently re-adjust the threshold value accordingly. Each LMA forwards `McastSend` and `McastRec` primitive events to its DMA that evaluates the filter expression upon receiving both events. The second filter, `Slow_Members`, waits to receive one `McastSend` and `McastRec` events from all LMAs in the domain. Then, the filter expression is evaluated. The `_ctr` and `_LMAs` are HiFi reserved key words and used to denote the number of the event occurrences and the number of LMAs in the domain, respectively. The filter expression evaluates to *true* if all RMS receivers in the domain send `McastRec` event from the indicated group name (*GrpName*) and the NackSch of one receiver or more is higher than the threshold. If the filter expression becomes true, then three actions are performed: (1) the average scheduled Nacks for receivers

[2] $NackSch = Number\ of\ NacksSent + Number\ of\ NacksCancelled$

EVENT= { `ModuleName`=RMS,`FuncName`=McastSend,`Immediate`; Machine="ANY", Domain="ANY", GrpName="ANY", TransRate=ANY, threshold= ANY } `McastSend.`
EVENT= { `ModuleName`=RMS,`FuncName`=McastRecv,`Immediate`; Machine="ANY", Domain="ANY", GrpName="ANY", KBrec= ANY, NackSch=ANY } `McastRec.`
EVENT= { `ModuleName`=DMA,`FuncName`=ANY,`Immediate`; Machine="ANY", Domain="ANY", KBrec= ANY, NackSch=ANY } `DomAVG.`

FILTER= [McastSend];
[McastSend.TransRate $<$ STHRESHOLD];
[ThisGrp=GrpName; `MOD` $Slow\_Memebrs.FX$=(McastRec.GrpName=ThisGrp);
`ADD` $Update\_Threhold$]]; `MonMcastSender.`

FILTER= [$(McastSend \wedge McastRec)$];
[$(McastRec.\_ctr = \_LMAs \wedge McastRec.GrpName =$ "*") $\wedge$
$McastRec.NackSch > McastSend.threshold$)];
[$CalcAvg$; $DomAVG$; `FORWARD`]; `Slow_Memebrs.`

FILTER= [$DomAVG$];
[$DomAVG.\_ctr = \_DMAs$];
[$UpdateThrehold$; $McastSend$]; `Update_Threhold.`

Fig. 3. Active Monitoring Example of Reliable Multicasting.

in same domain is calculated (CalcAVG), (2) the `DomAVG`, which represents an aggregate (summary) event, is sent to the containing DMA to reveal the domain average, (3) the `McastRec` event that matches the slow member criteria represented in the filter expression (i.e., NackSch $<$ threshold) is forwarded to the manager (RMS sender). The third filter, `Update_Threshold`, receives the `DomAVG` events from the DMAs and then calculate the total NackSch average, update the threshold and send the `McastSend` with new threshold to the LMAs/DMAs again. This filter can be a DMA task, instead of RMS senders. However, users must provide the action *UpdateThreashold* to this DMA which then can take care of updating the threshold dynamically while the RMS senders can take care of managing slow members problem. Since senders or receivers could be members in various multicast groups, the group name (GrpName) is used to limit the slow members monitoring activities (filters) on the multicast groups that suffers from this problem. Similarly, another filter can be used to deactivate `Slow_Memebers` filter when the *TransRate* becomes less then STHRESHOLD. In other words, the slow members filter can activated and deactivated dynamically based on the condition of the multicast group.

The slow members and NackSch average information are collected from each receiver via LMAs and then combined and propagated in hierarchical fashion via DMAs to the RMS of the sender. In addition to the dynamic feedback service offered in this example, this mechanism is scalable because it avoids the notifications implosion that may occur when `McastRec` are forwarded to one RMS sender

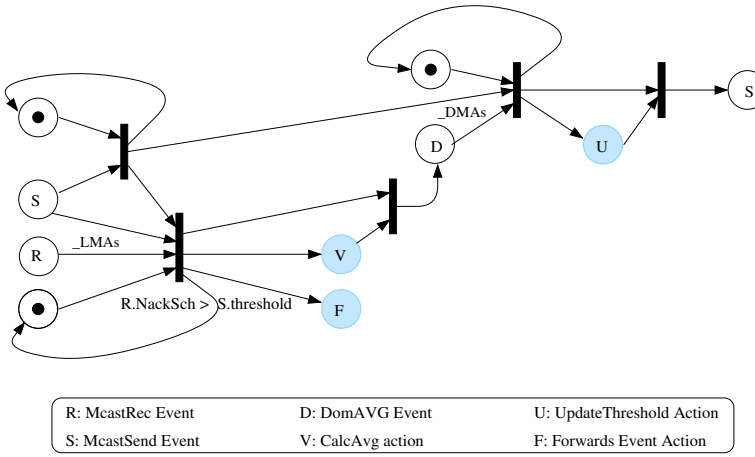

Fig. 4. The PN Filter Representation in HiFi Monitoring Agents.

from group of receivers. Furthermore, distributing the processing load such as calculating the average drop rate contributes to the monitoring performance.

6 Conclusion and Future Work

This paper describes a novel architecture for supporting active distributed monitoring. The monitoring system, called HiFi, uses a hierarchical monitoring agents architecture that distributes the monitoring load and limits the event propagation. The monitoring agents are programmable and can be reconfigured manually through users' interactions at run-time or automatically by the agents themselves based on the information of the detected events. Users utilize a simple language interface, called filter, to define their monitoring demands and associated actions. Users can also specify "general" monitoring tasks that can be customized automatically by the monitoring agents to perform specialized monitoring operations. We developed several techniques to support an efficient programmable agents environment for active monitoring. This includes *event incarnation* to enable event-filter-action programming model, *filter incarnation* and *filter registers* to enable automatic modifications and self-directed monitoring operations, and *dynamic subscription* that enables users to add, delete or modify their requests at run-time. We demonstrate an example of using HiFi for monitoring and steering a reliable distributed multicast server (RMS). This active monitoring architecture offers significant advantages in the scalability and performance of the monitoring systems. It also enables consumers to control the monitoring granularity, and thereby minimizing its intrusiveness.

Although HiFi is developed and used in real-life applications, it remains a prototype monitoring system. Many open issues remain to be addressed in the HiFi research plan. These include improving the filter incarnation mechanism

to provide higher abstraction such that users can specify the ultimate monitoring target without having to specify intermediate monitoring tasks, integrating system and network management, and extending the monitoring language to support temporal event relations.

References

1. Al-Shaer, E., Abdel-Wahab, H., Maly, K.: Application-Layer Group Communication Server for Extending Reliable Multicast Protocols Services. IEEE Int. Conference on Network Protocols. **10** (1997) 267–274
2. Al-Shaer, E., Abdel-Wahab, H., Maly, K.: Dynamic Monitoring Approach for Multi-point Multimedia Systems. International Journal of Networking and Information Systems, Vol. 2. **6** (1999) 75–88
3. Al-Shaer, E., Abdel-Wahab, H., Maly, K.: HiFi: A New Monitoring Architecture for Distributed System Management. Proceedings of International Conference on Distributed Computing Systems (ICDCS'99). **5** (1999) 171–178
4. Al-Shaer, E., Fayad, M., Abdel-Wahab, H., Maly, K.: Adaptive Object-Oriented Filtering Framework for Event Management Applications. To appear in ACM Computing Surveys.
5. Alexander, S., Kliger, S., Mozes, E., Yemini, Y., Ohsie, D.: High Speed and Robust Event Correlation. IEEE Communication Magazine. **5** (1996) 433–450
6. Bailey, M.L., Gopal, B., Pagels, M.A., Peterson, L., Sarkar, P.: PathFinder: A Pattern-Based Packet Classifier. USENIX Symposium on Operating System Design and Implementation. **11** (1994) 24–42
7. Gatziu, S., Dittrich, K.R.: Detecting Composite Events in Active Database Systems Using Petri Nets. Int. Workshop on Research Issues in Data Engineering: Active Database Systems. **2** (1994) 2–9
8. Goldszmidt, G., Yemini, S., Yachiam, Y.: Network Management by Delegation - the MAD approach. CAS Conference. (1991) 347–359
9. Object Management Group. The Common Object Request Broker: Event Service Specification, Tech. Rep. CCITT X.734/ISO 10164-5. (1993)
10. Joyce, J., Lomow, G., Slind, K., Unger, B.: Monitoring Distributed Systems. ACM Transactions on Computer Systems, Vol. 5. (1987) 121–50
11. Marzullo, K., Cooper, R., Wood, M.D., Birman, K.P.: Tools for distributed Application Management. IEEE Computer, Vol. 24. **8** (1991) 42–51
12. Merwe, J.V.D., Rooney, S., Leslie, I., Crosby, S.: The tempest: A practical framework for network programmability. IEEE Network Magazine. **6** (1998)
13. Parulkar, G., Schmidt, D.C., Kraemer, E., Turner, J., Kantawala, A.: An architecture for monitoring, visualization, and control and gigabit networks,. IEEE Network Magazine, Vol. 11, **10** (1997) 32–38
14. Schroeder, B.: On-line Monitoring: A Tutorial. IEEE Computer, Vol. 28. **6** (1995) 72–78
15. Sloman, M. editor. Network and Distributed System Management. Addison-Wesley, Reading, Massachusetts (1994)
16. Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J., Minden, G.J.: A Survey of Active Network Research. IEEE Communications Magazine, Vol. 35. **1** (1997) 80–86