FLEXIBLE NOTIFICATION SERVICE FOR GRID MONITORING EVENTS

S. Andreozzi, E. Fattibene, G. Misurelli, G. L. Rubini, INFN-CNAF, Bologna, Italy
C. Aiftimiei*, S. Fantinel†, INFN, Padova, Italy
G.Cuscela, G. Donvito, A. Pierro, INFN, Bari, Italy
N. De Bortoli‡, G. Tortone, INFN, Napoli, Italy

Abstract

An aspect to consider as regards the monitoring activity of Grid systems is the efficient match and delivery of notification events to subscribers. In this paper, we describe the implementation of a notification model for events that relies on publish/subscribe system and on recent evolution in XML document filtering. We also describe its integration in the context of the GridICE monitoring tool.

INTRODUCTION

In Grid computing, the monitoring activity plays the essential role of measuring important parameters in order to provide relevant information of a Grid system related to aspects such as usage, behavior and performance. One of the basic requirements for a monitoring service is the capability of detection and notification of fault situations and user-defined events. As regards this aspect, we describe the architecture and implementation of a flexible notification service designed to be incorporated, in a modular way, into a Grid monitoring tool.

A Grid notification service should be able to: (1) receive data from several resources, (2) filter them against a set of users specifications, (3) aggregate and customize filtered results and, finally, (4) deliver them only to interested users. A suitable model is represented by the publish/subscribe system based on event-driven mechanism and useful for distributed data, regardless the recipients identity or location. In such a model, the involved entities are publishers and subscribers that exchange messages through a broker; in a general way, messages from publishers are named events, while messages from subscribers are named subscriptions. The broker implements a filter algorithm in order to execute matching between events and subscriptions.

Today, the increasing success of XML as a standard for data representation and exchange over the Internet has led to a consequent increasing interest in filtering and content-based routing of XML data. Events are formalized as XML documents, while subscriptions are expressed through a language able to specify constraints over both events structure and content. After the description of both requirements and architecture, we present a multi-thread implementation of our notification service. We also report on the integration of this system with the GridICE [1], a distributed monitoring tool designed for Grid systems.

NOTIFICATION SERVICE REQUIREMENTS

In this section, we describe the main requirements for a notification service. The GridICE monitoring tool has been developed by INFN (Istituto Nazionale di Fisica Nucleare) [2] as part of the European DataTAG project [3], and then integrated into the Large Hadron Collider (LHC) Grid Computing middleware version 2 [4]. Later, it has been redesigned for better usability, stability and reliability in the context of the EGEE project [5]. The current notification service included in the GridICE production service provides basic capabilities, since it offers only the notifications of a predefined set of events (i.e., disk space depletion, i-node availability lower than a certain threshold, transition of a process from down to up and vice versa).

The growing interest for more flexible and scalable notification capabilities from LHC experiments has led us to study a more suitable solution satisfying their needs. The following set of requirements were inferred after having investigated a number of use cases: (1) enable each final user to define the precise set of events of interest; (2) final users should be able to add and delete notification requests; (3) the service should be able to filter all the events matching the defined subscriptions and deliver them to the interested users; (4) final users should be able to set the frequency at which notifications are sent (e.g., daily, hourly); (5) the service should be able to send the notifications using an asynchronous delivery model (e.g., publish/subscribe model).

Publish/subscribe systems represent an appealing solution for a possible communication paradigm in notification services. In particular, they enable to decouple recipients from senders both in time and space by using event-driven mechanisms and special function-

* on leave from NIPNE-HH, Romania
† also affiliated to INFN-LNL, Legnaro, Italy
‡ Contact author: natascia.debortoli@na.infn.it

alities for data distribution [6]. Among the various possibilities of filters and notification routing strategies that exist for this type of systems, in the recent years a growing interest has been observed in area of filtering and content-based routing of XML data. In an XML filtering system, events are represented by XML documents. According to the content-based routing approach, matching between events and subscriptions is a two-steps process: structure matching and predicates processing. This implies that it is meaningful to express the subscriptions using languages such XPath or XQuery that enable to specify constrains over both structure (path expressions) and content (value-based predicates).

A number of XML filtering algorithms and implementations have been proposed. Within the design and the implementation of our notification service, the adopted filter engine is YFilter [7]. It evaluates path expressions and predicates over streaming XML data via a Nondeterministic Finite Automata (NFA).

THE NOTIFICATION SERVICE ARCHITECTURE

This section provides a general overview of the notification service architecture based on the selected publish/subscribe system model and depicted in Figure 1.

It is important to define a set of key terms in order to outline the main tasks of the service: 'event', that is a piece of information received by the service (i.e., an XML document); 'subscriber', that is a user subscribing to the service in order to receive notifications about matching events; 'subscription', that is a standing request that expresses user preferences about a specific event; 'notification', that is a message containing the information related to a subscription and its matching events; 'filtering', that is the activity of matching between events and subscriptions.

There are several tasks that give a general overview of how a notification service should work: collect and store subscribers identities and related subscription data, collect event data, periodical match of events against subscriptions, process of filtered data in order to compose and send notifications to involved subscribers.

Publisher Module

The purpose of this module is to provide XML events that will drive the filtering algorithm as described in YFilter [7]. According to the different levels of abstraction in a Grid system (e.g., VO level, site level and operations domain level), several kinds of XML events are periodically generated. Each XML event sent to the Filter Engine is an abstraction of the information present in the monitoring data collection.

Subscriber Module

To compose and deliver notifications, the notification service must be able to get data about users and events of interest. The Subscriber is responsible for the management of this information and offers an API that enables collection of subscriber and subscription data. During the registration process performed by a potential user (the subscriber), information about the identity is gathered and a unique identifier together with a default profile are assigned. This profile contains the definition of the delivery frequency, that is the minimum interval between two notifications related to the user subscriptions. The service has no limit about the number of maximum subscribers.

Filter Engine Module

The Filter Engine is based on the YFilter implementation. The input are: (1) XML events generated by the Publisher module and driving the filter mechanism (each event is processed only once); (2) Subscriptions expressed by XPath and used to build the NFA. The Filter Engine outputs a set of filtered events together with their matching subscriptions.

Notification Manager Module

The Notification Manager module is responsible to compose end send notifications to the subscribers. It is based on a synchronous process that operates on the set of filtered events provided by the Filter Engine module. This module performs the following tasks: (1) first, it identifies the involved users, (2) than it aggregates the matched events and subscriptions, (3) after that it composes notification reports according to the users profiles and the subscriptions properties, (4) finally it sends the notification messages to the involved users.

IMPLEMENTATION DETAILS

In this section, we describe our design choices for the implementation of the notification service presented in this paper. The proposed implementation reflects the modularity of the architecture, requires a unique configuration file for the whole set of components and relies on the multithreading features of the Java programming language. For each module part of the notification service, we describe the internal details and we highlight how they interoperate to each other. We also explain how the notification service has been integrated with the GridICE monitoring tool.

Publisher Implementation

The Publisher is responsible for the generation of the monitoring events, therefore it must interact with

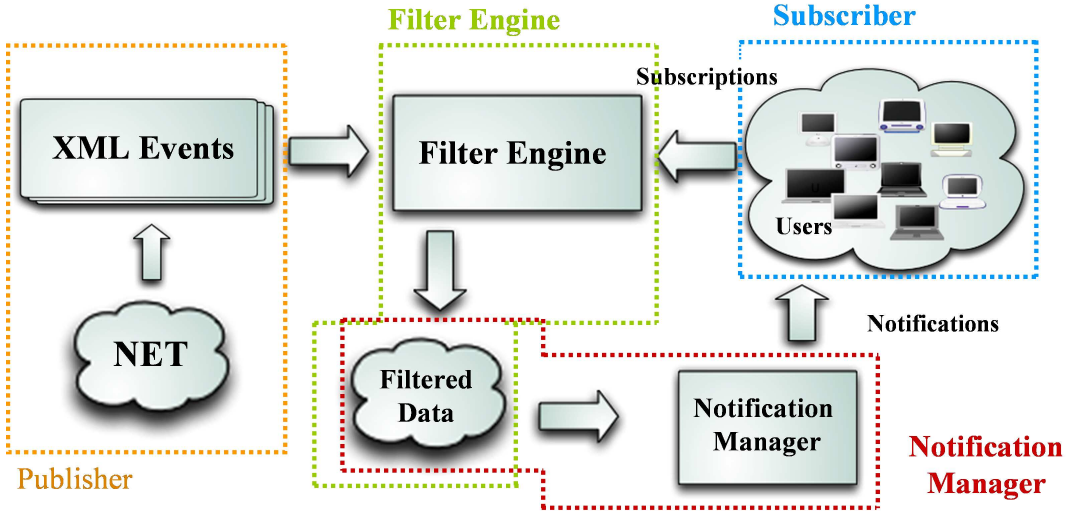

Figure 1: Architecture

some source of information. Our approach is to implement this service using a pull-based strategy that gathers XML documents reprenting monitoring information using the HTTP protocol. In order to accomplish its task, the Publisher requires the following information: the type of events to load, the target URL for the HTTP request and finally the frequency of event requests (i.e., frequency of HTTP GET operation for each event type). This information is specified in the configuration file as follows:

```
# Available event types
available_events=event_1,event_2,...,event_n

# Event frequences (as seconds)
event.event_j.freq=180

# Active events (i.e., they are being observed)
event.types=event_1,event_5

# Base URL for HTTP requests
event.baseUrl=http://GridICEhostname:port/baseURL

# Relative URL to get a specific XML document
event.event_1.url=dir_1/dir_2/page_1.php?attr_1=val_1
event.event_5.url=dir_3/dir_6/page_7.php
```

For each type of events, an indipendent instance of the Publisher is activated and performs requests according to the schedule defined in the configuration file. Each instance is responsible to provide for the Filter Engine module with the gathered set of events.

By integrating the notification service with the GridICE monitoring tool, it has been possible to access the following types of events: 'Site', i.e., summary information about computing and storage resources; 'GRIS', i.e., summary information about primary sources part of the Grid Information Service (GIS); 'Host Services', i.e., summary information about the status and characteristics of each monitored host together with the important processes part of Grid services (e.g., processes part of the Workload Manager Service or WMS) . An example of 'Site' event produced by the Publisher is given by the following XML fragment:

```
<SiteData Name="SiteUniqueID">
   <Country DnsCode="CountryCode">CountryName</Country>
   <QueuesNum>1</QueuesNum>
   <SlotsNum>2</SlotsNum>
   <FreeSlots>2</FreeSlots>
   <SlotsLoad>0</SlotsLoad>
   <GateKeeperNum>1</GateKeeperNum>
   <RunningJobs>0</RunningJobs>
   <WaitingJobs>0</WaitingJobs>
   <JobsLoad>0</JobsLoad>\
   <MonitoredHosts>3</MonitoredHosts>
   <BogoMips>5000</BogoMips>
   <WorkingNodesNum>2</WorkingNodesNum>
   <CPUNum>2</CPUNum>
   <CPULoad>20</CPULoad>
   <StorageAvailable>616144</StorageAvailable>
   <StorageUsed>759135</StorageUsed>
   <StorageTotal>1375279</StorageTotal>
   <StorageLoad>55</StorageLoad>
</SiteData>
```

Subscriber Implementation

The Subscriber is responsible for the management of both the subscriber and subscription data. This activity can be decomposed in two independent sub-tasks: the first is the management of insertion/modification or removal of subscriptions from users; the second is the periodical access to the available data in order to update the active set of subscriptions considered by the Filter Engine. As regards the implementation, the former task is currently performed by the notification service administrator upon receipts of requests from the interested subscribers; the latter task is called 'Subscription Manager' and verifies the presence of new, modified or deleted subscriptions on a

periodic fashion as defined in the configuration file:

```
# Time period between two executions of the
# Subscription Manager (as milliseconds)
subscriptionManagerFreq = 120000
```

An example of subscription related to the 'Site' event is:

```
/SiteInfo/SiteData[@Name="SITE_A"]
                  [RunningJobs[text()>"4"]]
```

This subscription states an interest for 'SITE_A' when the number of running jobs is greather than 4.

Filter Engine Implementation

The Filter Engine is responsible for the matching process between the set of subscriptions and the stream of incoming events. As stated in the introduction, we adopt YFilter [7], a filter engine designed and developed at the University of California at Berkeley. This component provides fast, on-the-fly matching of XML events against simultaneous subscriptions. Subscriptions consist of path expressions written in a subset of XPath 1.0. This task starts when event, subscriber and subscription data are available; then the Filter Engine continuosly evaluate arriving events against subscriptions. The matching process returns the set of filtered events, each one related with matched subscriptions.

YFilter APIs have been extended in order to include more advanced features. The Filter Engine is now able to continuosly receive data from both the Publisher and the Subscriber running instances; the YFilter subscription specification language has been extended in order to increase the kinds of recognizable subscriptions. Finally, the Filter Engine produces the set of filtered events and matched subscriptions for afterwards processing.

Notification Manager Implementation

The Notification Manager is responsible for composing and sending notifications messages to involved subscribers, according to their profile specification. This activity depends on the process of the filtered data provided by the Filter Engine. In a general way, a notification references a subscriber, his satisfied subscriptions and involved events. Moreover, notification is not composed and sent as soon as its content is identified. The subscriber define the 'notification interval', the interval time between two consecutively notifications. At each running time, only data that satisfy subscriber specification are processed in order to compose notification mail and then sent to the referenced subscriber. Remaining data will be processed, updated if necessary, and used to compose a notification mail as soon as subscriber's timing specification is satisfied. At last, each composed notification is sent by mail to involved subscribers.

CONCLUSION

In this paper, we have described the implementation of a flexible and scalable notification service, based on the model offered by the publish/subcribe systems. We also have given some details of its integration within the GridICE monitoring tool; with regard to this integration, a small efforts is needed to implement some new features satisfying inferred requirements, such as make available the notification service by a web interface.

REFERENCES

[1] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. L. Rubini, G. Tortone, M. C. Vistoli. GridICE: a Monitoring Service for Grid Systems. In Future Generation Computer Systems (FGCS) Journal, Elsevier, 21(4):559-571.

[2] Istituto Nazionale di Fisica Nucleare (INFN) - Italy. http://www.infn.it

[3] Research and technological development for a Data TransAtlantic Grid Site. http://datatag.web.cern.ch/datatag/

[4] LHC Computing Grid Web Site. http://lcg.web.cern.ch/LCG/

[5] Enabling Grids for E-sciencE Site. http://public.eu-egee.org/

[6] Eugster, Patrick Th;Felber, Pascal A;Guerraoui, Rachid;Kermarrec, Anne-Marie The many faces of publish/subscribe. ACM Computing Surveys (CSUR) Volume 35 N2, June 2003 , pp 114-131

[7] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. ACM Transactions on Database Systems (TODS). Volume 28, Issue 4. December 2003.