International Journal of Web Services Practices, Vol.1, No.1-2 (2005), pp. 158-166

Fine Grained SEDA Architecture for Service Oriented Network Management Systems

Shaurabh Bharti, Vikrant Kaulgud, Srinivas Padmanabhuni*, Akash Saurav Das, Venkat Krishnamoorthy, Naveen Krishnan Unni, Niranjan V.

Infosys Technologies Limited, Hosur Road, Bangalore, India 560100
{shaurabh_bharti, Vikrant_kaulgud, Srinivas_p, akash_das, venkatesan_k, naveenku, niranjan_v} @infosys.com

Abstract— **This paper presents a unique perspective of bringing service orientated architecture (SOA) to network management systems (NMS) with an aim to improve extensibility and flexibility, which are key requirements in current day dynamic environments. However, a key concern of scalability arises with bringing in of SOA, which we propose to address by incorporating SEDA, a scalable model for event driven architectures. Refining this further, we address the shortcomings of pure SEDA based event driven SOA approach by introducing the concept of Fine Grained SEDA (FGSEDA), which prescribes a finer level of granularity. We illustrate this model for fault management, a key NMS feature by delving in detail about implementation of such architecture with finer issues like thread management being explored in depth for an FMS as part of an overall NMS. We also discuss in detail the logical architecture of an implementation of an FGSEDA for a DSL network use case, leveraging Mule, an event driven SEDA based enterprise service bus, and highlight further extensions to this work.**

Index Terms— **Event Driven Architectures, SEDA, Network Management Systems, Service Oriented Architecture, Fault management System**

I. Introduction

THE last two decades have seen increased role of technology in the telecom sector. The network is evolving and with requirements for unified access, transports and voice switching. Additionally the network elements are now geared towards ferrying high-speed data and video. Further, with the continuous emergence of new technologies like Voice over Internet Protocol (VoIP), the underlying network elements keep changing. The high degree of complexity accompanying the network element technology necessitates rich network management systems (NMS) which can harness and control the usage of technology while hiding the inherent complexity. As operators begin to add new networks and expand existing networks to support new technologies and products, the necessity of scalable, flexible and functionally rich NMS systems arises.

NMS systems are crucial fort both day-to-day network operations management (higher staffing and operation-expenditure requirements) and strategic network growth planning. There exists an urgent need for a flexible, intelligent and scalable NMS architecture that can increase automation of network operations, while being able to provide long-term planning inputs. Another factor influencing NMS architectures is the clear trend of mergers and acquisitions among the key vendors. A key input into the successful integration of these various products into an NMS suite is the ease of integration, a key impediment in the traditional hierarchical NMS architecture. These growing demands of interoperability, flexibility and scalability fuel the need for an architectural framework that will address these issues seamlessly.

II. Network Management System Overview

A. *Elements of an NMS*

Elements of a NMS require at the minimum 5 areas of management, together termed as FCAPS, representing Faults, Configuration, Accounting, Performance and Security [8]. Each of these areas is described below.

1) Fault Management System (FMS) - This consists of all functions of the NMS related to network problems or faults that occur among the various elements of the managed network. The fault related functionalities include detection, notification, inference gathering/correlations, and possible automated rectification.

2) Configuration Management - This encompasses functionality dealing with configuration of network elements and their inter-linkages. This could include operations such as monitoring the configuration of a particular resource, providing interfaces for authorized operators to make configuration changes, logging and auditing of the configuration changes.

3) Accounting Management - This deals with the measurement of utilization of various network parameters associated with specific users or user groups. This monitoring and analysis of data enables the operator to charge the customer as per usage and customer's charge plans, and to understand capacity utilization of the network itself.

4) Performance Management System (PMS) - This deals with the measurement, aggregation, analysis and reporting of various aspects of network performance. The analysis of this data enables reactive and preventive maintenance, capacity planning and future capacity projections.

5) Security Management - This deals with controlling

*Corresponding author

access to all the resources within the network using prescribed

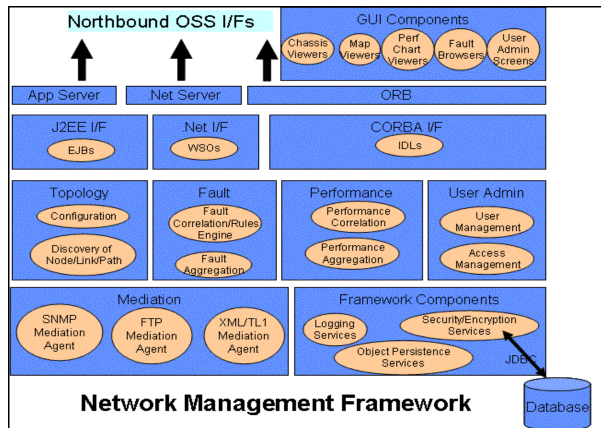

Figure 1. Traditional NMS Architecture

access control rules and guidelines to prevent unauthorized access. This encompasses authentication and authorization of users/groups that verifies the user credentials and permits operations on the various resources.

A typical NMS system has component(s) that provide functionality in the above areas, while drawing inputs from the network elements being managed as well as other functional components within the NMS.

B. Traditional NMS Architecture

The various modules that typically form the NMS are:

1) Framework Components: These provide basic platform functionality, including persistence, logging, and security (authentication, authorization and encryption).

2) Mediation Components: These components implement protocol specific interface functionality on the *Southbound* interfaces to communicate with the network elements. The list of components depicted in figure 1 are representative, and the specific protocols implemented may change based on the NMS requirements. All access to the network elements happens via these components.

3) Topology Components: These provide functions related to topology and configuration of network elements. These include discovering network elements, making this information available to other components, as well as adding/modifying/deleting the network element configuration.

4) Fault Components: Components in the module relate to fault collection, aggregation, correlation and reporting. These components interface with topology components for object and resource information, and with the mediation components to collect low-level network information.

5) Performance Components: These components collect, analyze, and report performance metrics collected from the network elements. The analysis components in these modules frequently interact with the topology components for resource information, as well as with the fault components for reporting of discrepancies and other performance issues.

6) Northbound Interface Components: These components implement functionalities on the *Northbound* interfaces to Operations Support Systems (OSS) applications. The list of components illustrated in figure 1 are illustrative, and the specific protocols implemented would change based on the OSS requirements.

7) GUI/HMI Components: These components provide the user-interface into the framework. They consist of user interaction screens that enable a user to query as well as perform operations on the various modules in the NMS.

The traditional NMS architecture depicted above has a hierarchical structure with information and process flows typically involving a GUI/Northbound Interface component (User input), a Core functionality component (one of Topology, Fault or Performance) and a Mediation component.

C. Limitations of the Traditional NMS Architecture

The traditional NMS architecture as depicted in Figure 1 places the described modules in a hierarchical order. This hierarchy represents a typical network-centric view of management with information flowing up from the network elements via a set of mediation and data transformation components. However, there are multiple limitations of this architecture:

1) Complex Information and Process flows: Ever-expanding management requirements from service providers as well as growing complexity in the network itself, has given rise to scenarios where the information and process flows in an NMS are much more complex and involve multiple components in the same hierarchical layer. A typical example of this is a *Service Correlation* module that is a mandatory component for a modern NMS. In a typical service correlation process flow, there are multiple functional components across fault, performance and topology domains that need to provide/consume information in order to achieve the end result (of the assessment of customer impact of network faults). The traditional hierarchical NMS architecture does not lend itself to easy modeling of these flows. In order to facilitate this, typical workarounds involve building of point-to-point interconnections between various modules. These workarounds lead to NMS modules tightly coupled to each other thus increasing the cost of maintainability, reusability and extensibility.

2) Interoperability: Another area where current NMS architectures are weak is interoperability. The NMS domain has traditionally never had universally accepted standard communication protocols. This is especially true of internal communication between NMS components, where standards-based component interaction has not been the norm. With the current trend of mergers between NMS software products, this lack of standardization causes very costly integration overheads of product components. Due to the inflexible and rigid nature of the traditional hierarchical NMS architectures, the time and effort spent in integrating a new system component is extremely high.

3) Scalability: With the deployment of new services like VoIP in networks, the number of network elements that need to be managed increases exponentially. Due to this, the scalability requirements from an NMS system have changed dramatically. There is a need for dynamic event collation, load balancing, coupled with near-real-time management of

internal request to resources. Current frameworks and systems do not address this optimally. The requirements of an adaptive framework to support huge volumes of requests based on priority is not inherently built into the architecture, and usually ends up being added at high development cost.

The proposed event driven service oriented architecture (EDSOA) attempts to overcome these limitations.

III. Event Driven Service-Oriented Architecture and SEDA concepts

A. Service Requestor, Service Provider and Service Registry

Service-oriented architecture (SOA) is a result of a great deal of research and development over the past few decades in distributed computing. SOA implementations revolve around the basic idea of a *service.* A service refers to a modular, self-contained piece of software, which has a well-defined functionality expressed in abstract terms independent of the underlying implementation. Any implementation of SOA has three fundamental roles: Service provider, Service requestor, and a Service registry. SOA involves three fundamental operations: publish, find and bind. The service provider *publishes* details pertaining to service invocation with a services registry. The service requestor *finds* the details of a service from the service registry. The service requestor then invokes (*binds*) the service on the service provider. The service registry is sometimes also referred to as the service broker, because it acts as a service broker between the requestors and providers. Service brokers and registries may be optional in many mainstream SOA implementations.

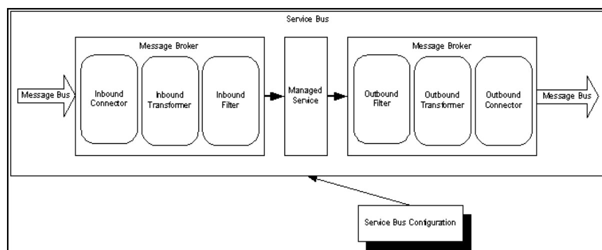

Figure 2. Elements of a Service Bus

A loosely coupled SOA implementation offers independence between the different participants so that each can act independently without requiring significant changes when one participant undergoes any change. Availability of service proxies and factories obviates the need for plumbing work on the part of the service requestor to configure the service provider, resulting in loose coupling between the provider and the requestor. This enables seamless interchange and extension of service-provider implementation code without affecting the service requestor implementation.[9]

B. Event Driven Architecture and Enterprise Network Service Bus

Event Driven Architecture, is a generic term used to represent architectures, where the core concept is that of an event, where a software monitors or is notified of the occurrence of an event. Event driven architectures should deploy mechanisms to detect these events, and act in response to these events. The detection could be in form of an approach where a small piece of software polls and waits for something to happen, or it could be based on a notification from the event generator.

An Enterprise Network Service Bus (ENSB) is a medium that provides the necessary plumbing required for event-driven interaction between various services implemented in disparate platforms such as .NET, J2EE, Web Services, C++ etc. This loose coupling enables seamless replacement and additions of the service implementation code as well as the service implementation platform and the remote protocol. An ENSB is based on message broker pattern [7] and the message bus pattern [6], and it facilitates this kind of loose coupling by providing an infrastructure for event-driven interaction between services thus removing the service consumer's dependencies on the service provider's implementation

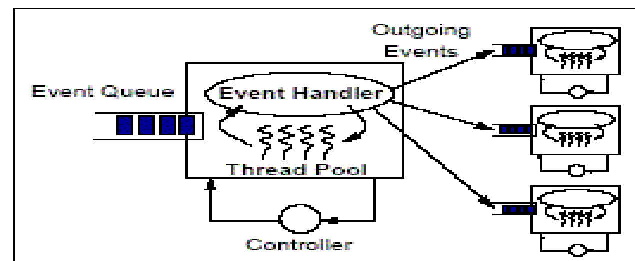

Figure 3. A SEDA Stage[2]

specific details.

Message Brokers act as inter-connects between the services managed by the service bus and between external applications and managed services. They consist of connectors, message transformers and message filters, which respectively externalize the protocol transformation logic, event data-format transformation logic and event filtering logic from the implementation logic of the core service functionality. This design makes the service implementation focused on business logic and hence better reusable.

Message bus provides the channels over which the events are transmitted between managed services and between external applications and the managed services. Examples of channels include JMS[10], SOAP[11] etc. The connectors are specific to each Message Bus.

The ENSB configuration consists of settings for inbound/outbound connectors, inbound/outbound event transformers and inbound/outbound event routers for each of the managed services. At start-up, the service bus reads this configuration information and all the managed services are wired up accordingly to communicate with each other.

C. Staged Event-Driven Architecture

A common approach to handling scalability is multi-threading, which handles concurrency by virtualizing the operating system resources available to a thread. However, this approach hides the fact that the system resources available are limited and shared, eventually resulting in a degradation of performance with increasing load. An alternate approach is to

provide control over the resource management to the application itself, thereby allowing it to performance tune itself. Staged-Event-Driven-Architecture (SEDA) [2] provides a blueprint for such an approach. In SEDA, applications are constructed as a network of stages, each with an associated incoming event queue and a pool of event handler threads.

Each stage in SEDA represents a robust building block that may be conditioned to load by threshold or filtering its event queue. Moreover, the granularity of a stage in SEDA may depend upon the complexity of stage itself. To account for stages with high load in the system, we further suggest breaking each SEDA stage into sub-stages. The criteria for creating sub-stages for a component are:

1) The stage involves many sub components, with each sub-component itself being complex.

2) The stage handles a heavy functional and computational load.

3) The sub-stages should be as much *decoupled* as possible. The partition should ensure *minimum message-passing* between sub-stages. They should be more or less *functionally independent* Each sub-stage should *share the load* in balanced way. Partition should ensure some sub-stages are not heavy loaded. Otherwise, it will degrade overall performance.

This refinement of SEDA architecture, we term as Fine Grained SEDA (FGSEDA), can improve certain performance metrics like scalability of stages, load conditioning etc. It helps applications to make informed scheduling and resource-management decisions as per varying request loads. This also puts insight into how SEDA can be used not just for a whole system, but also for certain high load parts of it.

In particular, we illustrate in context of an NMS, how FGSEDA may be implemented in Fault Management System only, without transforming the other components of traditional NMS architecture into SEDA stages.

D. Case for Service-Oriented Architecture based NMS Framework

This section aims at establishing the key design and architecture criteria required for a robust, scalable NMS. It correlates this with the standard characteristics of a SOA and makes the case for the suitability of SOA for designing a flexible and scalable NMS.

1) NMS as Event-Driven Systems: NMS s are primarily event-driven software subsystems that are modeled by workflow definitions. These workflows are triggered by events that are generated either from the *Southbound* side (i.e. the network elements or element management systems) or from the *Northbound* side (i.e., from the NMS users or from the OSS systems that interface with the NMS). Additionally, the triggered workflows, as part of their definition, require that the various NMS modules such as fault, configuration and performance support inter-module service requests. An event-driven architecture such as SEDA with its support for inbuilt message/request queues lends itself very well to design of a NMS system.

2) Scalability: Another key requirement for an NMS is scalability and performance. Traditionally, these requirements are addressed using two broad methods

a) By optimizing application design using high performance algorithms and data structures.

b) By deploying the application on higher end hardware, employ clustering and other hardware augmentation techniques. Though this is an effective solution, this is a brute force and a cost prohibitive approach.

While these are effective approaches to architect systems that manage small and medium sized managed networks, they face difficulties when required to scale up to manage large sized networks (upwards of 10000 network elements) in a cost-effective manner. This is partly due to the resource virtualization approach, followed in traditional NMS architectures. A SEDA based design approach, optionally coupled with application clustering, will provide a more adaptable solution for dynamically scaling up, while at the same time, offering consistent performance.

3) Integration and Extensibility: The network management domain is dealing with ever-changing network-element technologies. Due to this constant change, one of the key design requirements is to have a flexible and extensible set of external interfaces (Northbound and Southbound), as well as core business component interfaces that are loosely coupled to each other. Traditionally, this has been a failing of most NMS systems, and has resulted in extensive rework/enhancement requirements for supporting a new NE/OSS interface, or integrating a new business component from a different niche product. An SOA based architecture that inherently facilitates loose coupling and *pluggability* of new interfaces will significantly enhance the long-term value of the implemented NMS. This extensibility will also allow the system implementers to follow an evolution-upgrade approach, rather than a big-bang approach.

IV. Service-oriented architecture based NMS framework

A. EDSOA based NMS architecture

The traditional NMS architecture depicted in the preceding section is transformed into an SOA-based framework, with services that offer well-defined and scalable interfaces to the NMS functionality. The diagram in Figure 4 shows such an architecture. Each element has an incoming queue, and interacts through a common *Core NMS Service Bus* (ENSB). The key features this event driven SEDA based NMS architecture are:

1) *Core network service bus:* At the core of the NMS is the ENSB based service bus, which is responsible for managing the lifecycle of the services in the system. The bus consists of a combination of a broker and a message bus as already discussed in detail. The bus manages a lost of core services including the *Fault Service, Topology Service*, and the

Performance Service.

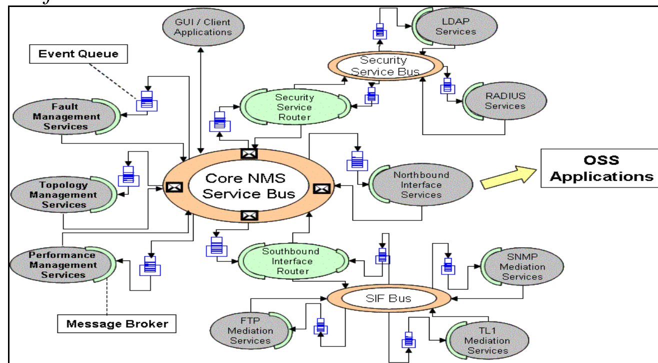

Figure 4. Elements of a EDSOA based NMS

(2). Southbound Interface Bus : This bus manages the various mediation services including TL/1 mediation service, SNMP mediation service, and FTP mediation service.

(3). *Northbound Interfaces*: The northbound services are responsible for the interface between OSS/BSS systems and the NMS.

(4). *Security Service Bus:* This bus manages the multiple security services like LDAP service, Radius Service, Encryption service etc.

B. Benefits of the EDSOA based architecture

The key benefits accrued by the SOA based NMS architecture can be outlined as:

(1). Cost savings by reuse: Many of the existing functionalities in existing NMS systems can be brought into use by wrapping an appropriate service layer. Likewise, infrastructural investments like LDAP directories, etc. will be reused with security services.

(2). Interoperability: By adoption of SOA based architecture, it is possible for multiple NMS systems to interoperate in a seamless manner.

(3). Scalability: Each of the managed services on the service bus is considered as a stage in SEDA. An event queue, an adaptive resource controller and event dispatcher threads are associated with each of them. The event queue may be an in-process (in-memory) queue or an out-of-process queue such as a JMS server. This addresses scalability issues in a very comprehensive manner.

(4.) Enhanced Security Paradigm: Another feature of the architecture is the integrated and extensible Security paradigm. The Security Service Router enables the framework to adapt easily to any external security framework mandated by the Service Provider

V. Fine Grained SEDA

A. Bringing Fine Granularity in to SEDA

A key issue in SEDA systems is *granularity* of the stages. While in some high performance activities in a SEDA environment, a stage may simply not be the right choice of granularity, as discussed earlier. To tackle the scalability of thread management, alongside providing more autonomy to applications in SEDA, we introduce the notion of Fine Grained SEDA (FGSEDA) where we propose breaking of a stage in SEDA into sub-stages. Each sub-stage represents a SEDA implementation of a critical functionality of the parent SEDA stage. For example, consider a parent SEDA stage having four critical functionalities. If the stage performance can be improved by *micro-managing these functionalities*, then each of these four functionalities can be implemented as SEDA sub-stage individually. Hence, the application, instead of having a single stage with competing threads from these functionalities, will now have fine-grained sub-stages. Each of the fine-grained sub-stages will now host threads performing a much smaller set of tasks. This design has the potential to improve application performance significantly.

The major criteria for creating sub-stages for a component are:

1) The stage involves many sub components, with each sub-component itself being complex.

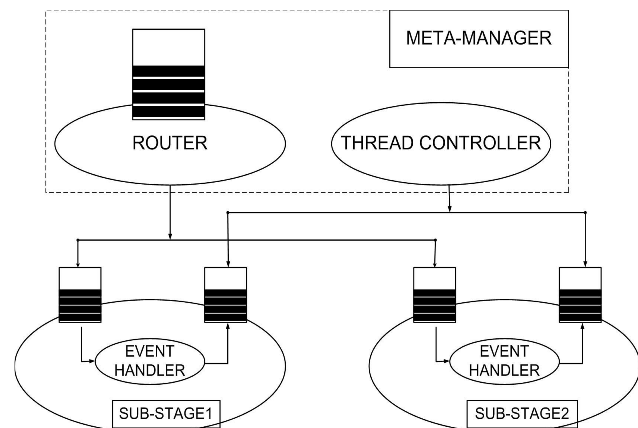

Figure 5. A Typical FGSEDA Stage. Thread controllers consume threads from thread-queues of sub-stages and execute them. Router handles event handling between sub-stages and other components of NMS, as well as between sub-stages. Together, they constitute the Meta-Manager(MM).

2) The stage handles a heavy functional and computational load in a NMS.

Incorporating fine-graininess in a SEDA stage requires additional components for building a system. Two issues that need to be tackled are, how to wrap the sub-stages and present a unified interface to the remaining system and secondly how to manage internal consistency between the sub-stages. It is pertinent to note that the sub-stages *collectively* provide the functionality of earlier single parent SEDA stage,

In conventional SEDA, controllers play a key role in managing events and threads. However, when we break the SEDA stage into sub-stages, merely a controller will not be sufficient to manage the complexity and coherence of different sub-stages.

To address the issue of managing and maintaining coherence across sub-stages, we propose the idea of a separate meta-manager (MM) for each FGSEDA stage.

A detailed view of an FGSEDA stage is shown in Figure 5. It contains its *sub-stages*, a *meta-manager(MM)* and an *event-queue*. Each sub-stage contains two queues : a separate *event-queue* and *thread queue*. Event-queue of a sub-stage contains all events to be consumed by the sub-stage. Threads to be executed from each sub-stage are queued in its thread-queue. Events meant to be consumed by the stage are stored in its own event-queue. A *router* parses the events in the event-queue of the stage and routes them to event-queues of appropriate internal sub-stage. Similarly, an output message/event from a sub-stage would be routed to the external system through *the router*.

Hence, unlike traditional SEDA stages, FGSEDA stages do not contain *event-controllers* but *routers.* Similarly, no *thread-controller* is needed for sub-stages. Sub-stages don't run their threads on their own. Instead, threads from each sub-stage are kept in its *thread-queue* (instead of a *thread-pool [2]*). *Thread-Controller* of the stage selects threads from those thread-queues, and finally execute them.

MM is made up of *Router* and *thread*-controller. MM may have more complex functions. MM can prioritize selection of threads from different thread-queues. This may help in *load conditioning* when events for particular correlators dominate., The FGSEDA stage is more robust than traditional SEDA stages. Since events are *quickly demuxed* to sub-stages, new events need not wait for old events to be consumed (which takes more time to empty the queue). This helps the stages accept higher load. Moreover, sub-stages may need to communicate between each other. MM would also manage all *intra-communication* needs of its sub-stages. They may be regarded as events, which are routed to other sub-stages with higher priority (as needed).

The suggested approach offers many advantages to NMS systems such as reusability of system components, extensibility to new components, ease of integration with other systems, scalability and enhanced security.

VI. FGSEDA applied to Fault Management in NMS

A. Fault Management in NMS

In the current architecture of NMS, each element is taken as single stage of SEDA. This implies threads from different stages are taken with equal probability for execution. However, among them, FMS and PMS accounts for the major loads in NMS. In addition, particularly FMS is complex involving multiple correlators and other sub-components. Optimal coordination between the correlators and sub-components is required for speedy fault localization and rectification. Therefore, it is wise to partition FMS to enhance its performance. We are leaving other elements as single complete stage, as they do not put heavy loads on NMS.

In this paper, Fault Management System (FMS) is taken as example to show how such architecture can be implemented. FMS is responsible to detect, diagnose and report all problems due to network faults. We will show that it is intuitive to partition such a system into three equivalent correlators, which can be implemented as SEDA sub-stages.

B. Need for three correlators

Event correlation is a one of the most important function of FMS. It involves linking between different *resources* and services. Resources consist of network elements (switches etc.) as well as system elements (server etc.). Service providers provide services to customers. Resources share dependencies upon each other. Services are dependent upon sub-services (low-level services) as well as resources (at finer level). Traditionally, event correlation involved only resources. After gathering faults (also termed as *resource events*), Root cause analysis (RCA) is run on the resource events to find *root* faults. Root faults are used to find all affected services, and appropriate alerts are issued. This kind of correlation may be termed as *Resource Correlation*. This can be referred to as a *bottom-up approach* to event correlation.

The increasing complexity of NMSs places the requirement of being able to locate and resolve faults as fast as possible. Moreover, if a resource fails, many other dependent resources are not likely to respond. This creates huge alarm flood to resource correlation. It becomes computationally intensive task to run RCA on them to find the root fault, and fix them.

Another approach to the same problem is to diagnose from responses. If a fault occurs at resource level, it is likely to be reflected on the *response* of the dependent services. Malfunctioning services give faulty outputs, and hence clients/service providers take a notice of them and generate alerts. According to changes in the *responses* and output of services, it is possible to track service faults down to root resource faults using dependency graph of services.

Services depend upon sub-services. Service Correlation uses hierarchical dependencies to locate faulty sub-services. In turn, sub-services can be used to locate antecedent resources,

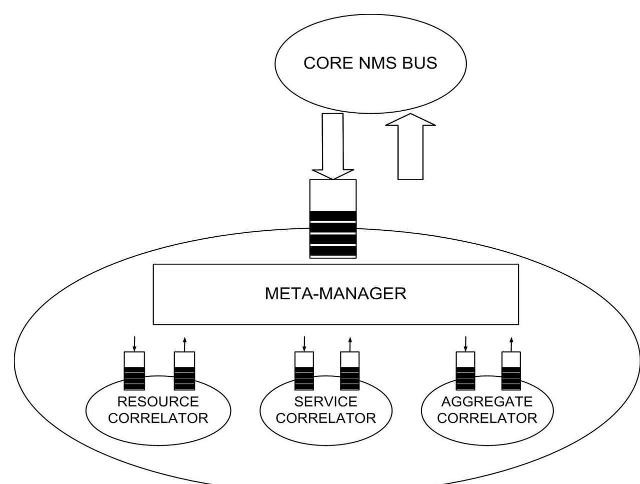

Figure 6. FGSEDA on FMS illustrating the SEDA sub-stages and other components of a FMS FGSEDA stage

which are likely to cause malfunctioning of the sub-service. Strong convergence can be achieved if most of the dependent services respond. This kind of correlation is termed as *Service*

Correlation. This may be regarded as *top-down approach*.

In actual scenario, both *resource correlators (RC)* and *service correlators (SC)* are present in FMS. RCs identify all malfunctioning higher services, and SCs help in finding the sub-services/resources that are likely to be the cause of the faulty services. Yet another correlator maps both results to each other. This is needed for two reasons: firstly to further reduce the number of faults, and secondly to locate the cause of faults precisely. This kind of correlation is termed as *Aggregate Correlation.* Sometimes, it could put heavy load on the system.

C. Fine Grained SEDA on FMS

An intuitive SEDA-based FMS design is to consider each of the fault components as a SEDA stage. However, as discussed earlier, to enhance performance, we suggest FGSEDA, which will involve breaking a stage into sub-stages.

. In the FGSEDA based architecture, FMS is broken into three major elements as shown in Figure 6 :

1)Resource Correlator
2) Service Correlator
3) Aggregate Correlator

Each correlator can be regarded as one FGSEDA sub-stage, which contains an *incoming queue*. The purpose of the queue is to decouple each correlator from others. All three queues receive messages from a *Router*. A Router is needed to parse the messages coming in the queue of FMS from other NMS stages/components. According the type of message, it is sent to the RC/SC/AC queue respectively. The design of the router will not be complicated. For example, all messages related to discovery of *resource faults* are directed to RC queue. Similarly, complaints from clients about faulty services can be directed to SC. A *proper message format* can even ensure partitioning of messages with just a demux.. All messages from correlators are simply directed to the router, which accordingly sends them to queues of other NMS stages.

A meta-manager maintains the coherence of the overall FMS system. *Priority* of correlators has to be maintained while execution of their threads. This will help conditioning the load as needed. For example, in case of alarm floods, RC may need more CPU time than AC or SC. SC might need to run complicated test cases, especially when RC fails to diagnose/resolve some faults. It wraps up all three correlators and presents it as single stage to other components of the NMS.

Finally, Router and Meta-Manager can be merged into single system. It may be called *FMS Interface.*

We are currently implementing this FGSEDA based FMS prototype in Mule [5], as described in the next section. We are in the process of identifying the relevant deployment issues and performance metrics to be captured. We anticipate significant gains by FGSEDA approach as opposed to plain SEDA approach.

D. Prototype Implementation

A DSL network is used to show a use-case of Service Correlation. The NMS consists of a presentation tier for administration console to be used for managing different events by administrators. The architecture of the system is shown in Figure 7. Any typical DSL network will have various network elements, as can be seen in [15]. The main components of this NMS implementation are:

(1)Fault Management System : This is implemented as a FGSEDA stage. It contains three major parts, namely Router, Resource Correlator (RC) and Service Correlator (SC). Event alerts from Mule ESB to FMS is passed to Router through FMS Queue. Router passes them to Resource Correlator via RC Queue. RC runs Root Cause Analysis (RCA) on the events. Drools Rule Engine is used to store rules governing dependencies of various events. Root events are passed to Service Correlator. Service Correlator issues alarms to different consumers that consume the affected services. Service dependencies upon these events are also stored in Drools rules engine.

(2)Open Fault Query System: It's an interface to query details of different events, service alarms etc.

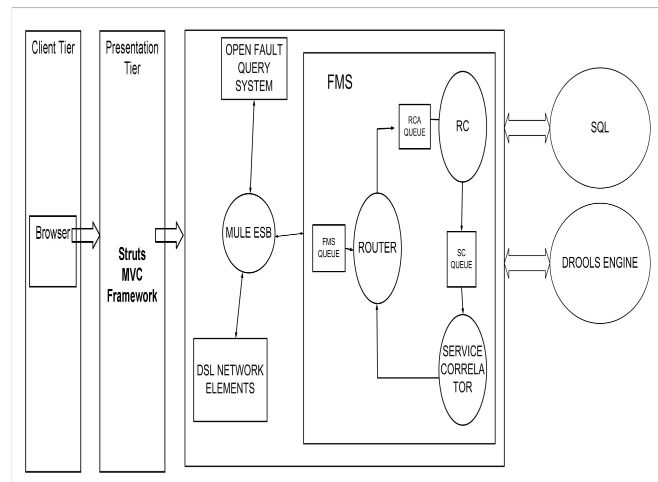

Figure 7. Architecture showing various components of the prototype system. It consists of a DSL network and an NMS implemented using FGSEDA architecture.

(3)Mule ESB [5]: This works as the ENSB for system. All message exchanges between network and FMS happen through Mule.

(4) SQL Database : It stores all details regarding raw events as well as processed events (using FMS).

(5)Drools Rules Engine[20] : It maintains rules governing relations between different kind of events.

(6)Presentation Tier: The presentation tier for administration console leverages Struts[19], the popular MVC framework.

E. Extensions to this work

1) Hybrid Architecture

This approach takes a more pragmatic view of the problem and rebuilds only those FCAPS components that need to be SEDA-based. Here the component under consideration is rebuilt in to a SEDA stage according the criteria for FGSEDA.

This component is then interfaced with the rest of the traditional NMS. The fine-grained SEDA stage uses Router and Meta-Manager (See Figure 5), that not only manage the internal sub-stages, but also provide a mechanism to interface a SEDA stage with the external system. This interface can be designed as per the external system characteristics. For example, in case of traditional NMS, the external system will be non-SEDA. Router and Meta-Manager need to work a non-SEDA system for achieving successful interfacing.

To explain this further we consider the case of a FMS implemented using FGSEDA architecture to enhance performance of FMS. The structure of this FGSEDA FMS is presented as *single* stage to external traditional NMS using the FMS Interface. The FMS Interface helps in managing execution threads and message sharing. Therefore, it can be installed even in traditional NMS systems, where other components are not implemented as SEDA stages. This may be termed as a *hybrid SEDA* NMS Architecture, where a fine-grained SEDA FMS is inserted in traditional NMS systems. However, issues like thread-management need to be handled for optimal performance of FMS. An internal message queue can be maintained to account for a SEDA queue. This implementation would have high value for deployment of SEDA architecture in current NMS systems. It will be easier for Service Providers to incorporate this architecture into their systems, instead of, implementing a full-blown SEDA based NMS that will require lots of changes in the system.

2) WS-Notification and WS-Eventing Standards

As the proposed NMS is based on event-driven architecture, most of the interactions are notifications based. However current ENSB implementations use proprietary bus and broker architectures to achieve event-orientation. However, just like standardization of protocols like SOAP have happened under the purview of Web services stack, work is also underway to standardize event-driven Web services in standards like WS-Notification[12] and WS-Eventing [14]. As these standards are currently still not standardized, conventional ENSB implementations are still proprietary. Specifically, WS-Notification allows different parts of system to be notified (and hence invoked for action) automatically, depending upon the nature of event. For example, certain critical events can invoke special modules, which puts them higher in the processing queue. This can reduce effort in root cause analysis. By using a notification broker, different modules of system can register for notifications. As the format of event-notifications become standardized based on WS-Notification/WS-Eventing, modules can register for specific events according to need. It increases flexibility of the system. Implementing event-driven nature of system can be made easier and interoperable by using WS-Notification/Eventing standards compliant middleware including ENSB implementations.

F. Deployment Issues

SEDA expects local clustering of components inside a stage. This is also important to decrease overhead of internal thread management. However, this is posing constraints to systems to be deployed on the same hardware unit, which shall be addressed by indirection via Mule[5], which will allow FMS components to interact on an ENSB. This may account for the not having clustered components.

G. Performance metrics

We are studying the impact of the architecture on scalability of the FMS. Particularly we are interested in studying the performance impact obtained by creating sub-stages for each correlator as against the single-stage FMS. Also of interest would be the scalability of the Router component in terms of the concurrent messages it can handle. In terms of latency, we intend to study the impact on the aggregate correlation latency.

VII. Related Work

Conventionally proprietary vendors have dominated NMS systems with limited interoperability though they supported standards like SNMP etc. These standards limited the integration capabilities of these conventional systems and provided limited interoperability. Though the emergence of web saw appearance of managed NMS systems leveraging the web, it still did not address true integration requirements of NMS systems. XML has appeared in recent literature and products as a key technology enabling interoperation of different NMS systems, and flexible NMS configurations. However, the integration challenge with other systems like legacy systems has only recently been tackled by adopting Web services [18][17]. Phillippe et al. [18] demonstrate the role of Web Services in extending JAMAP (Java Based Research prototype of a management platform), a research platform for standards based network management. Bin et al. [17] propose a web services based model of scalable distributed network performance management, but do not address the full applicability of generic SOA based infrastructure. Venkatesan et al. [1] proposed usage of SEDA based event driven Service oriented architecture for tackling the flexibility, and scalability required in tackling the integration requirements of today's NMS systems. In this paper, we further extend this work by introducing fine-grained SEDA stages into the core framework proposed in [1].

VIII. Conclusion

Implementing NMSs using Service Oriented Architecture makes them reusable and easy for integration and extensibility. SEDA architecture can be used to enhance its performance and scalability. As some of the components like FMS are bigger in NMS, further partitioning using SEDA is intuitive. Towards that end, we proposed FGSEDA that involves breaking of each SEDA stage into smaller sub stages. Further, we briefly provided insights towards incorporating SEDA in traditional systems. Hybrid architecture involving a combination of traditional and FGSEDA, or traditional and SEDA, ensures that bigger parts of NMS can be implemented using SEDA without affecting other systems. This can have high deployment value for current Service Providers.

References

[1] Venkatesan Krishnamoorthy, Naveen Krishnan Unni, and V. Niranjan. 2005. "Event-Driven service oriented architecture for an agile and scalable network management system". In *Proceedings of the first International Conference of Next Generation Web Services Practices.* Seoul, Korea, August 2005.

[2] M. Welsh, D. Culler, and E. Brewer. 2001. "SEDA : Architecture for well-conditioned scalable internet services". In *Proceedings of the 18th Symposium on Operating Systems Principle.* Alberta, Canada. Oct 2001. pp 230-245.

[3] Andreas Hanemann, Martin Sailer, David Schmitz. 2004. "Assured service quality by improved fault management". *In Proceedings of the 2nd international conference on Service oriented computing*. New York, USA. 2004

[4] Boris Gruschke. 1998. "Integrated event management : event correlation using dependency graphs". *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems :Operations and Management.* Newark, DE, USA. Oct 1998. 130-141.

[5] MULE. http://mule.codehaus.org/. Codehaus. (accessed Nov 12, 2005).

[6] *Message Bus Pattern.* http://www.eaipatterns.com. (accessed Nov 15, 2005).

[7] *Message Broker Pattern*. http://www.eaipatterns.com. (accessed Nov 15, 2005).

[8] *Network Management Systems : Best Practices.* Cisco Systems. http://www.cisco.com/warp/public/126/NMS_bestpractice.pdf. ((accessed Nov 09, 2005).

[9] Jeff Hanson. 2005. *Event-Driven Services in SOA.* Java World. http://www.javaworld.com/javaworld/jw-01-2005/jw-0131-soa.html. (accessed Nov 09, 2005).

[10] *Java Message Service*. Sun Microsystems. http://java.sun.com/products/jms/docs.html. (accessed Nov 03, 2005).

[11] *Simple Object Access Protocol*. World Wide Web Consortium. http://www.w3.org/TR/soap/. (accessed Nov 10, 2005).

[12] *WS-Notification.* IBM. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-notification/. (accessed Nov 19, 2005).

[13] *Ws-Addressing.* World Wide Web Consortium. http://www.w3.org/Submission/ws-addressing/. (accessed Nov 19, 2005).

[14] *WS-Eventing*. IBM. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-eventing/. (accessed Nov 19, 2005).

[15] *Common DSL Architecture.* Cisco Systems. http://www.cisco.com/univercd/cc/td/doc/product/dsl_prod/gsol_dsl/dsl_arch/gdsl.pdf. (accessed Nov 17,2005).

[16] John Shugart. *CNET Streaming Architecture.* http://www.appliedglobal.com/content/white_papers/ACFA847.pdf. (accessed Nov 20, 2005).

[17] Zeng Bin, Hu Tao, Wang Wei, and Li ZiTan. 2004. "A model of scalable distributed network performance management". *In Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN 04)*, pp.607

[18] Jean Phillippe Martin Flatin, Pierre Alain Doffoel, and Mario Jeckle. 2004. "Web Services for Integrated Management: A case study". In *Proceedings of the European Conference on Web Services,* September 27-30, 2004. pp September 27-30, 2004.

[19] Struts. http://struts.apache.org . Apache. (accessed Nov 12, 2005).

[20] Drools. http://drools.org . Codehaus. (accessed Nov 12, 2005).

Author Biographies

Shaurabh Bharti is a Junior Research Associate in Web Services Centre of Excellence group in SETLabs, Infosys Technologies, Bangalore, India - 560100. He can be reached at +91-9880570430, (e-mail: shaurabh_bharti @infosys.com)

Vikrant Kaulgud is a Senior Research Associate in SETLabs, Infosys Technologies, Pune, India - 411027. He currently works in P2P systems, mobility and wireless technologies. He can be reached at +91-20- 22978280, (e-mail: Vikrant_Kaulgud@infosys.com)

Srinivas Padmanabhuni heads the Web Services Centre of Excellence group in SETLabs, the R and D unit of Infosys Technologies Ltd., Bangalore, India - 560100. He can be reached at +91-80-51059507, (e-mail: srinivas_p@infosys.com)

Akash Saurav Das is a technical specialist with Web Services Centre of Excellence group in SETLabs, Infosys Technologies, Bangalore, India - 560100. He can be reached at +91-20-51059501, (e-mail:akash_das@infosys.com).

Venkatesan Krishnamoorthy is a technical specialist with Web Services Centre of Excellence group in SETLabs, Infosys Technologies, Bangalore, India -560100. He can be reached at +91-80-28520261, (e-mail:venkatesan_k@infosys.com).

Naveen Unni is with the Product Engineering division with Infosys Technologies, Bangalore - 560100. He can be reached at +91-80- 28520261, (e-mail: naveenku@infosys.com)

Niranjan V is a technical architect with Web Services Centre of Excellence group in SETLabs, Infosys Technologies, Bangalore, India - 560100. He can be reached at +91-80-28520261, (e-mail:niranjan_v@infosys.com).