# Towards Highly Available Three-Tier Monitoring Applications (Extended Abstract)

David Breitgand, Gleb Shaviner, and Danny Dolev

School of Computer Science and Engineering,
The Hebrew University of Jerusalem,
Givat-Ram 91904, Jerusalem

E-Mail: {davb, gleb, dolev}@cs.huji.ac.il

**Abstract.** Numerous important network management applications such as usage-based accounting, trend analysis, performance management, fault management, and others perform application-specific network monitoring tasks as part of their activities. This paper presents a work in progress that is aimed at providing a maximally transparent infrastructure increasing the availability of such management applications by improving the failure behavior of their constituent monitoring components.

**Keywords:** monitoring**,** high-availability middleware**,** CORBA, group communication, three-tier client/server architecture, network failures

## 1 Introduction and Motivation

The traditional two-tired structure of the management applications based on the rigid client/server paradigm with thick managers and thin target agents is being rapidly abandoned. This is motivated by the severe scalability and availability limitations of the traditional approach that is essentially centralized [1]. The fast progress that has been made recently in mobile code and distributed middleware technologies [11] makes more flexible architectures, such as the popular three-tier one, both technically feasible and attractive also for the management applications [2], [3].

Abstracting away the specific details a typical three-tier distributed management application looks as following. The target agents constitute the lowest tier and serve as the source of the management data. The monitoring components are dynamically dispatched at the middle tier. They monitor the target agents and other monitoring components accumulating and pre-processing the information collected from them. The consumers of this information constitute the uppermost tier.

It becomes increasingly popular to facilitate the communication among the entities residing in the first and second tiers using some middleware technology, such as CORBA [5], that hides away the heterogeneity of the network providing for smooth integration. Usually the middleware offers an abstraction of the *object bus* over which remote method invocations are performed.

The distributed three-tier organization of distributed applications offers considerable advantages [11]. In the new scheme, the overall availability of a management application is increased since different components of it may be executed at different network locations, and a failure of one of them would not imply the immediate unavailability of the whole service. However, since the dependencies among various components usually exist, even a single failure of a crucial component may bring the whole management application to a halt, or render it inaccurate.

As an example consider the following possible organization of a usage-based IP billing application that relies on its own network monitoring components to collect the needed information. The structure of this application is shown in Figure 1. In the lowest tier of the applications there are target agents capable of providing the raw information about their respective devices. In the second tier there are three types of monitoring components: flow collector, session collector, and flow preprocessor. The flow collector component monitors the forwarding devices and obtains the raw data about the IP flows from them directly.

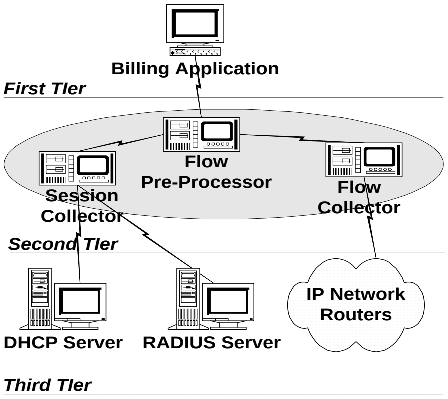


**Figure 1:** 3-tier application

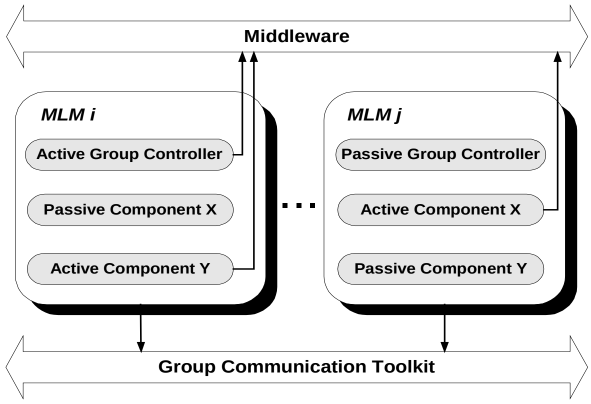


**Figure 2:** Structure of a HA-MLM

The session collector component monitors some Authentication, Authorization and Accounting server(s), such as RADIUS [12], collecting the user-session information. The flow pre-processor component monitors the flow collector and the session collector in order to correlate the users with their flows and reports this pre-processed information to the upper tier. The upper tier of the application consists of the billing component that generates the per-user bills.

Although a failure of any component at the second tier does not imply the *immediate* stoppage of *all* the application activities, the billing application, as a whole cannot proceed in a regular manner and eventually would halt, or become inaccurate if no additional measures are taken. For example, if the flow pre-processor component fails then at some point the flow collector will run out of the storage space and will be forced either to loose valuable information, or to seize its activity.

In light of the above we identify a clear requirement for increasing the availability of the critical monitoring components being part of the management applications. Given the complexity of handling distributed three-tier applications in a malicious environment prone to various network failures, it is both important and challenging to provide a maximally transparent infrastructure that allows a manager to deploy the needed monitoring components of the second tier in a highly available manner. This would improve the overall failure behavior of the management applications, and therefore contributes to the better provisioning of the network services in general.

In this paper we present an infrastructure that facilitates highly available application-specific monitoring components. Our infrastructure transparently handles very complex failure scenarios including network partitions and host failures. The deployment of the infrastructure requires only minor additions to the application code. Currently, we are not aware of any other proposal offering the similar features.

## 2 Architecture Overview

In the heart of the proposed infrastructure is the run-time environment for the second-tier entities, i.e., for the monitoring components. This run-time environment is provided by a set of *highly available mid-level managers (HA-MLMs)* where each HA-MLM being a *group* of daemon processes called *mid level managers (MLMs).* Figure 2 depicts the internal structure of a HA-MLM. Each MLM runs on a different host.

HA-MLMs are created and deleted on demand and are organized into a logical hierarchy. There is a single HA-MLM in the root of the hierarchy and all other HA-MLMs are created as either a sub-group within this HA-MLM or some of its descendants. The memberships of different HA-MLMs may overlap.

Member processes of a HA-MLM communicate among themselves using a group communication service that provides the partitionable membership semantics, the virtual synchrony execution model and the reliable FIFO message ordering [4], [7]. These features are common in today’s group communication services. In the future we expect the group communication to become an integral part of the standard middleware technology [8], [9], [10].

Although being physically a set of processes, HA-MLM appears as a single logical entity to the outside world. One of the HA-MLM members is assigned the function of a *group controller*. The group controller runs a special thread that exports the HA-MLM's interface over the middleware object bus.

The monitoring components of a management application are dispatched for execution within some HA-MLM through the HA-MLM interface provided by its group controller. At any given time any monitoring component is active at only one MLM from the HA-MLM. This MLM is termed *host*. It is also guaranteed that in case the HA-MLM is split due to the network partitioning, then only one component replica may be active at a time in any given network partition.

The host MLM executes a monitoring component as one of its threads. Other MLMs have the component's code and are kept updated about the component's *state* that is accumulated during its execution. When failures occur, the remaining MLMs detect them through the group membership service provided by the group communication tool. They transparently take over the failed counterparts, assuming the responsibility of the components whose hosts failed. This way the monitoring components remain available in spite of failures.

Restarting a failed component on a different MLM requires a consistent replication of the component state among the HA-MLM's constituent MLMs. Monitoring components are not involved in the replication details. Instead they periodically use the special interface provided by the host MLM requesting the synchronization of their states. This request is the only addition required for making a monitoring component highly available. The state is represented as an object whose internal layout and semantics are not interpreted by the MLMs. This object is simply propagated by the host to other MLMs in the HA-MLM upon a request using the group communication system.

During its execution a component may interact with the external entities. Each such interaction is termed *external even*t. In order to provide for consistency of the monitoring activities, the external events corresponding to the interactions between the entities from the first and second tiers and also among the peers at the second-tier are transparently controlled through the *interceptors* mechanism [5]. In particular, the external events may be delayed until some safety conditions are satisfied. These safety conditions are defined in such a way that the original semantics of a monitoring application concerning the external events, which generally is either *at-most-once* or *at-least-once*, will be preserved. Interceptors are available in the popular middleware technologies such as CORBA.

## 3 Benefits of our Approach

Our proposal offers the following additional advantages to the three-tier management applications:

- The crucial monitoring components are replicated for high availability transparently to themselves and to the external entities that use them;
- In case of the network partitioning when the monitored devices become available in different network partitions, multiple replicas of a monitoring component can continue the operation in a disconnected mode with one replica per partition; upon a re-merge the most updated, consistent state of a monitoring component with respect to all of its monitored devices can be obtained.

## 4 Consistency Maintenance Problem

The maximal transparency principle used in our approach makes the consistency maintenance of the components challenging. The full version of the paper investigates this problem in detail, provides the algorithm for solving it, and proves its correctness. We only outline the problem itself here.

Basically, it is two-fold. As a result of even quite simple failure scenarios a) more than one replica of a monitoring component may concurrently communicate with the same set of external entities, and b) the state of some replicas may contain the information referring to the same monitored devices, collected by replicas concurrently from the different network partitions. As will be shown in the full version of this paper both these sub-problems have feasible solutions.

## 5 Current Status

A prototype monitoring system based on our infrastructure is now being implemented in Java. We use CORBA-compatible Visigenic ORB [6] for Windows NT and Transis group communication toolkit [7]. The usage-based IP accounting management application serves as our primary test application.

## References


[1] Y. Yemini, *The OSI Network Management Model*. IEEE Communications Magazine pages 20-29, May 1993.

[2] Akhil Sahai and Christine Morin, *Towards Distributed and Dynamic Network Management*, Proc. NOMS'98, New Orleans, Louisiana, USA, February 1998.

[3] M. Feridun, W. Kasteleijn and J. Krause, *Distributed Management with Mobile Components*, Proc. IM'99, Boston, MA, USA, May 1999.

[4] ACM, *Communications of the ACM 39(4)*, special issue on Group Communications Systems, April 1996.

[5] Object Management Group, *Description of New OMA Reference Model*, OMG Document ab/96-05-02, May 1996.

[6] Inprise, *Visibroker for Java 3.1*, http://www.inprise.com/visibroker.

[7] Y. Amir, D. Dolev, S. Kramer and D. Malki, *Transis: A Communication Sub-System for High Availability*, Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing, July 1992.

[8] Silvano Maffeis, *Adding Group Communication and Fault-Tolerance to CORBA*, Proceedings of the USENIX Conference on Object-Oriented Technologies, Monterey, CA, June 1995.

[9] S. Mishra, L. Fei and G. Xing, *Design, Implementation and Performance Evaluation of a CORBA Group Communication Service*, to appear in the 29th IEEE International Symposium on Fault-tolerant Computing, Madison, WI, June 1999.

[10] OMG, *Fault Tolerance RFP*, orbos/98-04-01, April 1998.

[11] Robert Orfali, Dan Harkey, and Jeri Edwards "Client/Server Survival Guide", 3rd edition, John Wiley & Sons, 1999

[12] RADIUS, IETF RFC2138, RFC2139