A Business Continuity Monitoring Model for Distributed Architectures: A Case Study

Ewart J.H. Nijburg
University of Liverpool
United Kingdom
E-mail: ewart@nijsoft.com

Raul Valverde
Concordia University
Canada
E-mail: rvalverde@jmsb.concordia.ca

Abstract

Computerized services are the driving force behind every day business for many companies, it is of the utmost importance that these services are available during business hours because downtime costs serious money. Most of the computerized services today are based on a distributed architecture because of the many benefits of such an architecture. There is a downside to distributed architectures though; distributed architectures have an incomplete observability problem resulting in tough decision making and difficult control of the system build according to the architecture. This paper describes a design of a business continuity monitoring model, developed to cope with software, hardware, and operator failures by reducing the time required to detect, diagnose, and repair a problem in a distributed architecture. It is based on a three-tier model combined with five monitoring domains distilled from a standard distributed architecture. A prototype was developed to test the model in a real environment.

Keywords: Monitoring systems, distributed architectures, fault tolerant systems, distributed models, reliability.

Introduction

Computer systems are becoming more and more important to companies, for most of them, computer systems are the driving force behind every day business and unavailability of those systems, also known as downtime, results in huge losses in revenue and goodwill (Gartner 1999a). Historically, a computer system was comprised of only one machine and the software was executed linear, in a single process. The needs for resource sharing, openness, concurrency, scalability, fault tolerance, and transparency have resulted in the use of a distributed architecture in virtually all large computer-based systems today (Sommerville 2001). The complexity of distributed systems has made it difficult to detect problems and control the behaviour of the system in contrast to traditional centralized systems (Hoffner 1994c, Kramer 1997, Gartner 2001b).

In distributed computer systems, it is often hard to turn a component off or restart it because it is not known what has been connected to it over time (Venners 2003) and if one component stops functioning, it can bring the whole system down or even worse cripple the system without being detected. The primary reasons why a distributed architecture is more complex than a traditional centralized system are because a distributed architecture has no central point of control, no central point of observation, no central source of information, no central point of decision making and incomplete observability (Hoffner 1994a). Controlling a distributed system from a single point implies that the information required to make decisions is available from that same point. The process of gathering this data is called monitoring.

2. Problem Definition and Scope

Monitoring distributed architectures reduces the dynamic complexity of such architecture and therefore provides better insights into the system under observation. Reducing the complexity of distributed architectures aids root cause analysis in failure situations because it becomes clear where a problem originates, which on its turn helps to prevent unnecessary downtime. The aim of this research is to design a business continuity model, which will allow system administrators to observe a distributed architecture from a single point, and which will provide an insight into the dynamic complexity of a distributed architecture, with the added goal of reducing the downtime of the distributed architecture through early failure detection.

3. Research Methodology

In order to create the business continuity model, a greater understanding of the issues on monitoring distributed systems needs to be achieved, for this a review of monitoring terminologies, protocols and existing models is required.

The literature review will be used as the basis for a business continuity monitoring model and through an iterative prototyping phase both the design and the prototype monitoring system will be improved. The final design and prototype will be validated through a case study. The case study will look at what the strengths and weaknesses of the design are and if the design can actually reduce downtime.

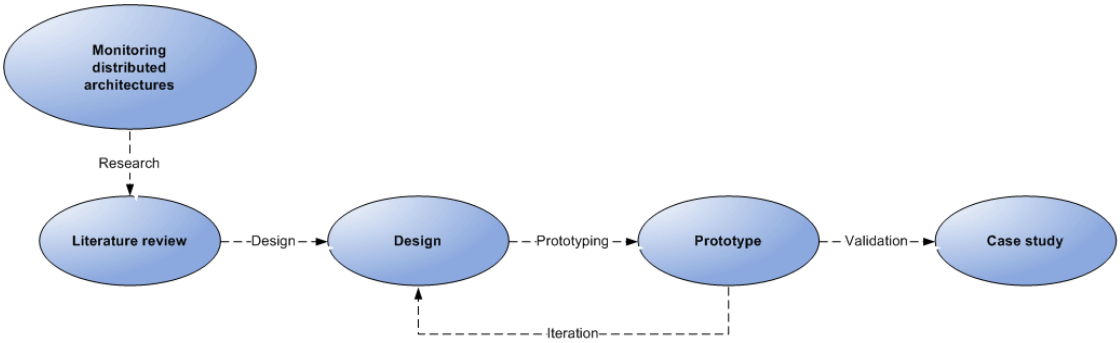

Figure 1: Research Methodology

4. Existing Monitoring Models

Many researchers have investigated the requirements of a business continuity monitoring model, in the following paragraphs five models are discussed that can be used to monitor distributed architectures.

4.1 Generic Monitoring Model

Mansouri-Samani and Sloman (1992) have created a monitoring model that they based on the Event Management Model of Feldkuhn & Erickson (1989). The model identifies four activities performed in a loosely coupled, object-based distributed system:

- **Generation**: Important events are detected, and event and status reports are generated. These monitoring reports are used to construct monitoring traces, which represent historical views of system activity.
- **Processing**: A generalized monitoring service provides common processing functionalities such as merging of traces, validation, database updating, combination / correlation and filtering of monitoring information. They convert the raw and low-level monitoring data to the required format and level of detail.
- **Dissemination**: Monitoring reports are distributed to users, managers or processing agents who require them.
- **Presentation**: Gathered and processed information is displayed to the users in an appropriate form.

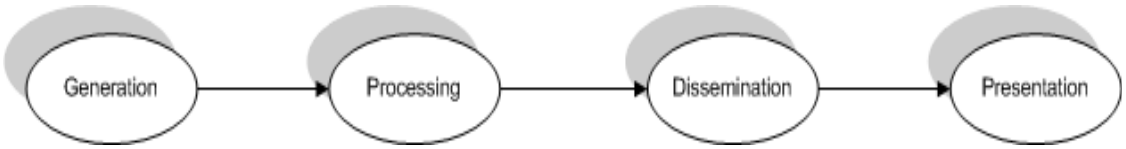

Figure 2: Generic Monitoring Model (Mansouri-Samani 1992)

4.2 General Monitoring Architecture

The General Monitoring Architecture has been defined by Zoraja, Rackl and Ludwig (1999). It is centred on what they believe to be three important concepts of monitoring; the event-action paradigm, a client-server architecture and layering.

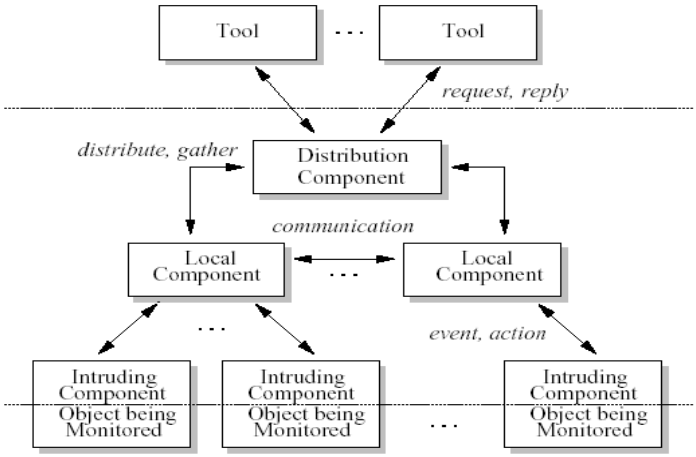

Figure 3: General Monitoring Architecture (Zoraja et. al 1999)

Like the Generic Monitoring Model from Mansouri-Samani (1992), this model identifies several activities in the monitoring process. These activities; Collecting, Processing and Presenting have been divided through the use of a three-tier client server architecture.

4.3 Multi-Layer Monitoring Model

The Multi-Layer Monitoring model has been created by Rackl (1999, 2000) as the basis for the Middleware Monitor (MIMO). MLM represents a basis for the systematic monitoring of complex middleware environments, it gathers data on several abstraction levels of a system and serves as an information source for all kinds of tools.

The MLM model is primarily designed as a model to monitor middleware technologies. The system to be monitored consists of six abstraction layers from which the monitor collects information and provides it to tools through the tool-monitor interface. All information gathered by the monitoring system is classified according to this layered model. For various middleware platforms, like DCOM, CORBA and JAVA RMI, this abstract model can be mapped to entity types related to the middleware environments.

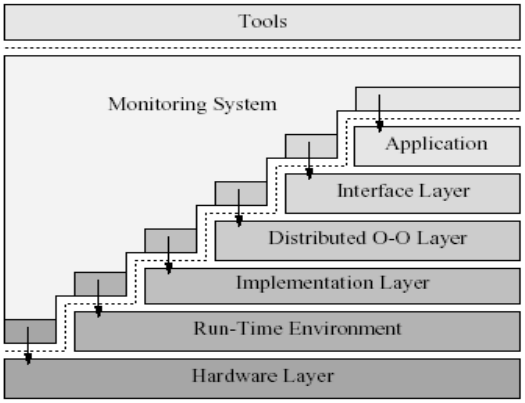

Figure 4: Multi-Layer Monitoring (Rackl 1999, 2000)

4.4 Hierarchical Filtering-based Monitoring Model

The Hierarchical Filtering-based Monitoring Model has been created by Al-Shaer (1999b) as the basis for the HiFi monitoring architecture. In this monitoring architecture, the task of detecting primitive and composite events is distributed among dedicated monitoring programs called monitoring agents (MA). HiFi has two types of MAs: local monitoring agents (LMA), and domain monitoring agents (DMA) (Figure 18). 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 that are beyond the LMA scope of knowledge. Within a machine, several producer entities are connected and provide information to the LMA. Every group of LMAs is attached to one DMA. DMAs are also connected to higher DMAs to form a hierarchical structure for exchanging the monitoring information.

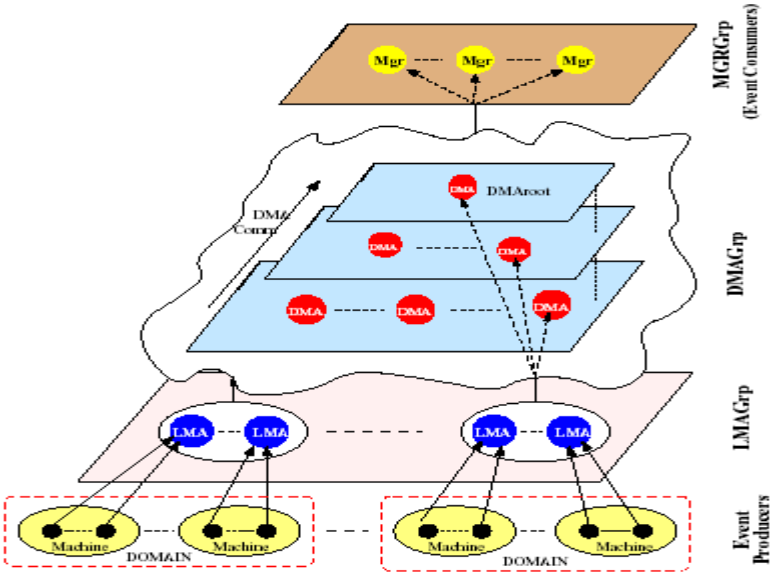

Figure 5: Hierarchical Filtering-based Monitoring Model (Al-Shaer 1999b)

4.5 OSI Event Reporting Model

The International Standards Organization (ISO) has developed a series of standards for the management of communications systems called Open Systems Interconnection (OSI). OSI defines managed objects as a representation of a managed resource, it uses the Common management Information Protocol (CMIP) to provide the primitives for supporting management operations that permit managers to control remote managed objects, query state and receive notifications (Mansouri-Samani 1992). The OSI Event Reporting Service performs both the processing and dissemination functions of the monitoring model in that it is responsible for both filtering of event reports and dissemination of the selected reports to chosen destinations. The components that constitute the service are shown in figure 6

Notifications are received by a local event detection and processing component which adds the event time, originating object class and instance to the notification message to form a potential event report. These are than sent to all local Event Forwarding Discriminators (EFDs).

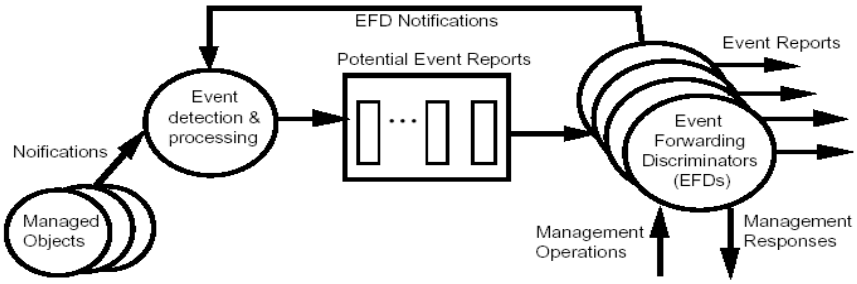

Figure 6: OSI Event Reporting (Mansouri-Samani 1992)

Existing models clearly show that monitoring on its own is a distributed activity which is divided into separate activities. Some of the models define three monitoring activities while others define four, but in general all the monitoring models define at least an event generation activity, an event processing activity, and an event presentation activity. The local monitoring requirement justifies the use of a three-layer model because monitoring information needs to be transported beyond machine boundaries. Except for the middleware-monitoring model, the models mentioned in the previous paragraphs focus on the activities of the monitoring process and they do not provide insights into which elements of a system should be monitored. As monitoring activities alone are not enough to build a monitoring system, further investigation in required to identify the monitoring domains of a distributed architecture.

5. Business Continuity Monitoring Model

This section introduces a business continuity monitoring model that extends the models mentioned by including the monitoring domains of a distributed architecture. The model inherits the four monitoring activities; generation, collecting, disseminating, presentation of events from the existing models and models these activities using three monitoring phases; collecting, processing, and presentation.

In a distributed architecture, monitoring the server tier and database tier is more important than monitoring the client tier because in general the latter resides outside the control of the organization on a separate network and only consumes the service.

Analysis of the server and data tier reveals that four monitoring domains can be identified;

- **Servers** – the actual hardware which is used to host the software that provides the service;
- **Database** – the hardware and software that is used to achieve data persistency;
- **Middleware** – the glue that binds different components into a service;
- **Network** – the physical connection between components and tiers.

Besides the four monitoring domains, one additional domain can be identified: **Software**. Software seems to be part of the server domain, which is comprised of hardware and software, but software has different monitoring needs compared to hardware. Hardware is very static and only requires a limited amount of monitoring, as opposed to software that is very dynamic and requires extensive amounts of monitoring in the form of tracing and/or sampling in order to cope with the complexity. Because of the huge difference in monitoring requirements, software is identified as a separate domain.

The model proposed in this research is depicted by figure 7 and combines both the monitoring activities as well as the monitoring domains, each domain is responsible for collecting the relevant information and sending it to the domain monitor by means of events. The domain monitor receives all the information and passes it on to the monitoring manager for dissemination to the management tool in the presentation layer.

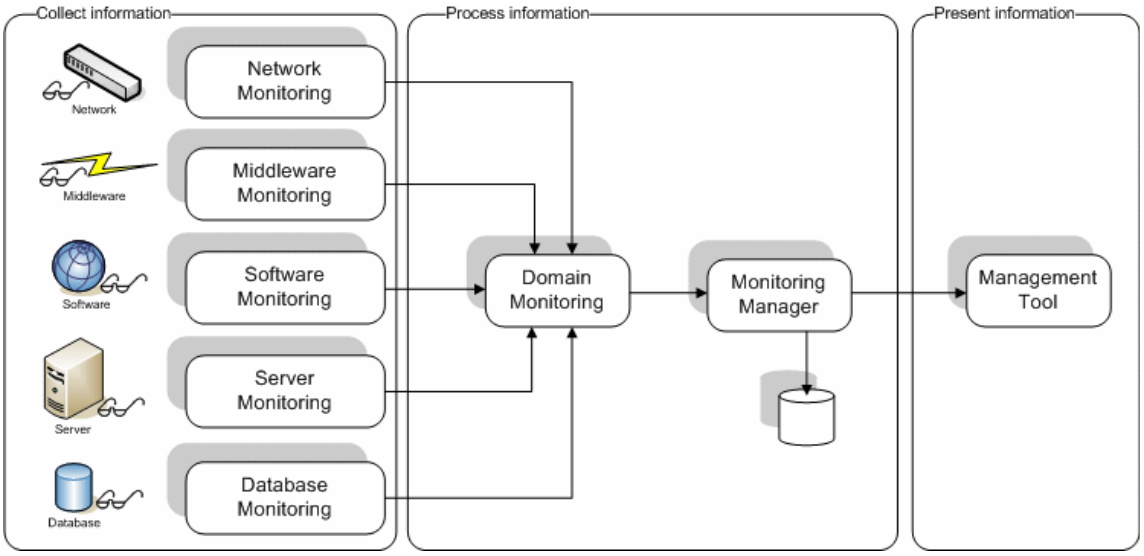

Figure 7: Business continuity monitoring model including monitoring domains

The four monitoring domains can be defined as:

- ***Network monitoring:*** Involves observation of the appliances used to create the networking infrastructure, typical parts of the infrastructure are switches, repeaters, bridges and routers. Monitoring these appliances enables early detection of failure, congestion problems and excessive bandwidth utilization.
- ***Middleware monitoring:*** Involves monitoring the communication between software. This communication can be either server-to-server, client to server or server to database in case the database is built on top of the middleware as well.
- ***Software monitoring:*** Involves two parts, the first part is observing the resources consumed by and allocated to processes in order to detect excessive use of resources. The second part of software monitoring involves software specific information such as errors, warnings and counters.
- ***Server monitoring***: Involves observing the hardware it contains and the operating system it is running. Typical elements that are monitored are resources the system can run out off such as disk space, memory, and CPU.
- ***Database monitoring***: Involves observing the behavior of the database, besides management problems such as low disk space and excessive CPU usage; the monitor should be able to detect abnormalities in query processing and duration.

Each monitoring domain has different monitoring requirements because they are built on different models of computation and engineering. The network domain for example most likely consists of COTS products which will commonly implement the SNMP protocol for monitoring. The software domain however most likely consists of tailored and COTS products combined and often implement no monitoring, or a propriety monitoring solution that is hard to generalize. Separating the different computational models into domains has the following advantages:

- Collecting information becomes modular, new domains can be added without affecting the functionality of the existing domains;
- A single domain can focus on a specific task, each domain can use its own specific logic for collecting data and then send it to the domain monitor;
- Collection information can be done close to the object under observation because the object is the only responsibility of the monitoring domain.
- Monitoring volume can be limited to only the events of interest and access to the objects can be maximized.

The task of the domain monitor is to receive all the events from the collection layer and translate them into a single event format that can be processed by the monitoring manager. Translating the events into a specific event format has the following advantages:

- Events of interest may occur at different parts of the system. Structures and processes that collect and order the information from the observed events are therefore necessary.
- Monitoring domains can provide their information in a variety of protocols to the domain monitor as long as the domain monitor is familiar with the protocol and knows how to translate the protocol into the final event format. Typical protocols, which the domain monitor should be familiar with, are the SNMP, WBEM, and CMIP standards.
- Observed events appear in a form which is not amenable for immediate use by the monitoring manager
- The volume of observed events may be such that it overwhelms the observer

The monitoring manager is responsible for dissemination of the generalized events to the presentation layer and since all events pass through the monitoring manager it is the ideal place to keep an audit trail of the events happening in the system under observation.

6. Case Study

The case-study system selected is Alex, an online stockbroker in the Netherlands. Alex has a complex distributed software architecture behind the services they provide, it represents a complicated case which can be generalized to other cases. Alex utilizes state of the art technologies such as Windows 2000, XP and 2003 server; Oracle 9i; Symmetric Multi Processing utilizing 2 processors; hundreds of servers; middleware; and most software consists of tailored applications written by their own development team. Software failures, hardware failures and operator errors are known to exist at Alex and the management of Alex would like to see the amount of downtime reduced to 0.5% or less.

A tool from Lucent Technologies called VitalSuite® is used for hardware and network monitoring, two tools from VERITAS, Indepth™ for Oracle and Inform for Alerts, are available for database tuning and monitoring, and the tailored software produces operational traces, performance traces, and error traces in the format of log files on the local machines. Monitoring requires a lot of information, a great deal of that information can be obtained from the information sources already available at Alex. Implementing the business continuity monitoring model at Alex (figure 8) requires that all of the 5 identified monitoring domains are able to provide their information to the processing layer, next the processing layer should be populated with the right tools for dissemination of events to the presentation layer and finally the presentation layer should be populated with a management tool.

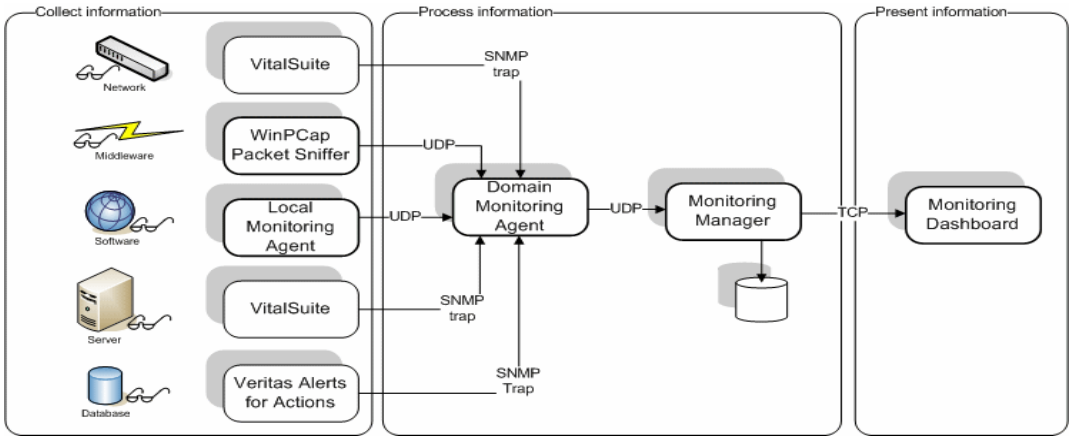

Figure 8: Alex specific monitoring design

Collecting information about the network and servers is one of the primary tasks of VitalSuite, the tool uses SNMP to retrieve information from the networking appliances such as switches, routers, and SNMP agents installed on the servers. Likewise, Inform for Alerts utilizes Indepth™ for Oracle to gather information about the state of the database. Both tools can send SNMP traps when a value drops below or rises above a predefined threshold, effectively limiting the events to only those of interest to the domain-monitoring agent. Because the tools satisfy the requirements for the domains and are already purchased by Alex they are reused.

The tailored software has already been instrumented in order to generate traces. The currently installed sensors generate a vast amount of data which is not useful in the processing layer, a software monitoring agent on each machine between the software and the processing layer will be used to filter the traces and send the events of interest. This will result in a minimal information flow between the two layers. Alex uses a messaging oriented middleware layer called TIBCO Rendezvous (TIBRV), this middleware layer uses UDP and TCP for communication. Communication between two components in the distributed architecture is based on a request reply scheme. The request is broadcasted using UDP to all components and through a consensus algorithm one of the receiving components agrees to process the request, when the component finishes the request it sends the reply back using TCP.

Valuable information can be distilled from these request reply messages as they can be used to chart the communication flows within the distributed architecture. TIBRV provides a way to receive the UDP traffic, but the TCP traffic is private to both components and cannot be read directly. As the request is a broadcast to all components and it is unclear which component is going to process the request, the request message cannot be used to chart the communication flow. Instead the TCP reply message should be used but that message is private and cannot be read using the standard TIBRV programming interface. In order to be able to receive the TCP reply messages an OSI level 7 packet sniffer will be used to peek at all TIBRV communications, both TCP and UDP. Each TIBCO message contains a header field that has several valuable fields. Both the request and reply messages have an "INBOX" field that indicates to whom the reply message should be send. By matching this "INBOX" field we know which reply belongs to which request and we can successfully chart the communication flow in the distributed architecture.

The entry point of the process information layer is the domain monitoring agent, besides being able to understand the propriety information from the middleware and software monitoring domains, it will be able to process SNMP traps from the network, server and database monitoring domains. After the agent has translated the incoming event into a homogeneous format it passes the event on to the event disseminator. The event disseminator forwards the events to all the subscribed clients in the presentation layer and stores the events they disseminate to the subscribers in a separate database as for audit purposes. The monitoring dashboard is the actual user interface the administrator is using for monitoring. It shows all monitoring domains and the elements under observation. If an event is received from the disseminator, it displays it to the observer.

7. Results

Analysis of the incident reports at Alex have shown that there are not many outages, but if one occurs it goes undetected for quite some time. Monitoring information isn't accessible and there aren't enough detection mechanisms in place to detect the problems, or the tools that are available aren't put to good use. A second issue emerging from the incident reports is that the time to repair is often quite large because it isn't clear where the problem originates and employees have to manually consult several monitoring tools and other sources of information before a solid problem statement can be delivered.

One incident report at Alex involves their quote distribution system; one of the components responsible for answering inquiries about quote information was not functioning properly causing some inquiries to fail while others were successful. The quote information servers, or Quote Request Servers (QRS) as they are called at Alex, are redundant servers that utilize a load-balancing scheme in order to divide the load amongst the servers. In this particular case the load balancer did not know of the faulty server and kept routing inquiries to the server, causing every fifth inquiry to return corrupted data. Upon investigation, it became apparent that one of the processes was not functioning as expected because the error trace for that process on that local machine contained numerous error messages. The software detected the problem at the moment it occurred, but no action was taken because nobody was notified that the failure had occurred. Active monitoring would have enabled administrators to disable the component and therefore prevent the load balancer from sending inquiries to that server. A prototype was implemented to test the proposed model. A simulator for collecting the information from the components was implemented as part of the prototype. This simulator has been used to replay the Alex incident, this time with active monitoring in place. Figure 9 shows how the process failure that was triggered manually in the simulator has propagated into a server failure that is sent to the domain monitor.

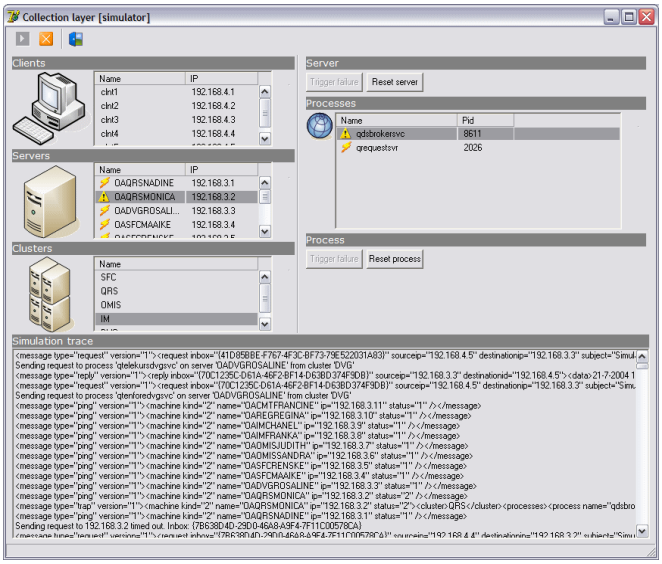

Figure 9: Simulating a process failure

In this simulation of the Alex incident, immediately after failing the qdsbrokersvc component, a state change is sent to the prototype's domain monitor who on its turn forwarded the event to the presentation layer to notify the administrator. The presentation layer received the event that was forwarded by the domain monitor within 2 seconds.

The presentation layer of the prototype consists on a monitoring dashboard that has three views; one represents the state of the servers based on their clustering, the second presents the state of individual servers, and the third shows a graph of the request-reply messages in the system. After the dashboard processed the event, it was clear which server had an erroneous state and therefore needed to be disabled.

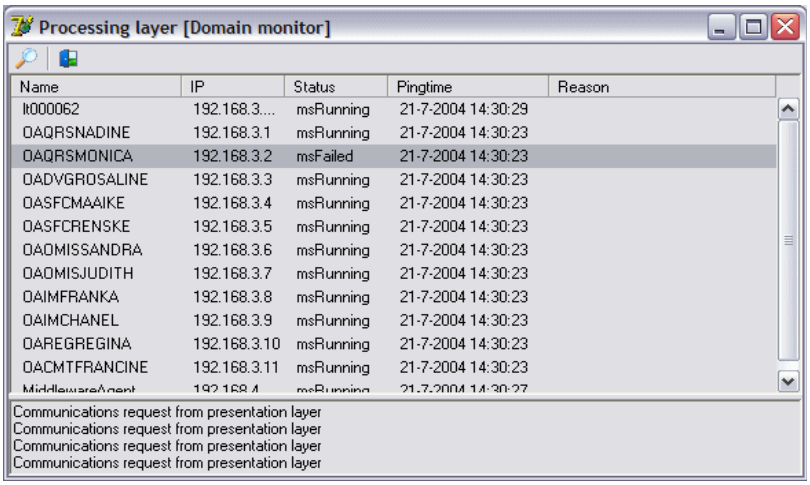

Figure 10: Domain monitor failure notification

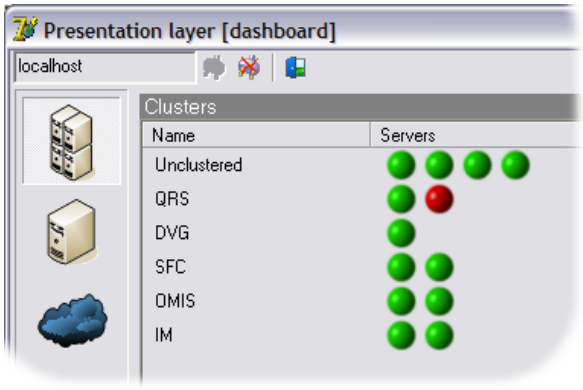

Figure 11: Presentation layer notification (cluster view)

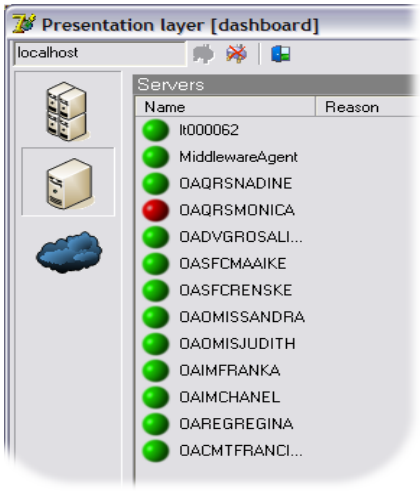

Figure 12: Presentation layer notification (servers view)

The most useful part of the prototype's dashboard is the visualization of the message flow in the observed system. The dashboard sends a request to the processing layer every 15 seconds and receives the observed communication flow from the processing layer. Figure 13 clearly visualizes the consequences of the process failure, two clients communicate with the component but as there is still one server left in the QRS cluster that has a valid component that can take over, the system keeps functioning. The administrators can solve the problem of the failed component without the system suffering any downtime.

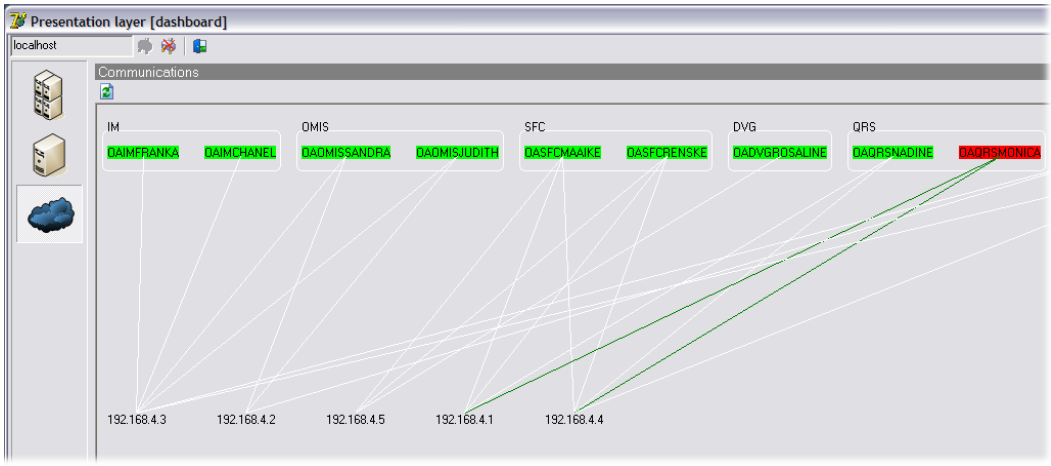

Figure 13: Presentation layer notification (communications view)

8. Conclusions

This research presented a business continuity monitoring model that has been designed to actively monitor a complete distributed architecture. The model is a combination of three activity layers with five monitoring domains meant to solve the problem of incomplete observability of the distributed architecture. The model targets several weaknesses of monitoring; volume and overhead are reduced through the use of events instead of sampling and by filtering out only the events of interest,

access to components is maximized by gathering information close to the observed components, and monitoring information can be actively presented to several observers at the same time. Monitoring domains have specific monitoring requirements; the model caters for them by allowing internal and external tools to integrate into the model. For example: an SNMP based network monitoring tool can be used for the network monitoring domain while a WBEM monitoring system such as Microsoft Operations Manager is used to monitor the server and software monitoring domains. The case study gives a good indication that downtime can be reduced by monitoring the distributed architecture.

In the case, the monitoring design enhanced the power of redundancy by quickly notifying administrators about the problem and allowing them to disable the faulty component from further participation. The monitoring design eliminated three out of four reasons for incomplete observability of the distributed architecture, reducing the dynamic complexity of such an architecture. The client server model of a distributed architecture makes it very suited for redundancy; any well design distributed architecture can implement redundancy and therefore benefit from the proposed business continuity monitoring model. The business continuity monitoring model targets three out of four reasons of incomplete observability, it provides a central point of observation, a central source of information, and a central point of decision. The model however, does not provide a way to control the distributed architecture. Controlling the distributed architecture from within the monitoring context does not require a model alteration; it is merely a case of implementing a management protocol that enables remote management of processes and servers.

References

Al-Shaer, E.S. (1998) 'A Hierarchical filtering-based monitoring architecture', doctoral dissertation, Old Dominion university.

Al-Shaer, E.S. (1999) 'Programmable agents for active distributed monitoring', DePaul University, Chicago.

Al-Shaer. E.S. (1999b) 'HiFi: A New Monitoring Architecture for Distributed Systems Management' available from Internet http://citeseer.ist.psu.edu/217578.html (26-6-2004)

Gamma, E. et all. (1994) Design Patters Elements of Reusable Object-Oriented Software, Addison-Wesley.

Gartner (2001a) "The High Cost of Achieving Higher Levels of Availability' Research not by D. Scott available from Internet http://www.availability.com/resource/pdfs/in3.pdf (21-04-2004)

Gartner (2001b) "NSM: Often the Weakest Link in Business Availability" Research note by D. Scott. Available from Internet http://www3.gartner.com/resources/99200/99205/99205.pdf (21-05-2004)

Hoffner, Y. (1994a) 'Monitoring in Distributed Systems' ANSA Phase III, available from Internet http://www.ansa.co.uk/ANSATech/94/Primary/100801.pdf (27-05-2004)

Hoffner, Y. (1994b) 'Management in Object-Based Federated Distributed Systems' ANSA Phase III, available from Internet http://www.ansa.co.uk/ANSATech/94/Primary/101801.pdf (27-05-2004)

Hoffner, Y. (1994c) 'Visualization of Distributed Systems' ANSA Phase III, available from Internet http://www.ansa.co.uk/ANSATech/94/Primary/101901.pdf (27-05-2004)

Performance technologies 'High Availability FAQ', available from http://www.pt.com/Datasheets/redundant_host_faq.pdf

Klamer, A. (2004) 'Agent or Agentless Fact and Fiction', Enterprise Network and services, available from Internet http://www.enterprisenetworksandservers.com/monthly/art.php/567 (1-06-2004)

Kramer, J. and Magee, J. (1997) 'Distributed Software Architectures', Department Of Computing Imperial College of Science, Technology and Medicine

Merriam Webster's Online (2004), available from Internet http://www.m-w.com/ (25-04-2004)

Mansouri-Samani, M. and Sloman, M. (1992) 'Monitoring Distributed Systems (A Survey)' Imperial College Research Report No. DOC92/23

Mansouri-Samani, M. and Sloman, M (1997). 'GEM A Generalised Event Monitoring Language for Distributed Systems' Distributed Systems Engineering Journal Vol. 4, No. 2 June 1997

Rackl, G. (1999) 'Multilayer Monitoring In Distributed Object-Environments' available from Internet http://citeseer.ist.psu.edu/245821.html (26-6-2004)

Rackl, G. (2000) 'Monitoring and Managing Heterogeneous Middleware', unpublished PhD thesis, Department of informatics, Technical university München.

Sommerville, I. (2001) Software Engineering, Sixth edition, Addison-Wesley.

Stallings, W. (1993) SNMP, SNMPv2, and CMIP. Addison-Wesley Publishing Company, Inc., 1993.

Venners, B. (2003) 'Visualizing Complexity, A Conversation with James Gosling, Part III', available from Interner http://www.artima.com/intv/visualize3.html (26-05-2004)

Veritas (2003) 'VERITAS™ Inform for Alerts' datasheet available from Internet http://eval.veritas.com/downloads/pro/inf/alert.pdf (27-05-2004)

Zoraja, I., Rackl, G. and Ludwig, T.(1999) 'Towards Monitoring in Parallel and Distributed Environments' In International Conference on Software in Telecommunications and Computer Networks SoftCOM '99, pages 133-141