Managing Distributed Legacy and Contemporary Service-based Applications: A White Paper
Table of Contents

1. **Introduction**  
   ____________________________________________________________ 4

2. **Application Performance Management Goals**  
   Reducing Complexity and Downtime  
   Improve Performance and Reliability  
   Automating and Enforcing Business Rules  
   ____________________________________________________________ 5

3. **Managing a deployed application**  
   Overview  
   NXTminder Benefits  
   In addition to off-loading the maintenance of existing home-grown solutions, NXTminder provides a large number of benefits for managing your applications:  
   ____________________________________________________________ 6
   Simplifies application complexity  
   Ensures application consistency and availability  
   Enables operations awareness  
   Increases operational productivity  
   Reduces risk  
   Provides Development Test Harnesses  
   Application Dependency and Relationship Management  
   Pro-Active and automatic system health management  
   Reliable fault-tolerant architecture  
   Advanced Management Console  
   Simple implementation  
   NXTminder Implementation  
   ____________________________________________________________ 7

4. **Core Capabilities**  
   Configuration and Planning  
   Monitoring and Alerting  
   Manage and Repair  
   ____________________________________________________________ 11

5. **NXTminder Use Cases**  
   ____________________________________________________________ 12

6. **Operational Efficiency and Minimized Risk**  
   ____________________________________________________________ 17

7. **Summary**  
   ____________________________________________________________ 18
1. Introduction

One of the biggest challenges facing companies that develop and maintain distributed or service-based applications is how to most effectively manage and monitor the interrelated components in a production environment.

Often, development teams implement simple process monitors or, for lack of a better solution, operation teams develop a tangle of scripts or JCL to start distributed application components in a correct order. Whether the system is based on legacy CORBA technology or Rest services based on Python and Ruby, dependencies and timings unique to the production environment need to be incorporated into these scripts – and that adds complexity.

Even if scripts are developed early in the life of the application, the knowledge used to develop them is not always easily understood, documented, or retrievable from a reliable repository. Over time, the infrastructure configurations and complex relationships become harder to replicate, applications become more difficult to manage, and the knowledge in the scripts becomes hard to use and leverage. This configuration of information can be very strategic to run-time operation.

Equally challenging is the performance levels and scalability required to make the application deliver its intended service levels and maintaining those levels as a system ages and is integrated with other systems.

At the same time, as systems age and know-how diminishes, businesses often take a “hands off” approach – absorbing risk, increased operational costs, and sometimes degraded performance in favor of “stability”.

eCube Systems provides NXTminder, an intelligent, configurable, application performance management tool that can insure the performance of complex enterprise application systems. NXTminder enables the definition of application environments, dependencies, scalability requirements, and timings that allow the applications to start, stop, and refresh in harmony with the rest of the enterprise. With NXTminder, the applications can evolve to meet new performance and service level requirements while ensuring the crucial dependencies and relationships between legacy business logic and contemporary platforms, architectures, and protocols.

Solving the challenge of managing enterprise applications with NXTminder enables companies to extend the life of existing business logic by defending it from “software hardening” and the loss of agility.
2. Application Performance Management Goals

This white paper outlines how companies can use NXTminder to automate the operation and performance of their enterprise applications through a single application performance management system.

NXTminder allows companies to define the environment and infrastructure of multi-platform enterprise applications and configure deployment, error handling, and performance rules to manage them. It also applies fault tolerance and automatic load leveling across multiple systems and services across systems.

Reducing Complexity and Downtime

When you employ an Application Performance Management tool like NXTminder, you reduce the application complexity by defining and managing a small number of definable, actionable, and configurable parameters that describe the application’s infrastructure requirements. By putting these parameters into a NXTminder configuration, you reduce the errors that occur during start up and shutdown. This minimizes the latency that occurs when you start and stop the application. The timesavings you gain can be quantified in cost savings in terms of machine time and operations time. Additionally, the complexity of adding more resources and scaling can be automated and scheduled.

Improve Performance and Reliability

Through the use of configurations, actions, and health scripts, NXTminder monitors and makes operational decisions based on the status, load, and effectiveness of the various application components. For instance, NXTminder makes it possible scale up the number of servers your application employs. By making your configuration dynamic and responsive to the load, you can ensure the performance and reliability of any application. Furthermore, you can load level the application across multiple servers to more effectively utilize all of your computing power. NXTminder supports predictive load leveling (scheduled) and dynamic load leveling using custom scripts to track performance and load.

Automating and Enforcing Business Rules

NXTminder’s most significant advantage is its ability to model the environment and the required operating process so simple repeatable configuration rules can ensure the system meets the targeted service level requirements. This capability makes the integration of automated enforcement of business rules, actions, and notifications into the operation of strategic applications possible.

The dynamic nature makes the use of a configuration tool, like NXTminder, essential as IT works to maintain alignment between IT capabilities and the needs of the business.
Managing Distributed Legacy and Contemporary Service-based Applications

In this paper, we’ll outline how NXTminder manages components, environment, and operational demand on strategic applications.

3. Managing a deployed application

Overview
Distributed enterprise applications are segregated into service and business logic components for scalability reasons. Typically, these components use interfaces to communicate with each other and other systems, such as databases and web servers. Most companies develop “home-grown” solutions to ensure that these services and components are available and that their applications work.

These “home-grown” solutions are often comprised of scripts, cron jobs, or other independent user generated mechanisms designed to ensure each component has access to its specific dependencies. Scripts developed for this type of solution loosely enforce dependencies in an asynchronous manner. When they break or complications arise, it is often difficult to identify the problem and fix it. Often, the entire deployed application (all of its services and components) needs to be restarted.

Scripts, like these, tend to be brittle and don’t migrate to newer systems easily because of hard-coded paths, timing differences, and environmental changes. Changing a script to handle new functionality, operating systems, or server instances can take hours if not days to deploy and test. The larger the system the more difficult and expensive it is to maintain.

NXTminder Benefits

In addition to off-loading the maintenance of existing homegrown solutions, NXTminder provides a large number of benefits for managing your applications:

Simplifies application complexity
- Manages a diverse set of objects: CORBA, RPC, Java, Python, Ruby, Web servers, databases, daemons, processes, and OS scripts
- Captures the complex relationships and dependencies between distributed components

Ensures application consistency and availability
- Maintains service levels by increasing up-time and reliability
- Policy driven restarts enforce rules based on defined relationships and dependencies
Managing Distributed Legacy and Contemporary Service-based Applications

- Automatically repairs and addresses changes in runtime requirements

**Enables operational awareness**

- Notifications and alerts keeps the operations team in the know
- Provides operations documentation
- Smart Phone client for mobile 24/7 access

**Increases operational productivity**

- Automatic service configuration, monitoring, alerting, starting, and restarting saves time
- Reduces errors by documenting exact system configurations, relationships, and dependencies
- Makes it easy to modify configurations and policies
- Can reduce issue resolution by more than 50%

**Reduces risk**

- Removes dependence on specific individuals with specific knowledge
- Configuration and policy framework limit impacts of human error or technical failure

**Provides Development Test Harnesses**

- Enables testing of complex environments within defined configurations
- Allows multiple test configurations for phased testing
- Speeds up testing by quickly allowing testers to start and stop different test configurations

NXTminder reflects many years of experience in designing Application Performance Management systems. NXTminder’s most strategic benefits are provided by fault tolerant design and advanced features for managing application systems:
Application Dependency and Relationship Management

- Implements cross platform application relationship
- Provides Application Groups for effective management
- Provides start up orders and dependencies to ensure orderly application activation.
- Timings and delays customized for each service component

Pro-Active and automatic system health management

- Implements operational rules and policies
- Ensures proper operations through health scripts and appropriate actions to prevent failures.
- Customizable policies enforce appropriate notification for and resolution of run-time issues

Reliable fault-tolerant architecture

- No single point of failure within NXTminder
- Through Group minimum, service levels can enforce fault-tolerance.
- Through Health scripts, Groups can implement load leveling.
- Node can independently enforce management policies

Advanced Management Console

- Expandable tree structured GUI provides application status in one glance.
- Easy to read color-coded icons providing Group and application status.
- Multi-platform agent configurations provided in one screen.
- Download-able to mobile devices

Simple implementation

- Automatically imports configurations/policies from Borland AppMinder. Easily replaces obsolete AppCenter, AppMinder, NetMinder, and home grown tools designed to keep strategic assets running
**NXTminder Implementation**

NXTminder is an agile, service-oriented enterprise application performance management system designed to manage the operation of distributed and SOA-style applications. NXTminder’s architecture is comprised of a Master Server that can be run on any server in the network and a series of agents, which manage the configurations on each server being managed. The console, which displays the status and configuration wizards for the applications, can be run as an application within a browser or hand-held device. The following diagram shows the architecture of NXTminder:
Here is a summary of the NXTminder functions:

- Its core functions include:
  - Monitoring
  - Notification
  - Repair
  - Managing the configuration of application components

- Next Generation features
  - Manages:
    - Python
    - Rails
    - Java Plain Old Java Object (POJO) executables
    - Executables
    - Web server and Web services availability
    - Database availability
    - The up-time of daemons, processes, and OS scripts
    - NXTera, Entera, and VisiBroker services
    - Legacy distributed services: Common Object Request Broker Architecture (CORBA), Remote Procedure Call (RPC), Distributed Computer Environment (DCE)

- Administers resources on Windows, UNIX, LINUX, MacOS, OpenVMS, Solaris, AIX, and HP-UX. While monitoring systems, repairs can be performed from the NXTminder console, the NXTminder mobile client, and via the NXTminder JavaScript API.

- Provides configuration and management security at all levels

- Replaces obsolete Borland AppCenter, AppMinder, NetMinder, and other home-grown tools designed to keep strategic assets running

- Requirements: Install agents on servers that have applications you wish to monitor. Access via a console or mobile browser
4. Core Capabilities
NXTminder provides a number of core capabilities and tools that are used in the management of deployed enterprise applications. These tools are used in conjunction with operational knowledge of the loads on each component to define and configure each server within its environment, consistent with its dependencies, and in a way that meets its minimum service levels. Prior to configuring NXTminder, operational knowledge and experience with the application is required to accurately predict demand loads, timings, and dependencies between each service component.

The following section summarizes the core capabilities of NXTminder and how they are used for management of application systems.

Configuration and Planning
- Captures component classifications within the application with Groups.
- Enables scalability through multiple application definitions and the use of timers.
- Defines/ knows exact environmental characteristics, dependencies, and resources needed for each service
- Automatic action taken to ensure policies and configurations are maintained

Monitoring and Alerting
- Detects and displays errors via the console, email, and messaging
- Intelligent actions performing based on individual error conditions
- Monitoring and alerting is automatic
- Tracks issues ensuring restorative actions can be taken

Manage and Repair
- Avoids downtime with automatic restarts
- Implements configurations and policies on restarts
- Promotes application integrity by pro-actively checking application health.
# 5. NXTminder Use Cases

The following table outlines the use cases for NXTminder.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Feature</th>
<th>Benefit</th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define and Discover Services and their Relationships</td>
<td>Defines and records which components are part of what applications</td>
<td>Ensures proper governance of distributed application components so operators can better diagnose issues and understand what components impact which applications.</td>
<td>Frees organizations of their dependence on specific individuals with specific knowledge. Speeds the process of issue resolution.</td>
</tr>
<tr>
<td>I need to set up a development environment, testing environment, and production</td>
<td>Reusable definitions of services and relationships</td>
<td>Ensures the environment being used in test, for example, is exactly what is deployed thus avoiding any errors</td>
<td>Enables the production environment to match the test environment exactly</td>
</tr>
<tr>
<td>I am responsible for a distributed application made up of numerous components and IT services. However, I don’t have a way to define the relationships between these components.</td>
<td>Provides a means to define relationships and record POLICIES to manage operations based on these relationships.</td>
<td>Ensures proper governance of distributed application components so the proper actions are taken in the proper order with the proper resource(s).</td>
<td>Frees organizations of their dependence on specific individuals with specific knowledge. Speeds the process of issue resolution.</td>
</tr>
<tr>
<td>I don’t know the order in which components/services must be started</td>
<td>Enables definition of start-up rules and policies for component start-up order.</td>
<td>Upon restart, the correct services start in the proper order</td>
<td>Application characteristics are stored and automatically applied as needed.</td>
</tr>
<tr>
<td>I don’t know how to properly stop a system or the order in which</td>
<td>Enables definition of shut-down rules and policies</td>
<td>At shut down, the correct services start in the proper order</td>
<td>Application characteristics are</td>
</tr>
<tr>
<td>Component/Service Issues</td>
<td>Feature</td>
<td>Benefit</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Components/services must be stopped for component shut-down order</td>
<td>Enables definition of rules and policies for component start-up configuration</td>
<td>Upon restart, the correct services start in the proper configuration</td>
<td></td>
</tr>
<tr>
<td>I don’t know how many components/services and what type need to be started to run my application</td>
<td>Recovers the configuration of currently managed services and automatically restarts them. The console displays service status</td>
<td>Provides a single point of truth for the state of the application</td>
<td></td>
</tr>
<tr>
<td>If a restart occurs, I don’t know if all the services are up and running again</td>
<td>Saves time and avoids problems created by configuration changes</td>
<td>Allows the admin to re-construct or remove changes in case of errors</td>
<td></td>
</tr>
<tr>
<td>Hard to track changes in the configuration</td>
<td>Configuration changes tracked in a log file.</td>
<td>Prevents the admin from having to manually enter each server and dependency.</td>
<td></td>
</tr>
<tr>
<td>I want to use an existing legacy AppMinder Configuration</td>
<td>Import AppMinder Configuration</td>
<td>Saves time in configuring if an AppMinder configuration is available</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prevents the admin from having to manually enter each server and dependency.</td>
<td></td>
</tr>
<tr>
<td><strong>Configuration and Planning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I put the service in the wrong group; do I have to enter all this information again?</td>
<td>Using the drag and drop feature, you can drag the managed service to another group or other part of the configuration</td>
<td>Reduces time for error prone operations</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Makes configurations more modular and easy to change.</td>
<td></td>
</tr>
<tr>
<td>I don’t know how services should be grouped</td>
<td>With dependency properties, relationship to other services can facilitate grouping of services</td>
<td>Time saved by organizing service/object structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operational engineers don’t have to have application specific knowledge</td>
<td></td>
</tr>
<tr>
<td>It is hard to define the minimum service level</td>
<td>The group feature allows you to</td>
<td>Problems are avoided by ensuring</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ensures runtime policy based</td>
<td></td>
</tr>
</tbody>
</table>
**Managing Distributed Legacy and Contemporary Service-based Applications**

<table>
<thead>
<tr>
<th>For each service/object</th>
<th>Configure minimum service levels for services/objects</th>
<th>Each group provides the minimum level of services/objects necessary to operate to capacity</th>
<th>Approach to meeting SLA commitments</th>
</tr>
</thead>
<tbody>
<tr>
<td>I have a large number of services to manage and I don’t want to manually enter each service.</td>
<td>Copy and paste objects feature allows you to replicate servers so that you only have to change the server name and starting command properties</td>
<td>Reduces time to enter a configuration</td>
<td>If the service configuration is similar, copy and paste functions will speed up the configuration process.</td>
</tr>
<tr>
<td>I have known usage peaks on a weekly, monthly, and annual basis that tax my system</td>
<td>Design resource management schedules that makes more resources available when needed</td>
<td>Automates the management of service availability with fail-over policies.</td>
<td>Aligns IT with business cycles and needs</td>
</tr>
<tr>
<td>When demand for services peak, I have to be available to replicate services and start new services/objects</td>
<td>Automatic start of additional services/groups as needed for fail-over and load-leveling</td>
<td>No need to plan for or watch for need to add services/groups</td>
<td>Ensure system availability at designed service levels</td>
</tr>
<tr>
<td>Hard to schedule maintenance restarts</td>
<td>Schedule maintenance restarts with timers</td>
<td>No need to be present to conduct maintenance restarts</td>
<td>Reduces cost and complexity of maintenance cycling</td>
</tr>
<tr>
<td>Resources sometimes do not get the priority they need to operate</td>
<td>Manages resource priorities</td>
<td>Resources operate efficiently since they get allocated the resources they need</td>
<td>Manages business functions based on IT importance</td>
</tr>
<tr>
<td><strong>Monitoring and Alerting</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard to track start and restarts on a regular basis, the state of applications, services, objects, and groups</td>
<td>Constantly tracks start and restarts of applications, services, objects and groups</td>
<td>Saves time and effort to track state and restart</td>
<td>Runtime record of system health and wellbeing</td>
</tr>
<tr>
<td>Don’t know status and problems</td>
<td>Automatic notification via Email, SMS,</td>
<td>No more unknown problems</td>
<td>Agile and flexible operations management</td>
</tr>
</tbody>
</table>
### Managing Distributed Legacy and Contemporary Service-based Applications

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Don’t know at what level there is a problem</td>
<td>Monitors systems at the configuration, group, and service/object level</td>
<td>Or where the problem exists</td>
</tr>
<tr>
<td>I don’t have a history of what has happened before</td>
<td>Access log files for all managed objects</td>
<td>Data available for study</td>
</tr>
<tr>
<td>I need to integrate my own health scripts and diagnostics</td>
<td>Open system to enable custom health scripts and diagnostics</td>
<td>User customizable</td>
</tr>
<tr>
<td>I need to monitor Master Groups, Groups, Java executables, Processes, Scripts, Process started by scripts, Python executables, and Daemons, NXTera, Entera Objects</td>
<td>Monitors Master Groups, Groups, Java executables, Processes, Scripts, Process started by scripts, Python executables, and Daemons</td>
<td>Monitors complex cross-platform environments</td>
</tr>
</tbody>
</table>

### Manage and Repair

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Solution</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>I want to restart service/objects, groups, and configurations in the appropriate order</td>
<td>Restart service/objects, groups, and configurations in the appropriate order based upon dependency rules</td>
<td>Avoids downtime by correctly starting in proper order</td>
</tr>
<tr>
<td>I want to run user generated scripts to resolve specific issues</td>
<td>Create scripts based upon your installation and configuration</td>
<td>Customizable</td>
</tr>
<tr>
<td>If an error occurs, I want to run my scripts for repair</td>
<td>Automatically runs repair scripts</td>
<td>Manage repairs without human intervention</td>
</tr>
<tr>
<td>I want triggers and alerts to launch specific repairs</td>
<td>Notifies designated people there is a problem</td>
<td>Management can just respond to problems and not</td>
</tr>
<tr>
<td>Feature</td>
<td>Need</td>
<td>Benefit</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>I want to manage rules for delayed, priority, and retry of repair actions</td>
<td>Create rules for repair priorities</td>
<td>Repairs performed in the correct order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supports operational independence and lowers the cost and expertise required</td>
</tr>
</tbody>
</table>

**Flexible and Agile**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Need</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>I want the monitoring to be distributed so, if the monitoring client goes down, the monitoring will continue</td>
<td>NXTminder can connect to any agent for monitoring</td>
<td>If the Master Agent or computer, which the console is connected to, goes out of service, the remaining agents will continue to work and meet the requirements outlined in the defined policies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ensures the redundancy of business critical systems</td>
</tr>
</tbody>
</table>
6. Operational Efficiency and Minimized Risk

Application Management has a cost but it shouldn’t represent additional operational risk or work for your operations staff. In many cases, IT organizations are building their own infrastructures to manage legacy applications. However, they are not taking into account the effort required to manage and maintain the variety of systems they operate. To make matters worse, there is the risk created by the homegrown scripts authors leaving the company without fully documenting how the scripts work. A true Application Management strategy provides a platform independent structure, which can easily support new components, service levels, environmental configurations, and changes in operational demand.

In the end, every business executive has to contend with risk. eCube is committed to mitigating risk with proven technology that automatically implements the operational intelligence of its clients. With NXTminder, applications can automatically be maintained and operated with the greatest level of reliability and agility needed to respond to changing business needs.
7. Summary

IT managers are facing a variety of business pressures that are forcing them to evaluate the cost and risk associated with the operation of legacy systems. Business requirements demand ever-lower operational costs, better performance, reduced risk, and compliance with industry and corporate standards.

NXTminder provides an intelligent, self-healing, and systematic approach to the manageability and performance of distributed enterprise and Web service-based applications. Supporting a wide variety of application service/components (including COM scripts, simple processes, Java objects, python, and executables -- such as web and database servers -- as well as services that enable .NET, J2EE, Web Services, CORBA/RPC, and REST-base applications), NXTminder is the flexible tool enterprise operations teams need to replace simple process, monitors, and hard to maintain home-grown solutions.

NXTminder, eCube’s Application Performance Management platform, enables companies to extend the value of existing distributed enterprise and new Web service-based applications by effectively managing service components for optimal reliability, performance, and total cost of operation.