BlackBird Monitoring System
Performance Analysis and Monitoring in Information Systems

João Paulo Henriques Germano

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Júri
Presidente: Doutor Mario Serafim dos Santos Nunes
Orientador: Doutor Alberto Manuel Rodrigues da Silva
Co-orientador: Doutor Fernando Henrique Corte Real Mira da Silva
Vogal: Doutor Paulo Jorge Pires Ferreira

Setembro de 2007
Acknowledgments

Now, in the final steps of this journey, it is time to thank all those that made it possible. First of all I wish to thank Prof. Alberto Silva and Prof. Fernando Mira da Silva for accepting this challenge and for their guidance in establishing a clear and attainable objective for this work. A special thanks to Vodafone Portugal for providing the opportunity and the resources for this work, and to my manager Luis Bento for continued support and orientation throughout all stages of this work. To my brother José Germano as the main instigator for this work, and for his constant support and invaluable feedback. To my team colleagues, especially José Fragata for their experienced advice on the applications. Thanks also to Pedro Tomás for the template for this document.
Abstract

Today’s Enterprise Applications are under constant pressure from demanding service level agreements, and achieving such service levels requires early identification and quick addressing of any performance problems. The fundamental tools for this task are monitoring systems capable of providing detailed Metrics on the application status and performance. However, Enterprise Applications have become extremely complex systems, under constant evolution in terms of architecture and technologies. Also, the ever increasing range of tasks to be performed has produced an almost endless variety of different applications. As a result, adequate monitoring systems are rarely available.

This work intends to address this problem by developing a monitoring system capable of interacting with the various technologies employed by the Monitored Application. The system must adaptable to the application architecture regardless of its complexity and follow it’s evolution with a minimum development effort. Capable of producing Metrics that closely relate to the service level goals established for the application.

To achieve these objectives, the BlackBird monitoring system employs several technology specific Interface Modules, and a central system for Metrics calculation and presentation. Interface modules provide technology support, they interact with the application using a specific technology but produce performance data in a common format, isolating the central system from the employed technologies. All performance data is then stored in a common repository and used for Metrics calculation. All aspects of the performance data gathering and processing can be specified using a set of pre-defined Configuration Objects, thus allowing the adaptation to any architecture and evolution.

Keywords

Application Monitoring, Systems Administration, Application Performance, Component, Adaptable.
Resumo

As aplicações empresariais actuais encontram-se sob constante pressão por parte de exigentes acordos de nível de serviço, e atingir tais níveis de serviço, implica identificar atempadamente problemas de performance. Para isto são necessários sistemas capazes de fornecer métricas detalhadas do estado de funcionamento e desempenho da aplicação. Contudo, as aplicações empresariais tornaram-se extremamente complexas e continuam em constante evolução em termos de arquitectura e tecnologias usadas. Além disso, o crescente número de tarefas a desempenhar origina uma infinidade variedade de aplicações. Como resultado, sistemas de motorização adequados raramente estão disponíveis.

Este trabalho pretende abordar este problema desenvolvendo um sistema de monitorização capaz de interagir com as diferentes tecnologias da aplicação a monitorizar. Um sistema adaptável à arquitectura da aplicação independentemente da sua complexidade e capaz de acompanhar a sua evolução com o mínimo de desenvolvimento. Capaz, também de produzir métricas directamente relacionadas com os objectivos de níveis de serviço estabelecidos.

Para atingir este objectivos, o sistema de monitorização BlackBird emprega módulos de interface de várias tecnologias e um sistema central para o cálculo de métricas e apresentação de resultados. Os módulos de interface fornecem suporte tecnológico, eles interagem com a aplicação usando tecnologias específicas mas produzem dados num formato comum, isolando o sistema central das tecnologias utilizadas. Estes dados são armazenados num repositório comum que é utilizado para o cálculo de métricas. Todos os aspectos de funcionamentos de sistema podem ser especificados utilizando um conjunto predefinido de objectos de configuração, possibilitando assim a adaptação a qualquer arquitectura e/ou evolução.

Palavras Chave

Monitorização de Aplicações, Administração de Sistemas, Performance da Aplicação, Componente, Adaptável.
# Contents

1 Introduction 1
    1.1 Motivation ............................................. 2
    1.2 Objectives ........................................... 3
    1.3 Results ............................................... 5
    1.4 Dissertation Outline ................................. 5

2 State of the art 7
    2.1 Historic Perspective ................................. 8
    2.2 Existing Systems .................................... 9
    2.3 Future Trends ....................................... 11
    2.4 Comparative Analysis ............................... 12

3 Requirements 15
    3.1 General Requirements ............................... 16
    3.2 Monitoring Requirement ............................ 17
    3.3 Actors ............................................... 18
        3.3.1 Deploy Monitoring Requirement Use Case ....... 19
        3.3.2 Visualize Monitoring Output Use Case ....... 20
    3.4 Additional Requirements .......................... 21

4 BlackBird Design and Implementation 23
    4.1 Application Modeling ............................... 24
    4.2 Architecture ....................................... 25
    4.3 BlackBird Components .............................. 26
        4.3.1 Adaptation Layer .............................. 27
        4.3.2 Aggregation Layer ............................ 28
        4.3.3 Presentation Layer ............................ 29
    4.4 Result Format ..................................... 29
    4.5 Data Model .......................................... 30
        4.5.1 Main Schema .................................. 30

vii
## Contents

4.5.2 Monitored Application Schema ........................................ 32
4.5.3 Client Schema Management ........................................... 34
4.6 Interface Modules ............................................................ 36
4.7 User Interface ................................................................. 36
  4.7.1 Monitoring interface .................................................. 39
  4.7.2 Graphics Pages ......................................................... 40
  4.7.3 Configuration Interface ............................................... 40

5 Application Scenarios .......................................................... 43
  5.1 Monitored Applications .................................................. 44
    5.1.1 Pre Paid Billing System ........................................... 44
    5.1.2 Mediation Device .................................................. 46
  5.2 Monitoring Configuration ................................................. 47
  5.3 Configuration Process .................................................... 48
  5.4 Configuration Example .................................................. 50
  5.5 Monitoring Output ........................................................ 54

6 Conclusions ............................................................................ 57
  6.1 Results ............................................................................. 58
  6.2 Future Work ...................................................................... 60

A Appendix A ............................................................................. 63
  A.1 PPB Monitoring Requirement ............................................ 64

B Appendix B ............................................................................. 67
  B.1 Configuration Parameters ................................................ 68
List of Figures

1.1 Monitoring System Overview. ............................................. 4
3.1 BlackBird Actors. .......................................................... 18
3.2 Deploy Monitoring Requirement use case. ............................ 19
3.3 Visualize Monitoring Output user case. .............................. 21

4.1 Application Model. .......................................................... 24
4.2 Domain Model. ............................................................... 25
4.3 BlackBird Components. ..................................................... 27
4.4 XML Schema. ................................................................. 30
4.5 Relational Entity from XML Schema. .................................. 30
4.6 Main Schema. ................................................................. 31
4.7 Monitored Application Schema. ........................................ 33
4.8 Application Schema Management. ..................................... 35
4.9 BlackBird Interface. .......................................................... 37
4.10 Application Owner Page. .................................................. 38
4.11 Application Page. ........................................................... 38
4.12 Graphics Page. ............................................................... 40
4.13 Configuration Page. ........................................................ 41
4.14 Edit Page: Create WEBServ Module. ................................. 42
4.15 Edit Page: Create Metric. ............................................... 42

5.1 Configuration Sequence Diagram. ....................................... 49
5.2 Edit Page: Create SQ_LOGIN. ............................................ 50
5.3 Edit Page: Create SQL_Module. ........................................ 50
5.4 Edit Page: Create Graphic. ............................................... 51
5.5 Edit Page: Create Metric. ................................................. 52
5.6 Edit Page: Create Page. .................................................... 52
5.7 PPB Monitoring Configuration Page. ................................... 53
5.8 PPB Queue. ................................................................. 54
List of Figures

5.9  Mediation Device Network Elements. ........................... 54
5.10 Mediation Device Streams. ................................. 55
List of Tables

2.1 Feature Comparison Table. .............................. 13
### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>Third Generation (mobile communication)</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>GPRS</td>
<td>General Packet Radio Service</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile communication</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java2 Platform Enterprise Edition</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>PPB</td>
<td>Pre Paid Billing system</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>WEB</td>
<td>Short for World Wide Web</td>
</tr>
<tr>
<td>WMI</td>
<td>Windows Management Instrumentation</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XQuery</td>
<td>XML Query Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
</tr>
</tbody>
</table>
1

Introduction

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Objectives</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Results</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Dissertation Outline</td>
<td>5</td>
</tr>
</tbody>
</table>
1. Introduction

In this chapter we introduce the context of enterprise applications and the difficulties in effectively managing these applications. Then we establish the main objectives for this work and the proposed final result.

1.1 Motivation

The telecommunications business, in which Vodafone is included, is one of the most competitive markets. As a provider of high technology services there is a constant pressure to implement new technologies that will allow the diversification of the provided services and the improvement of existing services. Like in most large scale and technology based business, the Information Technologies (IT) infrastructure of telecommunications companies has become the main base of support to business processes, and many Enterprise Applications are now considered mission-critical, reaching a point where the performance of these applications has a direct relation to the performance goals of the entire company.

The complexity and diversity of the business rules and provided services, together with the pressure for fast implementation demand a vast portfolio of different applications in the organization. These applications can be extremely diverse, in terms of complexity, architecture, base technologies and application provider. And as result of fierce competition environment all these applications are required to constantly evolve and adapt, in order to implement new business requirements and support new services.

In organizations such as Vodafone the teams responsible for the operation and management of these applications are faced with the challenge of assuring the best possible quality of service and the attainment of the negotiated Service Level Agreement (SLA). For this task it is essential to have monitoring systems capable of providing a comprehensive view of the application status, in order to closely monitor critical components of the application, anticipate performance problems and act before there is any impact on the quality of service and SLA. The currently available monitoring systems can provide efficient monitoring on the network and device level, however, due to the complexity and diversity of the applications, these systems are unable to provide the desired monitoring on the application level.

Most of the available monitoring systems specifically target applications or technologies that have a large user base, Database servers, WEB servers, Unix or Windows Servers. In these cases the monitoring systems is adapted to a fixed architecture, monitors pre-determined system parameters and expected system components. This kind of monitoring ignores all the functionality that is developed over the base
1.2 Objectives

The objective of this work is to develop and deploy a monitoring system that will have two main functions:

- provide the complete and detailed application level monitoring on the applications maintained by the DTSI of Vodafone Portugal;

- assist in the performance optimization and problem diagnostics of the applications maintained by the DTSI of Vodafone Portugal.

This monitoring system must be capable of interacting with a wide range of enterprise applications all based on different architectures and built on different technologies, obtain relevant working parameters and store them in a common data repository. The monitoring system must also be expandable, so that adding support for new technologies can be accomplished with minimal changes.

The monitoring system must provide a level of data processing that uses the data repository as source to the calculation of Metrics. These Metrics relate to the indicators of the quality of service demanded from the Monitored Application.
1. Introduction

From these Metrics the monitoring system must generate a set of web pages, updated in real-time, containing graphics and charts that will be used by the management teams to access the current working status of the application.

In order to be adaptable to the Monitored Application without any previous knowledge of it’s characteristics, the monitoring system must allow the Application Owner to specify how the application should be monitored. The monitoring system must provide a set of pre-defined Configuration Objects that can be combined to produce the desired result, and an interface for fast and easy configuration of the monitoring system. This way, the manager of the Monitored Application will be able to specify which working parameters are necessary, how the Metrics must be calculated and how they must be displayed. To be useful in performance optimization and problems diagnosis, the monitoring system must allow the manager of the Monitored Application to implement changes and add of new elements, quickly and autonomously.

The diagram in fig. 1.1 represents a general view of the monitoring system, it obtains data from a number of servers using different technologies, and generates visual representations of the system status and performance that are presented to the application operators.

Figure 1.1: Monitoring System Overview.
1.3 Results

As a result of this work it is expected the integration of the monitoring system in some of the major applications of Vodafone Portugal DTSI, providing continuous monitoring of these applications and being actively used to optimize performance and diagnose problems.

The most valuable capabilities of this monitoring system are:

- monitoring a range of applications as diverse as possible, so that it may be applied to the wide range of application in use at Vodafone Portugal;

- providing complex Metrics that relate to the main application goals, for complementing current monitoring systems currently deployed at Vodafone Portugal.

1.4 Dissertation Outline

This chapter presented an introduction to the context of Enterprise Applications and to the subject of monitoring systems. It also presents the motivation for this work, the proposed solution and expected results. The next chapter delivers an overview of existing monitoring systems and their characteristics, and evolution possibilities. Chapter 3 defines the high level requirements and the supported types of Monitoring Objects, details the use cases and additional requirements. Chapter 4 we begin by defining the concepts of application modeling, then we present the overall architecture and data format. To end we detail the data management implementation and discuss alternative implementations. Chapter 5 contains application monitoring examples implemented for some of the applications at Vodafone Portugal. In the last chapter we discuss the final result of this work against the initial objectives and propose future improvements.
1. Introduction
State of the art

Contents

2.1 Historic Perspective ........................................... 8
2.2 Existing Systems ................................................. 9
2.3 Future Trends ..................................................... 11
2.4 Comparative Analysis .......................................... 12
2. State of the art

This chapter presents a brief review on the evolution of monitoring system, followed by an analysis of current available solutions representative of the various types, and a discussion of future evolution possibilities.

2.1 Historic Perspective

The need for monitoring systems was born from the need to assure high availability of the first enterprise level systems, IBM, etc. These first monitoring systems were developed by the system provider as part of a services package. With the appearance of large scale enterprise networks it became also necessary to develop monitoring tools capable of monitoring these vast collections of computers. Thereby, it was network monitoring that originated the first vendor independent monitoring systems, such as HP Openview [1].

As the systems and networks evolved and matured so did the monitoring systems, Simple Network Management Protocol (SNMP) [2] became the most widely used management protocol and is currently the base to the most widely used network and device management tools, HP Openview [1] and Nagios [3].

However, due to the numerous programming languages and the almost infinite number of architectures and purposes, application level monitoring remains an extremely diversified field with no predominant protocols or methodologies. For application level monitoring it remains the developer’s responsibility to provide the application with the interfaces that will facilitate its monitoring. Due to development costs, time constraints or technology constraints, the monitoring capabilities provided by Enterprise Applications can vary greatly:

- application specific monitoring tools provided by the developer, Microsoft Operations Manager [4], Oracle Enterprise Manager [5];
- proprietary APIs to be used by general purpose monitoring tools, Manage Engine [6];
- integration modules for standard monitoring protocols, SNMP [2];
- implementation of technology specific monitoring protocols, Java Management Extensions (JMX) [7], Windows Management Instrumentation (WMI) [8];
- non existing, small user base application, or rapid development applications.
2.2 Existing Systems

The purpose of any monitoring system is to provide information about the current working status and performance of a given Enterprise Application. The Monitored Application is a computer program that performs businesses related work, it may be running on multiple servers, and it may interact other applications. In order to facilitate the monitoring process, the Monitored Application must provide an interface for an external program to obtain the appropriate working parameters and performance data. The monitoring system will then use the provided interface to gather information from the Monitored Application, apply a set of data processing rules and deliver the summarized data to the application manager.

The currently available monitoring tools are as diverse as the application they target, and the differences between tools can range all aspects of the monitoring process:

- general architecture;
- application instrumentation;
- targeted applications.

In terms of architecture, most monitoring systems employ a Manager-Agent architecture, there is usually a central management system and several agents each one dedicated to a target resource or application. A complete monitoring system will have several types of agents specifically developed for the interfaces and architectures of the various types of monitored resources. The agent will usually reside on the same host as the monitored resource, it will communicate with the monitored resource and relay the obtained data to the central management system using a protocol such as SNMP [2]. The agent may be scheduled to continuously gather data providing a constant stream of data, or it may act only upon a command from the central management system. Depending on the number of the monitored resources, some architectures may also include sub-managers that will aggregate and filter data from a group of agents, referred as a domain, and forward the relevant information to the central manager. This is the architecture that emerged from the first network management systems and is currently the base of all monitoring systems based on the SNMP protocol. It is especially adequate for monitoring of vast numbers of distributed resources such as computer networks Openview [1], Nagios [3] or grid computing systems MonALISA [9].

As more applications evolved from centralized to distributed and from raw processing to providing services, a new monitoring architecture became possible, Agent-
2. State of the art

Agentless Monitoring. In this case there is no need to deploy an agent on the application host, all data gathering is accomplished by remote access using standard protocols and logins. There is only a central management system that executes calls to the services provided by the Monitored Application, and by remotely accessing applications resources like databases, servers and file systems.

Agentless Monitoring typically provides lightweight monitoring, but with limited depth of data gathering or monitoring capabilities. However, it is less intrusive, easier to deploy and does not require continuous development. Agentless Monitoring is especially adequate for services based application and is the only option for proprietary and closed source applications. One example of significant agentless monitoring tool is Longitude \[10\] from Heroix.

Since the first monitoring systems one key part of the monitoring process has been application instrumentation, which consists in modifying existing applications in order to collect additional data during run-time. The basic instrumentation technique is to insert instrumentation code at key points of interest in the program. During execution, the instrumentation code is then executed together with the original program code, producing events that that will give the main monitoring tool information about the application’s current status.

The importance of application instrumentation for the management of complex distributed application has resulted in several technology level standards like JMX for Java and Java2 Platform Enterprise Edition (J2EE) and WMI for Microsoft Products and the Microsoft .NET framework. These two protocols became base of all monitoring applications aimed at these frameworks.

Other major difference between the various available tools is the range of applications they target, monitoring tools can be application specific or general purpose. Application specific tools are usually developed by companies that wish to provide a high level of monitoring to it’s product range, the monitoring tool will then be included in a support package or as an additional component. Naturally, application specific tools can provide the best monitoring of any application. However, it becomes impossible to combine the monitoring of applications that work together. Most of all, developing dedicated monitoring tools is a costly process that can only be supported by large companies with a significant application portfolio. Examples of such tools are Microsoft Operations Manager \[4\] and Oracle Enterprise Manager \[5\]. Third party companies will only risk developing application specific tools for applications that can guaranty a large user base. Quest Software provides versions of the Spotlight \[11\] monitoring tool for BEA WebLogic Server, Oracle, etc.

The option taken by most companies for adding monitoring capabilities to their
2.3 Future Trends

applications is to implement a standard management technologies such as SNMP, JMX or WMI, the organization employing the application will then use a general purpose monitoring tool. General purpose monitoring tools aim to provide monitoring services to a range of applications as wide as possible. For this they will implement support for standard monitoring technologies and protocols, and for proprietary protocols used in applications with a large user base. One example is ManageEngine [6] from AdventNet.

Due to the development costs and the limited market, application specific tools tend to be more expensive and limited to only a few applications. Organizations tend to employ application specific tools only for the most critical applications, or for widely used applications such as servers and databases. General purpose tools will be used for the remaining low user base applications found in large organizations.

One feature that originated from the SNMP based monitoring systems and is being implement by ever more monitoring tools is auto discovery and self configuration. Monitoring tools will use the directory capabilities of monitoring technologies and detect compatible resources available in the network and automatically configure a typical set of monitoring functions for those resources. This allows for a much faster deployment, especially for network assets or distributed applications, unfortunately it may also detect unwanted targets, such as testing or development resources.

2.3 Future Trends

Extensive work has been aimed at improving the monitoring support of Enterprise Applications, with the main focus on improving the monitoring extensions provided by the monitored application. Most of this work is focused on the JMX [12] technology which is part of J2SE platform, JMX defines an architecture in which client applications can provide a remotely accessible management interface that exposes their internal structures and resources. Although, it is only applicable to Java and the J2SE platform. It has a close resemblance to the WMI for the .NET framework, and the main concepts are applicable to other programming languages and architectures, including legacy applications [13]. The use of standard management architectures and instrumentation techniques on enterprise applications opens the way to automated application management and self managed applications [14]. These have one common characteristic, they target component based applications, and because of the component organization and the management interfaces provided by the base technologies it is possible to obtain a good detail of application
monitoring.

Even applications that do not employ a strict component based architecture can usually be modeled as a set of distinct programs. Whatever the purpose or technology used the complexity of Enterprise Applications dictates that the full task to be executed must be split between simpler tasks, that will be performed by different program modules.

By considering a definition of Component less restrictive than the one usually associated with Component Based Software it should be possible to model any application as component based, where each component may have a number of parameters that can be used as indication of the general application health and performance. And, by modeling the application, it should be possible to capture a more abstract level of application functionality, that is closer to the business logic and to the main application goals of quality of service.

2.4 Comparative Analysis

The Monitoring Systems referenced in the previous section are some of the most widely used and represent a representative sample of the existing monitoring solutions. The table 2.1 summarizes the main features of these systems.

The objectives for this work clearly requires a general purpose monitoring application, and considering the context of Vodafone Portugal, the agentless architecture seems the most adequate, as it the less intrusive on the monitored systems. Also, in order to minimize the impact on the Monitored Applications, adding instrumentation code to existing applications is not an option. However, the monitoring system must be capable of using existing instrumentation interfaces namely JMX and WMI. These restrictions force the proposed Monitoring System to rely on the protocols provided by the target application. Therefore, it must support those interfaces and use them for data gathering.

Although most of these systems provide some form of support to user defined Metrics, in most cases these Monitoring Systems allow the definition of Metrics based on different data sources only for reporting purposes, which inadequate for real-time monitoring.

Component based monitoring provides the best base for defining Metrics related to the application main goals. In order to extend the concept of component based monitoring to applications that were not developed as Component Base Applications, this work must define a simplified component definition that allows modeling most applications as component based.
### Table 2.1: Feature Comparison Table.

<table>
<thead>
<tr>
<th></th>
<th>Targeted Systems</th>
<th>Supported Protocols</th>
<th>Agent Based</th>
<th>Interface</th>
<th>Alerts and Notifications</th>
<th>User Defined Data Gathering</th>
<th>User Defined Metrics</th>
<th>Component Based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HP Openview</strong></td>
<td>General Purpose</td>
<td>SNMP</td>
<td>Y</td>
<td>Thin Client</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Nagios</strong></td>
<td>Network and Host</td>
<td>Network services</td>
<td>Optional</td>
<td>WEB</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td><strong>Microsoft</strong></td>
<td>Microsoft Applications</td>
<td>WMI</td>
<td>Y</td>
<td>WEB and Client</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Operations Manager</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Oracle</strong></td>
<td>General Purpose</td>
<td>ODBC and Network services</td>
<td>Y</td>
<td>WEB</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Enterprise Manager</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Spotlight</strong></td>
<td>Dedicated</td>
<td>JMX</td>
<td>Y</td>
<td>Thin Client</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Manage Engine</strong></td>
<td>General Purpose</td>
<td>Multiple</td>
<td>Optional</td>
<td>WEB</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Longitude</strong></td>
<td>General Purpose</td>
<td>Multiple</td>
<td>N</td>
<td>WEB</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>BlackBird</strong></td>
<td>General Purpose</td>
<td>Multiple</td>
<td>N</td>
<td>WEB</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
2. State of the art
## Requirements

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 General Requirements</td>
<td>16</td>
</tr>
<tr>
<td>3.2 Monitoring Requirement</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Actors</td>
<td>18</td>
</tr>
<tr>
<td>3.4 Additional Requirements</td>
<td>21</td>
</tr>
</tbody>
</table>
3. Requirements

The previous chapter present the main characteristics of current monitoring tools and areas of future development, on this chapter we will specify which of these characteristics are more adequate to the objectives of this work and Vodafone. This will determine the main requirements and use cases of the Blackbird framework, and additional requirements.

3.1 General Requirements

This work intends to produce a monitoring system with the following five key features:

- monitor a wide range of applications, adaptable to the architecture and technologies of the Monitored Application;
- provide in depth application level monitoring, component based;
- easily adaptable to the evolution of the Monitored Application;
- low impact on the monitored system, agentless and without additional application instrumentation;
- graphic interface for data visualization and configuration.

The monitoring system must be capable of communicating using the interfaces provided by the Monitored Application, executing predefined commands and store the output data in a format that is independent of the Monitored Application or the interfaces used. It must be easily expandable, so that adding support new technologies is possible without changes to the base architecture or to existing functionality. It must be capable of simultaneously executing as many different commands as necessary to obtain all the relevant data from the Monitored Application.

As discussed in the previous chapter modeling the Monitored Application in components can provide a level of monitoring closer to the application goals and performance. All data gathered must be directly related to a specific application component and all stored data must be accessible by referenced to a component identifier. The monitoring system must be capable of aggregating the stored data using component identifiers and timestamps to calculate complex Metrics that represent the application goals and performance indicators.

The monitoring system must include a configuration interface that will allow the Application Owner to modify all aspects of the application monitoring or to specify the complete monitoring of a new application. The Blackbird framework must be
3.2 Monitoring Requirement

capable of handling multiple applications while maintaining complete separation of
data and enforcing access control.

Considering the large number of different applications at Vodafone Portugal,
adding agents or instrumentation to even a small number of these applications would
hardly be cost effective, especially in the case of proprietary and closed source applica-
tions. It would also be extremely disruptive to the current development lifecycle
with possible impact on the implementation of new business requirements.

Finally, the monitoring system will provide a graphical interface for presenting
the graphics and tables that represent the Metrics. Because this tool must pro-
vide monitoring service to a large number of applications under the responsibility of
different departments or teams, it must allow multiple users accessing different re-
sources and implement access control on all visualization and configuration actions.
One of the most important uses for the monitoring system is to provide information
to the on call support, therefore it is necessary a lightweight client that allows many
users to simultaneously execute different tasks.

3.2 Monitoring Requirement

The Blackbird system aims to provide a monitoring service to an application
without any previous knowledge of the architecture. Knowledge of the Monitored
Application must be introduced by the Application Owner in the form of Moni-
toring Objects that describe operations to be performed by the BlackBird System.
The Application Owner has knowledge of the application goals and performance
requirements and will be responsible for defining the monitoring requirements of the
Monitored Application and detailing the Monitoring Objects necessary to specify
the complete monitoring of the application.

A monitoring requirement may include any number of the following types of
Monitoring Objects:

**Commands:** Commands to be executed using the interfaces provided by the mon-
titored application, the response will be stored and used to caculate Metrics.
The output must be a list of application components of the same type and a
set of parameters that are associated with each of those components.

**Metrics:** A formula to be executed on the stored data to produce a result that
is representative of the application’s status and performance. The formula
will reference and aggregate the stored date using the component ID and the
3. Requirements

results timestamp. Consequently the result may also be a set of parameters associated with a component of the monitored system.

**Alerts**: A condition to be evaluated on a Metric, if true it will trigger a notification message.

**Graphics**: A visual representations of a Metric, it can be a table or various types of graphics, line, pie or bar charts. Will use the Metric as a data source and plot the data according to the type and format. May contain links to a page for problems drill down and root cause analysis.

**Pages**: Screens that contain graphics relative to the same application component or type of component. Provide the base to a structured representation of the monitored architecture.

3.3 Actors

The monitoring process using the Blackbird Monitoring System involves the actors represented in fig. 3.1.

![BlackBird Actors](image.png)

Figure 3.1: BlackBird Actors.

**Application Owner**: Manager of an Enterprise Application that will be monitored using the Blackbird system. Creates monitoring requirements for new applications, updates existing monitoring configurations.

**Blackbird Administrator**: Responsible for the maintenance and administration of the Blackbird system. Grants access to users and configures the BlackBird system according to monitoring requirements.
3.3 Actors

**Operator**: Accesses the monitoring pages generated by the Blackbird system and responds to alarms.

### 3.3.1 Deploy Monitoring Requirement Use Case

The initial deployment of a Monitoring Requirement will always be the most complex part of using the BlackBird System, it involves the configuration of all configuration objects necessary to gather the required data, compute Metrics and alarms, format graphics and pages. Thereby, it should be performed by a user with extensive knowledge of the BlackBird System, the Blackbird Administrator. Figure 3.2 presents the use case for deploying a new Monitoring Requirement.

![Deploy Monitoring Requirement Use Case](image)

**Figure 3.2**: Deploy Monitoring Requirement use case.

**Access Application List**: Access a screen containing a list of all applications currently being monitored. This screen allows access to the main screens and
3. Requirements

management screens for each of these applications. Access to the manage-
ment screens is restricted to users with an administrator role.

**Access Monitoring Object List:** Access the management screen that lists all
Configuration Objects currently deployed and list available edit actions. For
each type of Monitoring Object the available edit actions are: create, copy,
update and delete. Once the the Blackbird Administrator selects one of these
actions on one of the objects the user will be directed to an edit screen.

**Edit Monitoring Objec:** Access the edit screen, which is a form requesting all the
parameters necessary for creating a new instance of the requested Monitoring
Object, form content will depends on the action selected and the type of
Monitoring Object.

**Apply Changes:** Once confirmed, all changes must immediately take effect with-
oout interruption to the monitoring of other applications. Data gathering of
new commands is automatically started, new Metrics are immediately avail-
able for query, monitoring screens and graphics are rebuilt according to the
new format.

The Update Monitoring use case performed by the Application Owner involves
basically the same actions as the Deploy Monitoring Requirement use case but will
usually involve a small number of changes and Configuration Objects.

### 3.3.2 Visualize Monitoring Output Use Case

Once the Monitoring Requirement is deployed and the BlackBird System is
started, it will initiate data gathering and produce the graphics and pages that
compose the final result of the monitoring process. Now, application operators will
access these pages and visualize the graphics in order to determine the status of the
application. The use case for Visualize Monitoring Output is detailed in fig. 3.3.

**Access Application List:** Access a screen containing a list of all applications cur-
rently deployed and available to the user. This screen allows access to the main
screens for each of these applications.

**Main Application Screen:** Access a screen containing basic information about
the selected application, active alarms and a list of all graphics screens de-
ployed for the selected application.
3.4 Additional Requirements

Figure 3.3: Visualize Monitoring Output user case.

**Graphics Screen:** Access a screen containing a set of graphics and tables updated in real time that represent the output of the performance Metrics.

**Follow Drill Down Link:** Each graphic or table may have links to a screen containing more detailed information related to this graphic or table.

3.4 Additional Requirements

All data gathering must be performed without any changes to the Monitored Application and without installing additional components in the monitored system. The only exception may be scripts or program modules that can be deployed without rebuilding the application and are executed only upon a call from the monitoring system.

For the purpose if this work, the BlackBird System must be able to gather data from the Monitores Application by executing two kinds of commands, remote execution of SQL and invocation of WEB Services.

Databases are central components to every major application at Vodafone, they allow remote access in a controlled and secure way using the vendor independent protocols ODBC and JDBC. Most importantly, Databases implement great part of
3. Requirements

the business logic and are probably the best source of information about application status and performance. The BlackBird System must be capable of remotely access a database execute an arbitrary SQL statement and store the statement output in a normalized format that can be used as input for Metrics calculation.

WEB Services are a widely used technology at Vodafone DTSI, they provide support for many business critical applications and have enormous potential for usage expansion. The BlackBird System must capable of interpreting a Web Services Description Language (WSDL) description of a WEBService, invoke the required method using the arguments provided in the Monitoring Requirement, and store the output in a normalized format.
BlackBird Design and Implementation
4. BlackBird Design and Implementation

On this chapter we begin by presenting the application modeling definition to be used for this work and the data format for exchanging and storing performance data. Next, the Blackbird main architecture and the implementation is described, with emphasis on the management of stored data. To end we present the BlackBird interface for visualization and configuration.

4.1 Application Modeling

The objective of the Blackbird monitoring application is to provide a monitoring service to a range of applications as wide as possible. But in order to build a program that will interact with these applications and perform any useful task, it is necessary to define a minimum set of characteristics that are common to all applications. As proposed in chapter 2, by modeling the Monitored Application as a set of interacting components it should be possible to obtain a more detailed view of the application status and performance, also, it should be easier to obtain Metrics that closely relate to the application performance goals.

For monitoring purposes, which is the scope of this work, we consider the following simplified definition of application component:

- executes a well defined task within the application;
- can be univocally referenced;
- has a set of working parameters which can be obtained using the application interfaces.

Based on this definition of application component the model in fig. 4.1 represents the Monitored Application:

![Figure 4.1: Application Model.](image-url)
4.2 Architecture

Each of the required Configuration Objects defined in the requirements chapter is implemented by a dedicated class, except for Commands that is split between the Module class and subclasses dedicated to specific technologies, DatabaseScript and WebserviceRequest. The dataStore attribute of Module is a set of database objects that will store all data produced by the associated command and provide support to Metrics calculation. Figure 4.2 presents the high level Domain Model for the BlackBird System.

![Diagram](image-url)

**Figure 4.2:** Domain Model.

The monitoring process in the BlackBird System begins with data collection from the monitored application, this is performed by the combined work of the Module class and sub classes. The Module class is a virtual class that is used for controlling the execution of the sub classes and contains the dataStore where result data is stored. Each of the Module sub classes is designed to use specific protocols and may have technology specific attributes that must be provided in the Monitoring Requirement. For each Command specified in the monitoring requirement there is
4. BlackBird Design and Implementation

an instance of the appropriate Module sub class.

The Module sub classes provide an adaptation layer that isolates the BlackBird architecture from any technology details. The Module sub classes will handle all technology specific logic like establishing a connection, authentication, formatting the command, obtaining and validating the response. Then, the Module sub classes convert the command result to a normalized Extensible Markup Language (XML) document and deliver that document to the dataStore. Adding support for additional protocols to the BlackBird System only requires the creation of additional Module sub classes and requires no changes to existing components.

The Metric class provides the data processing and aggregation functionalities of the BlackBird System by computing the formula specified in the Monitoring Requirement. In order to provide a good base for Metrics calculation the dataStore is designed to be accessed as a relational entity having the same name as the Module it belongs, this way the Metric can be completely defined by a SQL select statement to be performed on the dataStore of existing Module objects.

The Graphic class provides the visual presentation to the Metric objects, it will use the output of the Metric formula as a data source and apply the type of graphic and the format request in the Monitoring Requirement. A Graphic may be a table of values or a chart as specified by the type attribute. A Graphic of type chart can also have different formats, line chart, bar chart and pie chart. The page_id attribute specifies to which Page this Graphic belongs, and the link attribute designates another Page containing more detailed information for root cause analysis.

The Page class provides the base for generating the monitoring interface that will be accessed and navigated by the operators. The Page object contains only a title and basic layout definition for placing the required Graphic. By combining the information from the Page and Graphic objects the BlackBird system generates the required interface, a WEB Applicaton containing the requested charts and tables.

The Alert class provide automatic notification of performance problems. It is defined as boolean condition to be evaluated, if true the Alert update its status and sends a notification message to the list of contacts defined for the application. This condition is defined as an SQL statement.

4.3 BlackBird Components

As required, the blackbird system will be implemented using an agentless architecture, it is composed of an Adaptation Layer itself composed by a variable number local Interface Modules designed for specific protocols that will handle command ex-
4.3 BlackBird Components

eecution, an Aggregation Layer that will handle data storage and Metrics calculation, and a Presentation Layer for generating the monitoring pages and graphics. The Component Diagram of the BlackBird System is represented in fig. 4.3.

![Component Diagram of BlackBird System](image)

Figure 4.3: BlackBird Components.

### 4.3.1 Adaptation Layer

The Adaptation Layer provides one of the main features of the BlackBird System, adaptation to the technologies of the Monitored Application, interfaces the Monitored Application using any protocols it provides, and generates performance data in a normalized format. On the BlackBird System all technology and protocol specific processing is performed by interface modules. Each type of interface module handles a specific technology or protocol, performs all the tasks necessary for executing the requested command, converts the command output to the normalized format and delivers it to the dataStore.

The Adaptation Layer also performs the first step for providing a monitoring service adapted to the architecture of the monitored system. By allowing multiple Module to execute independently it becomes possible to specify as many data sources as required for compiling a complete repository of performance data that will allow the calculation of any relevant Metrics.

Although only two types of Interface Modules were developed for this work, SQL execution and WER Services call, the BlackBird architecture is expandable to other
4. BlackBird Design and Implementation

types of protocols. Adding support for national protocols to the BlackBird system requires no changes to existing components only the addition of new components:

- develop a new interface module to handle the required protocol;
- create two new configuration tables and associated views;
- create new stored procedures for the edit operations on the new type of module.

The ModuleManager component controls execution of the Interface Modules and manages the `dataStore` of those objects. According to the Monitoring Requirement, it creates as many Interface Modules of each type as necessary, and creates or updates the `dataStore` for the Interface Modules whenever there is a change to the format of the result returned by the command.

4.3.2 Aggregation Layer

The Aggregation Layer stores all performance data in an organized an easily accessible form, performs Metrics calculation and alerts verification. The main component of the Aggregation Layer is the database, for this work we choose Microsoft SQL Server 2005, the primarily reason for this choice was the advanced support for XML data [15]. The Aggregation Layer contains the `dataStore` attributes from all existing `Module` objects implemented as a database tables and views, the combination of all these objects constitutes a complete repository of all performance data gathered from the Monitored Application. And, since all this data is accessible through queries, it should be possible to implement any Metric required by the Application Owner. The Aggregation Layer is responsible for other of the main features of the BlackBird System, Component Based Monitoring. Since all performance data gathered from the application can be used as input for the Metric objects, it is possible aggregate the data collected by different Modules to produce a set of Metrics that provide a complete view of all aspects for an Application Component or type for a class of Application Components.

The Adaptation Layer also contributes to adapting the monitoring service to the architecture of the Monitored Application by allowing the calculation of any Metrics necessary to represent all functions of the Monitored Application.

The MetricManager component is responsible for creating and updating the implementation of Metric formulas, and actively evaluating alert conditions.
4.3.3 Presentation Layer

The Presentation Layer is responsible for the final output of the BlackBird Monitoring System, which are monitoring pages containing visualizations of the status and performance of the Monitored Application.

The monitoring pages are generated from the Graphic and Page objects and deployed on an Application Server as a WEB Application. However, the automated generation of the WEB Application was considered to be out the scope of this work, so the Webcockpit application is used to generate the monitoring pages based on a configuration file generated by the BlackBird System from the existing Page and Graphic objects.

The PageManager component is responsible for managing the generation and deployment of the WEB Application that provides the monitoring interface.

4.4 Result Format

In order to isolate the BlackBird architecture from any technology details it is necessary to define a normalized format for the results data, this format must be:

- capable of containing any result;
- easily convertible to a relational format.

According to the application model defined at the beginning of this chapter fig. 4.1, and to support component level monitoring, the output of a command executed on the monitored system may be defined as a tree of components each containing any number of working parameters, including a component identifier. Also, considering that all components in a result are of the same type, all components in a result must have the same working parameters. According to this definition, any result may be formatted as a XML document with the schema from fig. 4.4.

The root element RESULT may contain any number of ROW elements, each ROW element represents an application component. Each ROW element may contain a variable number of different elements. The name and type of these elements can only be determined at run time from the command result, the diagram from fig. 4.4 shows an example where there are thee string parameters the first being the component identifier. The elements contained in ROW represent the component_ID and the associated working parameters. The attributes module_id, timestamp and response_time provide additional information related to the command execution.
4. BlackBird Design and Implementation

This format can be easily mapped to a relational entity whose attributes are the \texttt{ROW} sub elements, the resulting database entity from the schema in fig. 4.4 is presented in fig. 4.5.

4.5 Data Model

The BlackBird Database serves two purposes, storage and processing of Performance Data from the Monitored Applications, and support to the configuration and control of Monitoring Objects. The BlackBird System is intended for an enterprise environment where security and role separation is always a major concern, because of that for each Monitored Application there is a dedicated database schema for contain the implementations of \texttt{dataStore Metric} and \texttt{Alert} that exist for that application. The main BlackBird schema contains the Monitoring Objects definitions for all monitored applications.

4.5.1 Main Schema

The BlackBird schema presented in fig. 4.6 is used for controlling the monitoring process, contains Monitoring Objects definitions, and additional entities for implementing application separation and access control.
4.5 Data Model

Figure 4.6: Main Schema.
The application_owner entity represents Application Owners which may have any number of applications under their responsibility. Contains a database user for managing the monitoring configuration of applications it owns, group information and contacts.

The application entity represents applications being monitored by the BlackBird System, contains general information about the application including contacts, and a database user for managing Configuration Objects. Except for the application_owner and application, all entities have an owner attribute, that identifies to which application they belong.

The entities Metric, graphic, page and alert contain the definition for all Configuration Objects of the corresponding class. The Module class and sub classes are split into entities module, command and login. The command contains the protocol specific details of the command to execute, login contains protocol specific authentication details, and module provides a single point of control to different Commands independent of protocol specific information.

For this work only SQL and WEB modules where developed, their configuration entities are SQL_login, SQL_command, WEBSrv_login, WEBSrv_command. Additional Module sub classes for different protocols require no changes to existing tables, only the creation of two additional tables for commands and logins.

4.5.2 Monitored Application Schema

For each application there is a separate schema that contains the result data obtained from the Monitored Application. This schema will contain the database objects that implementation the dataStore and Metric, the creation and modification of all database objects in the Client Schema is performed automatically by the BlackBird System as a result of changes to the Configuration Objects. Figure 4.7 presents an example schema containing the implementation of two dataStore and two Metric objects.

The dataStore provides a vital function of the BlackBird System, it receives data in the form of XML documents and then provides access to that same data through a relational interface. This way, from the Application Owner point of view, for each configured Module there is a database entity of the same name that contains all result data, and all result parameters can be accessed as entity attributes.

The implementation of this function was a major issue since the beginning of this work and several implementation options where considered. The various available processes for loading XML data in relational tables, where considered either too inflexible or inadequate for real time monitoring. On the other hand, options based on
relational tables where also considered too inflexible. In the event of changes to the result format, which is frequent given the constant evolutions of Enterprise Applications, the adaptation to the new format would require alterations to already existing tables. These alterations where considered to complex to manage automatically.

The final implementation of dataStore is composed of a Results Table and a Translation View, result data is stored XML format in the Results Table, then based on the format of the XML result the BlackBird System creates a view for selecting all XML elements as columns of result set.

The Results Table is named results_{module_id} and contains all results produced from the command execution on the Monitored Application. It contains only two columns, result which is the XML document representing the result of a single command execution, timestamp that contains the date and time of the command execution. For improved performance, Results Tables have a primary XML index and a PROPERTY index on the result column [16][17].

The Translation View provides a relational like interface to the Results Table. In SQL Server, querying XML instances stored in an XML column requires the use of XML data type methods that take XML Query Language (XQuery) expressions for referencing elements in the XML column. In order to simplify Metrics definition,
the Translation View implements a query that decodes the XML ROW elements into result set rows and the XML parameter elements into columns.

The complex implementation of the dataStore is rewarded by an extremely simple implementation of the Metric objects, a database view is created using the SQL formula defined in the Metric. The created view will automatically provide the required output whenever queried.

The data stored in the Results Tables is provided by the Modules, therefore it is necessary a safe and controlled way for the modules to store their results on the Results Table. On each Application Domain Schema there is a stored procedure insert_result that acts as the single point of entry for results data, it takes as arguments the module_id and the XML result document, performs the necessary validations including detection of SQL injection attempts [18] [19], and inserts the XML document in the results table of the specified module. Module will login using the application database user and execute the insert_result procedure providing their module_id and the result document.

4.5.3 Client Schema Management

One of the most important functions of the BlackBird System is the management of the dataStore. Changes to a Module must be automatically applied to the corresponding dataStore. This task is performed by the ModuleManager program which also controls modules execution.

The creation of a new Module requires the creation of a new Results Table and the associated Decoding View. The Results Table has a fixed definition so it’s creation posses no problems, the Decoding View definition however, depends on the results format. Because it performs the mapping of XML elements in relational columns, the definition of the Decoding View requires information about the name and type of all sub element of the ROW element in the result document. Changes to an existing Module that alter the result format require and update to the translation view. Changes to the formula of a Metric also require an update to the view that implements that Metric. The diagram in fig. 4.8 describes the interactions that occur upon the creation of a new Module.

The ModuleManager periodically checks for changes to the configured Monitoring Requirement. If a new Module is created, the ModuleManager creates an new Module java object of the appropriate sub class and instructs the Module object to connect to the Monitored Application. Next, the ModuleManager requests a XML Schema Definition (XSD) which describes the XML document that the Module will produce for delivering command results. To determine the result format the Module will
execute the command on the Monitored Application for the first time and use the first command response to produce the XSD schema that will describe the result XML document. Based on the XSD schema the ModuleManager will obtain the name and type of all sub elements of the ROW element and compose a select statement for mapping the XML elements in relational columns. Based on that select statement the ModuleManager creates the Decoding View and Results Table.

Once the dataStore is created the Module is ready for continuous data collection the ModuleManager issues a start command. The Module will begin to periodically execute the command on the Monitored Application, convert the response to a XML result and send the XML result to the dataStore by invoking the insert_result...
stored procedure. The Module will continue to collect results until it receives a stop command, then the Java Object will terminate but the dataStore will remain in the database.

Changes to an existing Module produce the same interactions described in this diagram except that the dataStore is only updated. The Results Table is maintained with all its data, only the Decoding View is updated for mapping the new element structure.

The management process of Metric is much simpler, upon changes to the formula, the existing view is dropped and recreated according the new SQL select statement.

4.6 Interface Modules

For this work only two modules where developed, SQL execution on a remote database and for WEB Services call. These technologies are base to most of Vodafone applications, and allow the demonstration of the BlackBird capabilities in combining data obtained using different technologies.

The SQL module uses JDBC to connect to the monitored Database, and execute an SQL statement. This requires the installation of the appropriate Driver, authentication data and the SQL statement to execute. All major Database vendors provide JDBC drivers so this module should be capable of gathering data from any application that includes a database.

The WEB module uses the Apache Axis Framework for dynamically invoking Web services. It only requires the location of the WSDL definition of the Web Service, the method to invoke and the required arguments. The module will interpret the WSDL definition build the appropriate SOAP message, executes the Service Call and interprets the results.

4.7 User Interface

The BlackBird interface supports two main functions, visualization of the final output of the monitoring process, and configuration of the Configuration Objects. The interface is implemented as a WEB application running on an Apache Tomcat application server. Figure 4.9 represents the page structures of the BlackBird interface.

The starting page for both monitoring and management is the Application Owner Page, fig. 4.10. For each Application Owner it presents a list of all applications currently being monitored and provides links to the Application Pages and to the
Figure 4.9: BlackBird Interface.

classification page.

For each Monitored Application the Application Page, fig. 4.11, presents general information including contacts for alerts and a list of active alerts each with a links to a related graphic page. The left bar presents a link the Configuration Page and to all Graphics Pages configured and for this application.
4. BlackBird Design and Implementation

**BlackBird Monitoring Framework**

**Billing & Mediation Applications**

<table>
<thead>
<tr>
<th>Application</th>
<th>Active Alerts</th>
<th>Support Mail</th>
<th>Support Phone</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARBOR, production</td>
<td>0</td>
<td><a href="mailto:AppSUppARBOR@corp.vodafone.pt">AppSUppARBOR@corp.vodafone.pt</a></td>
<td>919012345</td>
<td>Billing &amp; Mediation</td>
</tr>
<tr>
<td>Portability, production</td>
<td>0</td>
<td><a href="mailto:AppSUppMed1@corp.vodafone.pt">AppSUppMed1@corp.vodafone.pt</a></td>
<td>919012345</td>
<td>Billing &amp; Mediation</td>
</tr>
<tr>
<td>PP8, production</td>
<td>0</td>
<td><a href="mailto:AppSUppPP8@corp.vodafone.pt">AppSUppPP8@corp.vodafone.pt</a></td>
<td>919012345</td>
<td>Billing &amp; Mediation</td>
</tr>
<tr>
<td>Mediation Device, production</td>
<td>2</td>
<td><a href="mailto:AppSUppMed1@corp.vodafone.pt">AppSUppMed1@corp.vodafone.pt</a></td>
<td>919012345</td>
<td>Billing &amp; Mediation</td>
</tr>
</tbody>
</table>

**Billing & Mediation Users**

<table>
<thead>
<tr>
<th>User</th>
<th>Username</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day support</td>
<td>md2</td>
</tr>
<tr>
<td>24h support</td>
<td>md2</td>
</tr>
</tbody>
</table>

Figure 4.10: Application Owner Page.

**Mediation Device, production**

Application Owner: It - Billing & Mediation
Support email: AMBILL@corp.vodafone.pt
Support phone: 919050020
Database User: NPM_MD

**Active Alerts**

<table>
<thead>
<tr>
<th>alert_id</th>
<th>alert_description</th>
<th>start_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit_size_delay_100</td>
<td>bit_size_delay 100 bytes</td>
<td>2007-01-01 00:00:00.00</td>
</tr>
</tbody>
</table>

Figure 4.11: Application Page.
4.7 User Interface

4.7.1 Monitoring interface

The final output of the monitoring process is an WEB application containing charts and tables that provide a visual representation of Metrics related to the performance goals of the monitored application, where these charts and tables are organized in pages according to the application architecture and are updated in real time. One of the main requirements for this work is that the this WEB application must be dynamic, all pages, charts and tables are generated from their definition in the database and updated when their definition is changed.

The development of such an interface can be by itself the subject of a thesis and goes beyond the scope of this work. There is already a wide range of applications capable of performing these tasks, so it was decided to employ an existing application to implement the monitoring interface. This application must have the following characteristics:

- web based;
- generate charts and tables;
- automatically update contents from a database;
- simple configuration;
- updatable without code generation.

For this work it was chosen the Webcockpit application. It generates complete Java Server Pages (JSP) based web applications which contain charts and tables, who’s contents are retrieved using database queries. Webcockpit configuration requires only an XML configuration file that specifies each JSP page, containing charts, tables and their interconnection.

The PageManager generates the Webcockpit configuration file from the existing Page and Graphic objects, executes the Webcockpit application for producing the WEB application package and deploys that package on the application server. Whenever there is an update to the definition of any page chart table, the PageManager generates and deploys a new package for that application.

The complete process of rebuilding the monitoring interface according to the new configuration is performed automatically and takes no more than a minute. This allows for the Application Owner to use the BlackBird System for performance tuning and for actively investigating performance problems.
4. BlackBird Design and Implementation

4.7.2 Graphics Pages

Graphics Pages, fig. 4.12, will contain the charts and tables specified in the monitoring configuration, the left bar presents a link to the Configuration Page and to all Graphics Pages configured and for this application.

![Figure 4.12: Graphics Page.](image)

Although the Webcockpit application allows many types of charts not all are applicable to the purpose of this work, the only chart types supported by the BlackBird System are bar, line, timeseries and pie.

4.7.3 Configuration Interface

Configuration pages, fig. 4.13, use the same application server and all active operations will also be implemented using JSPs, however these are static pages independent of any monitoring configuration.

Application Owners will access the Configuration Page by selecting the update configuration link of a specific application in the Application Owner page, the Application Owner will be prompted for the database user and password for this application. And upon successful access, this page presents a list of all Configuration
Objects of the various types currently configured for this application. The Application Owner may select an object to edit and select the edit operation options, update, copy and delete. Selecting any of these actions will take the Application Owner to the Edit Page, fig. 4.14.

The contents of the Edit page, fig. 4.15, depends on the type of monitoring object and the requested change. As described in the Main Schema section of this chapter, section 4.5.1, for each type of Monitoring Object there is a set of stored procedures for performing all valid changes, and a view for obtaining all relevant configuration data. Based on the type of Monitoring Object the JSP that implements this page determines which view and stored procedures must be used. From the output of the configuration view it determines the name and type of the required configuration parameters and generates a form for collecting these parameters. Upon confirmation from the user executes the appropriate stored procedure for applying the changes.

Because the Edit Pages are prepared determine the necessary management stored procedures from the monitoring object type, the addition of new Interface Modules does not require any changes to the Configuration Pages or Edit Pages, it only requires the creation of the appropriate stored procedures and views.
4. BlackBird Design and Implementation

Figure 4.14: Edit Page: Create WEBServ Module.

Figure 4.15: Edit Page: Create Metric.
5 Application Scenarios

Contents

5.1 Monitored Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
5.2 Monitoring Configuration . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
5.3 Configuration Process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
5.4 Configuration Example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
5.5 Monitoring Output . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
5. Application Scenarios

On this chapter we will present application examples that illustrate the use of the BlackBird Monitoring System for application monitoring and how the main features of the BlackBird System help improving the monitoring of Enterprise Applications. Next we detail the configuration process for one example, and present the final output of the BlackBird System.

5.1 Monitored Applications

The application examples focus on two applications currently under the responsibility of the Billing & Mediation team at Vodafone Portugal.

- Pre Paid Billing System (PPB).
- Mediation Device.

Both are Business Critical applications that require 24 hour support and monitoring. The monitoring system currently used for monitoring the applications is HP Openview Operations, this system provides complete network and device monitoring, and also application level monitoring based on application specific scripts and programs. These examples aim to extend the detail level of monitoring, by providing real time graphics and alerts on the performance goals of the applications, which is only possible through the combination of multiple sources of performance data.

As intended the BlackBird System is currently integrated in the production environment of these applications, all the results presented here where obtained from the production environments of the applications. Before migrating to the production environment all new Module objects are first tested on the application System Test environment. These testes are mainly on load tests, a high number of the new Module is configured using aggressive pooling intervals and timeout limitations, then a pre-defined set of normal application tasks is executed. The execution of these tasks under load is compared to a base line to determine if there is significant impact from the new Module definition.

5.1.1 Pre Paid Billing System

The first example will focus on the PPB System from Vodafone Portugal. The PPB application is responsible for lifecycle management of all pre-paid clients, interacts with SIEBEL which is the Customer Relationship Management (CRM) solution employed by Vodafone Portugal, and with the Intelligent Network system form Alcatel-Lucent that handles real time rating and the management of the costumer
balance. Communication between these systems is provided by a middleware of WEB Services from WebMethods. The PPB application was developed at Vodafone Portugal, it contains an Oracle database and was developed using C and PL/SQL.

There is already a dedicated monitoring system for PPB which was developed together with the main application it provides performance information and alerts. Although it delivers extremely valuable information, it implements a static set of Configuration Objects and is not applicable to any other applications. The present example will add monitoring of recent functionalities of the PPB application that have not yet been included in the dedicated monitoring system, currently only basic unavailability alerts are provided.

**Pre Paid Billing System Monitoring Example**

One of the tasks performed by the PPB application are credits to the costumer balance in the IN platform. Recent Business Requirements "Vita Recarga" and "SOS Extra" have introduced new scenarios on which it is necessary to apply credits to the customer balance. When the condition defined in the Business Requirement the PPB system generates a request for balance credit, this request is then added to a queue for processing. The processing queue exists as a database table containing requests to be processed, and the processing of each request requires a call to a WebService for confirming current balance.

The main performance criteria for this task are the average processing time and maximum processing time for these requests, the current monitoring includes alarms if thresholds on these parameters are exceeded. For monitoring this aspect of the PPB application, problems diagnosis and performance tuning, it would be extremely helpful to have a line chart presenting the following parameters:

- **proc_time**: average request execution time;
- **web_time**: average WebService response time;
- **ppb_time**: proc.time - web.time, which represents the contribution of the PPB System to the total processing time.

This is a relatively simple example, however, the presented parameters have direct relation to the application performance goal. Furthermore, by splitting the application goal proc.time into sub parameters, it becomes easier to identify the contribution of each application component to the final result. By monitoring these parameters and implementing additional alerts it should be possible identify problems before they produce an impact on the negotiated SLA.
5. Application Scenarios

The output of this graphic is also helpful for performance tuning. Since it illustrates the impact of configuration changes on each of the parameters that contribute to the application performance goal, it should be easier to establish a compromise that produces the best overall results.

5.1.2 Mediation Device

The second example applies to the Mediation Device Rating System also from Vodafone Portugal. The Mediation Device processes billing records of all traffic types, GSM, 3G, GPRS, SMS, etc. Collects billing records from all Network Elements, validates and selects billable records, pre-rates and sends billable records to the billing systems ARBOR and PPB. The Mediation Device was also developed at Vodafone Portugal, it contains an Sybase database and was developed using C and Transact SQL.

Currently all monitoring and alerts on the Mediation Device Application are provided by scripts and programs developed for providing full coverage of all application tasks, all integrated in the Openview Operations System. The present example will use some of the same scripts as input and use the data processing and data processing capabilities of the BlackBird System for providing Metrics directly related to the application performance goals.

Mediation Device Monitoring Example

The fundamental task of the Mediation Device is the continuously processing of call records from all Network Elements and deliver the billed records to the destination billing systems. The most critical SLA defined for the Mediation Device are all related to the delay between record generation on the network element and its delivery to the destination billing systems. Given these, the fundamental indicator of performance for the Mediation Device is the number of records processed per unit of time.

The Mediation Device processes various types of records refereed to as Data Stream, and for each Data Stream, there are various Network Elements. Thereby, the Mediation Device architecture can easily be modeled into components. The basic components are the Network Elements that have parameters such as curet delay and records processing rate. These elements can be grouped into the main application components the Data Streams, whose parameters are an aggregation of the sub element parameters.

Existing scripts and programs integrated in Openview provide alerts on the processing status of processes and processing delay. This example uses the existing
scripts as data source for creating charts on the delay associated with network ele-
ments and records processing rate according to type of traffic. In this case all the
necessary information can be obtained from the Mediation Device Database using
only SQL Modules.

5.2 Monitoring Configuration

Once defined the required output, the Application Owner must elaborate the
Monitoring Requirement in order to specify the Configuration Objects necessary
for producing the desired monitoring output. The Monitoring Requirement for the
PPB example contains the following Configuration Objects:

log_PPDB: Login details for the PPB database.

log_PPWEB: Login details for WEB Service calls.

cmd_queue_time: SQL command for obtaining the average processing time, where
the result contains the parameter proc_time.

cmd_web_time: WEB command for obtaining the Web Service response time, where
all results have a response_time attribute.

mtr_queue_time: Metric for combining data from both commands.

graph_queue_time: A line chart that uses mtr_queue_time as input.

pg_queue: A page for containing the graph_queue_time graphic.

The task of creating the Monitoring Requirement may seem an excessive burden
on the Application Owner. However, there was a great effort in order to reduce the
required configuration details to the absolute minimum necessary for defining the
complete monitoring of an application of which only the Application Owner has the
required knowledge.

The Module object is intended to execute commands on the Monitored Application. The definition of the commands necessary for obtaining all the required data
must to be performed by someone with detailed knowledge of the target application.
The best source for this knowledge are the Application Owner or the Application
Developers. The Page and Graphic Objects only require simple format parameters. The definition of Metric objects is one task that specifically requires advanced
knowledge, it requires good understanding of Transact SQL. However, the Black-Bird System is to be used by the support and operations teams, which are usually required to have such knowledge.

By providing a simple set of Configuration Objects, the BlackBird System frees the Application Owner from most implementation details and allows him to focus on the creation of more complex monitoring Metrics.

The complete Monitoring Requirement for the PPP application example can be found in Appendix A.

## 5.3 Configuration Process

The configuration of the Monitoring Requirement can be performed either by the BlackBird Manager or the Application Owner using the Configuration Interface described in the previous chapter fig. 4.13 and fig. 4.14.

The diagram in fig. 5.1 describes the sequence of actions necessary to the configuration of the of a Monitoring Object. Starting from the Configuration Page the user selects and action to be applied on an existing Configuration Object or the creation of a new object. Then the user is presented with a form for providing or updating the configuration parameters of the object. Upon edit confirmation the user is returned to the Configuration Page and the BlackBird Manager proceeds to execute all the necessary actions such as creating and updating the objects in the Application Schema.

The user must first create all necessary Module objects, the creation of a new Module automatically triggers the creation of the database objects that compose the dataSource, that will be used by the Metric objects. Next the user defines all Metric objects so that the corresponding database views are ready to be used by the Graphic and Alert objects. Finally the user creates the Page Objects.
5.3 Configuration Process

Figure 5.1: Configuration Sequence Diagram.
5. Application Scenarios

5.4 Configuration Example

Next we present some of the Edit Pages for the configuration of the PPB example. The configuration on the SQ_Login log_PPBDDB, fig. 5.2, requires only the basic parameters for a JDBC connection, class of the appropriate JDBC driver, a JDBC connect string a user and password.

Figure 5.2: Edit Page: Create SQ_Login.

The configuration of the SQL Module cmd_queue_time, fig. 5.3, requires the query to be executed, and like all modules requires a login of the appropriate type, pooling interval in minutes and timeout in seconds.

Figure 5.3: Edit Page: Create SQL_Module.
The configuration of the page `Queue Time`, fig. 5.4, requires the type Chart or Table, for charts it requires the format, timeseries, pie, bar, etc. Requires the Metric that will be the source of data and the Page where it will be displayed. Finally some additional layout information, a title for the graphic, size and position.

**Figure 5.4: Edit Page: Create Graphic.**
5. Application Scenarios

The configuration of the Metric mtr_queue_time, fig. 5.5, requires only the SQL select statement that is the formula.

![BlackBird Monitoring Framework](image)

**PPB: Create Metric**

| metric_id: | mtr_queue_time |
| metric_description: | Queue processing times |
| query: | SELECT 'prod_time', cmd_queue_time.timestamp, cmd_queue_time.prod_time FROM cmd_queue_time, cmd_web_time WHERE cmd_web_time.timestamp > cmd_queue_time.timestamp AND cmd_queue_time.timestamp > GETDATE() - 1 union SELECT 'web_time', cmd_queue_time.timestamp, DATEDIFF(msec,cmd_queue_time.response_time)*4 / 1000 as web_time FROM cmd_queue_time, cmd_web_time WHERE |

Figure 5.5: Edit Page: Create Metric.

The configuration of the Page pg_queue, fig. 5.6, requires a title for the page and two dimension parameters that specify the separation of the page in virtual rows and columns for containing Graphics.

![BlackBird Monitoring Framework](image)

**PPB: Create Page**

| page_id: | pg_queue |
| page_description: | PPS Queues |
| page_title: | PPS Queues |
| dimension_x: | 0 |
| dimension_y: | 0 |

Figure 5.6: Edit Page: Create Page.
Figure 5.7 shows the monitoring configuration page for the PPB application after all this example is completely configured.

Figure 5.7: PPB Monitoring Configuration Page.

The complete description of all configuration parameters for the various Configuration Objects can be found on Appendix B.
5. Application Scenarios

5.5 Monitoring Output

The final output of the PPB monitoring example is shown in figure 5.8.

![Figure 5.8: PPB Queue.](image)

This chart shows the evolution of the requests processing time and the contributions of the WEB Service call and PPB processing. This capture shows normal conditions, where the contribution of WEB Service call to final processing time is negligible. Next, figures 5.9 and 5.10 show the output pages for the Mediation Device application. In this example the BlackBird System was used for fast implementation of a detailed monitoring service, and to generate Metrics that aggregate data obtained using different technologies, Database access and WEB Services call.

![Figure 5.9: Mediation Device Network Elements.](image)
5.5 Monitoring Output

The chart from figure 5.9 shows the evolution of the delay in Call Records processing for the Network Element components of the Mediation Device application, only the ones that present a delay above a defined threshold are shown. This is a batch process, because of that the delay tends to evolve in steps, this figure was captured during a peak hour so there is a tendency for an increase in processing delay. On the bottom there is a table that shows current processing status of Network Elements.

![BlackBird Monitoring Framework](image)

Figure 5.10: Mediation Device Streams.

The second figure 5.10 shows a chart presenting the evolution of number of Call Records processed per hour for the Stream components of the Mediation Device application, this figure was captured during a maintenance operation that required stopping the processing of some types of traffic.

In this example the BlackBird System was used to improve the existing monitoring to detail the information on the component level, and provide real time visualization of Metrics directly related to the SLA and performance indicators.

These examples are not enough to demonstrate the full capabilities of the BlackBird System, all aspects the architecture are aimed at flexibility, so that an Application Owner may implement any monitoring he regards as necessary. However, these simple examples are enough to show that the BlackBird System can be effectively used for providing an advanced monitoring service to real world enterprise application.
5. Application Scenarios
Conclusions

Contents

6.1 Results .................................................. 58
6.2 Future Work ............................................. 60
This chapter summarizes the entire work. On the results section we review the proposed objectives, then we discuss how the various components of the BlackBird architecture contribute to achieving those results. Finally, we analyze the application examples. On the Future Work section we analyze improvements for the existing functionalities of the BlackBird, and propose additional functionalities to be developed based on the BlackBird architecture.

6.1 Results

The final result of this work is an application that will be used at Vodafone Portugal to assist in the managing of the applications under the responsibility of the DTSI. It will be used for:

- monitoring a range of applications as diverse as possible, so that it may be applied to the wide range of application in use at Vodafone Portugal;

- providing complex Metrics that relate to the main application goals, for complementing current monitoring systems currently deployed at Vodafone Portugal.

Based on the analysis of existing monitoring systems and the targeted environment, it was determined that for performing these tasks the BlackBird System must provide these features:

- monitor a range of applications as diverse as possible, and adaptable to the architecture and technologies of the Monitored Application, so that it may be applied to the wide range of application in use at Vodafone Portugal;

- providing complex Metrics that relate to the main application goals, for complementing current monitoring systems currently deployed at Vodafone Portugal.

- provide in depth application level monitoring, component based;

- easily adaptable to the evolution of the Monitored Application;

- low impact on the monitored system, agentless and without additional application instrumentation;

- graphic interface for data visualization and configuration.
6.1 Results

The use of an Adaptation Layer of technology specific Interface Modules, allows data collection using any interfaces provided by the application. Combined with the possibility of using any number of independent Module, it becomes possible to adapt to any system simply by creating all the Module required for collecting all relevant data from the monitored application.

By defining a normalized data format for the information gathered from the Monitored Application, and by organizing all data performance data in an easily accessible repository, it becomes possible to define any required Metric.

The use of a simplified definition of component for application modeling allows the creation of Metrics on component performance parameters. Metrics that can be used for describing how the various components contribute to the global performance goals.

Also because of the modules based Adaptation Layer, the BlackBird System can be adapted to changes on the Monitored Application that require additional data to be gathered. And, as a result of the automated management of the data repository, the BlackBird System can also adapt to changes on the format of the data obtained from the target application.

All data gathering is performed by remote access to the interfaces provided by the from the Monitored Application, no changes are required on the Monitored Application, only general access for command execution is required.

The monitoring interface is based on the Webcockpit that provides a rich set of chart formats and tables, based on the output of the defined Metric. The configuration interface allows full control over all aspects of the monitoring process, and fast implementation of configuration changes.

Finally, the application examples of BlackBird System on production Enterprise Applications have demonstrated that the BlackBird System can be used for:

- fast implementation of a detailed monitoring service;
- generating Metrics that aggregate data obtained using different technologies;
- improving the existing monitoring to detail the information on the component;
- providing real time visualization of Metrics directly related to the SLAs an performance indicators.
6. Conclusions

6.2 Future Work

In terms of data collection and interface to the Monitored Applications, the natural improvement is the development of additional Interface Modules for additional protocols: Remote Shell for access to unix systems, JMX for J2EE applications, WMI for .NET application.

Although the data repository provides easily accessible source of data, the definition of Metrics is still the most demanding configuration task. This task can be greatly simplified by the addition of a query building interface, either as an applet on the Edit Page or as a separate application.

For visualization purposes, it could be possible to have better drill down capabilities and root cause analysis by automatically generating a hierarchical view of the Monitored Application, such as the one provided by the Spotlight [11] monitoring system. This would require an improved definition of application modeling for describing also the relations between applications components.

The alerts functionality is not yet completed, and it would advantageous to integrate these alerts on the existing Openview system.
Bibliography


Appendix A

Contents

A.1 PPB Monitoring Requirement .......................... 64
A. Appendix A

A.1 PPB Monitoring Requirement

**WEBServ login**

| login_id:   | PPB          |
| login_description: | PPB mw user |
| client_user:  | ppb          |
| client_password: | xyz         |

**WEBServ Module**

| module_id:       | cmd_web_time       |
| web_service_wsd: | VfIN_Flow_ECommerceGateway_RequestBalance.wsdl |
| web_operation:   | RequestBalance    |
| web_args:        | 351918217915&&&&&&&1 |
| output_fields:   |               |
| pooling:         | 1              |
| timeout:         | 30             |

**SQL Login**

| login_id:   | ppb          |
| login_description: | PPB database, production |
| JDBC_driver: | oracle.jdbc.driver.OracleDriver |
| JDBC_connect_string: | jdbc:oracle:thin:@vfpppppp:1521:ppbppb |
| DB_user:     | userppb      |
| DB_password: | passppb      |
### SQL Module

| module_id: | cmd_queue_time |
| module_description: | Queue mean processing time |
| query: | select trunc(avg((LAST_UPDATE_DATE - EFFECTIVE_CREATION_DATE)*24*60*60)) "proc_time" from SERVICE_ACTIONS_HISTORY where EFFECTIVE_DATE >= sysdate - 1/24/60 and ACTION_TYPE = 'LR' |
| pooling: | 1 |
| timeout: | 30 |

### Metric

| metric_id: | Valid |
| metric_description: | Queue processing times |
| query: | SELECT 'proc_time', cmd_queue_time.timestamp, cmd_queue_time.proc_time /ncmd_queue_time.proc_time /nFROM cmd_queue_time, cmd_web_time WHERE cmd_web_time.timestamp = cmd_queue_time.timestamp AND cmd_queue_time.timestamp >= GETDATE() - 1 |

### Page

| page_id: | pag_queue |
| page_description: | PPB Queues |
| page_title: | PPB Queues |
| dimension_x: | 1 |
| dimension_y: | 1 |
### Graphic

| graphic_id: | grph_queue_time |
| graphic_description: | Queue processing times |
| graphic_type: | 1 |
| graphic_format: | timeseries |
| graphic_title: | Queue processing times |
| page_id: | pag_queue |
| metric_id: | mtr_queue_time |
| position_x: | 1 |
| position_y: | 1 |
| size_x: | 600 |
| size_y: | 300 |
Appendix B

Contents

B.1 Configuration Parameters .......................... 68
## B.1 Configuration Parameters

### WEBServ login

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_id</td>
<td>An identifier, max 50 char, with no spaces.</td>
</tr>
<tr>
<td>login_description</td>
<td>Object Description, max 50 char.</td>
</tr>
<tr>
<td>client_user</td>
<td>Username for WEB service call</td>
</tr>
<tr>
<td>client_password</td>
<td>Password for WEB service call</td>
</tr>
</tbody>
</table>

### WEBServ Module

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module_id</td>
<td>An identifier, max 50 char, with no spaces.</td>
</tr>
<tr>
<td>web_service_wSDL</td>
<td>URL/Path of the WEB Service WSDL.</td>
</tr>
<tr>
<td>web_operation</td>
<td>WEB Service method to invoke.</td>
</tr>
<tr>
<td>web_args</td>
<td>Call arguments separated by &amp;.</td>
</tr>
<tr>
<td>output_fields</td>
<td>Output parameters to retrieve, if blank, retrieves all parameters.</td>
</tr>
<tr>
<td>pooling</td>
<td>Pooling Internal in minutes.</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout in seconds.</td>
</tr>
</tbody>
</table>

### SQL Login

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_id</td>
<td>An identifier, max 50 char, with no spaces</td>
</tr>
<tr>
<td>login_description</td>
<td>Object Description, max 100 char.</td>
</tr>
<tr>
<td>JDBC_driver</td>
<td>Class of the JDBC driver appropriate to the target Database</td>
</tr>
<tr>
<td>JDBC_connect_string</td>
<td>JDBC connect string for the target Database</td>
</tr>
<tr>
<td>DB_user</td>
<td>Username for the target database.</td>
</tr>
<tr>
<td>DB_password</td>
<td>Password for the target database.</td>
</tr>
</tbody>
</table>
## B.1 Configuration Parameters

### SQL Module

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module_id</td>
<td>An identifier, max 50 char, with no spaces</td>
</tr>
<tr>
<td>module_description</td>
<td>Object Description, max 100 char.</td>
</tr>
<tr>
<td>query</td>
<td>An valid SQL statement to be executed on the target database</td>
</tr>
<tr>
<td>pooling</td>
<td>Pooling Internal in minutes.</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout in seconds</td>
</tr>
</tbody>
</table>

### Metric

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric_id</td>
<td>An identifier, max 50 char, with no spaces</td>
</tr>
<tr>
<td>metric_description</td>
<td>Object Description, max 100 char.</td>
</tr>
<tr>
<td>query</td>
<td>An valid SQL statement to be executed on the monitored application schema.</td>
</tr>
</tbody>
</table>

### Page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>page_id</td>
<td>An identifier, max 50 char, with no spaces</td>
</tr>
<tr>
<td>page_description</td>
<td>Object Description, max 100 char.</td>
</tr>
<tr>
<td>page_title</td>
<td>Page title, max 100 char.</td>
</tr>
<tr>
<td>dimension_x</td>
<td>Number of virtual rows for containing Graphics.</td>
</tr>
<tr>
<td>dimension_y</td>
<td>Number of virtual columns for containing Graphics.</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
</tr>
<tr>
<td>graphic_id</td>
<td>An identifier, max 50 char, with no spaces</td>
</tr>
<tr>
<td>graphic_description</td>
<td>Object Description, max 100 char.</td>
</tr>
<tr>
<td>graphic_type</td>
<td>Graphic type, 1 for Chart, 2 for table.</td>
</tr>
<tr>
<td>graphic_format</td>
<td>Chart format permitted values are: timeseries, horizontalBar, line, pie, verticalBar, xy</td>
</tr>
<tr>
<td>graphic_title</td>
<td>Graphic title, max 100 char.</td>
</tr>
<tr>
<td>page_id</td>
<td>Page for containing this chart.</td>
</tr>
<tr>
<td>metric_id</td>
<td>Metric to be used as source.</td>
</tr>
<tr>
<td>position_x</td>
<td>Virtual row of the containing page.</td>
</tr>
<tr>
<td>position_y</td>
<td>Virtual column of the containing page.</td>
</tr>
<tr>
<td>size_x</td>
<td>Graphic width.</td>
</tr>
<tr>
<td>size_y</td>
<td>Graphic height.</td>
</tr>
</tbody>
</table>