uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Bo Zhan

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

An Integrated Quality Assurance Framework for Enterprise Performance Management Systems

TITRE DE LA THÈSE / TITLE OF THESIS

Liam Peyton

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Thomas Tran (absent)

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

D. Amyot

M. Weiss

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

An Integrated Quality Assurance Framework for Enterprise Performance Management Systems

Bo Zhan

Thesis

Submitted to the Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the degree of Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering

University of Ottawa

Your file Votre référence
ISBN: 978-0-494-34118-6
Our file Notre référence
ISBN: 978-0-494-34118-6

Abstract

Increasingly, large organizations, such as governments and hospitals, are deploying complex on-line enterprise performance management systems, which support business performance management by analyzing the data collected from their operational systems into data warehouses. Those systems are often run on a network of centralized servers and are accessed by thousands of geographically dispersed users via browser-based interfaces.

To be successful, an enterprise performance management system must be scalable and it must be reliable to provide critical business services 24 hours per day x 7 days per week. The system also must be accepted by users, which means it will be really used so that the value of the system to the business can be realized. Data in the performance management system can be sensitive; hence, the system must protect privacy when delivering data and reports to users.

This thesis defines an integrated quality assurance framework which addresses scalability, reliability, usage and privacy issues for enterprise performance management systems. Compared to black box testing, the integrated quality assurance framework has several advantages. It uses basic technologies (metadata modeling, reporting, portal, etc.) of an enterprise performance management system to provide deep analysis on system logs to measure quality both during development of the system and when it is in production. This thesis performs a case study using our approach by simulating the enterprise performance

management system for a major teaching hospital in our lab environment and integrating our quality assurance framework.

Often the data in the production environment is highly sensitive and its access strictly regulated by privacy law. To accurately assess the quality of the performance management system in a test environment, this thesis presents a strategy for generating test data with similar characteristics to the real sensitive data, and we have a Java-based tool implemented and applied in our research.

Acknowledgment

First, I would like to thank my family. Throughout my study and research in past years, your love and support, and endless patience have been truly inspirational.

The completion of this thesis would not have been possible without the direct support of Dr. Liam Peyton who introduced me to the field of business intelligence and performance management and gave me important guidance and support during my research. I owe a lot of thanks to Dr. Daniel Amyot and Dr. Thomas T. Tran for their advice and support.

This thesis is a part of the project "Evolving E-Health Business Process Around Accessible Data Warehouse" which was funded by the ORNEC (Ontario Research Network for Electronic Commerce). I would acknowledge the whole project team for their help in my research.

I would also like to acknowledge the Ontario Research Network for Electronic Commerce (ORNEC), Cognos and IBM, who provided funding, software, hardware, and training for my research.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AJAX	Asynchronous JavaScript and XML
BI	Business Intelligence
BPM	Business Performance Management
CSV	Comma Separated Value (File Format)
DTD	Document Type Definitions
ETL	Extract, Transform, Load (data)
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IIS	Internet Information Service
IT	Information Technology
JDBC	Java Database Connectivity
KPI	Key Performance Indicator
MS	Microsoft
OLAP	Online Analytical Processing
PK	Primary Key
QA	Quality Assurance
QoS	Quality of Service
SOA	Service Oriented Architecture
SQL	Structured Query Language
XML	Extensible Markup Language

Chapter 1. Introduction

Today's business environment requires the capability of sensing and responding to business situations in a timely fashion [Jeng2006]. Increasingly, large organizations are deploying complex on-line enterprise performance management systems that are used to run and manage critical business processes [Jarrar2000]. On-line enterprise applications that are used by thousands of geographically dispersed users present special challenges for quality assurance.

To be successful, an enterprise performance management system must be scalable. An effective architecture should meet the demands of enterprise-scale IT environments, report data on millions of business event and serve the needs of thousands of business users. Scalability is critical.

To be successful, an enterprise performance management system also has to be reliable. It plays a very important role in business day-to-day operations, and must be run in a 24*7 manner. On the other hand the complexity of IT infrastructure in organizations produces uncertainty on reliability especially as the IT infrastructure changes and the data itself evolves.

To be successful, an enterprise performance management system has to be used. Reports are useful for a business to measure the health of the whole business, for example, quarterly financial and operating results. They are also useful for a business to measure specific activities, for example, a report to measure drug usage for infection control in hospital environments. But the performance management system actually creates more pres-

sure on the people within the organization. They have to deal with the increased volumes and competitive pressures that technology creates, which make people reluctant to accept the system and use reports provided; therefore, there has to be a way to monitor the use of the system to ensure that the reports and data delivered by the system are actually being used.

To be successful, an enterprise performance management system has to protect privacy when delivering data and reports to users. Many enterprises store very large volume customer information, which is sensitive to the public. For example, name, telephone, address, social insurance number, credit card number, etc. Privacy violation will pose a risk to the life of an individual; also and it creates liability and reputation risks for an organization. To protect both enterprises and customers, privacy protection must be applied in a performance management system and the quality must be ensured.

This thesis has been built into following sections. Section one addresses the research issues and contributions. Section two introduces the research background and related works with respect to enterprise performance management systems, and quality assurance frameworks. Section three presents the integrated quality assurance framework and section four explains our new mechanism for generating test data for data warehouses. Section five presents a case study using our approach by simulating the enterprise performance management system for a major teaching hospital in our lab environment and integrating our quality assurance framework. Section six provides an evaluation on the quality assurance provided by the traditional framework and the integrated framework, and an evaluation on the test data generation performed by the tools adopted. Finally, in section seven we present our conclusions and directions for future work.

1.1. Issues Addressed

Development of enterprise performance management systems relates to several aspects (business methodology, goals/objectives, operational system, reporting system and process, data collection system and process, IT infrastructure), and those aspects are in continuous change to support increased business need and provide better services. Currently technology experts use professional testing tools [PerformanceTester2007] [OpenSTA 2007][Htmlunit2007] to perform black box testing to ensure the scalability and reliability of web-based performance management systems, but those testing tools have several limitations:

- None of them address the service oriented architecture of the performance management system since they only capture responses at the user interface level that reflect symptoms of quality issues, and cannot easily address which components and services inside the system may be the source of the quality issue. From an administrator's point of view, the latter is more valuable than the former because she needs to know exactly which parts of the system cause quality problems and then solve these problems.
- Black box testing tools can collect test result into logs and provide reports on data in log, but they do not provide deep analysis on that data. To make deep analysis on data in logs, reports with complex aggregations on those data are needed. Due to a lack of data integration, modeling and reporting technology, the data reported by these tools are quite simple and insufficient for users to notice a quality issue and investigate and determine the cause of an issue.

- Current testing tools do not provide portals that make it easy to navigate and organize test results in easy to understand reports, in the way a performance management system does for enterprise reporting.
- Black box testing typically only measures performance during development of the performance management system, but much of the change and the evolution of the performance management system happens in production on sensitive data as reports are developed and data collected over time. There is no mechanism for keeping quality assurance framework in production.
- Administrators are not alerted when exceptions on system qualities happen so that necessary actions can be performed to avoid system failure. To provide this function, metrics which reflecting system qualities have to be defined and monitored as the system is running. However we did not find any implementation of this function in current testing tools.
- Performance management system usage is not tracked in black box testing since it is based on simulations of users. Detail information about how reports are being used is needed to do this task.

1.2. Thesis Contributions

The contributions of this thesis include:

- An integrated quality assurance framework for enterprise performance management systems. The integrated framework enables deep analysis on logs made by different components/services across the architecture of the performance management system to ensure scalability, reliability, usage of the system as well as

addressing privacy issues both during the development the system and when it is in production.

- A data warehouse architecture for quality assurance data analysis. This architecture leverages data warehouse technology to organize quality assurance data and has a "quality assurance portal" to help users create and manage quality assurance reports providing analysis on quality assurance data both at detail and aggregation levels. Users can customize the portal to combine different graphs and reports for different goals and explore the quality of the system and its components to locate problems and bottlenecks.
- A model for test data generation for data warehouses and data marts. This model uses a formal XML-based specification to generate time-based fact tables in a data warehouse. It explicitly defines a data warehouse star schema with dimension tables and fact tables. A test data generation tool has been developed against this model.

These contributions have been validated in a case study applied to the enterprise performance management system and quality assurance issues of a major teaching hospital in Ontario.

Chapter 2. Background

2.1. Business Performance Management

Business performance management (BPM) enables an organization to understand the status of business processes across business and IT, in context against goals and trends, in order to improve business operations [Jarrar2000]. BPM is often used to enable specific management initiatives like balanced scorecards, Total Quality Management, Six Sigma, and Sarbanes-Oxley.

In [Jeng2006] BPM is organized into five stages: Sense, Detect, Analyze, Decide and Execute. "Sense" is the stage when a BPM system interacts with business solutions and provides data extraction, transformation, and loading capabilities for the sake of preparing qualified data (metrics) that is to be further monitored and analyzed. "Detect" is the stage of detecting business situations or exceptions occurring in the business solutions. An example of situation detection could be lower-than-expected cycle time performance in certain manufacturing processes. "Analyze" is the stage when a BPM system performs business analytics such as risk-based analysis of resolving business exceptions. The output of this stage often comes with recommendation of potential resolutions to decision makers. "Decide" is the stage when a decision maker will make decision about what to respond to business situations. A decision maker can be either human or software agent. "Execute" is the stage when a BPM system carries out actions for the purpose of enforc-

ing the decisions made by decision makers. Actions can be of many forms such as sending email or as complicated as a sub-process invocation.

2.2. Data Warehouse

A data warehouse is a subject-oriented, integrated, time-variant, non-volatile collection of information that is optimized for monitoring and analysis in support of management's decision-making processes [Inmon1992]. Considering the ability to consolidate and summarize data from heterogeneous legacy systems, more organizations are interested in integrating data from operational systems into data warehouse to supports rich query for answering complex business questions.

To build a data warehouse, data is extracted from existing legacy or operational systems, transformed into a common universal format, and loaded (ETL) into databases, usually at a regular frequency (e.g. daily). Since operational data varies in completeness, format, storage and place, professional tools are often adopted in ETL processes.

Data in a data warehouse can be further organized in data marts. A data mart is defined as "a logical subset of the complete data warehouse" and "is probably sponsored by and built by a single part of the business, and a data mart is usually organized around a single business process"[Kimball2002]. Data marts contain a snapshot of operational data which reduce the data complexity to users with different needs.

To enable high performance access to the data in data warehouse, the data has to be presented in a standard framework. The dimensional model [Innon1992][Kimball2002] is a widely accepted approach, and strategies and extensions on dimensional modeling have been addressed [Berndt2001][Alan2005][Lawyer2004]. A dimensional model is com-

posed of one fact table and a set of related dimension tables. A fact table consists of one or more numerical facts which can be used to measure a business process, as well as foreign keys connecting to primary keys of related dimension tables. A dimension table, by contrast, contains attributes that are often descriptive textual information structured hierarchically as well as a primary key. This characteristic star-like structure is often called a star join schema, or star schema. In a star schema, "the dimension attributes are the source of most of the interesting constraints in data warehouse queries and are virtually always the source of the row headers in the Structured Query Language (SQL) answer set [Kimball2006]". This structure provides a logical data cube which enables On Line Analytical Processing (OLAP) which can quickly provide answers to dimensional analytical queries.

Data warehousing is a complicated task. Operational data comes from many sources which can be widely decentralized, and it is delivered in many forms such as paper, files and databases, which need much effort for extracting, cleaning and formatting data. In specific areas, it may present unexpected challenges; for example, in a hospital environment, medical standards and coding schemas are very different among hospitals and many of them are incompatible and require careful translations. Data warehouse modeling is a complicated task, which involves both knowledge of business processes, as well as familiarity with the structure and behavior of operational information systems. After setting up a data warehouse, the performance of data warehouse is another big issue, for example, as data volume grows on analytical systems, queries will take more time.

2.3. Enterprise Performance Management Systems

Enterprise systems are often geographically distributed with thousands of users and they are increasingly strained by the sheer volume of data that is consumed, generated, and manipulated during the course of everyday operations. Enterprise performance management systems that support BPM provide services to track, manage, and report on data collected from across the organization's business operations into enterprise-wide data warehouses and data marts.

Business performance management aims to resolve business integration issues by:

- providing an integrated framework,
- enabling an organization to understand the status of business processes across business and IT,
- placing that understanding in context against goal and trends and taking timely actions to improve business operations [Jarrar2000].

BPM helps organizations plan where the business is heading and manage progress against plans, convey strategy and key deliverables, assign accountability across the business, analyze business information to understand what is driving trends and spot anomalies, track key business metrics and gauge the health of the organization, and save time and money.

Recently researchers have been trying to improve the performance of healthcare services by analyzing data collected from operation systems. In those researches, key performance indicators (KPI) are used for performance measurement [Tawney2005][Berler2005]. Based on KPIs, performance metrics are created in the performance management system according to business goals, and the performance management system is responsible for

managing metrics, detecting hospital situations and exception occurring in healthcare services, and performing business analysis to help hospital executive staff to make decisions.

2.4. Service Oriented Architecture

Enterprise performance management systems are often run on a network of centralized servers in a service oriented architecture (SOA) [OASIS2006] that leverages web services technologies [W3C2004], accessed by thousands of geographically dispersed users via browser-based interfaces throughout the Internet. These applications integrate collections of services which are self-contained and independent of the context or state of other services within the distributed system architecture.

SOA encourages several critical development practices, but the most important ones are establishing and adhering to service contracts and splitting interface from implementation, which means that the service's consumers know only the contract and remain ignorant of its implementation details [Vinoski2007]. In general, SOA leads to loosely coupled systems which are implementation independent.

2.5. Quality of Service

Quality of service (QoS) is commonly an underestimated requirement that proves to be crucial to improve the user's experience of networked time-sensitive applications [Ditze2005]. QoS management in networked enterprise systems optimizes system resources and activates computing mechanisms in order to satisfy QoS requirements of many concurrent applications in a network. QoS requirements are expressed with QoS

characteristics in various categories including scalability, reliability, security, timeliness [Wang2004]. This thesis focuses on the scalability, reliability, usage and privacy of the performance management system.

Scalability

Enterprise performance management systems are often deployed in a wide range of scales, in terms of number of users and services, quantities of data stored and manipulated, rates of processing, numbers of nodes, geographical coverage, and sizes of networks and storage devices. Scalability means not just the ability to operate, but to operate efficiently and with adequate quality of service, over the given range of configurations [Jogalekar2000].

It is essential that as the customer base increases, and therefore the system has to deal with significantly increased loads, that the system is prepared to handle the increased traffic so that the users do not encounter unacceptable system performance [Weyuker2002]. In practice, scalability of a performance management system can be ensured by load testing, which determines the system's behavior under various workloads. Metrics to determine the system scalability include response time, throughput, CPU load, memory usage, storage usage, and so on.

Reliability

In [IEEE 610.12-1990], reliability is defined as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." In [IEEE 982.1-1988] Software Reliability Management is defined as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and the use of measurements to maximize

reliability in light of project constraints such as resources, schedule and performance". The terms errors, faults and failures are often used interchangeably, but do have different meanings. In software, an error is usually a programmer action or omission that results in a fault. An observable fault is a software defect that causes a failure, and a failure is the unacceptable departure of a program operation from program requirements [Rosenberg1998]. In practice, reliability can be ensured by monitoring system exceptions/errors during load testing. Metrics to determine the system reliability include HTTP errors, web server errors, application errors, service errors, and so on.

2.6. Usage Monitoring

When or whether a product is fit for wide release is a leading point of contention within the software industry. The dilemma can be resolved by having an effect process in place for product testing and by establishing precise metrics that will create a high degree of certainty that the application will perform as intended, and be free of errors that will cause users to reject it [Linda1998]. For the performance management system, usage monitoring will answer questions such as how many report requests per day, which report is popular (useful) to users and who browses a certain report at what time. By answering these questions, the usefulness of the performance management system to a business will be finally discovered. To monitor the usage of web-based applications, data mining on click stream data recorded in logs is the most popular technique for analysing users' behaviours to the system [Ting2005].

2.7. Privacy

"Privacy is the claim of individuals, groups and institutions to determine for themselves, when, how and to what extent information about them is communicated to others" [Westin1967]. Large organizations usually store a very large volume of customer information, and particularly some of them, such as governments and hospitals, can keep highly sensitive personal information, for example, social insurance number, medical issue, job status, income, religious belief, and so on. Privacy is a right, not a preference. The European Union Directive on Privacy and Electronic Communication [EUDPEC2002] is the leading example of comprehensive privacy legislation. As of January 1, 2004, all Canadians businesses are required to comply with the privacy principles set out by Personal Information Protection and Electronic Documents Act (PIPEDA) [PIPEDA2000]. Particularly in the health care area, in Ontario, Canada, the Personal Health Information Privacy Act (PHIPA) within the context of PIPEDA came into effect on November 1, 2004, and sets out the rules that health information custodians must follow when collecting, using and disclosing personal health information [PHIPA2002]. Similarly, the United State has the Health Insurance Portability and Accountability Act (HIPAA) to address the security and privacy of health data [HIPAA1996].

Useful work has been done to address the relationship between privacy enhancing technology and privacy legislation [Peyton2004] by logging and auditing of information transfers between businesses in B2B networks. Key principles for compliance with privacy legislation were identified as:

1. Organizations must identify how they intend to use personal data and receive consent from individuals.

2. Organizations must establish internal procedures to document and safeguard their use of data.
3. Individuals must be given access to their data and have recourse to challenge its accuracy and use.

2.8. Quality Assurance

A traditional approach to ensure scalability and reliability is to perform load testing and stress testing in a black box manner. Black Box Testing is a testing strategy in which an application is tested using its outward-facing interface and does not need any knowledge of internal design or code. Load testing is used to determine the behaviour of such a system under various workloads, while stress testing determines the breaking point or unacceptable performance point of a system to discover the maximum service level it can achieve.

Modern testing tools, for example IBM Rational Performance Tester [PerformanceTester2007], OpenSTA [OpenSTA2007] and Htmlunit [Htmlunit2007], can launch load testing and stress testing on large online business systems by providing simulated user behaviours to the system. Typical functions provided by those tools include recording user actions to a web system and replaying them, creating a number of virtual users and simulating concurrent access to the system, and reporting on the test results. By using a good testing tool, time and cost for quality assurance for a performance management system are saved, but as we discussed in Section 1.1, they have obvious limitations when performing quality assurance for performance management systems.

Large software systems often keep logging system events into files or databases. Tools like AspectJ and Log4J have worked well for logging events in Java applications, which can incorporate systematic logging with low overhead. Log files are very useful when analyzing whether a run of a program reveals faults in the system. Log file analysis is not a comprehensive approach to testing, but rather a general framework for reliably automating one part of the testing task (test result evaluation). Quality assurance approaches based on analyzing system logs are integrated in some researches which discuss how log files can be used in software testing [Andrews1998] [Andrews2003] and system security enhancement [Kowalski2006].

Audit trails that record the details of user activity can be used to monitor privacy compliance. In [Yip2006], an extensible information security specification format is used as a compliance audit mechanism based on audit trails to enforce user rules and security policies when users try accessing information. In [Chen2005], the design of a HIPAA compliant auditing system for automatically monitoring the data flow and the work flow of a medical imaging system based on security requirements is outlined.

Database and data warehouse technology has been adopted to design and develop a repository of metrics for monitoring the qualities of software projects [Subramanyam 1999] [Palza 2003] [Ruiz 2005], and the dimension model has been used to model quality data captured during the execution of processes to enable Online Analytical Processing (OLAP). OLAP lets quality assurance users easily explore quality data to reflect on the different aspects of target software projects by sorting, filtering, drilling up/down, etc. and then make decisions. Those proposals focus on providing assistance on managing software projects with quality data defined by development teams and collected by spe-

cific tools or by hands. Unfortunately we have not seen any discussion on using a data warehouse as a repository of quality data for quality assurance for such software projects, especially for a performance management system which is already available to be used to analyze and report on quality assurance data in a data warehouse.

Chapter two introduces the background of this thesis, containing business performance management, data warehouse, enterprise performance management system, service oriented architecture, quality of services, usage monitoring, privacy and quality assurance for enterprise performance management systems. In next chapter we present the integrated quality assurance framework for performance management systems.

Chapter 3. An Integrated Quality Assurance Framework

3.1. Overview

Development of an enterprise performance management system is a complex process, which involves business methodology, goals, operational system, reporting system, data collection and IT infrastructure. Several characteristics of this process make quality assurance of scalability, reliability, usage and privacy challenging:

- To save time and cost, an organization usually sets up a performance management system based on its existing information systems. Information systems in large organizations such as governments can be very complex due to numerous factors including scale, decentralization, heterogeneity, mobility, dynamism, and so on.
- To develop a performance management system, a variety of off-the-shelf components are assembled including operating systems, databases, application servers, reporting engines, analytical engines, ETL systems, and content management systems. The combinations of components are varied and there is no guarantee on the quality of possible integrations, even on a single component.
- To develop a performance management system, custom development of data warehouses, data marts, reports, analyses and portals are needed, which themselves have an impact on quality. Further complicating the situation is the fact that

this development is ongoing and typically takes place after the initial system has been deployed into production. In particular, the volume of data within the data warehouses and data marts is constantly growing and the volume of data processed and returned when running reports varies over time while in production.

Black box testing for the performance management system by professional web testing tools is not enough to ensure the quality of the system because it does not perform quality assurance across the architecture of the performance management system. In a black box approach, web testing tools only focus on the quality of the whole system, and cannot assess quality at the level of integration of components/services inside the system, for example, web server, application server, data warehouse, etc. Hence, it cannot provide a deep analysis of quality of service. By itself, black box testing does not provide alerts of quality issues in real time. In production, administrators need to monitor real time qualities of the system so that they can adjust the system to avoid failures, but black box testing can only be performed separately from production.

In this chapter, we will demonstrate the architecture of performance management system first, and then present our integrated quality assurance framework for enterprise performance management systems and a data architecture for quality assurance data analysis.

3.2. Integrated Quality Assurance for Performance Management Systems

3.2.1 Performance Management System Architecture

Figure 1 shows a common architecture of the performance management system, which contains user, enterprise performance management portal, performance management services in a service oriented architecture (SOA), data warehouse and operational systems.

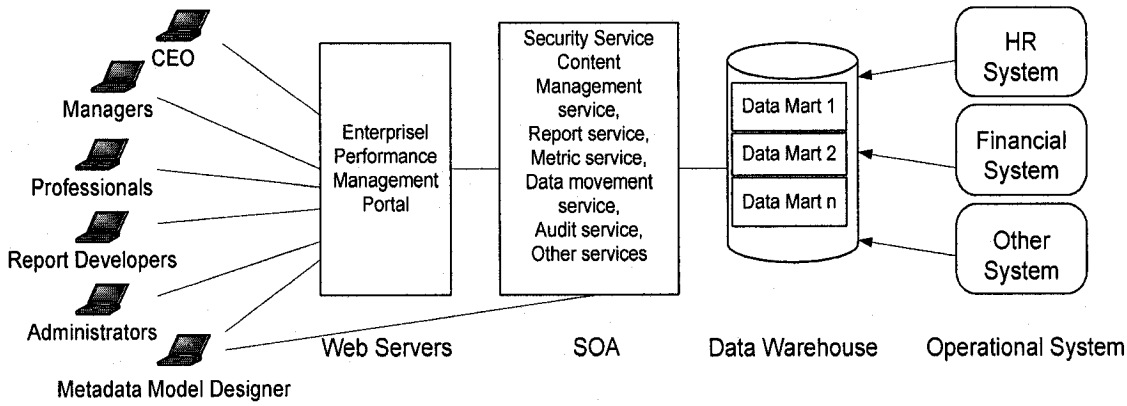


Figure 1 Architecture of the performance management system

Users

In daily business, users of the performance management system can be professionals, managers and executive officers, etc., and different users are interested in data that cuts across traditional divisions in the businesses at different levels of detail. For example, a sales clerk cares about how many products he or she sold in the past three months; a financial manager needs to monitor the balance of investments and profits of the business; a chief executive officer want to see the business performance in several areas such as financial, market share, products, etc., at a glance and on the go.

There are also other more technical users that maintain the performance management system. Users can be metadata model designers who prepare metadata of operational data collected from operational systems into the data warehouse, model metadata as a star schema and prepare the business view based on business requirements. Metadata is finally published for reporting. Users can be developers of reports providing day-to-day event monitoring, data aggregation in different level, etc., which satisfy the needs of business users. Users can be system administrators maintaining the system and monitoring quality of service every day.

Enterprise Performance Management Portal

An enterprise performance management portal is a browser-based user interface for accessing the reporting, monitoring, and analysis functionality of the performance management system. The "portal" view can be customized for each user to define their particular business view of enterprise data and highlight the individual reports and analyses that are most relevant to them. Users access the "portal" on web servers, and requests are forwarded further to the performance management services which provide corresponding services by interacting with each other and the data warehouse.

Performance Management Service

Performance management service is the core of a performance management system, providing a set of services developed in service oriented architecture (SOA). Typical services include:

a. content management service which allows users to create metadata models giving a common foundation for information sharing on an enterprise scale;

b. report service which processes ad hoc queries against different data source and creates reports;
c. authentication and authorization service for managing system security;
d. audit service for tracing important user actions.

Operational System

An operational system is a system which supports the operation of customer business. Operational systems are process oriented and often not integrated [Lawyer2004]. Big organizations, for example hospitals, may have several operational systems to manage different business processes, such as financial system, human resource system, production system, clinic system, and so on. For each operational system, there may be an operational database which stores day-to-day operation data, and those databases are often normalized to maximize accuracy and minimize redundancy. To keep high performance, usually these databases only keep recent data and move history data into backup devices. It is difficult to report from an operational database because some business questions may involves a large number of tables and queries may create heavy load on database systems.

Data Warehouse

The architecture of data collection from operational system into an enterprise data warehouse is shown in the following figure.

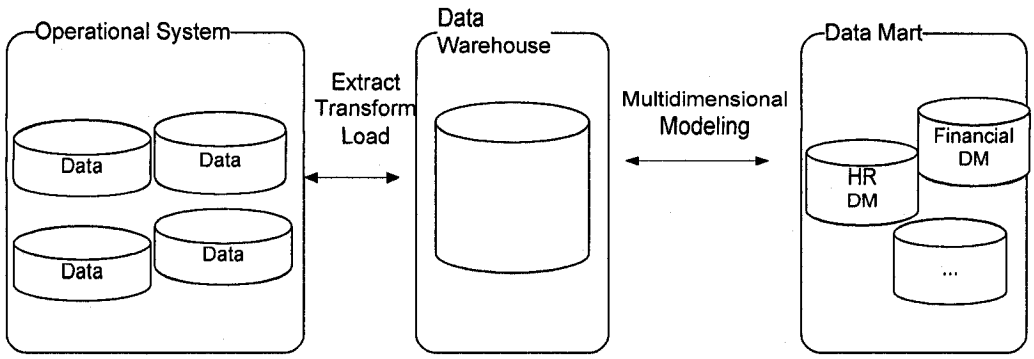


Figure 2 Operational data collection architecture

To build a data warehouse, data in operational systems is extracted from existing legacy or other operational system, transformed into a common universal format, and loaded (ETL) into databases, usually in daily manner. The star schema is likely the most popular dimensional modeling and it organizes data in a data warehouse as fact tables with their related dimensions. Business decisions are made according to specific business activities and areas; hence it is not efficient to dig into all raw data collected in the data warehouse. Data Mart stands for highly-focused version of a data warehouse and it follows the same lead as data warehouse in a small scale. It provides access to critical data faster than the enterprise data warehouse, allowing business managers to initiate business intelligence strategies and reap the rewards sooner [Houari2004]. For example, in an enterprise, departments such as finance, marketing and sales, can have their own data marts. In the environment of a performance management system, we treat a data mart as a combination of data, packages (metadata), reports, portals and any other components related to one department or business logic.

3.2.2 Basic User Scenarios in the Performance Management System

The three main user interactions with the performance management system are: browsing and running reports, authoring reports, designing a data mart model. Testers often test the system by simulating these user interactions with the system. During a test, services in SOA keep logging interaction events that can be collected for assessing quality for the system, and this is explained in more detail in Figure 4 in Section 3.2.3.

Browsing and running reports

Browsing and running reports is the user scenario which is most often tested. In daily business, end users such as professionals, managers and CEOs, log on to the system, browse and run reports through the enterprise performance management portal and then log off the system. The following interaction diagram shows detail interactions between services in this scenario.

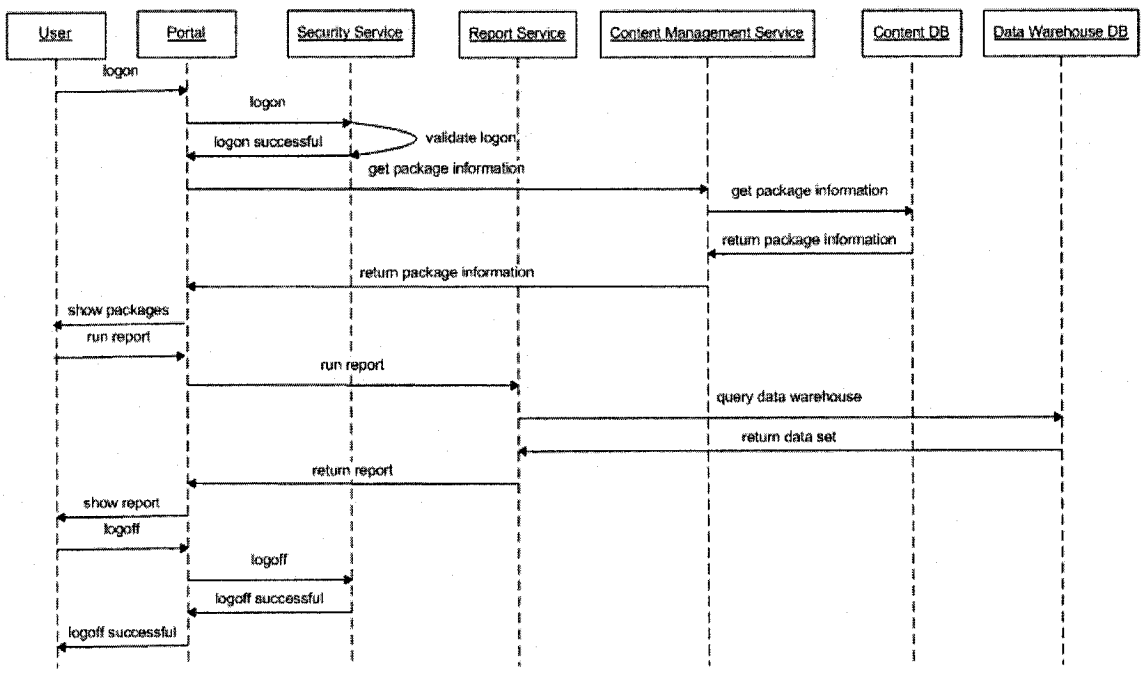


Figure 3 Interaction diagram for running reports

Authoring reports

Using report designing tools, to develop a report, developers first retrieve package metadata from the content store, and then select proper objects to create reports. Typical functions provided by reporting tools include organizing data in lists and tables, presenting data graphically using charts and maps, drilling up/down in dimensions, drilling through reports, filtering data and adding additional calculations on data, and so on. By using these functions, developers can enhance or limit data that users can see based on users' needs, roles and other aspects.

Designing a data mart model

To do reporting on the data collected from operational systems into the data warehouse, the performance management system needs a data mart model to group data in fact tables with related dimensions. By using a metadata modelling tool, designers can model the data in the data warehouse in a star schema, prepare business views on a metadata model and publish metadata objects in a business view as packages for authoring reports. Access control can be applied to packages to limit metadata objects that users can access in authoring reports.

3.2.3 Integrated Quality Assurance Framework Architecture

In this section, we propose a quality assurance framework that leverages existing information and logs from system components and services to build a data mart that can be used to analyze test runs and production runs. The integrated quality assurance framework of the performance management system is fully integrated into the performance management system itself across the whole architecture of the performance management

system. It allows deep analysis on system logs by providing detail and aggregation information of each log. It also examines the concept of the quality assurance portal for communicating the quality state of the performance management system both during the development of the performance management system and when it is in production.

Figure 4 shows the architecture of the quality assurance framework for the performance management system. Through a quality assurance log loader, a number of component/service logs are loaded into a quality assurance data mart in the data warehouse.

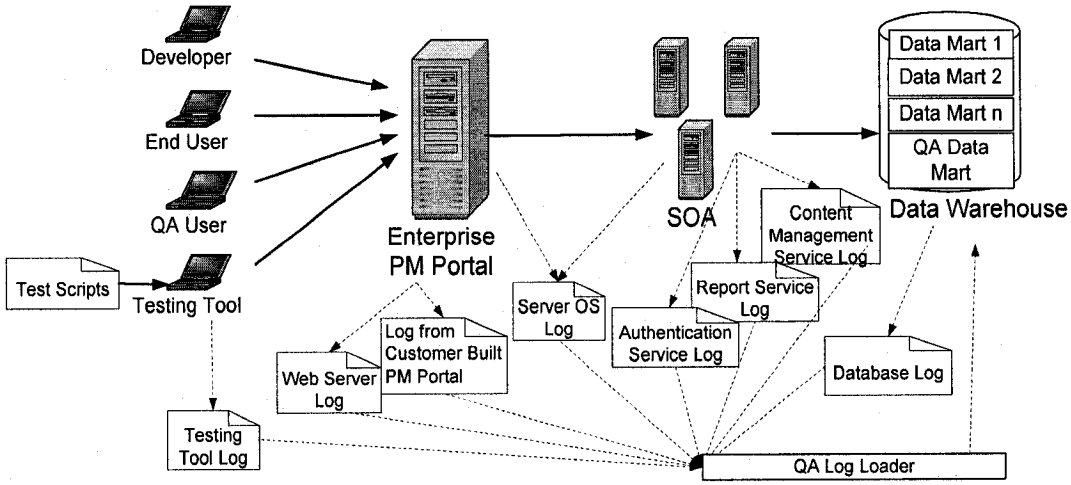


Figure 4 The architecture of the integrated quality assurance framework

Logs

Most components and services in the performance management system have their own logs. Common information to an event recorded in logs contains when an event happens, on what component, why it happens and what is the status of the component. Typical logs that can be captured in the performance management system include:

- Testing tool log

Testing tools keep logs for actions to the target system and their feedbacks after sending, for example, system response time to each request, number of concurrent users versus elapsed time, HTTP errors versus elapsed time, and so on.

- Web server log

  A web server often logs the connection information retrieved from request headers, for example, IP address, target URL, etc. It also logs its own running status, for example, current number of connections and maximum number of connections reached.

- Web application server log

  A web application server often logs its running status, for example, exceptions when loading applications responding to user requests.

- Customer built application log

  Customers may develop their own applications to customize or expand the functions of the performance management system. Those applications may use an independent log to keep critical information in their service process.

- Performance management service log

  Each performance management service in a service oriented architecture (SOA) logs their working state, for example, values of input parameters, results after service, etc. Those logs can be used to analyze the system status on component/service level.

- Server operating system performance log

  The operating system of servers can be configured to log the CPU, memory and storage usage which can be used to analyze the scalability of the system.

- Database log

A database can keep logs to monitor transactions. For example, creating, updating and deleting database or tables, and executing queries.

QA Log Loader

A QA log loader is a program responsible for extracting, transforming and loading data from logs into the data warehouse in the performance management system. Since different kind of components may use different policies for logging, this results in different data formats in logs, and the QA log loader should allow users to configure which logs will be collected.

3.3. A Model for Quality Assurance Data Analysis

To analyze the quality data collected in the integrated quality assurance framework, we present a model that applies data warehouse technology to the data collected to create quality assurance data marts. A quality assurance portal can be set up to help users create and manage quality assurance reports based on these data marts. The model follows the five stages of BPM in [Jeng2006] for system performance management. In the "Sense" stage, it interacts with quality assurance goals, collects log data into the data warehouse, models the data in star schema, and prepares quality data (metrics) that is to be further monitored and analyzed. In "Detect" stage, it detects system status or exceptions occurring in the system and alerts quality issues to users. In the "Analyze" stage, a quality assurance portal is used to communicate the system status with quality assurance reports to help users investigate causes of system exceptions across the system architecture. In the "Decide" stage, users make decision about what to respond to quality issues and actions such as database tuning are performed in the "Execute" stage.

3.3.1 Quality Assurance Data Modelling

The data collection process for quality assurance is basically the same as the operational data collection process shown in Figure 2 except the content to be collected is not operational data but logs recorded by components in the performance management system. Technologies used on operational data, for example collecting, modeling and analyzing, are also suitable for data in system logs. This means an enterprise performance management system can be used to communicate performance of the system itself. The following figure shows the details of this architecture.

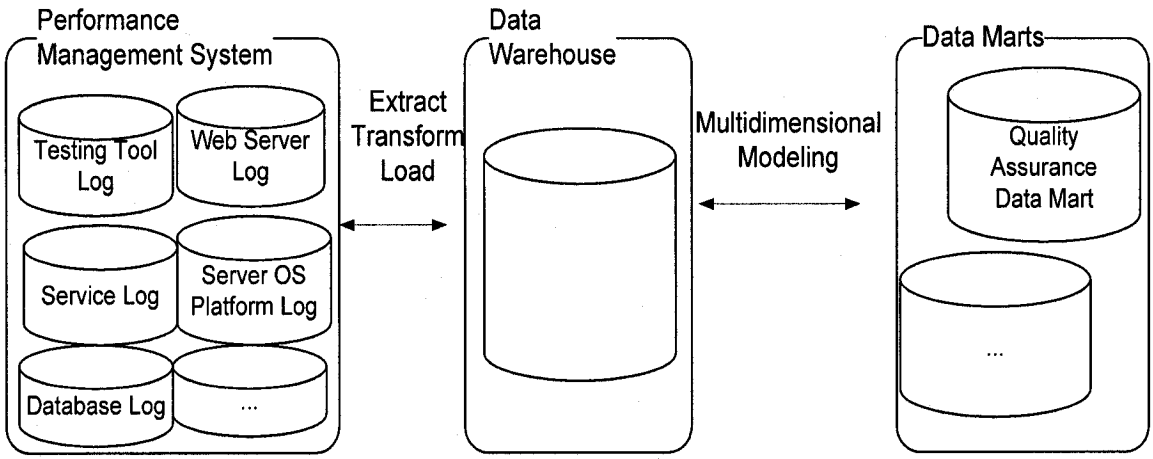


Figure 5 Quality assurance data collection architecture

Components and services in the performance management system often log critical time-based events during system runs. Typical events happen when there are inputs, exceptions/errors and output coming. Useful information is recorded to monitor the status of the system, which can be used to ensure the qualities of the performance management system. Data in logs is first extracted, transformed and loaded into the data warehouse in the performance management system by log loaders. Then, data is organized in the qual-

ity assurance data mart with a multi-dimensional model to enhance data presentation and navigation.

Basic dimensions of the quality assurance data mart include the time dimension, the user dimension and the report dimension. Those dimensions can be related to more than one fact table. The time dimension is organized in a calendar hierarchy, and its granularity is usually on milliseconds to be compatible with different logs. The user dimension can be extracted from operational databases and organized in an organizational hierarchy. Common attributes of this dimension include user id, user name, user role, user level, user group, department name, territory, and so on. The report dimension comprises detailed information of reports contained in the performance management system, and it is often organized in package-folder-report hierarchy. Attributes of this dimension include report name, report type, report size, folder name, package name, etc. There can be other dimensions extracted from operational databases and log data, which depend on the goal of the quality assurance and the availability of related information.

A fact table is generated by a single log or a combination of several related logs which have common dimensional information, and one or more fact tables are used to measure a specific quality of the system.

This dimensional model helps users to organize the quality data logically without searching in many tables, and it allows deep analysis on logs by enabling OLAP which quickly provides answers to analytical queries that are dimensional in nature and allows flexible navigation and pivoting of the data along different dimensions at different levels of detail.

3.3.2 Quality Assurance Portal

Online Performance Management Systems provide enterprise performance management portals for business users to access and manage reports. Often they allow users to create personal portal for particular business goals. With this capability, we examine the concept of a quality assurance portal for communicating the state of quality of a performance management system. A quality assurance portal contains suitable reports and data to show the trends and exceptions of system qualities. Towards monitoring different quality assurance goals, users can have their quality assurance portal customized with particular graphs and reports to make their analysis on the quality data more efficient. By providing OLAP functions in reports, users can sort, filter, aggregate quality data and drill up/down in dimensions, which can dramatically reduce the efforts to notice quality exceptions and to investigate and determine the cause of those exceptions.

Particularly, because the quality assurance portal is integrated into the performance management system, the product can be used both during the development of the system and when it is in production. During the development of the system, a quality assurance portal can be set up by the development team to monitor the quality of the system in test environment, and this would help the team figure out the quality problems in early stages. The development team can even have a quality assurance portal as an integrated function in the final release and transfer it to end customers. In production stage, customer can either accept the quality assurance portal integrated in the product or set up their own quality assurance portal according to the architecture of their information systems.

3.4. Quality Assurance Methodology

The integrated quality assurance framework should be capable of analyzing necessary logs which will happen both during the development of the system and when it is in production and it should allow users to turn on/off a component in the framework for analyzing a specific log according to its quality assurance goals. For example, unlike quality assurance on scalability and reliability of the system that can be simulated during the development of the system, quality assurance on usage and privacy of the system are mainly for the production. But all components in the framework have to be implemented and tested during the development of the system to make sure they will work in the target environment. Moreover, some components/services allow users to configure the level of details in a log; hence the framework should be compatible with this functionality and allow users to configure it.

3.4.1 Quality Assurance for the Performance Management System under Development

The integrated quality assurance framework can be set up at the beginning of development of the performance management system, and the scalability and reliability of the system are what testers care about in this stage. A typical quality assurance process by this framework has five steps.

Step 1.Configure the system for logging

Before testing the performance management system, the system must be configured to make logging function available in services/components. To avoid affecting the perform-

ance of the performance management system, users need to make decisions on which log will be collected in what level of detail information. To analyze the impact of the logging, users can enable a specific log in a specific detail level at one time and use the integrated framework to report and analyze the system performance.

Step 2.Simulate user actions

Simulation of user actions to the performance management system can be done by developers with some programs directly accessing services or by testers who make some inputs to the UI. But the most popular way is using a professional testing tool to record different user behaviors in some business scenarios into scripts and replay those scripts to simulate a real use of the performance management system while considering the total number of possible users, the maximum number of concurrent users and the possible categories of user actions.

Step 3.Collect logs

After the simulation of user behaviors to the performance management system, logs happened in the test process are extracted, transformed and loaded into the data warehouse by the QA log loader.

Step 4.Model quality assurance data in data warehouse

Having log data in the data warehouse, a quality assurance data mart would be created with a dimensional model of quality data, combining the fact tables (quality facts) and necessary dimensions (time, user, etc.), to demonstrate the quality of the performance management system in different levels and aspects. The metadata model of the quality

assurance data mart will be published usually as packages for users to create reports on quality data by choosing relevant query items (i.e. columns).

Step 5.Analyze the test result

A quality assurance portal, integrated in the enterprise performance management portal, is created by leveraging the technologies of the performance management system. It allows users to create and organize valuable reports and related information and explore detail and aggregation information by drilling up/down dimensions in reports to visualize the scalability and reliability of the performance management system.

3.4.2 Quality Assurance for the Performance Management System in Production

The integrated quality assurance framework, as a component of the performance management system, will be deployed in customer environments. The infrastructure of the performance management system in the production environment has no difference with the one in the test environment. Real users (professionals, managers, etc.) instead of testers or testing tools will access the system, administrators instead of testers care about qualities of the system.

To perform quality assurance for the system in production, administrators turn on the logging functions for certain components in the system and configure the level of detail information in logs based on their quality assurance goals. Administrators then configure the QA log loader to collect these logs. A decision is needed about how often components log events and how often the QA log load logs since they are relevant to the performance of the system. The quality assurance portal and reports can be customized by administra-

tors according to their own needs. Besides the scalability and reliability of the system should continuously be ensured in production, usage and privacy of the system should also be ensured.

3.4.3 Addressing Quality Issues

Scalability, Reliability and Usage

In the quality assurance data mart, facts reflecting the scalability of the system include response time to users' request in testing tool and web server logs; CPU and memory usage of servers in server operating system logs; report execution time in report service log and database query execution time in database log; etc. Facts reflecting the reliability of the system include HTTP errors in testing tool and web server logs; exceptions in web server, service and database logs; etc. Facts to measure the usage of the system include number of users online per day, number of requests per day, times of a report has been executed, and so on, which can be found in performance management audit service log. Quality assurance reports are created based on those facts and related dimensions so that users can ensure the scalability, reliability and usage of the system directly by viewing and analyzing reports.

Privacy

As we discussed in the background section, audit trails that record the details of user activity can be used to monitor privacy compliance, and it has to combine information policy. The user scenarios described in section 3.2.2 can raise several privacy concerns about the way of modelling metadata, developing reports and browsing reports. Before we dis-

cuss these privacy concerns, we first summarise four levels of privacy when dealing with data in a data warehouse:

- Personal Detail. Personal private information should not be published to the public, for example, name, social insurance number, phone number, address, etc.
- Inferable Personal Detail. In some conditions, personal detail can be inferred from related data, for example, one can find out personal detail of a person from his/her family member's data.
- Aggregate Inferable Personal Detail. One of the most important functions of report is performing aggregation on data, for example, grouping data by day or month. Aggregation on data may allow one to narrow and locate information related to a person. For example, if there is only one female in an organization, then data aggregation by gender may discover the detail information of this person, such as salary, personal working performance, etc.
- Aggregation Summary

  Aggregation summaries are often secrets of businesses and not allowed to be discovered by public or competitors, for example, average cost of a production and total profit of an organization.

According to the privacy level discussed, we summarize processes with privacy concern following the data architecture of the performance management system shown in the following figure.

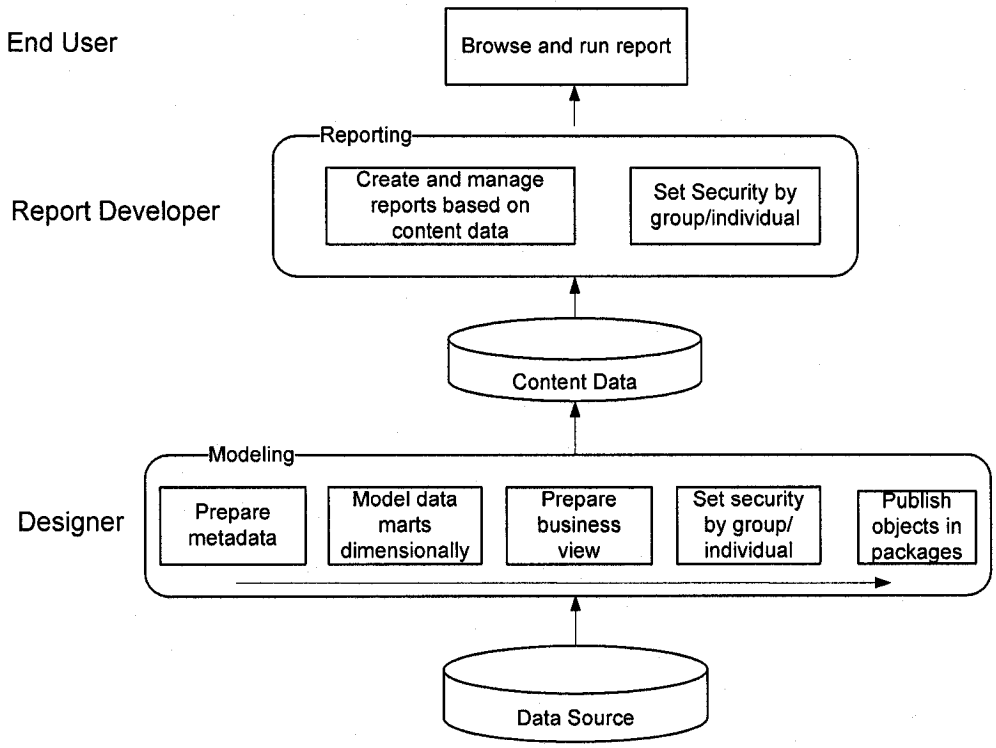


Figure 6 Data architecture of the performance management system

- Metadata modeling. Reporting needs data collected from operational systems, but not all operational data are necessary for performance management. To create a data mart, designers have to make decisions on what data in the data warehouse and in what level of detail data are necessary for business performance management. Designers also need to make sure whether or not the data are allowed to be published from a privacy view. To publish packages, designers have to analyze business processes to limit query objects for reporting, for example, a package for performance management of financial department should only contain query objects which are related to this department.
- Developing reports. To create a report under a package, developers need to know what query items/columns can be used in a report and what kind of aggregation

can be applied to which query items/columns according to its usage and data privacy.

- Set security and permission to user groups. Report owners or administrators often set up security for data sources/packages/reports by user groups. For example, people in one department can be assigned to a group that can only see reports related to this department. However it is still difficult to make a perfect job here due to unclear group definition, complex group hierarchy and continuous change in an organization.
- Set security and permission to individuals. Users can be assigned into a group to inherit its right to data sources/packages/reports; also users can be directly assigned with privileges to certain reports. In this process, who can view which report and when must be strictly defined.

To address privacy issues, organizations must identify how they intend to use personal data and receive consent from the individual. Privacy policies in documents for accessing data in the data warehouse have to be defined. To address privacy issues, an audit trail should provide at least detailed information about who access what report at when. Other useful information includes:

- What data sources are used in creating data marts, by whom?
- What data columns are used for constructing fact tables and dimension tables, by whom?
- What data columns are published within packages, by whom?
- What data columns are used in reports, by whom?
- Etc.

With that information available, reports can be created for users to analyze user activity and discover privacy issues against their privacy policy. For example, in a hospital, a report showing patients' diagnosis can be only viewed by physicians. We can design a report on report execution history to see which user accessed which report. This report can have report name, user name, user role, and so on; hence privacy violations can be easily discovered by grouping report name and user role.

This chapter presents the integrated quality assurance framework for performance management systems, a data warehouse architecture for quality data analysis, and a methodology to use the integrated framework in quality assurance. In next chapter we introduce a new mechanism for generating test data for data warehouses.

Chapter 4. A Model for Test Data Generation

Quality assurance for the performance management system needs "real" data in large volumes to validate accuracy of reports, scalability, and reliability, etc., which means "test" data used is similar to real data in terms of both volumes and actual values so that reports run in our test environment would have the similar processing characteristics as in production. But real data in production is very sensitive, subject to privacy and ethical considerations; it can be difficult to get reasonable test data in large volumes.

In this section, we describe how test data can be generated and have a test data generation tool developed to generate reasonable test data in real world volumes while protecting sensitive information. We first present the common structure of data in the data warehouse in the performance management system, and present challenges in generating test data for quality assurance for the system. We then present a specification-based strategy for test data generation and describe the specification language used.

4.1. Structure of the Data Warehouse

The data warehouse in the performance management system can consist of one or more databases that contain data collected from business operational systems. In order to facilitate and control the analysis operations, data in the data warehouse is often organized as data marts, as we discussed in background information. The data mart contains a fact ta-

ble and a set of dimensions. A dimension table is a category of attributes organized in hierarchy, for example, a time dimension can have day, month, year, and so on. A fact table is a table that contains the primary keys of related dimension tables as well as the measures of interest with appropriate granularity. For example, sales amount would be such a measure. A dimensional model example in a healthcare data warehouse is shown in the following figure.

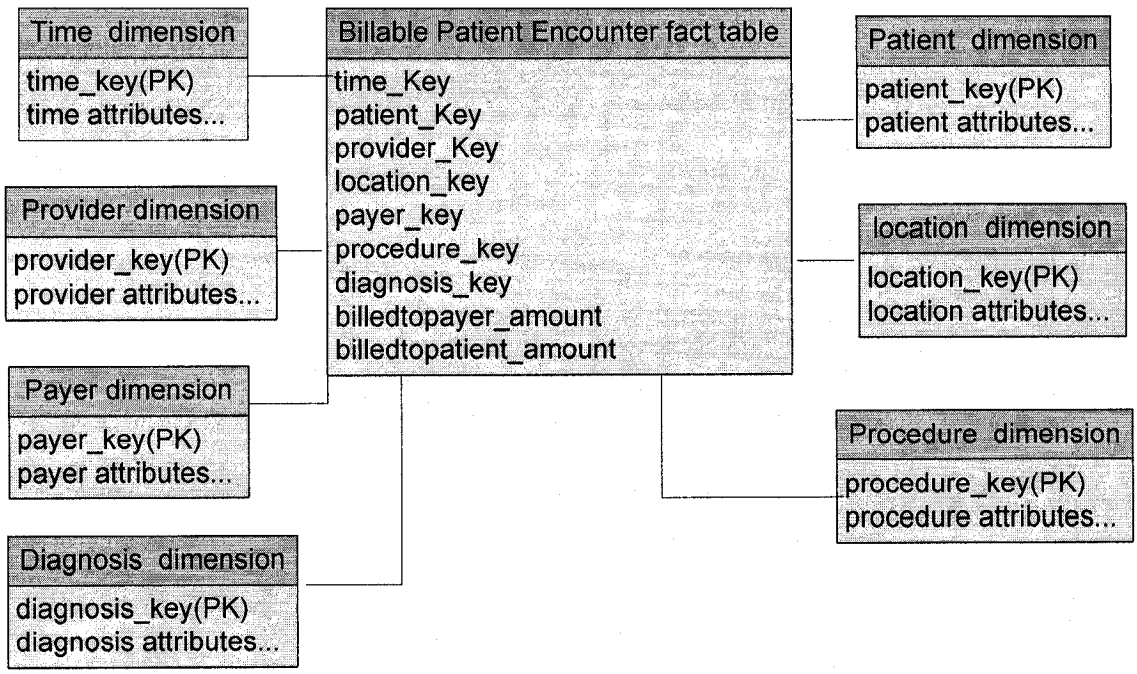


Figure 7 Dimension model example [Kimball 2002]

4.2. Challenges in Generating Test Data for Performance Management Systems

To generate high quality test data for quality assurance for the performance management system, the main challenges we have faced include:

- Obfuscating sensitive information. Sometime we can obtain real sample data, but we cannot use it directly in testing. For example, in our research, a small sample data set from a major teaching hospital is available for use to analyzing data structure of its data warehouse. But physician names and hospital names are sensitive to public; therefore, obfuscation on this kind of information is needed before using it.
- Generating time-based fact table with meaningful fact data. Real data in the data warehouse of the performance management system is collected from business operational system in day-to-day operations; hence fact tables are basically time-based. Sometimes there can be certain relations among facts, for example, the "start time" and "end time" of a prescription. Those relations have to be implemented to make test data accurate for reporting.
- Generating data with similar characteristics to real data. Test data must have similar "report" characteristics as real data, for example, same number of rows, similar distribution, similar aggregation result on columns, and so on.

Commercial or open source software applications/tools have been developed for populating test data into databases. But according to our evaluation of some of them (see detail in Section 5), we have not seen any of them that can overcome all these challenges in generating test data for the performance management system.

4.3. A Specification-Based Strategy for Test Data Generation

We study a strategy to generate test data for the performance management system while considering the above requirements and issues. The following figure shows the architecture of our strategy.

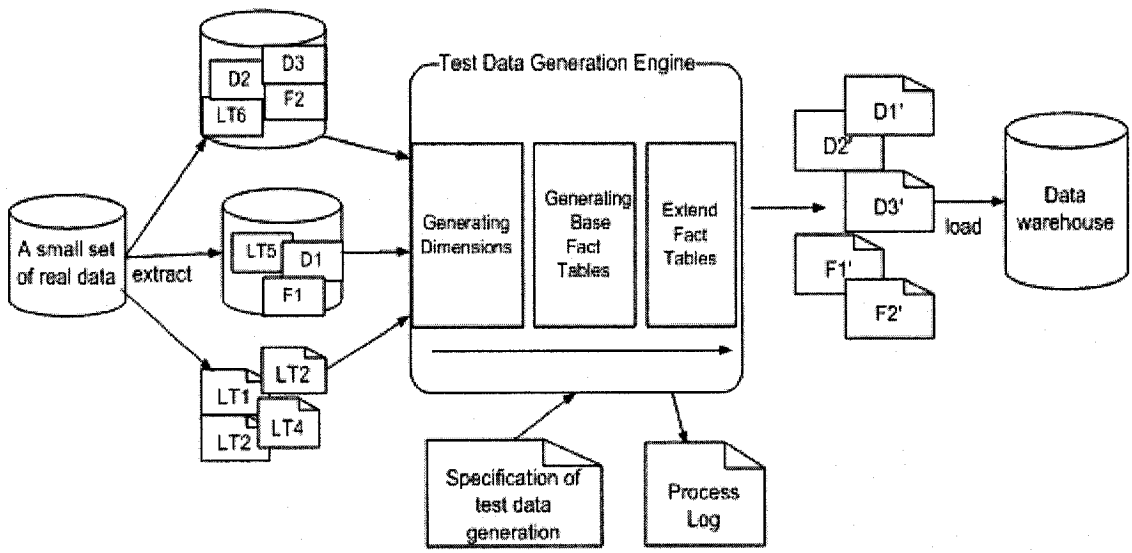


Figure 8 Architecture of test data generation

(Note: LT_n stands for Lookup Tables extracted from a small set of real data. D_n represents a dimension table in a database, which is empty and available to access. F_n represents a fact table in a database, which is empty and available to access. D'_n represents a dimension table in a CSV file filled up with test data. F'_n represents a fact table in a CSV file filled up with test data.)

We assume that the structure of the target data warehouse is available for a test team to set up a similar but empty data warehouse in the test environment, which means the test team should be able to access the dimension tables (D_n) and fact tables (F_n) in databases

to get their definitions. We also assume all fact tables are time-based, i.e. there is at least one column in a fact table that has time-type value in a business period.

To make the test data look like "real" data, we have to refer to real data. We first select a small set of real data to get the meaning of the data and understand the structure of the data warehouse. It will also be used as the kernel data for generating more test data which has the same structure and similar characteristics. To make the test data real, we should use as much real data set as possible. For example, in a real healthcare data set, information such as time, service name and department name are not sensitive so that it can be directly used as test data. But physician name is sensitive, which must be obfuscated before using it. Data in the small data set can be treated as a set of lookup tables (LT_n). A lookup table can be a list of non-sensitive data in a column of a flat file, or a column in a database table. Data will be extracted from these lookup tables when generating test data into target dimension and fact tables.

We then have a test data generation engine which uses a specification as input. The specification describes how to generate test data for target dimension and fact tables using available data (lookup tables). This engine will follow the scenarios defined in the specification to generate test data for dimension tables first, and then use both dimension tables and lookup tables to generate test data in the earliest business period for fact tables, and then expand fact tables in a full-range of time period. The output of the test data generation is a set of dimension tables and fact tables in flat files. Using importing or loading functions of commercial database software, data in those flat files can easily be loaded into target dimension tables and fact tables in the data warehouse.

4.4. Specification Language for Test Data Generation

In this part, we study a specification-based approach for test data generation for the performance management system. This approach uses a formal specification to describe functional scenarios illustrated in our strategy with all necessary program paths in a program. The specification is oriented towards a data warehouse structure, and it explicitly illustrates a star schema with dimension tables and fact tables, so users can easily understand its structure and make their own specifications.

We adopt XML to create specifications because XML has self-documenting format, which describes structure and specific values, and this format is both human-readable and machine-readable. In addition, by using parsing tools, verification on job specification is extremely simple, efficient, and consistent. The following figure shows the structure of the specification.

```
<source>
  <databases>
      ...
  </databases>
  <files>
      ...
  </files>
</source>
<dimensions>
  ...
</dimensions>
<facttables>
  ...
</facttables>
<expands>
  ...
</expands>
```

Figure 9 Sections in specification language

The defined specification includes four sections: source, dimensions, fact tables and expands. In the source section, databases and files are defined as resources. Lookup tables will be generated by extracting data from database tables and files. The dimensions and fact tables defined in the following sections will refer to target tables in databases to get their definitions. The "dimensions" section defines how to generate test data for dimensions using lookup tables. The "facttables" section defines how to generate test data for time-based fact tables using both dimensions and lookup tables. We call the fact table generated in this section *basic* fact table, which should contain test data in the earliest period of business operations. In the "expands" section, we define how to expand fact tables to have test data in a full-range time period by cloning data in their basic fact tables.

4.4.1 Source Section

Our specification structure describes all resources available such as distributed databases and local flat files.

```
<source>
    <databases>
        <database id="d_amrdb" name="AMRDB_LARGE" type="SQL2005" host="id1g"
        port="1433" user="sa"/>
    </databases>
    <files>
        <file id="f_physicians" path="D:\temp\data\tbl Physicians.csv" header="true"
        delim="," qualifier="&amp;quot;"/>
        <file id="f_services" path="D:\temp\data\tbl Services.csv"/>
        <file id="f_hospitals" path="D:\temp\data\tbl Hospitals.csv" header="true"/>
        <file id="f_session" path="D:\temp\data\tbl Session Types.csv"/>
        <file id="f_contancts" path="D:\temp\data\tbl referring Contacts.csv"/>
        <file id="f_code" path="D:\temp\data\tbl Code.csv" header="true"/>
    </files>
</source>
```

Figure 10 Example code of source section

Basic elements in the "source" section include:

- <database> defines a database with a unique ID, the database name, the type of database, the location of the database, and a user name needed to sign on this database. A java program can use those attributes to connect to a database via Java Database Connectivity (JDBC).
- <file> defines a local flat file with the file name and the path to its location in the local file system, an indicator for column headers, the type of delimiter and the type of text qualifier.

Given a small set of real data, non-sensitive data in database tables or flat files will be directly used as lookup tables to provide test data. For example, the numeric values and the name of common services in healthcare data. Sensitive data must be replaced with fake but meaningful data to protect privacy, for example, replacing a real physician name with a random name. Lookup tables will be referred in "dimensions" and "facttables" sections by a designed notation. For example, "f_physicians.Physician Name" refers to the column "Physician Name" in file "f_physicians". The databases defined in this section also provide the definitions of target dimension and fact tables, which will be referred in the following sections

4.4.1 Dimension Section

This section defines how to generate test data for dimension tables using databases and files defined in the source section.

```
<dimensions>
    <dimension id="dim_physician" ref="d_amrdb.Tbl ConsultingPhysicians"
    output="c:\temp\Tbl Consult-ingPhysicians.csv">
        <column name="ID" by="sequence"/>
        <!--get data from a look up table-->
        <column name="Physician Name" by="lookup" lookup="f_physicians.Physician Name"/>
        <column name="Service" by="lookup" lookup="f_services.column1"/>
        <column name="Site" by="lookup" lookup="f_hospitals.Hospital"/>
    </dimension>
</dimensions>
```

Figure 11 Example code of dimension section

Basic elements in this section include:

- <dimension> has a unique ID and it refers to the target dimension table in a database to get its definition. The attribute "output" defines the output file that will store test data.
- <column> defines a dimension column with a unique name which must be the same as a column name existing in the target dimension table. The attribute "by" defines the way test data will be generated for a column, and it has three values here. The value "lookup" means it will get data from a lookup table which is defined in another attribute "lookup"; the value "sequence" means the data will be a sequence of integers; the value "random" means the data will be a random value according to the real type of the column. The attributes "min" and "max" define the minimum and maximum values of numeric columns

Dimension tables are generated based on the lookup tables extracted from files or database tables defined in "source" section. The execution result of this section should be a number of dimension tables stored in CSV files.

4.4.2 Fact Table Section

This section defines how to generate test data for time-based fact tables using generated dimension tables in flat files and lookup tables extracted from source databases and files.

```
<facttables>
    <facttable id="fac_data" ref="d_armdb.tbl Data" start="01/01/2003 00:00:01"
    end="12/31/2003 23:59:59" rows="1000000" append="false"
    output="c:\temp\tbl Data.csv">
        <column name="SessionID" by="sequence"/>
        <!--get data from a dimension table -->
        <column name="Service" by="dim" dim="dim_physician.Service"/>
        <column name="Session Type" by="lookup" lookup="f_session.column1"/>
        <column name="Number of Attnewendees Consulting" by="random" min="0" max="110"/>
        <column name="Date" by="time"/>
        <column name="Duration" by="random" min="0" max="360"/>
        <column name="Consulting Contact" by="dim" dim="dim_physician.Physician Name"/>
        <column name="Consulting Site" by="dim" dim="dim_physician.Site"/>
        <column name="Referring Site" by="dim" dim="dim_physician.Site"/>
        <column name="Cost" by="random" min="0" max="135" distribution="normal"/>
        <column name="Code" by="lookup" lookup="f_code.Code"/>
        <column name="Referring Contact" by="lookup" lookup="f_contancts.column1"/>
        <column name="Number of Attnewendees Referring" by="random" min="0" max="110"/>
    </facttable>
</facttables>
```

Figure 12 Example code of fact table section

Basic elements in this section include:

- <facttable> has a unique ID and refers to the target fact table in a database to get its definition. The attributes "start" and "end" define the time period in which test data should be generated. The attribute "rows" defines how many rows of test data will be generated. The attribute "output" defines the output file that will store test data, and the attribute "append" means whether data in the output file should be deleted before outputting.
- <column> defines a fact table column with a unique name which must be equal to a column name existing in the target fact table. The attribute "by" defines the way

test data will be generated for a column. For example, the value "lookup" means it will get data from a lookup table which defined in another attribute "lookup"; the value "sequence" means the data will be a sequence of integers; the value "random" means the data will be a random value according to the real type of the column; the value "time" means the data will be a time between the "start" and "end" attributes defined for the fact table; the value "timedepend" means the data (time) will depend on the time in another column; the value "expression" means the data will be a result of a Java expression [Singular2007]. Two optional attributes "min" and "max" define the minimum and maximum values of numeric columns. The attribute "distribution" defines in which distribution a fact will be. Possible distributions include random distribution, standard normal distribution which creates a bell curve on data graph in the whole data range, etc.

The execution result of this section should be a number of fact tables in CSV files with test data in the earliest time period, for example, the year 2003. The "timedepend" attribute in the <column> element will generate relationships between time-type columns; the "expression" attribute will generate relationships between numerical-type columns.

4.4.3 Expand Section

This section defines the process to expand basic fact tables generated in the previous section to other time periods while considering the trend of facts in different time periods.

```
<expands>
    <!--expand the fac_data to have 1000000 new records in year 2004
    and the aggregation of cost will be 10 percent more than the one
    in year 2003-->
    <extable id="fac_data" start="01/01/2003 00:00:01" end="12/31/2003 23:59:59"
        newstart="01/01/2004 00:00:01" newend="12/31/2004 23:59:59">
        <excolumn name="Cost" trend="110"/>
    </extable>
    <!--expand the fac_data to have 1000000 new records in
    year 2005 and the aggregation of cost will be 20 percent more
    than the one in year 2003-->
    <extable id="fac_data" start="01/01/2003 00:00:01" end="12/31/2003 23:59:59"
        newstart="01/01/2005 00:00:01" newend="12/31/2005 23:59:59">
        <excolumn name="Cost" trend="120"/>
    </extable>
    <!--expand the fac_data to have 1000000 new records in year 2006
    and the aggregation of cost will be 15 percent more than the one
    in year 2003-->
    <extable id="fac_data" start="01/01/2003 00:00:01" end="12/31/2003 23:59:59"
        newstart="01/01/2006 00:00:01" newend="12/31/2006 23:59:59">
        <excolumn name="Cost" trend="115"/>
    </extable>
</expands>
```

Figure 13 Example code of trend section

Basic elements in this section are:

- <extable> refers to a basic fact table generated in the previous section. The attributes "start" and "end" define the period for which the data will be cloned. The attributes "newstart" and "newend" defines the time range for the new data.
- <excolumn> refers to a fact in a fact table. The attribute "trend" defines the change between the data being cloned and new data. For example, the value "110" means there is a ten percent increase on aggregation of the fact "Cost" in new data comparing to the data being cloned.

The result of this section will be a number of final fact tables filled up with test data in a full range of business periods showing aggregate level trends.

This chapter introduces a specification-based strategy for test data generation and has the specification language explained. In next chapter we present a case study by simulating the enterprise performance management system for a major teaching hospital in our lab environment and integrating our quality assurance framework.

Chapter 5. Hospital Performance Management System Case Study

Our integrated quality assurance framework for enterprise performance management systems and our tool for generating dimensional test data were validated in a case study based on the performance management system for a major teaching hospital. Hospitals are significant enterprises with a diverse set of stakeholders and quality assurance issues to be addressed. All the characteristics and analysis of enterprise performance management systems that have been described in this thesis apply to hospital performance management systems. In addition, privacy is a critical issue for hospitals so the requirements around usage monitoring and protecting privacy are particularly relevant.

5.1. Case Study

This research is a part of the project "Evolving E-Health Business Process around Accessible Data Warehouses [EHealth2007]" at the University of Ottawa. The project has created a simulation environment in which the business processes of a major teaching hospital are modeled and the performance management infrastructure of the hospital is duplicated. In particular, a data warehouse has been created with the exact same schema and structure as the teaching hospital and the same volumes of data to create comparable reports and analyses that can be validated with health researchers at the hospital. The aim is

to demonstrate how explicitly modeled business processes can be measured and adapted to leverage performance management infrastructure in order to improve the efficiency and effectiveness of health care.

In our case study, we first worked with hospital researchers to analyze three hospital business processes and generate test data related to these processes in our hospital data warehouse. Then we designed user scenarios for how healthcare workers would interact with the hospital performance management system that we could simulate with test scripts. With the test data, researchers in the ORNEC team developed reports and data marts and portal interfaces relevant to the hospital business processes that were modeled. With this case study environment, we were able to evaluate our proposed integrated quality assurance framework.

In order to evaluate our integrated quality assurance framework we first used OpenSTA, an off-the-shelf black box testing tool to see how well it could provide quality assurance of scalability, reliability, usage and privacy. Then we used our integrated framework to address the same quality assurance issues and compared it to the standard off-the-shelf approach.

We also compared the tool we created for generating test data with three off-the-shelf data generation tools.

5.1.1 Performance Management System

The infrastructure of the performance management system is shown in the following figure.

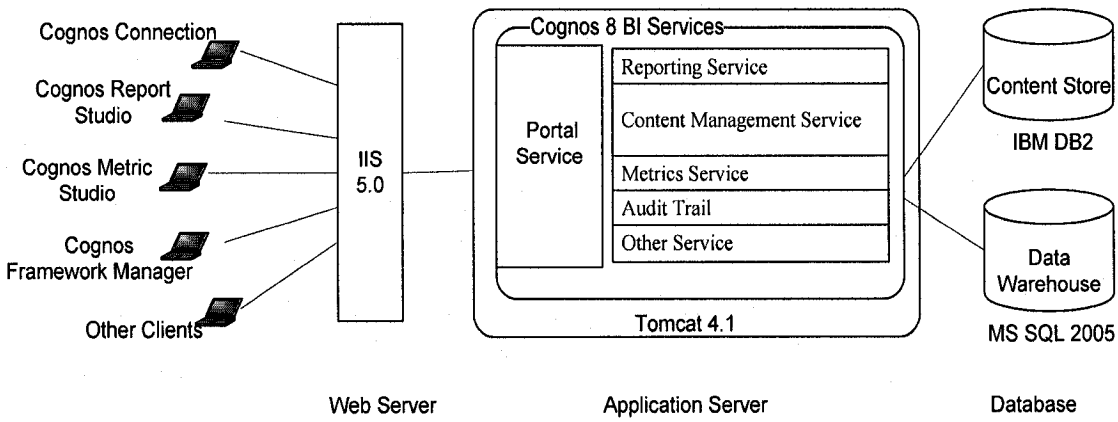


Figure 14 Performance management system infrastructure

In the target performance management system, there are several kinds of clients:

- Designers, who use Cognos Framework Manager, a thick client windows application to build and design multidimensional data marts presenting the information in one or more data sources and publish metadata model of data marts as packages to the content store to support performance management reporting.
- Report developers, who use Cognos Report Studio to create reports to measure and communicate the performance of business processes. They also use Cognos Metric Studio to organize the data into balanced scorecards in order to define and report on metrics across the entire organization. Developers also use Cognos Connection to set up personalized portals for end users to focus on a particular subset of performance management reporting. Cognos Report Studio, Cognos Metric Studio and Cognos Connection are all browser-based interfaces written in AJAX technology.

- End users, who access reports and scorecards on personalized portals through Cognos Connection.

5.1.2 Performance Management Portals and Data Marts

Three healthcare processes: Tele-Health, Infection Control and Discharge processes are currently under study by researchers in our team, who need three corresponding data marts in the data warehouse for reporting. Our test data generation tool can generate test data for these data marts (see section 5.3). Based on our test data, reports for measuring business performance based on Key Performance Indicators in these three business processes are created by researchers in our team. Three portals are created to provide users a place to access and organize related reports.

5.2. Integrated Quality Assurance Framework

5.2.1 Framework Infrastructure

Our infrastructure for the integrated quality assurance framework is shown in the following figure.

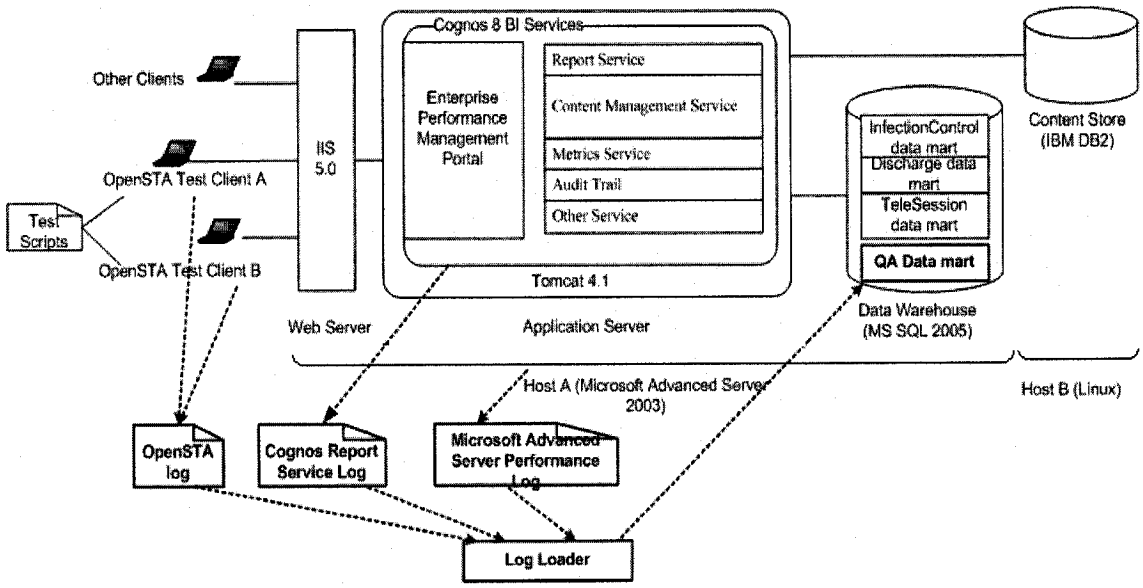


Figure 15 Integrated quality assurance framework infrastructure implemented

In our lab topology, we have the web server, application server and the data warehouse running on Host A and have the content store in Host B. Two client computers are used to simulate user actions to the system. The network has a 100 Megabit bandwidth. The detail software information is listed in the following table.

Table 1 Detailed software and hardware information in lab environment

Computer	Component (Software)	Description
Host A	Operating system (Microsoft Server 2003)	Off-the-shelf; Logs have to be explicitly enabled in Microsoft Server 2003's performance monitor and can be saved into flat files or databases; Logs contain CPU/memory/page file usage at a certain time point.
	Data warehouse (SQL Server 2005)	Off-the-shelf.
	Performance management services in SOA (Cognos 8 BI)	Off-the-shelf; Service logs have to be enabled explicitly by configure Cognos log service and logs are automatically written into database (MS SQL 2005 in this experiment); The level of logging must be configured for each service; Among service logs, the report service log contains detail information of report operations, for example, report name, package name, user name, operation type, execution time, etc.; Controlled by the setting in the Cognos log service.

	Application server (Tomcat 4.1)	Off-the-shelf (open source)
	Web server (IIS 5.0)	Off-the-shelf
	Log loader (Java Program)	Written by us in Java and running on the Host A, collecting operating system performance logs.
Host B	Operation system (Linux)	Open source.
	Content store (IBM DB2)	Off-the-shelf; Store metadata models of the performance manage system.
Test Client A, B	Operating system (Windows-XP)	Off-the-shelf.
	Testing tool (OpenSTA)	Off-the-shelf (open source); Logs are written by OpenSTA and can be exported into Excel files or CSV files, containing server response information in HTTP protocol.
	Log loader (Java Program)	Written by us in Java, running on the test clients for collecting OpenSTA logs.

5.2.2 Off-the-shelf Black Box Testing

We choose OpenSTA, an open source application designed to easily launch load testing for web applications. The newest version of OpenSTA is 1.4.3 released two years ago. Testing is performed using the record and replay metaphor common in most commercially available toolsets. Recordings are made in the tester's own browser producing simple scripts that can be edited and controlled with a special high level scripting language.

These scripted sessions can then be played back to simulate many users by a high performance load generation engine. Using this methodology a user can generate realistic heavy loads simulating the activity of one to thousands of virtual users [OpenSTA2007].

Simulation of User Actions

In the hospital, the performance management system is expected to be used by more than one thousand professionals and executive stuffs. A concurrency rate of 10% is assumed, with 10% of concurrent users making requests at a given moment. In other words, for 10,000 users, the assumption is that 1,000 users will be actively using the system at the same time, and that 100 users will be actively making requests. Both common (browse reports through portals) and complex (create/edit reports) user operations will be simulated during the quality assurance process. Running reports will execute database queries and creates heavy load to the performance management system; hence, in practice reports are scheduled to run every day, week or month during off hour to catch up-to-date data, and refreshed reports are cached by Cognos server for users to browse in daily operation. A simple report relates to no more than 10000 records in database tables, and a complex reports related to more than 10000 records in database tables. The following table shows the different test scripts that were used and the number of virtual users that were running each test script.

Table 2 Number of virtual users in different scenarios

Test Script	Virtual users
Browse cached results from several reports using Report Viewer	50
Browse cached results from several complex reports using Report Viewer	20
Running several simple reports from Cognos Connection	10
Running several complex reports from Cognos Connection	10
Author and modify report definitions using Query Studio	10
Total:	100

To create loads on the system, we used OpenSTA to record users' actions into test scripts and created 100 virtual users in two client computers (50 in each computer) to replay those test scripts for one hour.

Data Collection

OpenSTA collects server responses into log files. Once test runs are complete, a couple of reports are generated by OpenSTA to help analysis on qualities of the target performance management system, but OpenSTA neither provides dynamic aggregations on data in graphs nor drill through functions between related reports. Reports (see Figure 16) on the scalability of the system include:

- HTTP Response Time vs. Number Of Responses

- HTTP Response Time vs. Elapsed Time

From these two reports, we can see that the response time of the system is very high at the beginning because each report needs to query databases to retrieve data. Once a report runs, it will be cached on the server and the corresponding response time goes down.

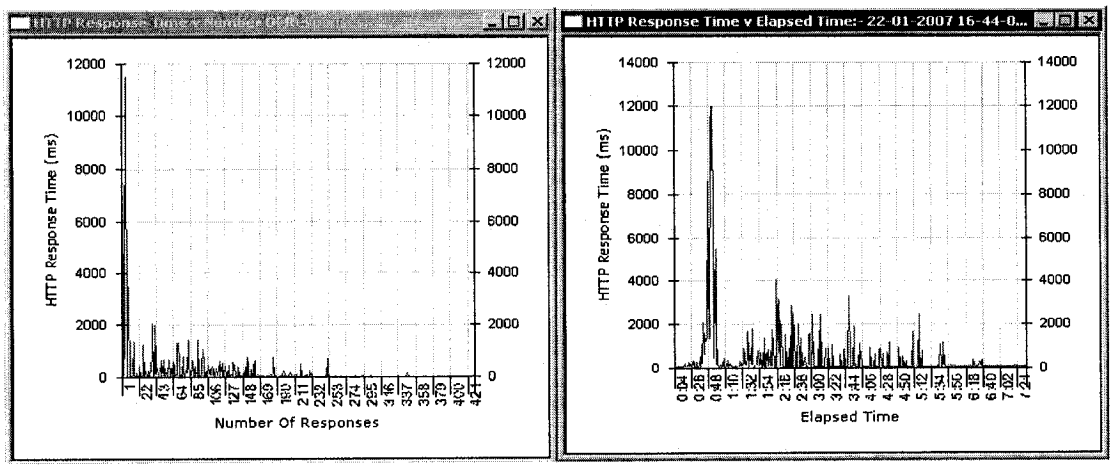


Figure 16 Scalability analysis by OpenSTA

Reports (See Figure 17) on the reliability of the system include:

- HTTP Errors vs. HTTP Requests
- HTTP Errors vs. Elapsed time

These two reports show that at the beginning, with the number of requests increases and the time elapsed, the number of HTTP errors increases.

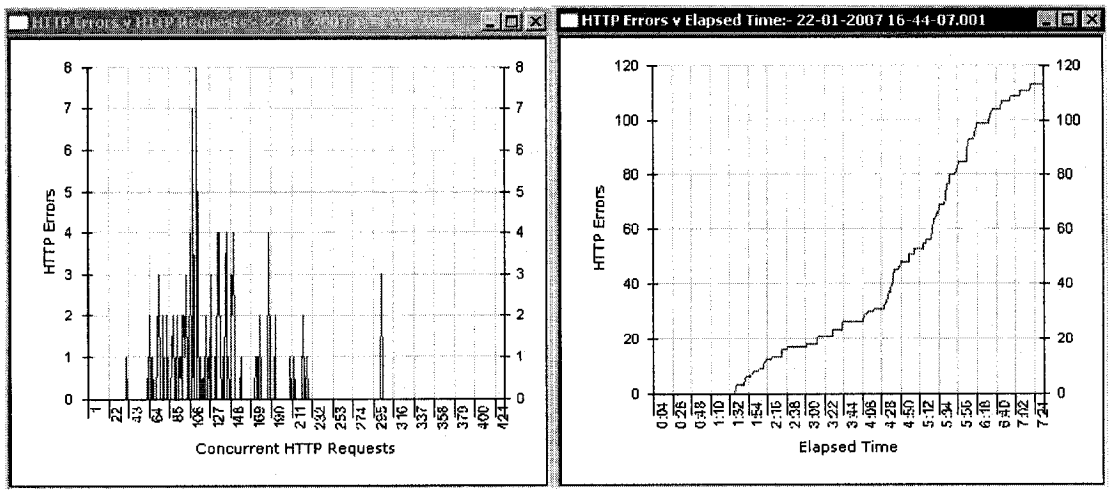


Figure 17 Reliability analysis by OpenSTA

OpenSTA clients are able to keep generating 100 users' traffic to the performance management system, and the system is reliable over long periods of time. However, OpenSTA data collection and reporting was not reliable and cannot handle the load. We repeated using OpenSTA to collect data in 1 hour, 20 minutes and 10 minutes, and only the test in 10 minutes collected data and returned reports. It obviously needs a more robust data collection architecture as most performance management systems.

5.2.3 Integrated Quality Assurance Framework

Configuring the System for Logging

Before testing the performance management system, the system must be configured to make the logging function available in services/components. To avoid affecting the performance of the performance management system, we only have the system to log information useful for the quality assurance on the scalability, reliability, active use and privacy of the performance management system. We enabled Cognos 8 BI report service log

in detail level and enabled server operating system (MS Advanced Server 2003) performance log and configured it to log CPU/Memory/Page File usage at each 15 seconds.

Simulation of User Actions

During the development of the system, testing tools or programs can be used to simulate user actions to the system. In the previous black box testing framework, user actions have been recorded by OpenSTA and corresponding test scripts and tasks have been executed and saved in two client computers. In this experiment, we used OpenSTA to replay those test scripts and tasks to simulate 100 real users accessing the system simultaneously in one hour.

Data Collection

After the simulation of user behaviours to the performance management system, logs happened in this process are available for collection. We checked several component/service logs in the system. Some logs have useful and well formatted information. For example, Cognos 8 Business Intelligence records detailed information on report operations and it can save that information directly into a database. Some logs have useful information, but they are not well formatted or facts need to be further extracted by a log loader, a java program responsible for extracting, transforming and loading data from logs into the data warehouse. For example, the operating system of the server, Windows Server 2003, can log the usage of system resources such as processor and memory. However, since it can not directly save logs into the data warehouse database (MS SQL 2005), we have to export logs into flat files and import them into the data warehouse manually. Moreover, the time in flat files can not match the "timestamp" column type in the database, which needs the log loader to format it. Another example is OpenSTA logs. Time in

OpenSTA logs are recorded as "mm:ss", which only has second (ss) and minute (mm) information, and the log loader need to format the time to be "dd/MM/yyyy hh:mm:ss", which has second, minute, hour, day, month and year information. Some logs have mass format or useless information. For example, the application server, Tomcat 4.1 records its running status, for example, loading java programs. But this information is too complex for customers to analyze it status. The data warehouse database (SQL 2005) logs don't have performance information, for example, how long it takes a query to execute.

Finally we collected OpenSTA logs, server operating system performance logs and Cognos report service logs. Tables corresponding to those logs are created in the database.

Data Modeling

After collecting logs into the data warehouse, a quality assurance data mart is created to organize quality data. Cognos 8 Business Intelligence provides a modeling tool, Framework Manager, to create a virtual dimension model over relational data sources. Dimensions and fact tables are extracted from relational data sources and then business views are created with necessary dimensions and facts to support analysis on specific business goals. In this experiment we use the Framework Manager to model the quality data and publish a package containing necessary query items (columns) which will be used to generate quality assurance reports. The following figure shows a sample dimension model on quality data.

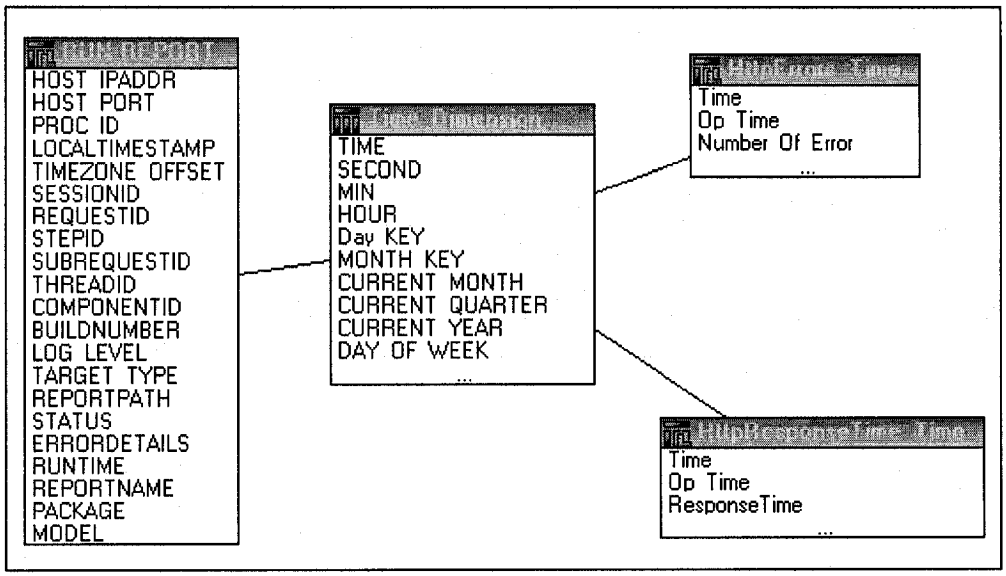


Figure 18 A dimension model of quality assurance data

Data Analysis

Based on data mart models in the published quality assurance package, a number of quality assurance reports are created using Cognos 8 reporting functions. A quality assurance portal (See Figure 19) is created and integrated into the enterprise performance management portal to manage those reports. The top-left pane of this portal is a report that shows how reports have been used, and the top-right pane shows a report on the number of report requests per day. These two reports are at the top because they are the most interesting reports to hospital users. The bottom-left pane shows links to other quality assurance reports and the bottom-right pane shows links to information related to the testing. Through the quality assurance portal, users can analyze the scalability, reliability, and usage of the performance management system and locate quality issues.

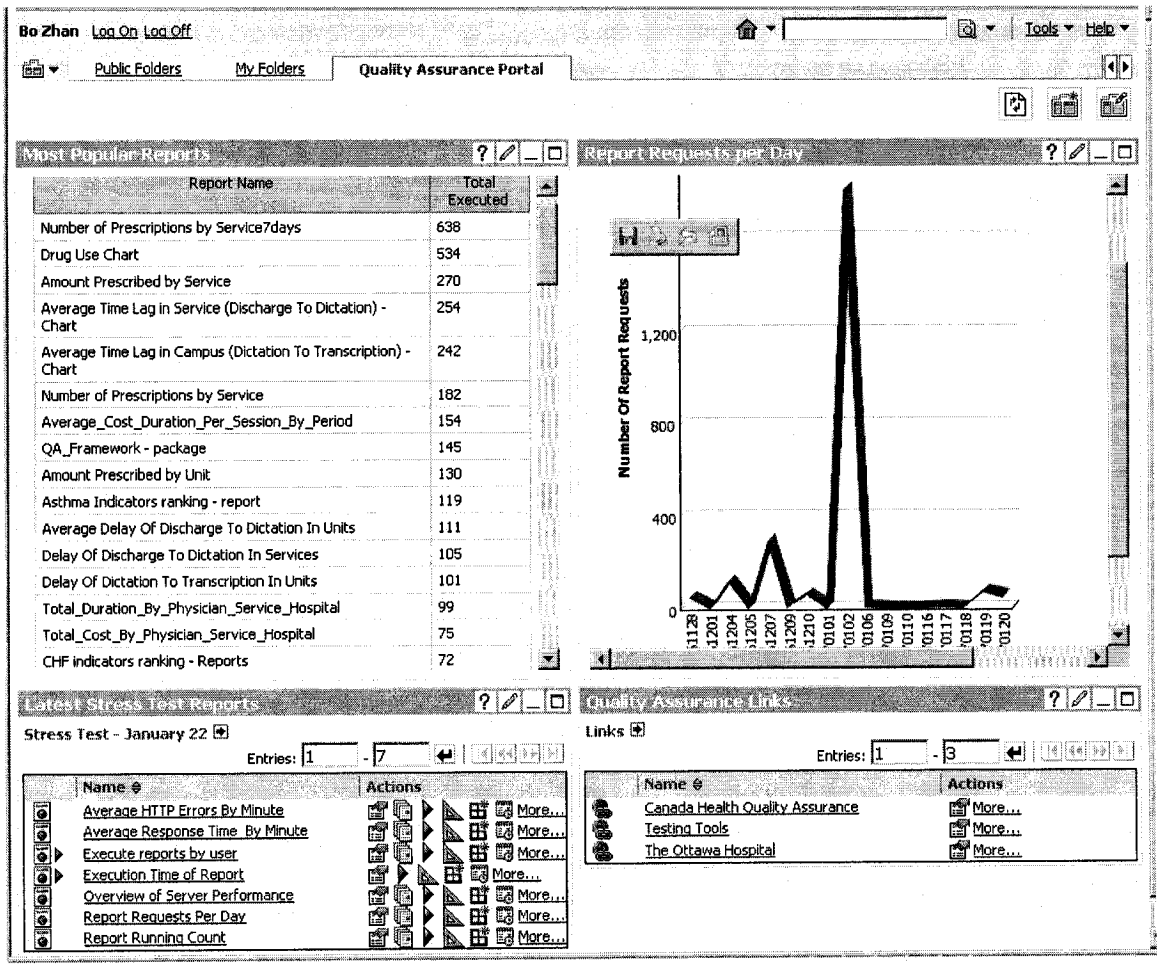


Figure 19 The quality assurance portal

Several reports (See Figure 20 and Figure 21) can be used to show the scalability of the performance management system under a certain environment. For example, "Average Response Time by Minute" is extracted from the OpenSTA logs and shows the same information as the report "HTTP Response Time vs. Elapsed Time" in Figure 16. "Report Execution Detail" is extracted from Cognos logs, which shows the execution time of each report running. "Overview of Server Performance" is based on the server platform logs, which shows the CPU usage, memory usage and paging file usage of the server under the testing.

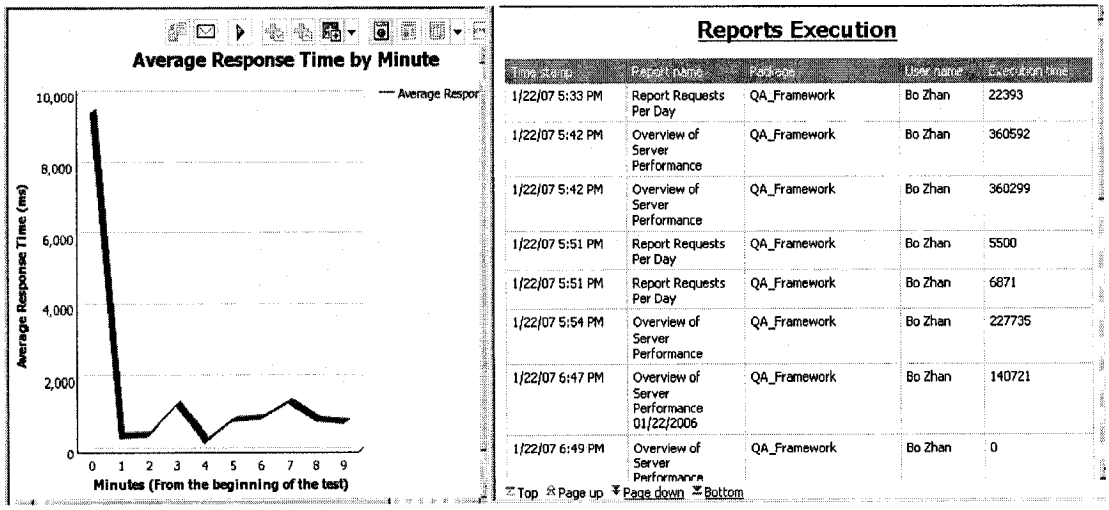


Figure 20 Report "Average Response Time by Minute" and "Reports Execution"

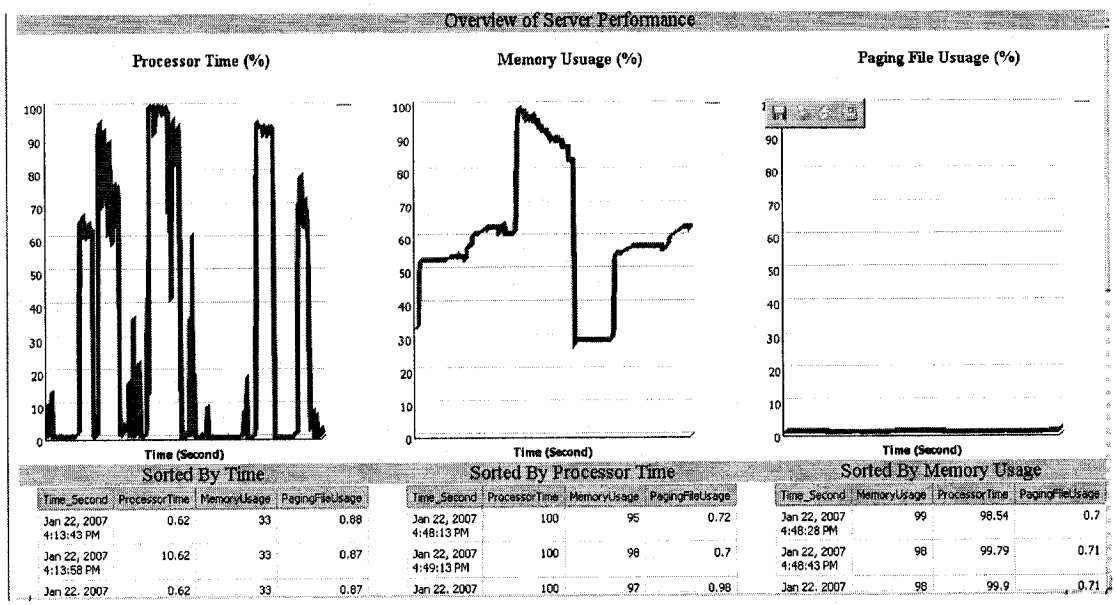


Figure 21 Report "Overview of Server Performance"

From the report "HTTP Errors by Minute" in Figure 22 based on the OpenSTA logs, we can see the reliability of the performance management system during stress testing. The curve in the graph shows that errors are increased continuously in 10 minutes. This report

does not reflex the correct information on the reliability of the system, but we use it as an example to show that the integrated quality assurance framework can collect and report logs from testing tools or programs.

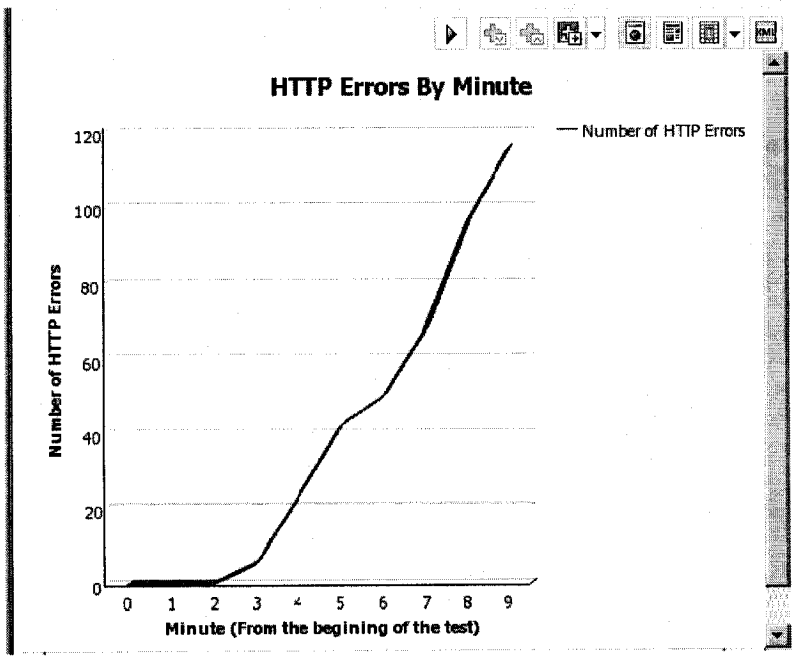


Figure 22 Reliability analysis from the quality assurance portal

To ensure the active use of the performance management system, we have reports "Most Popular Reports" and "Report Requests per Day" extracted from Cognos logs on the quality assurance portal (See Figure 19). These two reports show to what extent users access reports every day and what kind of reports are interesting to users. Another report, "Execute Report by User" (See Figure 23) extracted from Cognos logs shows who access which report and when. All those three reports tell use how the system is being used.

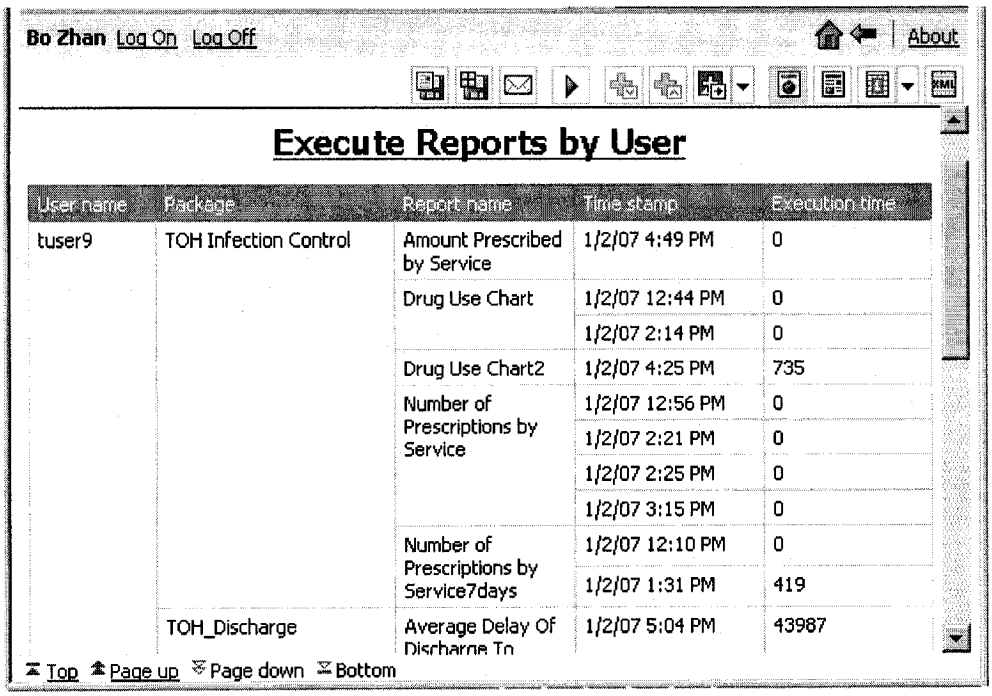


Figure 23 Report "Execute Reports by User"

Reports on audit trail logs can not only be used for ensuring the usage of the system, but also for monitoring the privacy of the system. At the present, we don't have solution to match privacy policies to audit trail directly, but with well designed policy documents and audit trail reports, users can monitor the privacy of the system in an easy way. For example, based on policy documents, users can summarize who in which departments can view which reports in which packages into a database table. Combining this table and the report "Execution Reports by Users", users can easily discover privacy issues.

5.2.4 Quality Assurance for the Performance Management System in Production

In addition to the quality assurance that was done with OpenSTA simulations in development, the performance management system has also been used "in production" for a

series of demonstrations to hospital professionals and ORNEC staff. Limited access was given to users from these two institutes to try the three portals that were set up.

In this situation, the black box testing framework using OpenSTA is not suitable for quality assurance for the system because scalability and reliability of the system have to be measured under the actions of the actual users. But the integrated quality assurance framework works for the system in this situation. Cognos BI reporting service log and the server operating system performance log were continuously enabled and these logs were collected for measuring qualities of the system. At one point, we were able to find out that the performance of the system was slowing down due to lack of memory on the server by looking at the server performance report (see Figure 21).

5.3. Test Data Generation

To drive the performance management portal and accurately assess quality in a test environment, we need test data that has similar volumes to production with similar trends and aggregate results. Real data in a hospital data warehouse is highly sensitive with privacy concern and is protected. We do not have full access to it, but real information such as number of physicians and name of hospital departments are available, which is not sensitive and can be directly used to generate test data. Moreover, we can collect small samples of data through our interaction with researchers and doctors at the hospital. With the small sample data set, our test data generation tool would generate test data for the data warehouse.

5.3.1 Test Data Requirements

As required by other researches, three data marts are needed to be constructed for three healthcare processes: Tele-Health, Infection Control and Discharge process. All of three data marts have a time-based fact table which has facts related to certain business periods. The fact table will use primary keys of several dimension tables as foreign keys.

The Tele-Health Data Mart

The Tele-Health fact table (See Figure 24) has a total of four million records separated into four years (from 2003 to 2006). It has a normal distribution of the fact "cost", and it also has a yearly trend on the aggregation of that fact, for example, the aggregation of "cost" in year 2004 is 10 percent more than the one in year 2003. A small real data set saved as Microsoft Access database file is available to use, but physician name and hospital name contained in this data set were obfuscated before using it.

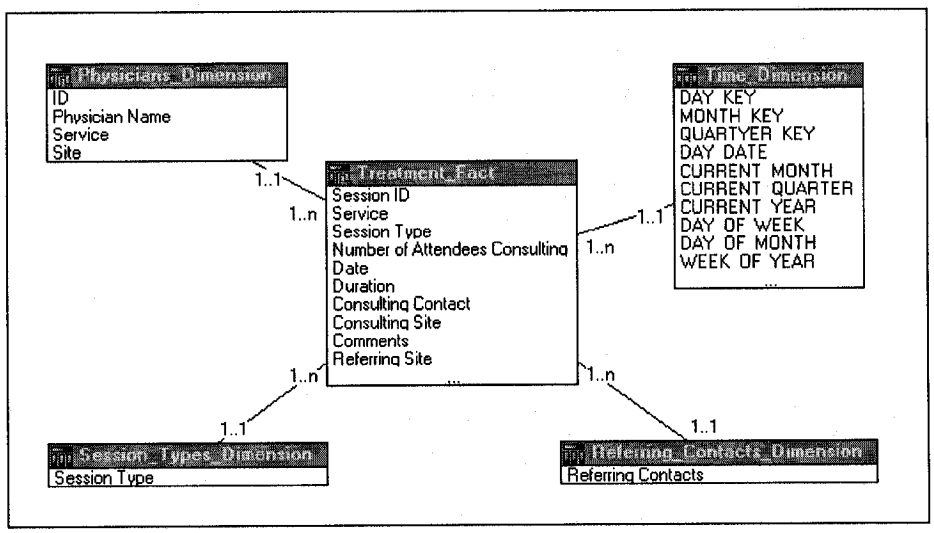


Figure 24 Dimension view of the Tele-Health data mart

The Infection Control Data Mart

The Infection Control fact table (See Figure 25) has ten thousands records distributed in year 2006, and six more thousands records in November of that year. In each record, "End Time" must be after "Start Time", and "Total Amount of Drug" is the multiplication of "Hourly Amount" and "Total Hours".

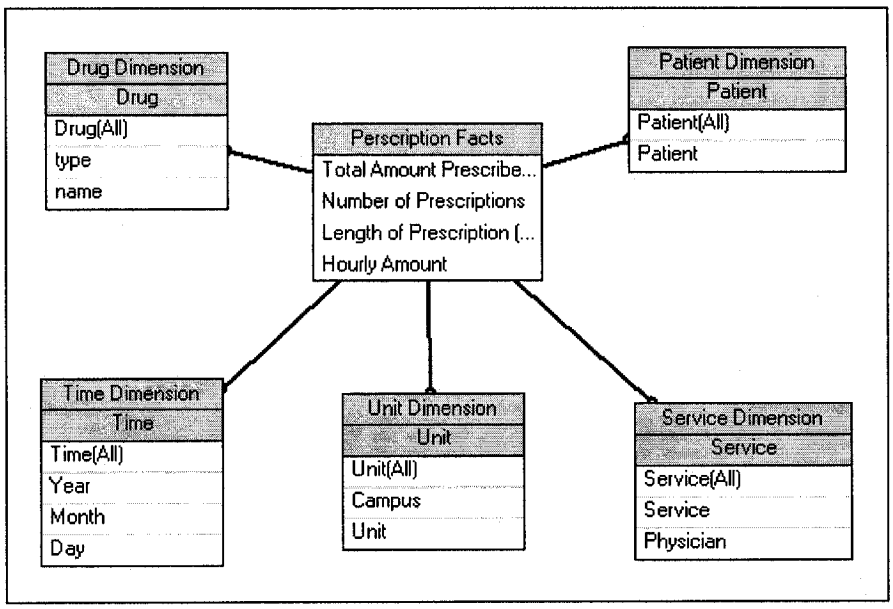


Figure 25 Dimension view of the Infection Control data mart

The Discharge Data Mart

The Discharge fact table (See Figure 26) has ten thousands records distributed in November of 2006. Especially "TimeToDischarge", "TimeToDictation", and "TimeToTranscription" should all be empty in five hundreds records. In eight thousands records "TimeToDischarge" should be exactly in November of 2006; "TimeToDictation" should be after "TimeToDischarge" in 0 to 15 days; "TimeToTranscription" should be after "TimeToDischarge" in 0 to 10 days. In other records it asks: "TimeToDischarge" should

be exactly in November of 2006; "TimeToDictation" should be after "TimeToDischarge" in 1 to 3 months; "TimeToTranscription" should be after "TimeToDischarge" in 0 to 10 days.

Real information, for example, number of physicians, name of hospital departments are collected from the hospital and that real information is not sensitive and can be directly used to generate test data.

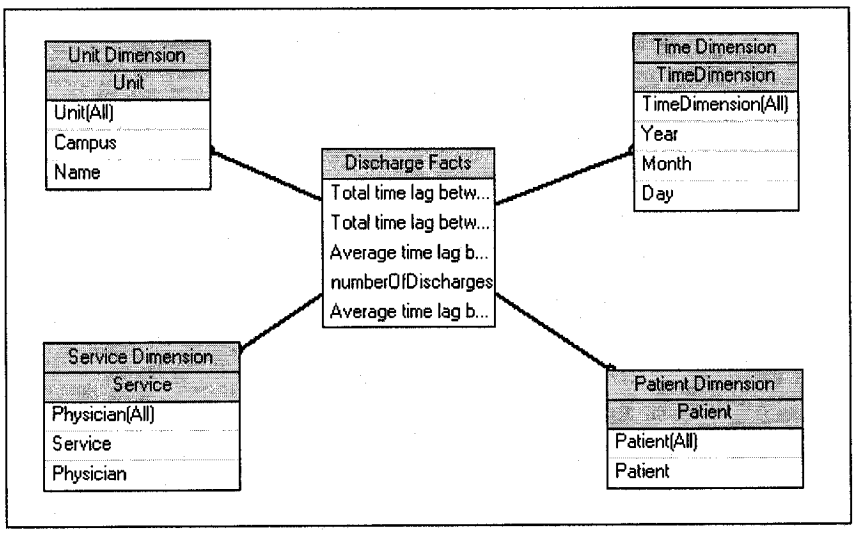


Figure 26 Dimension view of the Discharge data mart

5.3.2 Test Data Generation Using off-the-shelf Tools

Three off-the-shelf tools have been evaluated in generating test data for the performance management system. The Advance Data Generator [ADG2007] supports connections both to database and local flat files. It can generate test data in large volume and output data into flat files. It has a GUI which makes it easy to learn and use. It support lookup tables only in flat files with only one column, which means that you cannot use a column in a database table or flat files with multiple columns as lookup tables; hence, it failed to use the real data set for Tele-Health Session. It does not support data warehouse star

schema explicitly, which means we can only configure and generate one table at a time and this make it complex to resolve the linkage from a fact table to dimension tables. It cannot keep data consistency between fact tables and dimension tables. It does not accept time as a parameter to generate time-based facts table, and it does not support generating test data in normal distribution. It can expand a table in database or flat files by appending new data into it, but new data has no relation with existing data in this table; hence there is no defined trend can be applied to facts in fact tables. It can save a test data generation task and reuse it, but it can only execute one task each time.

The DTM Data Generator [DTM2007] has functions similar to the Advance Data Generator. Moreover it supports using lookup tables from other database and flat files so that columns in the real data set can be treated as lookup tables to generate test data. There is no explicit configuration for data warehouse star schema with fact tables and dimension tables. It can generate time-type columns in target table, but there is no relation between those time-type columns can be set; therefore, it failed to generate the fact tables of the Infection Control and Discharge data marts because these two tables have more than one time-type columns with relations between them. It has same problem as the Advanced Data Generator when expanding a fact table. It does not support generating test data in normal distribution or using expression to generate data, and it can only execute one task each time. The GUI of this tool is very complex and difficult to learn and use.

The TurboData [TurboData2007] has a GUI which helps users to manage project with connections to data sources, and it is easy to learn and use. Except for the functions we mentioned for DTM Data Generator, it supports generating data from other columns to derive the value for a column by applying an expression written in Microsoft Visual Ba-

sic Script, which means it support the relation between columns in fact table, so it is suitable to generation time-based fact table. However, it does not support expanding a fact table with a trend applied on facts, so it fails to generate the fact table of the Tele-Health Session data mart. It neither supports generating test data in normal distribution nor executing batch tasks.

5.3.3 Dimensional Test Data Generation

Since the failure of three existing tools in generating test data for the target performance manage system, we developed a Java-based test data generator, following the test data generation model presented in Chapter 4. Our tool accepts XML-based test data generation specifications written in our specification language as input and executes the specification to generate test data for data warehouse.

It can connect multiple distribute databases via JDBC and local flat files and define data sources in the source section of the specification. Columns in database tables and flat files can be used as lookup tables. It explicitly support configuration for fact tables and dimensional tables in data warehouse star schema, and it can generates test data both for dimensions and time-based fact tables. It supports relations between time-type columns in fact tables and it uses Java Math Expression Parser (JEP) [Singular2007], an open source application, to support generating data by an expression, and it also supports generating fact data in a normal distribution. Finally it expands fact tables into a full-range time period by cloning existing data in fact tables and applying changes on numeric facts to produce a trend of business performance. It can execute batch tasks each time.

This tool successfully generates test data for all of three healthcare data marts. In the Tele-Health data mart, the time dimension table was generated with the time granularity at the level of day for the year from 2003 to 2006. The physician dimension table has one thousand records of fake but meaningful physician's information categorized in twenty healthcare services and distributed in two hospital campus. The fact table has been generated with one millions records for each year from 2003 to 2006. The aggregation on the "cost" in the fact table is following the required trend: total cost in 2004 is 10 percent more than the one in 2003; total cost in 2005 is 20 percent more than the one in 2003; total cost in 2006 is 15 percent more than the one in 2003.

All tables generated have been output to CSV files and imported into the data warehouse by using the import function of MS SQL Server, and then used by researchers both in the hospital and the University of Ottawa in reporting to help business process modeling.

In this chapter we performed quality assurance for a simulated enterprise performance management system using a black box testing tool and our integrated framework separately. In next chapter, we compare these two frameworks on their capabilities to perform quality assurance for performance management systems and efforts required based on this case study.

Chapter 6. Analysis of Results

Based on experiments in Chapter 5, we evaluate both black box testing by OpenSTA and the integrated quality assurance framework. There are two aspects to consider in evaluating a quality assurance framework. One aspect is what its capabilities are, for example, what aspects of quality assurance can be measured and what quality issues can the framework address. We focus on the scalability, reliability, usage and privacy of the performance management system since those qualities have the biggest impact on whether or not the system would be successful. The other aspect is how much effort is required to implement/maintain the quality assurance framework, and notice/investigate quality issues, as well as who can do it and how much skill/knowledge do they need. We also evaluate tools used in generating test data for quality assurance for the performance management system.

6.1. Quality Assurance Capabilities

Criteria for evaluating capabilities of a quality assurance framework/tool to perform quality assurance for the performance management system are listed as follow:

- Quality assurance for scalability, reliability, usage and privacy of the performance management system.
- Discover detail information of data collected.

- Enable aggregations on quality data by drilling up/down in different dimensions, for example, time, services, etc.
- Perform quality assurance across the architecture of the performance management system.
- Quality assurance of the performance management system in production.

The following table shows the summary of the evaluation of black box testing with a professional tool and the integrated quality assurance framework in doing quality assurance for the target performance management system.

Table 3 Comparison of integrated framework with black box testing

Criteria		**Black Box Testing**	**Integrated Framework**
Ensure non-functional qualities	Scalability	yes	yes
	Reliability	yes	yes
	Usage	no	yes
	Privacy	no	yes
Discover detail information of data collected		yes	yes
Enable aggregations on quality data by drilling up/down in different dimensions		no	yes
Alert system quality exceptions		no	can be done
Perform quality assurance across the architecture of the performance management system		no	yes
Quality assurance in production		no	yes

We can see that black box testing with OpenSTA can be used to simulate real user behaviors to the performance management system and report HTTP feedbacks returned by web servers with graphs and tables showing detail information of data collected. Useful reports are provided to show scalability and reliability of the performance management system on the whole system level.

But black box testing does not access the underlying architecture of the performance management system; hence qualities of components/service inside the system infrastructure can not be tested and deep analysis on the system qualities can not be done. It can not ensure usage and privacy of the system due to lacking of related data. It only provides simple aggregations on data due to lacking of metadata model on data. By OpenSTA there is no way to alert users for system exceptions and users have to view reports after a test run is completed. Moreover it is not suitable for quality assurance for the system in production.

In contrast, the integrated quality assurance framework is fully integrated into the performance management system and across the system architecture. It collects logs happened during test into the data warehouse and models quality data in multi-dimensional schema so that it can provide detail information of quality data and allow users to drill up/down in hierarchies of dimensions. It can ensure scalability, reliability and usage, and monitor privacy of the performance management system. Monitoring business performance metrics and alert business performance issues is basic functions of the performance management system, although we did not implement the function to alert users to system exceptions due to time limit. Several ways can be used to implement alert function. For example, we can use email function of the system to notice a new report ready for users,

or we can set up quality metrics and their ranges and make the system monitor those metrics against data and send email for data out of range. Finally, the integrated quality assurance framework can be used in production with quality assurance knowledge accumulated.

6.2. Quality Assurance Implementation Effort

Criteria for evaluating efforts of using a quality assurance framework/tool to perform quality assurance for the performance management system are listed as follow:

- Efforts to implement the quality assurance framework
- Efforts to maintain the quality assurance framework
- Efforts to notice a quality issue
- Efforts to investigate and determine cause of an issue
- Additional efforts to perform quality assurance in production

Efforts we used in our experiment to perform quality assurance for the performance management system are shown in the following table.

Table 4 Effort comparison

Criteria		**Black Box**	**Integrated Framework**
implement		1 person day	1 person month
maintain		1 person day	1 person week
notice an issue	scalability	View reports	Alert to view reports
	reliability	View reports	Alert to view reports
	usage	Cannot be done	Alert to view reports
	privacy	Cannot be done	Analyze reports and privacy policy documents
determine cause	scalability	Manual inspection distributed logs.	Analyze reports.
	reliability	Manual inspection distributed logs	Analyze reports.
	usage	Cannot be done	Analyze reports
	privacy	Cannot be done	Analyze reports and privacy policy documents
quality assurance in production		Cannot be done	Zero

Efforts to implement the quality assurance framework

In the black box testing framework, we do not need to be familiar with the infrastructure of the performance management system as we only target the web pages. There are efforts needed to analyze user scenarios, record user actions as test scripts. The test scripts can be saved in OpenSTA project files and copied between client computers. Test scripts can be replayed by client computers to create required traffic on the system, and then re-

ports are automatically created after test runs for analyzing collected data. I finished this part in one day.

The integrated QA framework needs lots of efforts in collecting log data into data warehouse. First, we need to enable log functions for system components. Second, we need to analyze logs and implement a log loader for them. The ETL process involves running the log loader and executing database SQL commands or functions, and double check the data collected is needed to ensure the data quality. Third, we used a metadata modeling tool to model collected data dimensionally in the data warehouse and publish metadata model for reporting. Quality assurance reports and portal need to be designed and created manually. I used one month to set it up.

Efforts to maintain the quality assurance framework

For the black box testing framework, changes on web pages will cause OpenSTA to redo the work of recording and replying test scripts, but it is still quite simple and can be done in one day.

For the integrated quality assurance framework, efforts are needed when there are changes in the system infrastructure, for example:

- Changes of applications will bring different logs into the quality assurance framework. The log loader has to be compatible with those logs and the metadata model of the quality assurance data mart has to be changed too.
- Changes on the granularity of logs of a component will cause configurations on the log loader and relation changes between metadata objects in the quality assurance data mart.

- Changes on the goal of quality assurance may ask for new reports or modification on existing reports, or the quality assurance portal.

All this work can be done in one week. It could be much fast if the integrated framework is designed to be compatible with different services/components configuration.

Efforts to notice a quality issue

With the black box testing framework, the effort needed to notice a quality issue is usually high. First, there is no alert provided to users for system exceptions, users need to find them out manually. Second, reports are integrated into the testing tool, but there is no relationship set up between reports and there is no portal to help users easily go through reports. To find out quality issues, users need go through all reports and figure out the data relation between reports; this is difficult even for quality assurance experts. Due to the limitation on data collection, it cannot notice quality issues on usage and privacy of the system.

Reports in the integrated quality assurance framework are created based on the quality assurance data mart model, a star schema, and reports content can be customized to allow users to easily find out data and graphs that they are interested. Users can drill up and down in dimensions to see the system qualities in different levels and drill though different reports to analyze related information. The quality assurance portal can also be customized by users to organize reports, which help users to focus on their goals. In our experiment, we noticed a system exception from the curve in the report "Overview of Server Performance". The integrated quantity assurance framework can notice issues on scalability, reliability, usage and privacy of the system.

Efforts to investigate and determine causes of an issue

The black box testing framework cannot perform quality assurance across the performance management system architecture; hence, it cannot report what happened inside the box. To investigate and determine causes of an issue, users need manually inspect system components' logs distributed in the system. Those logs often have complex structure, complex information and very long content, and users have to look through the content line by line. To do this, time and efforts tend to be huge. Without related data collected, it cannot investigate and determine cause of a usage or privacy issue of the system

The integrated quality assurance framework can perform quality assurance across the performance management system architecture. The framework can collect data from logs automatically by the ETL process, so data in the quality assurance data mart is well structured, cleaned and meaningful. The quality assurance portal allows users to view quality data in a single place and reports can let users find the interested data very quick. In our experiment, the report "Overview of Server Performance" has data sorted by time, so we figured out what we have done at that day and found the cause. The integrated quality assurance framework can notice issues on scalability, reliability, usage of the system. The integrated quality assurance framework does not provide an automatic way to notice a privacy issues to users, but it provides much help for users on analyzing data and report usage and discovering privacy problems with privacy policies in documents. It is true there is effort and cost to implement and integrate this framework, but both producers and customers will get much benefit without extra cost and effort in production.

6.3. Evaluation of Test Data Generation Tools

Based on our experiments on generating test data for the performance management system, we compared three off-the-shelf tools with our own tool. Criteria for evaluating test data generation tools are listed as follow:

- Support connections both to distribute databases and local flat files
- Generate test data in large volume.
- Support the data warehouse star schema. Understand the star schema of data warehouse and configure data generation for dimension and fact tables.
- Generate test data for time-based fact tables. Given a period of time, generate test data for a time-based fact table while make relations between time-related columns to be consistent.
- Expand time-based fact tables. Given a period of time and a data trend, expand a fact table to contain rows for a given time period so that the aggregation of those rows over the time period matches a defined trend or formula.
- Generate fact data in normal distribution.
- Generate fact data using an expression. Generate data for a fact column based on the value of other columns.
- Execute batch tasks. Given the description of three tasks (data generation for Tele-Health, Infection Control and Discharge Process), execute all tasks as one sequence.
- Output data into flat files which are ready to be loaded into database

According to our experiments in Chapter 5, the result of the evaluation is shown in the following table.

Table 5 Evaluation of test data generation tools.

Feature	**Advanced Data Generator**	**DTM Data Generator**	**Turbo Data**	**Our tool**
Support data warehouse star schema explicitly	No	No	No	Yes
Support connections both to distributed databases and local flat files	Yes	Yes	Yes	Yes
Generate data in large volume	Yes	Yes	Yes	Yes
Generate data for time-based fact table while keeping relations between time-type columns	No	No	Yes	Yes
Expand time-based fact table with a trend applied to facts	No	No	No	Yes
Generate fact data in normal distribution	No	No	No	Yes
Generate fact data given an expression	No	No	Yes	Yes
Execute batch tasks	No	No	No	Yes
Output data to flat files	Yes	Yes	Yes	Yes

None of the three off-the-shelf tools supports data warehouse schema explicitly and none of them can generate time-based fact tables or expand time-based fact tables with a trend on fact applied. Our test data generation tool supports data warehouse schemas explicitly and is especially designed for generating time-based fact tables and applying distributions and trends on facts. Compared to off-the-shelf tools, our tool has some limitations. For example, at the present it only supports random and standard normal distribution on facts. It can be extended to support more distributions in further development. It only supports

percent value in defining trend, but it is easy to make it to support Java Math Expression Parser (JEP) [Singular2007], which gives users more choices on defining a trend. It does not have a GUI that asks users for XML knowledge and it only supports data sources accessible via JDBC. However, it can be further developed to include a friendly GUI and to support ODBC or other data sources.

In this chapter, we evaluated both black box testing by OpenSTA and the integrated quality assurance framework for their capabilities to perform quality assurance for performance management systems and efforts required. In next chapter, we make our conclusion for this thesis and discuss future work.

Chapter 7. Conclusion

The aim of this thesis has been to create an integrated quality assurance framework to ensure the scalability, reliability, usage and privacy of enterprise performance management systems as well as address issues about building representative test data. We have focused on issues that have not yet been adequately addressed by other existing quality assurance frameworks. We also validated our work with a case study based on the enterprise performance management system for a major teaching hospital in Ontario in which we compared our integrated framework to existing black box testing approaches and compared a test data generation tool we created to commercially available data generation tools.

Compared to the black box testing, our framework has several advantages:

- Quality assurance can be done not only at the system/user interface level but also on a component by component basis within the SOA of the performance management system.
- It uses the tools of the enterprise performance management system to collect, store, report and analyze log data to measure system qualities.
- Besides showing the detail log information, it can combine more than one log and aggregate the detail in different levels and aspects to help users make appropriate decisions.

- Scalability, reliability, usage and privacy can be ensured under one quality assurance framework.
- It can monitor and communicate system exceptions to users according to predefined quality metrics.
- It can be used both during the development of the performance management system and when the system is in production; hence, there can be good transfers on experience and knowledge between these two stages.

The integrated framework can be used on any performance management systems in SOA. The usefulness and effectiveness of the integrated framework depends on several factors. First it depends on the data quality of logs. In terms of the effort needed to extract, transform and load the log data into the data warehouse, the best way for logging is to save logs into databases. Good data format of log files can save much effort in ETL process. However software applications often use flat files for logging and it is worse that many of them do not adopt a good policy for logging, for example, non-standard time format, ambiguous acronyms, non-standard encoding type, missing lookup table value, and so on. The log loader has to be compatible with different log formats which cause it to be complex in programming. Bugs in Log Loader may affect the data quality or produce wrong results.

Second, enabling log functions of software applications can dramatically affect their performance, which may also affect the performance system. This is the reason most software applications limit their log to a high level, but deep analysis on the qualities of the performance management system asks for detail information in the logs; hence the per-

formance of the performance management system and the performance of the integrated quality assurance framework have to be balanced.

Thirdly, this framework tries to collect logs into the data warehouse for a long period and this will increase the size the data warehouse very quickly, which asks for more hardware and software support and query speed tends to be slow. For example, we may ask a server platform to record the processor, memory usage in each fifteen second. In this case the number of records per month would be huge. Finally the integrated framework depends on the functions provided by the performance management system to monitor its qualities. The function failure of the performance management system will cause the failure of the integrated framework, for example, it will not provide alerts to users for system exceptions if the system is in a failure.

7.1. Contributions

This thesis has three contributions. First, we established a quality assurance framework for ensuring scalability, reliability, usage and privacy of the performance management system based on its architecture. Second, we created a data warehouse architecture for quality assurance data analysis. This architecture leverages data warehouse technology to organize and present quality assurance data for reporting. With a quality assurance portal, it can communicate the state of quality of a performance management system both during development of the system and when the system is in production. Thirdly, we developed a model for generating test data for the performance management system, and the model can be suitable for systems with a data warehouses environment.

7.2. Future work

There is still work needed on addressing privacy issues by our framework. A privacy policy is usually not highly visible and tends to be too long, and it may link to another policy. This makes it difficult to apply privacy policies in the performance management system and to have an automatic process to show who violated which privacy policy in which operation. It is possible to define a data mart to define the policies around access to data in the performance management system and policies can be related to user groups, user roles and individual users; therefore, the integrated quality assurance framework can provide a more comprehensive approach to measure privacy of the performance management system.

In this research, we focus on logs either in structured files or in databases, but the efforts for the ETL process for these logs still involve much programming work. To reduce it, applying professional ETL tools is needed. Some tools have automatic processes for organizing data resource, clean and transform data in data staging. To a large degree, a lot of these issues could be handled if tools in industry were shipped with the quality framework already integrated and ensured that the files are consistent. This can be done at least for the SOA components of off-the-shelf performance management systems. There are log files from separate technologies, for example, test client, portal server, database, operating system, etc. It would be good if the industry as a whole could define a standardized logging format and conventions (e.g. time stamps). There are already standardized libraries available like Log4J. More work needs to be done to understand this issue for enterprise software as a whole and SOA in particular.

References

[ADG2007] Upscene Productions, Advanced Data Generator 1.7.0, HTTP://www.upscene.com/, accessed 2007/05.

[Alam1997] Alam, M.S., Chen, W.H., Ehrlich, W.K., Engel, M.E., Kropfl, D.A., Verma, P., Assessing software reliability performance under highly critical but infrequent event occurrences, Proceedings of The Eighth International Symposium On Software Reliability Engineering, Volume , Issue , 2-5 Nov1997 Page(s):294 – 307.

[Alan2005] Alan J. Forster, The Ottawa Hospital Data Warehouse: Obstacles and Opportunities. IT in Healthcare Series, June 15, 2005.

[Andrews1998] Andrews, J.H, Testing using log file analysis: tools, methods, and issues, Automated Software Engineering, 1998. Proceedings of 13th IEEE International Conference, Oct. 1998.

[Andrews2003] Andrews, J.H., Yingjun, Z., General test result checking with log file analysis, Software Engineering, IEEE Transactions on Volume 29, Issue 7, July 2003.

[Berler2005] Berler, A., Pavlopoulos, S., Koutsouris, D., Using key performance indicators as knowledge-management tools at a regional health-care authority level, Information Technology in Biomedicine, IEEE Transactions on Volume 9, Issue 2, June 2005.

[Berndt2001] Berndt, D.J., Fisher, J.W., Hevner, A.R.., Studnicki, J., Healthcare data warehousing and quality assurance, Computer- Volume 34, Issue12, Dec. 2001.

[Chen2005] Chen X., Zhang J., Wu D., Han R., (2005) HIPPA's compliant Auditing System for Medical Imaging System, proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, Shanghai, China, Sept1-4.

[Ditze2005] Ditze, M., Jahnich, I., Towards end-to-end QoS in service oriented architectures. Industrial Informatics, 2005, INDIN '05. 2005 3rd IEEE International Conference on 10-12 Aug. 2005. Page(s):92-97.

[DTM2007] DTM Soft, DTM Data Generator 1.15.04, HTTP://www.sqledit.com, accessed 2007/05.

[EHealth2007] Evolving E-Health Business Process around Accessible Data Warehouses, http://cserg0.site.uottawa.ca/twiki/bin/view/EHealth/WebHome, accessed 2007/05.

[EUDPEC2002]European Union Directive on Privacy and Electronic Communications. European Parliament, Brussels, Belgium, 2002. http://register.consilium.eu.int/pdf/en/02/st03/03636en2.pdf, accessed 2007/05.

[HIPAA1996] Health Insurance Portability and Accountability Act (HIPAA), United States Congress, United States, 1996. http://aspe.hhs.gov/admnsimp/pl104191.htm, accessed 2007/05.

[Houari2004] Houari, N., Far, B.H., An intelligent project lifecycle data mart-based decision support system, Electrical and Computer Engineering, 2004. Canadian Conference on Volume 2, May 2004.

[HtmlUnit2007] Gargoyle Software Inc, HtmlUnit version 1.11, HTTP://htmlunit.sourceforge.net/, accessed 2007/05.

[PerfomanceTester2007] IBM Rational Performance Tester, http://www-306.ibm.com/software/awdtools/ tester/performance/index.html, accessed 2007/05.

[IEEE Std 610.12-1990] IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terms. IEEE Computer Society, 1990.

[IEEE Std 982.1-1988] IEEE Std 982.1-1988. IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Computer Society. 1988.

[Inmon1992] Inmon, W. H., Building the Data Warehouse, John Wiley & Sons, Inc, 1992.

[Hsien2005] I-Hsien T., Chris K., Daniel K., "UBB Mining: Finding Unexpected Browsing Behaviour in Clickstream Data to Improve a Web Site's Design" In Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2005), France, 19-22 September 2005, pp. 179-185.

[Jarrar2000] Jarrar, Y.F., Al-Mudimigh, A., Zairi, M., ERP implementation critical success factors -the role and impact of business process management, IEEE International Conference on Management of Innovation and Technology, 2000.

[Jeng2006] J.J. Jeng, Service-Oriented Business Performance Management for Real-Time Enterprise, E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services. The 3rd IEEE International Conference on 2006 Page(s):28-28.

[Jogalekar2000] Jogalekar, P., Woodside, M., Evaluating the scalability of distributed systems, Parallel and Distributed Systems, IEEE Transactions on Volume 11, Issue 6, June 2000.

[Kimball2002] Kimball R. and Ross M, The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, Second Edition, Wiley, 2002.

[Kowalski2006] Kowalski, K., Beheshti, M, Analysis of Log Files Intersections for Security Enhancement. Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on April 2006.

[Lawyer2004] Lawyer, J., Chowdhury, S., Best practices in data warehousing to support business initiatives and needs. System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on 5-8 Jan. 2004.

[Linda1998] Linda R., Software usage metrics for real-world software testing, IEEE Spectrum archive Volume 35, Issue 4, April, 1998.

[OASIS2006] OASIS, OASIS SOA Reference Model TC, HTTP://www.oasis-open.org/ committees/tc_cat.php?cat=soa, August 2, 2006.

[OpenSTA2007] OpenSTA (Open System Testing Architecture) 1.4.3, HTTP://www.opensta.org, accessed 2007/05.

[Singular2007] Singular Systems, Java Math Expression Parser, http://www.singularsys.com/jep/exampleapplets.html, accessed 2007/05.

[Peyton2004] L.Peyton, M. Nozin, Tracking Privacy Compliance in B2B Networks, Sixth International Conference on Electronic Commerce, Delft, The Netherlands, October, 2004.

[PHIPA2004] PHIPA, Personal Health Information Protection Act, Government of Ontario, Canada, 2004. http://www.e-laws.gov.on.ca/DBLaws/Statutes/English/04p03_e.htm, Accessed February 2007.

[Palza2003] E. Palza, C. Fuhrman, A. Abran, "Establishing a Generic and Multidimensional Measurement Repository in CMMI context". 28th Annual NASA Soft. Eng. Workshop (SEW'03), pp.12-20. IEEE, Dec. 2003.

[Rosenberg1998] Rosenberg, L., Ted H., and Jack S., "Software Metrics and Reliability." 9th International Symposium, Germany, Nov. 1998. Greenbelt, MD: NASA Software Assurance Technology Center, 1998.

[Ruiz2005] Ruiz, D.D.A., Becker, K., Novello, T.C., Cunha, V.S., A data warehousing environment to monitor metrics in software development processes, Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on 22-26 Aug. 2005 Page(s):936 – 940.

[Tawney2005] Tawney, B., Analyzing the patient load on the hospitals in a metropolitan area. Systems and Information Engineering Design Symposium, 2005 IEEE 29 April 2005 Pages(s):215-221.

[PIPEDA2000] The Personal Information Protection and Electronic Documents Act (PIPEDA), Department of Justice, Canada, 2000. http://laws.justice.gc.ca/en/P-8.6/text.html, Accessed 2007/05.

[TurboData2007] Canam Software, TurboData 4.0.6, HTTP://canamsoftware.com, accessed 2007/05.

[Vinoski2007] Vinoski, S., REST Eye for the SOA Guy, IEEE Internet Computing, Volume 11, Issue 1, Jan.-Feb. 2007.

[Subramanyam1999]V. Subramanyam, S.V.B. Sharma, "HPD - Query tool on Projects Historical Database", SEPG Conference in Bangalore, Feb 1999.

[Wang2004] Wang, G., Chen, A., Wang, C., Fung, C., Uczekaj, S., Integrated quality of service (QoS) management in service-oriented enterprise architectures, Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International.

[W3C2004] W3C Working Group, Web Services Architecture, Note 11 February 2004, http://www.w3.org/TR/ws-arch, accessed 2007/05.

[Wei2005] Wei, P., Tao, L., Sheng, M., Autonomic Computing, 2005, ICAC 2005, Proceedings, Second International Conference, June 2005.

[Westin1967] Westin, A., 1967, Privacy and Freedom, New York: Atheneum.

[Weyuker2002] E. J. Weyuker, A. Avritzer, A metric to predict software scalability, Proc. of the Eighth IEEE Symposium on Software Metrics, 2002.

[Yip2006] Yip, F., Ray, P., Paramesh, N., (2006) Enforcing Business Rules and Information Security Policies through Compliance Audits; XISSF - A Compliance Specification Mechanism, Business-Driven IT Management, BDIM '06.

Appendix 1: OpenSTA Script and Sample Report

To test the scalability and reliability of the performance management system, we used OpenSTA to record and reply user actions to simulate real user actions in test environment. The following script is a sample test script recorded by OpenSTA during testing. In this script, a user first logs on to the system from a login page, and then clicks on the Cognos Connection to browse the list of packages and reports, and then runs a report, and finally logs off from the system. Since the script is very long, we simplified it to show its structure with many of script deleted to save space. The missing script is basically used to extract parameters of the system environment.

```
!Browser:IE5
!Date : 5/8/2007
Environment
        Description ""
        Mode            HTTP
        Wait            UNIT MILLISECONDS

Definitions
        ! Standard Defines
        Include                 "RESPONSE_CODES.INC"
        Include                 "GLOBAL_VARIABLES.INC"

        CHARACTER*512           USER_AGENT
        Integer                 USE_PAGE_TIMERS
        CHARACTER*256           MESSAGE

        Timer                   T_TEST

        Comment:  The following code is for getting cookies

        CHARACTER*1024          cookie_1_0
        CHARACTER*1024          cookie_4_0
        CHARACTER*1024          cookie_4_1
```

```
CHARACTER*1024      cookie_4_2
CHARACTER*1024      cookie_4_3
CHARACTER*1024      cookie_5_0
CHARACTER*1024      cookie_5_1
CHARACTER*1024      cookie_15_0
CHARACTER*1024      cookie_15_1
CHARACTER*1024      cookie_15_2
CHARACTER*1024      cookie_15_3
CHARACTER*1024      cookie_24_0

CONSTANT            DEFAULT_HEADERS = "Host: id1g^J"    &
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
2.0.50727)"
CONSTANT    S_cookie_1_0    = "cogbktbsug_cratab=msrch"
```

Code

```
!Read in the default browser user agent field
Entry[USER_AGENT,USE_PAGE_TIMERS]

Start Timer T_TEST
```

Comment: The following code is for connecting to the login page of the performance management system

```
PRIMARY GET URI "http://id1g/cognos8/cgi-bin/cognos.cgi HTTP/1.0" ON 1 &
        HEADER DEFAULT_HEADERS &
        ,WITH {"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, " &
                        "application/vnd.ms-excel, application/vnd.ms-powerpoint, applica
                        tion/msword, " & "application/x-shockwave-flash, */*", &
                "Accept-Language: en-us", & "Cookie: "+S_cookie_1_0, &
                "Connection: Keep-Alive"} Load Response_Info Header on 1            &
                Into cookie_1_0  &,WITH "Set-Cookie,CRN"

WAIT 203
```

Comment: The following code is to submit username "cognos8" and password "ehealth" to log on to the system

```
PRIMARY POST URI "http://id1g/cognos8/cgi-bin/cognos.cgi HTTP/1.0" ON 4 &
        HEADER DEFAULT_HEADERS &
        ,WITH {"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, " &
                        "application/vnd.ms-excel,  application/vnd.ms-powerpoint,  applica-
                        tion/msword, " & "application/x-shockwave-flash, */*", &
                "Referer: http://id1g/cognos8/cgi-bin/cognos.cgi", &
                "Accept-Language: en-us", &
                "Content-Type: application/x-www-form-urlencoded", &
                "Connection: Keep-Alive", &
                "Content-Length: 170", &
                "Pragma: no-cache", &
```

"Cookie: "+cookie_1_0+"; "+S_cookie_1_0} &
,BODY "CAMName-space=SiteNTLM&CAMNamespaceDisplayName=SiteNTLM&CAMUsername=cognos8&CAMPassword=ehealth" &
"&h_CAM_action=logonAs&encoding=UTF-8&m=portal%2Fmain.xts&m=portal%2Fmain.xts"

WAIT 783

Comment: The following code is for clicking the link of Cognos Connection to show the performance management portal

PRIMARY GET URI &
"http://id1g/cognos8/cgi-bin/cognos.cgi?b_action=xts.run&m=portal/cc.xts&gohome= HTTP/1.0" ON 5 &HEADER DEFAULT_HEADERS &
,WITH {"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, " &
"application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, " &"application/x-shockwave-flash, */*", &
"Accept-Language: en-us", &
"Pragma: no-cache", &
"Connection: Keep-Alive", &
"Cookie: "+cookie_4_2+"; "+cookie_4_0+"; "+cookie_4_1+"; "+cookie_4_3+"; "+S_cookie_1_0}
Load Response_Info Header on 5 &
Into cookie_5_0 &
,WITH "Set-Cookie,cc_session"
Load Response_Info Header on 5 &
Into cookie_5_1 &
,WITH "Set-Cookie,caf"

.....

Comment: The following code is for clicking a report to run a report, called "Average Dictation To Transcription Time"

PRIMARY POST URI "http://id1g/cognos8/cgi-bin/cognos.cgi HTTP/1.0" ON 21 &
HEADER DEFAULT_HEADERS &
,WITH {"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, " &
"application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, " &
"application/x-shockwave-flash, */*", &
"Referer: " &
"http://id1g/cognos8/cgi-bin/cognos.cgi?b_action=xts.run&m=portal/" &"report-viewer.xts&ui.action=run&ui.object=storeID(%22i04C41655239D4E9E99BEE846E4125382%22)&ui.h" &
"eader=false&ui.toolbar=false&ui.backURL=%2fcognos8%2fcgi-bin%2fcognos.cgi%3fb_action%3dxts.run%" &
"26m%3d%2fcps4%2fcommon%2fback.xts%26url%3d%252fcognos8%252fcgi-bin%252fcognos.cgi%253fb_action%" &
"253dxts.run%2526m%253dportal%252freport-viewer.xts%2526ui.action%253drun%2526ui.object%253dstor" &

```
"eID(%252522i04C41655239D4E9E99BEE846E4125382%252522)%2526ui.header%25
3dfalse%2526ui.toolbar%253" & "dfal
se%26action%3drun&m_session=57FC7323ECC188973321577F29FAE09C", &
        "Accept-Language: en-us", &
        "Content-Type: application/x-www-form-urlencoded", &
        "Connection: Keep-Alive", &
        "Content-Length: 3785", & "Pragma: no-cache", &
        "Cookie: "+cookie_4_2+"; "+cookie_4_0+"; "+cookie_4_1+"; "+cookie_5_0+";
        "+cookie_5_1+"; "+ & S_cookie_1_0}        &
,BODY
        "m_tracking=CAFS3c00000170FAAAAC3dvH7nr5atGps8UC2AVeNYYZq4w
        lw95yjFraTvqmhQ9SmN*uibgzI_H4sIAAAAA" &
        "AAAAK2SsW6DMBCG5-
        YpInaMIZACIlmo-
        hCo1U5auFDtgNbHBd4nx25fIUkWjdkni8Xz3f99ZLj5PkDdKnrmGGoWSpZL
        IR1yM" &
        "IHK0PV97-3R4i-
        F4kJBPjWuvQ*zzIDDGELMkSrdBRGkYfGzfd03Hj7UvJGAtG*5tnp8up7ikCj
        bDjJADaiHbW4NhyFhVsS5" &
        "jqYkHuy0rA-YwDFkM1pZxdWYxszbqisCx5yZTBp7gkTZG6a9p3MFc-
        BxY7-dCCrQ71Fy22P1GC3nz8yaUUge9RszxUjH*9v" &
        "rIfaeLlU8TP1z5YULCiNCILOlLkqbOxhHnDr1WDQe41rhnd4f6CXY0V-
        zjB2**AWM1YM79AgAA&ui.conversation=CAFS" &
"3c000003d0FAAAAC3dvH7nr5atGps8UC2AVeNYYZq4ZdpMlaDNDryMHQK-
hQKzwSFU3cU_H4sIAAAAAAAAAOVY3U-bMBB-Z" &
        "n8F6sO0PdD0C0S7glRo0CqVFbUwJlV7uKZH48mxg*3047-
        HjknWAhI0Acq6vCSK7bvf3e-ufHZzFMkGgwB355I01CLEo8Jc" &
        "NqQShE0Ku-OAMtnQQ0cFX6mw4Tiz2aw4qxa5mDiVUqns-
        DrvDjwfA9gjTCpgHhaORcSaTiL4uGm**OgPeuoClL*kyAxIBOH" &
        "5ZmCgNVLsxROzapaKC*y0v3y*jbj6Rkq101r5YH*-
        Uq23a27drddPXPewduDWypX96mHFTvtqwf6FeA85VIQzuYR30Gtd7L" &
        "khstIWCREeN-
        uKgWMlMvUE1bfsFz6nUhOZqbiZSxLFvWw5BDGesyHg8zjY*0-
        p3D8aSd*mkRh8MDXmpderOSEc4rAsnr5X" &
"sVO84ngWdHjsijI6ia7SotLV41xipSHKIoenzAu9StIXOKMiJ7qVLUDJEyxF6kwUktBuAJ5CjR
6EPCjXB7JDvYGqESLM0aV" &
"0ucY-l5E5iDO1Twh*cEJ5TGZZ1wE8GJK3y5V0*o4LOdK1Oz*SAx-FBa5C-f3y-
PumYBJgEytRuBOvhjd6oITCh6E-0KxUSL" &
"KWWvaoGA7SRxry15Kofln5m8EqHbTNVF*i9J2CnlNOkEumOdbZ3dY5mbrOUKX1WwoM
UkAYnENRF36AqXP6XiNNCWZXZMdcj" &
"U3oW9cajdOKtAZLGSHeTQa44VluI*3Ecr-
owTb1tZlHoQyomB*bWdF5o8NXadAP7F8I2YwzrIxvppptm3c3qxWCjzf9J2Jq" &
"etwnXTVG4E*AokHtdc4w21nHtv2eL3jW96TTHa0q0egPJy22OKq39lOTueSXvW7H59NMS1
qqM-xeH--Z*-77OVfCEKbpVC8" &
"3-1fqvKnATrMdw3oWDNkiB65IV68-
b2bJXY3GSzrfiVzFCg0x54HyaBXxtsGjjXMzEGmRTJCjwqmR9Ma7wBBt4U*GxcAAA_" &
"_&m=portal%2Freport-
viewer.xts&b_action=xts.run&ui.name=Average+Dictation+To+Transcription+Time" &
"+Lag+In+Services&ui.errURL=%2Fcognos8%2Fcgi-
bin%2Fcognos.cgi%3Fb_action%3Dxts.run%26m%3D%2Fcps4" &
"%2Fcommon%2Fback.xts%26url%3D%252fcognos8%252fcgi-
bin%252fcognos.cgi%253fb_action%253dxts.run%2" &
"526m%253dportal%252freport-
viewer.xts%2526ui.action%253drun%2526ui.object%253dstoreID%28%252522" &
"i04C41655239D4E9E99BEE846E4125382%252522%29%2526ui.header%253dfalse%2526ui.to
olbar%253dfalse%26" &
```

"action%3Drun&ui.backURL=%2Fcognos8%2Fcgi-
bin%2Fcognos.cgi%3Fb_action%3Dxts.run%26m%3D%2Fcps4%2F" &
"common%2Fback.xts%26url%3D%252fcognos8%252fcgi-
bin%252fcognos.cgi%253fb_action%253dxts.run%2526" &
"m%253dportal%252freport-
viewer.xts%2526ui.action%253drun%2526ui.object%253dstoreID%28%252522i04" &
"C41655239D4E9E99BEE846E4125382%252522%29%2526ui.header%253dfalse%2526ui.toolb
ar%253dfalse%26act" &
"ion%3Drun&ui.object=storeID%28%22i04C41655239D4E9E99BEE846E4125382%22
%29&searchPathForURL=URLID" &
"%28%22MTM2ODoxMTY0MTY3NjY0OTIxOjg2NjQ%3D%22%29&ui.action=wait&m_sourc
e=storeID%28%22i04C4165523" &
"9D4E9E99BEE846E4125382%22%29&m_class=report&m_write=true&m_save=true&m_execu
te=true&objectClass" &
"=report&ui.gateway=http%3A%2F%2Fid1g%3A80%2Fcognos8%2Fcgi-
bin%2Fcognos.cgi&run.outputFormat=HTM" &
"LFrag-
ment&m_session=57FC7323ECC188973321577F29FAE09C&ui.toolbar=false&ui.navlinks=1&p
reviousSta" &
"te=&packageBase=%2Fcontent%2Fpackage%5B@name%3D%27TOH_Discharge%27%5D&m
odelPath=%2Fcontent%2Fpa" &
"ckage%5B@name%3D%27TOH_Discharge%27%5D%2Fmodel%5B@name%3D%272006-11-
22T14%3A26%3A49.984Z%27%5D&" &
"ui.folder=%2Fcontent%2Fpackage%5B@name%3D%27TOH_Discharge%27%5D%2Ffolder%5
B@name%3D%27Reports%2" &
"7%5D&m_ro_saveAsName=Report+View+of+Average+Dictation+To+Transcription+Time+La
g+In+Services&sav" &
"e_how=&ps_nav_stack=&ps_nav_source=portal%2Freport-
viewer.xts&ps_nav_op=&m_ro_saveAsPath=&tempP" &
"ath=&saveAsPathText=&ifrmcmd=execute&deliverySection=true&m_return=&changed_m_ro_
prompt=0&m_ro_" &
"prompt=false&m_ro_saveOutput=&m_ro_recordingEnabled=&m_ro_saveAsType=&specify_for
mat=checked&ru" &
"n.outputLocale=en-
us&BIline1=Your+report+is+running.&BIline2=Please+wait...&m_ro_outputF_HTML_T" &
"ype=true&m_ro_outputFormat_HTML=HTML&continueConversation=&_promptControl=&_a
utosubmitParameter"
&"=&run.prompt=true&promptOnRerun=true&ui.sh=cr&ui.header=false&ui.username=cognos8"

WAIT 533
.....

Comment: The following code is to log off from the system

WAIT 329

GET URI "http://id1g/cognos8/ps/portal/js/cookie_jar.js HTTP/1.0" ON 22 &
HEADER DEFAULT_HEADERS &
,WITH {"Accept: */*", & "Referer: " &
"http://id1g/cognos8/cgi-
bin/cognos.cgi?b_action=xts.run&m=portal/" &
"logoff.xts&h_CAM_action=logoff", &
"Accept-Language: en-us", &
"Cookie: "+cookie_5_1+"; "+cookie_24_0+"; "+S_cookie_1_0, &
"If-None-Match: ~<22>80474493237c61:82c~<22>", &

```
                    "Connection: Keep-Alive"}

            WAIT 126

            GET URI "http://id1g/cognos8/ps/images/space.gif HTTP/1.0" ON 23 &
                HEADER DEFAULT_HEADERS  &
                ,WITH {"Accept: */*", &  "Referer: " &
                                "http://id1g/cognos8/cgi-
                                bin/cognos.cgi?b_action=xts.run&m=portal/" &
                                "logoff.xts&h_CAM_action=logoff", &
                        "Accept-Language: en-us", &
                        "Cookie: "+cookie_5_1+"; "+cookie_24_0+"; "+S_cookie_1_0, &
                        "If-None-Match: ~<22>801a1383237c61:82c~<22>", &
                        "Connection: Keep-Alive"}
            SYNCHRONIZE REQUESTS
        End Timer T_TEST
Exit
ERR_LABEL:
        If (MESSAGE <> "") Then
            Report MESSAGE
        Endif
Exit
```

Sample Quality Assurance Report

We have seen the report "Overview of Server Performance" in Chapter 5; in the following figure we show the report design process in Cognos Report Studio. The left part of the window shows the metadata of the quality assurance data mart, containing query items in dimension tables and fact tables. To create a report, a developer simply needs to drag a query item from a fact table or a dimension table and drop it in the proper position in the right window, the report content. The tool will automatically validate the relationships between query items chosen. Users can run the report before saving it to the content store.

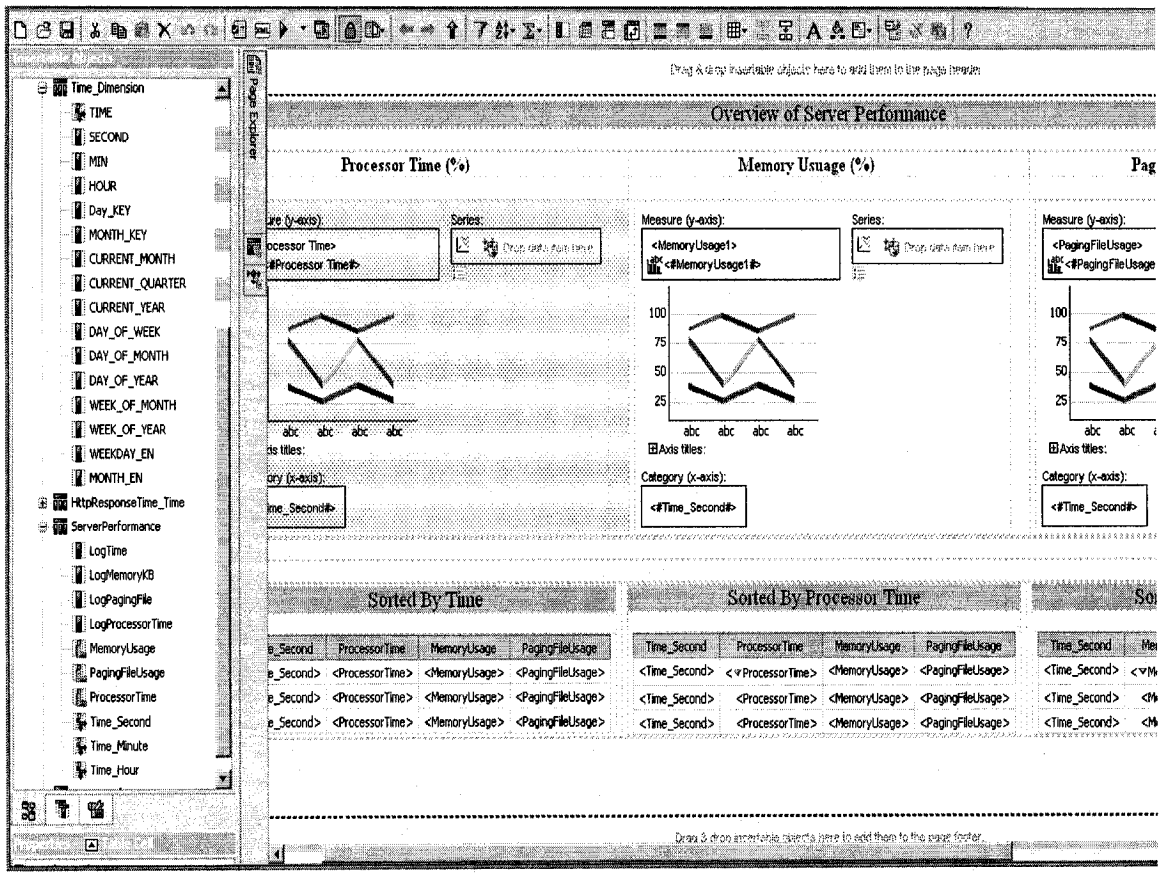


Figure 27 Sample quality assurance report in report design process

Appendix 2: XML DTD for Test Data Generation Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT project ((source, dimensions, facttables, expands))>
<!ATTLIST project
        name CDATA   #REQUIRED
>
<!ELEMENT source ((databases, files))>
<!ELEMENT databases ((database+))>
<!ELEMENT database EMPTY>
<!ATTLIST database
        host CDATA   #REQUIRED
        id    CDATA   #REQUIRED
        name CDATA   #REQUIRED
        port CDATA   #REQUIRED
        type CDATA   #REQUIRED
        user CDATA  #REQUIRED
>
<!ELEMENT files ((file*))>
<!ELEMENT file EMPTY>
<!ATTLIST file
        id    CDATA #REQUIRED
        path CDATA #REQUIRED
        header (true|false ) #IMPLIED
        delim    CDATA #IMPLIED
        qualifier CDATA #IMPLIED
>
<!ELEMENT dimensions ((dimension*))>
<!ELEMENT dimension ((column+))>
<!ATTLIST dimension
        id      CDATA #REQUIRED
        output CDATA #REQUIRED
        ref     CDATA #REQUIRED
>
<!ELEMENT facttables ((facttable+))>
<!ELEMENT facttable ((column+))>
<!ATTLIST facttable
        id CDATA #REQUIRED
        ref CDATA #REQUIRED
        output CDATA #REQUIRED
```

```
        append (true|false ) #IMPLIED
        rows CDATA #REQUIRED
        start CDATA #REQUIRED
        end CDATA #REQUIRED
>
<!ELEMENT column EMPTY>
<!ATTLIST column
        name CDATA #REQUIRED
        by (time | sequence | random | lookup | dim | timedepend | expression)  #REQUIRED
        dim CDATA #IMPLIED
        distribution (random|normal) #IMPLIED
        lookup CDATA #IMPLIED
        depend CDATA #IMPLIED
        max CDATA  #IMPLIED
        min CDATA #IMPLIED
        unit CDATA #IMPLIED
        trend CDATA #IMPLIED
        expression CDATA #IMPLIED
>

<!ELEMENT expands ((extable*))>
<!ELEMENT extable ((excolumn))>
<!ATTLIST extable
        id CDATA #REQUIRED
        newstart CDATA #REQUIRED
        newend CDATA #REQUIRED
        start CDATA #REQUIRED
        end CDATA #REQUIRED
>
<!ELEMENT excolumn EMPTY>
<!ATTLIST excolumn
        name CDATA #REQUIRED
        trend CDATA #REQUIRED
>
```

Appendix 3: Infection Monitoring Data Generation and Sample Report

---

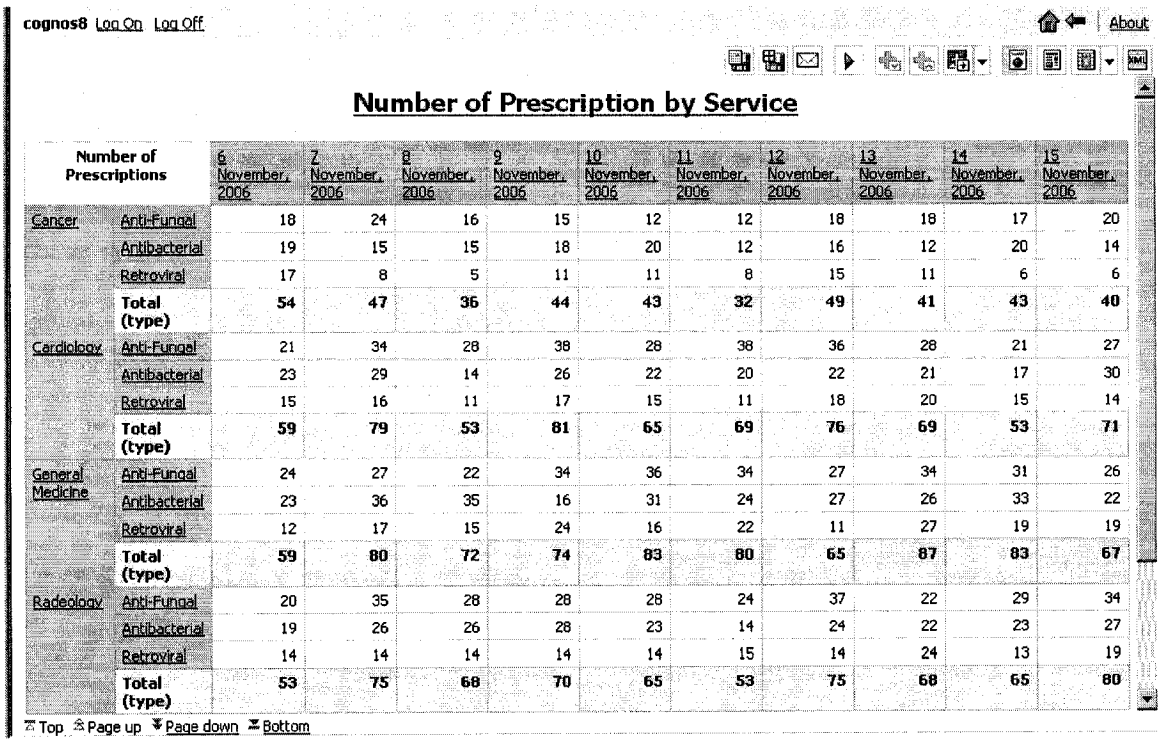
Our tool executed the following specification to generate test data for Infection Control data mart according to data requirements.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE project SYSTEM "C:\workspace\ve\G5\model\model.dtd">
<project name="Infection Control Data Mart">
        <source>
                <databases>
                        <database id="d_toh_dw" name="toh_dw" type="SQL2005"
                        host="id1g" port="1433" user="sa"/>
                </databases>
                <files>
          </files>
        </source>
        <dimensions>
                <!-- dimension table Unit, Physician, Drug is availabe in
                the database, no need to generate-->
        </dimensions>
        <facttables>
                <facttable id="fac_prescription" ref="d_toh_dw.prescription" start="01/01/2006
                        00:00:01" end="12/31/2006 23:59:59" rows="10000" append="false"
                        output="c:\temp\data\prescription.csv">
                        <column name="PrescriptionID" by="sequence"/>
                        <column name="PatientID" by="random" min="1" max="1000"/>
                        <column name="UnitID" by="lookup"
                                lookup="d_toh_dw.unit.UnitID"/>
                        <column name="PhysicianID" by="lookup"
                                lookup="d_toh_dw.physician.PhysicianID"/>
                        <column name="DrugID" by="lookup"
                                lookup="d_toh_dw.drug.DrugID"/>
                        <column name="StartTime" by="time"/>
                        <!--three case for TotalHours, (5 days, 10 days or 15 days)*24 -->
                        <column name="TotalHours" by="random" min="1" max="3"
                                unit="120"/>
                        <!--the end time is in minute -->
                        <column name="EndTime" by="expression" expres
```

```
                    sion="StartTime+(TotalHours)*60"/>
                <!--the hourly amount is random from 1 dose to 10 dose-->
                <column name="HourlyAmount" by="random" min="1" max="10"/>
                <column name="TotalAmount" by="expression" expres
                    sion="HourlyAmount*TotalHours"/>
            </facttable>
            <facttable id="fac_prescription" ref="d_toh_dw.prescription" start="11/01/2006
                    00:00:01" end="11/30/2006 23:59:59" rows="6000" ap-
                    pend="true" output="c:\temp\data\prescription.csv">
                <column name="PrescriptionID" by="sequence"/>
                <column name="PatientID" by="random" min="1" max="1000"/>
                <column name="UnitID" by="lookup"
                    lookup="d_toh_dw.unit.UnitID"/>
                <column name="PhysicianID" by="lookup"
                    lookup="d_toh_dw.physician.PhysicianID"/>
                <column name="DrugID" by="lookup"
                    lookup="d_toh_dw.drug.DrugID"/>
                <column name="StartTime" by="time"/>
                <!--three case for TotalHours, (5 days, 10 days or 15 days)*24 -->
                <column name="TotalHours" by="random" min="1" max="3"
                    unit="120"/>
                <!--the end time is in minute -->
                <column name="EndTime" by="expression" expres
                    sion="StartTime+(TotalHours)*60"/>
                <!--the hourly amount is random from 1 dose to 10 dose-->
                <column name="HourlyAmount" by="random" min="1" max="10"/>
                <column name="TotalAmount" by="expression" expres
                    sion="HourlyAmount*TotalHours"/>
            </facttable>
        </facttables>
        <expands>
            <!--no expand-->
        </expands>
</project>
```

In this specification, dimension tables such as "Unit", "Physician", "Drug" have been created manually by researchers in the E-Health team, and primary keys of those dimension tables, such as "physicianID", "unitID" and "drugID", are used as lookup tables in generating the fact table. To generate facts "EndTime" and "TotalAmount", expressions supported by Java Math Expression Parser (JEP) [Singular 2007] are used. In the fact table section, it will first generate 10000 normal records for year 2006 and then generate 6000 more records for November of 2006 with different characteristics on facts.

The following screenshot is a sample report created by researchers in the E-Health team to measure the performance of the infection control process based on test data generated.



cognos8 Log On Log Off About

Number of Prescription by Service

Number of Prescriptions		6 November, 2006	7 November, 2006	8 November, 2006	9 November, 2006	10 November, 2006	11 November, 2006	12 November, 2006	13 November, 2006	14 November, 2006	15 November, 2006
Cancer	Anti-Fungal	18	24	16	15	12	12	18	18	17	20
	Antibacterial	19	15	15	18	20	12	16	12	20	14
	Retroviral	17	8	5	11	11	8	15	11	6	6
	Total (type)	**54**	**47**	**36**	**44**	**43**	**32**	**49**	**41**	**43**	**40**
Cardiology	Anti-Fungal	21	34	28	38	28	38	36	28	21	27
	Antibacterial	23	29	14	26	22	20	22	21	17	30
	Retroviral	15	16	11	17	15	11	18	20	15	14
	Total (type)	**59**	**79**	**53**	**81**	**65**	**69**	**76**	**69**	**53**	**71**
General Medicine	Anti-Fungal	24	27	22	34	36	34	27	34	31	26
	Antibacterial	23	36	35	16	31	24	27	26	33	22
	Retroviral	12	17	15	24	16	22	11	27	19	19
	Total (type)	**59**	**80**	**72**	**74**	**83**	**80**	**65**	**87**	**83**	**67**
Radeology	Anti-Fungal	20	35	28	28	28	24	37	22	29	34
	Antibacterial	19	26	26	28	23	14	24	22	23	27
	Retroviral	14	14	14	14	14	15	14	24	13	19
	Total (type)	**53**	**75**	**68**	**70**	**65**	**53**	**75**	**68**	**65**	**80**

Top Page up Page down Bottom

Figure 28 Report "Number of Prescription by Service"

We use the next screenshot to show the metadata model of this report. From the left side of the window, we can see metadata models of dimensions such as Drug, Patient, Service, Unit and Time, also the Prescription fact table. A crosstab was created in the right side of the window. Rows of this crosstab are a list of services, and columns of this crosstab are a list of days in November 2006, and the measure is number of prescriptions. The crosstab will automatically do aggregation on the measure in both service dimension and time dimension.

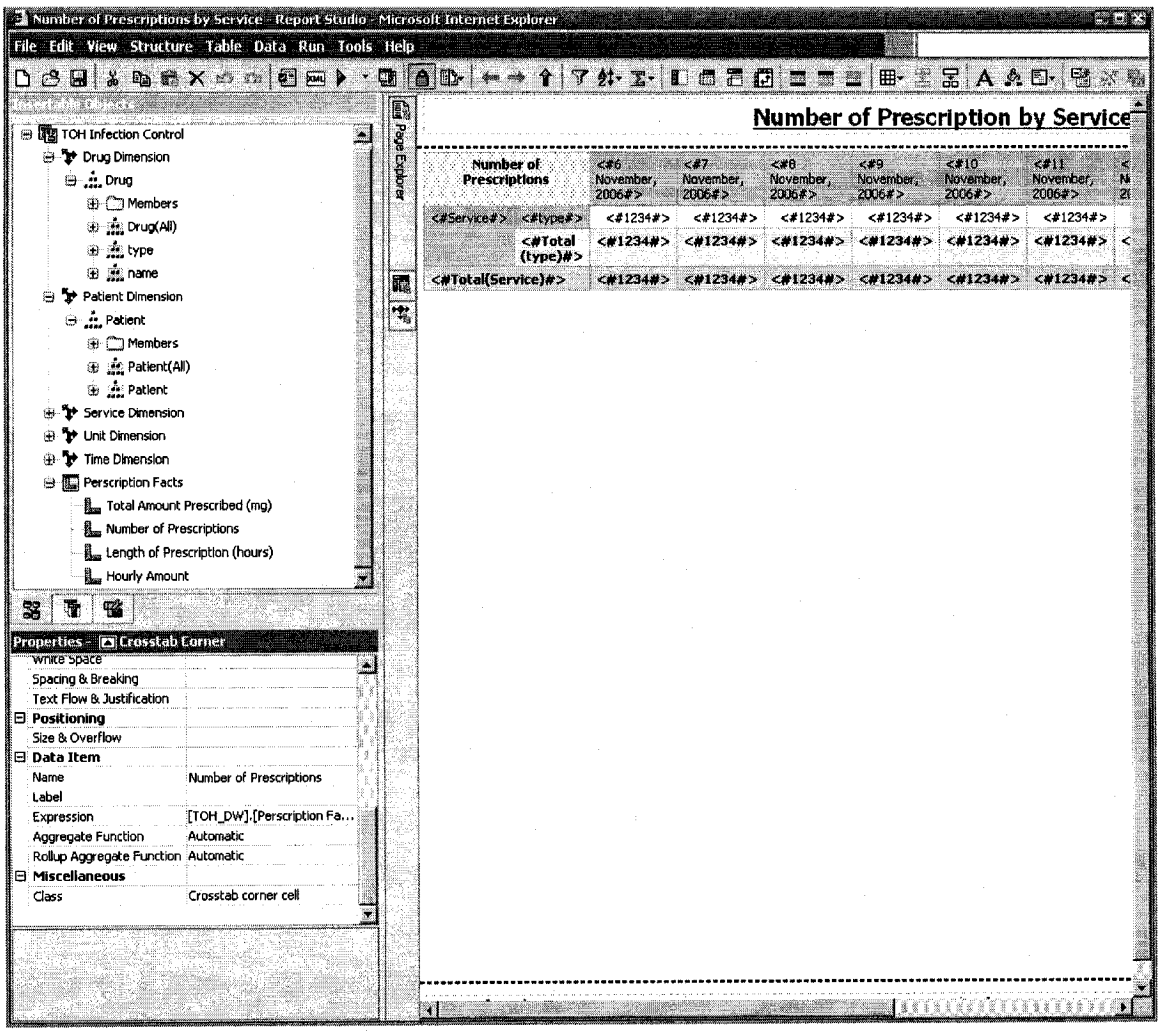


Figure 29 Report "Number of Prescription by Service" in design

Appendix 4: Tele-Health Data Generation and Sample Report

Our tool executed the following specification to generate test data for the Tele-Health process according to data requirements. It used a small real data set in several flat files to generate 1000000 rows across four years with a specified trend on the fact "cost".

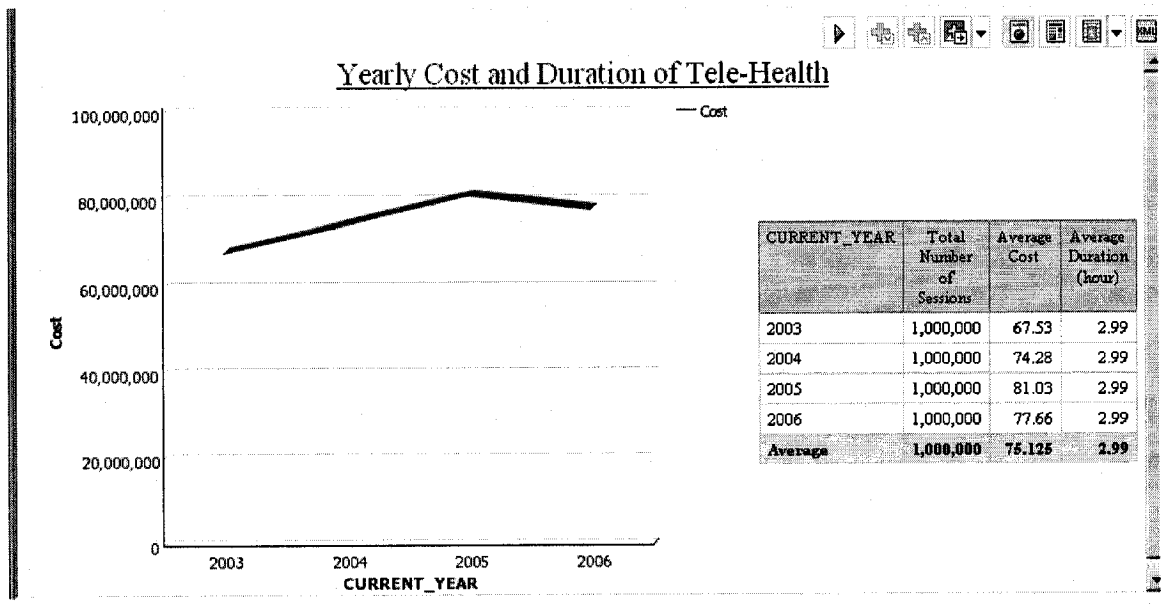
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE project SYSTEM "C:\workspace\ve\G5\model\model.dtd">
<project name="telehealth">
        <source>
                <databases>
                        <database id="d_amrdb" name="AMRDB_LARGE" type="SQL2005"
                        host="id1g" port="1433" user="sa"/>
                </databases>
                <files>
                        <file id="f_physicians" path="D:\temp\data\tbl Physicians.csv"
                                header="true" delim="," qualifier="&amp;quot;"/>
                        <file id="f_services" path="D:\temp\data\tbl Services.csv"/>
                        <file id="f_hospitals" path="D:\temp\data\tbl Hospitals.csv"
                                header="true"/>
                        <file id="f_session" path="D:\temp\data\tbl Session Types.csv"/>
                        <file id="f_contancts" path="D:\temp\data\tbl referring Contacts.csv"/>
                        <file id="f_code" path="D:\temp\data\tbl Code.csv" header="true"/>
                </files>
        </source>
        <dimensions>
                <dimension id="dim_physician" ref="d_amrdb.Tbl ConsultingPhysicians" out
                        put="c:\temp\Tbl Consult-ingPhysicians.csv">
                        <column name="ID" by="sequence"/>
                        <!--get data from a look up table-->
                        <column name="Physician Name" by="lookup"
                                lookup="f_physicians.Physician Name"/>
                        <column name="Service" by="lookup" lookup="f_services.column1"/>
                        <column name="Site" by="lookup" lookup="f_hospitals.Hospital"/>
                </dimension>
        </dimensions>
        <facttables>
```

```
            <facttable id="fac_data" ref="d_armdb.tbl Data" start="01/01/2003 00:00:01"
                    end="12/31/2003 23:59:59" rows="1000000" append="false" out
                    put="c:\temp\tbl Data.csv">
                    <column name="SessionID" by="sequence"/>
                    <!--get data from a dimension table -->
                    <column name="Service" by="dim" dim="dim_physician.Service"/>
                    <column name="Session Type" by="lookup"
                            lookup="f_session.column1"/>
                    <column name="Number of Attnewendees Consulting" by="random"
                            min="0" max="110"/>
                    <column name="Date" by="time"/>
                    <column name="Duration" by="random" min="0" max="360"/>
                    <column name="Consulting Contact" by="dim"
                            dim="dim_physician.Physician Name"/>
                    <column name="Consulting Site" by="dim"
                            dim="dim_physician.Site"/>
                    <column name="Referring Site" by="dim" dim="dim_physician.Site"/>
                    <column name="Cost" by="random" min="0" max="135" distribu
                            tion="normal"/>
                    <column name="Code" by="lookup" lookup="f_code.Code"/>
                    <column name="Referring Contact" by="lookup"
                            lookup="f_contancts.column1"/>
                    <column name="Number of Attnewendees Referring" by="random"
                            min="0" max="110"/>
            </facttable>
    </facttables>
    <expands>
            <!--expand the fac_data to have 1000000 new records in year 2004
            and the aggregation of cost will be 10 percent more than the one
            in year 2003-->
            <extable id="fac_data" start="01/01/2003 00:00:01" end="12/31/2003 23:59:59"
                    newstart="01/01/2004 00:00:01" newend="12/31/2004 23:59:59">
                    <excolumn name="Cost" trend="110"/>
            </extable>
            <!--expand the fact_data  to have 1000000 new records in
            year 2005 and the aggregation of cost will be 20 percent more
            than the one in year 2003-->
            <extable id="fac_data" start="01/01/2003 00:00:01" end="12/31/2003 23:59:59"
                    newstart="01/01/2005 00:00:01" newend="12/31/2005 23:59:59">
                    <excolumn name="Cost" trend="120"/>
            </extable>
            <!--expand the fac_data to have 1000000 new records in year 2006
            and the aggregation of cost will be 15 percent more than the one
            in year 2003-->
            <extable id="fac_data" start="01/01/2003 00:00:01" end="12/31/2003 23:59:59"
                    newstart="01/01/2006 00:00:01" newend="12/31/2006 23:59:59">
                    <excolumn name="Cost" trend="115"/>
            </extable>
    </expands>
</project>
```

In this specification, columns in several files are treated as lookup tables in generating a dimension table and a fact table. The fact "Cost" is defined to be generated in a stardard normal distribution in a range of value. Paticularly, the fact table with test data in year 2003 generated in the "facttables" section is furthre expanded in "expands" section to generate test data for year 2004, 2005 and 2006 with a trend applied on the aggregation of the fact "Cost". For example, the aggregation of the "Cost" in 2004 would be 10 percent more than the one in year 2003.

A Sample Report in Tele-Health Data Mart

Besed on the test data generated, reports are created to show the performance of the Tele Health process. The following screenshot is the running result of the report "Yearly Cost and Duration of Tele-Health", from the curve, we can see the trend of the total cost across four years.



CURRENT_YEAR	Total Number of Sessions	Average Cost	Average Duration (hour)
2003	1,000,000	67.53	2.99
2004	1,000,000	74.28	2.99
2005	1,000,000	81.03	2.99
2006	1,000,000	77.66	2.99
Average	**1,000,000**	**75.125**	**2.99**

Figure 30 Report "Yearly Cost and Duration of Tele-Session"

Appendix 5: Discharge Process Data Generation and Sample Report

---

Our tool executed the following specification to generate test data for the discharge process according to data requirements. There is no trend required but the fact table has special charateristics on time-related facts.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE project SYSTEM "C:\\workspace\ve\G5\model\model.dtd">
<project name="Discharge Data Mart">
 <source>
  <databases>
   <database id="d_toh_dw" name="toh_dw" type="SQL2005" host="id1g" port="1433"
       user="sa"/>
  </databases>
  <files>
  </files>
 </source>
 <dimensions>
  <!-- dimension table Unit, Physician is availabe in the database, no need to generate-->
 </dimensions>
 <facttables>
  <facttable id="fac_discharge" ref="d_toh_dw.discharge" start="11/01/2005 00:00:01"
       end="11/30/2005 23:59:59" rows="8000" append="false"  out-
       put="c:\temp\data\discharge.csv" >
   <column name="DischargID" by="sequence" />
   <column name="PatientID" by="random" min="1" max="1000"/>
   <column name="UnitID" by="lookup" lookup="d_toh_dw.unit.UnitID"/>
   <column name="PhysicianID" by="lookup" lookup="d_toh_dw.physician.PhysicianID"/>
   <column name="TimeToDischarge" by="time" />
   <!--TimeToDictation is 0 to 15 days after TimeToDischarge-->
   <column name="TimeToDictation" by="timedepend" depend="TimeToDischarge" min="0"
       max="15" unit="Calendar.DATE"/>
   <!--TimeToDictation is 0 to 10 days after TimeToDictation-->
   <column name="TimeToTranscription" by="timedepend" depend="TimeToDictation"
       min="0" max="10" unit="Calendar.DATE"/>
  </facttable>
  <!--500 more special records with empty time in TimeToDischarge,TimeToDictation,Time
```

```
ToTranscription-->
<facttable id="fac_discharge" ref="d_toh_dw.discharge" start="01/01/1900 00:00:00"
      end="01/01/1900 00:00:00"  rows="500" append="true"
      output="c:\temp\data\discharge.csv" >
 <column name="DischargID" by="sequence" />
 <column name="PatientID" by="random" min="1" max="1000"/>
 <column name="UnitID" by="lookup" lookup="d_toh_dw.unit.UnitID"/>
 <column name="PhysicianID" by="lookup" lookup="d_toh_dw.physician.PhysicianID"/>
 <column name="TimeToDischarge" by="time" />
 <column name="TimeToDictation" by="time" />
 <column name="TimeToTranscription" by="time"/>
</facttable>
<!--1500 more special records.-->
<facttable id="fac_discharge" ref="d_toh_dw.discharge" start="11/01/2005 00:00:01"
      end="11/30/2005 23:59:59"  rows="1500" append="true"  out-
      put="c:\temp\data\discharge.csv" >
 <column name="DischargID" by="sequence" />
 <column name="PatientID" by="random" min="1" max="1000"/>
 <column name="UnitID" by="lookup" lookup="d_toh_dw.unit.UnitID"/>
 <column name="PhysicianID" by="lookup" lookup="d_toh_dw.physician.PhysicianID"/>
 <column name="TimeToDischarge" by="time" />
 <!--TimeToDictation is  1 to 3 months after TimeToDischarge-->
 <column name="TimeToDictation" by="timedepend" depend="TimeToDischarge" min="1"
      max="3" unit="Calendar.MONTH"/>
 <column name="TimeToTranscription" by="timedepend" depend="TimeToDictation"
      min="0" max="10" unit="Calendar.DATE"/>
</facttable>
</facttables>
<expands>
 <!--no expand-->
</expands>
</project>
```

Similar to the specification of infection control data generation in Appendix 3, in this specification, dimension tables exist in the data warehouse, and primary key columns of dimension tables are used as lookup tables in generating a fact table. 8000 normal records will be generated first in the fact table; 500 more records will be generated with "01/01/1900 00:00:00" in all time-type facts; 1500 more records will be generated with different relations between time-type facts.

A Sample Report in Discharge Data Mart

Based on test data generated for the Discharge data mart, the following report was designed to show the average lag time from discharge to dictation in the discharge process.

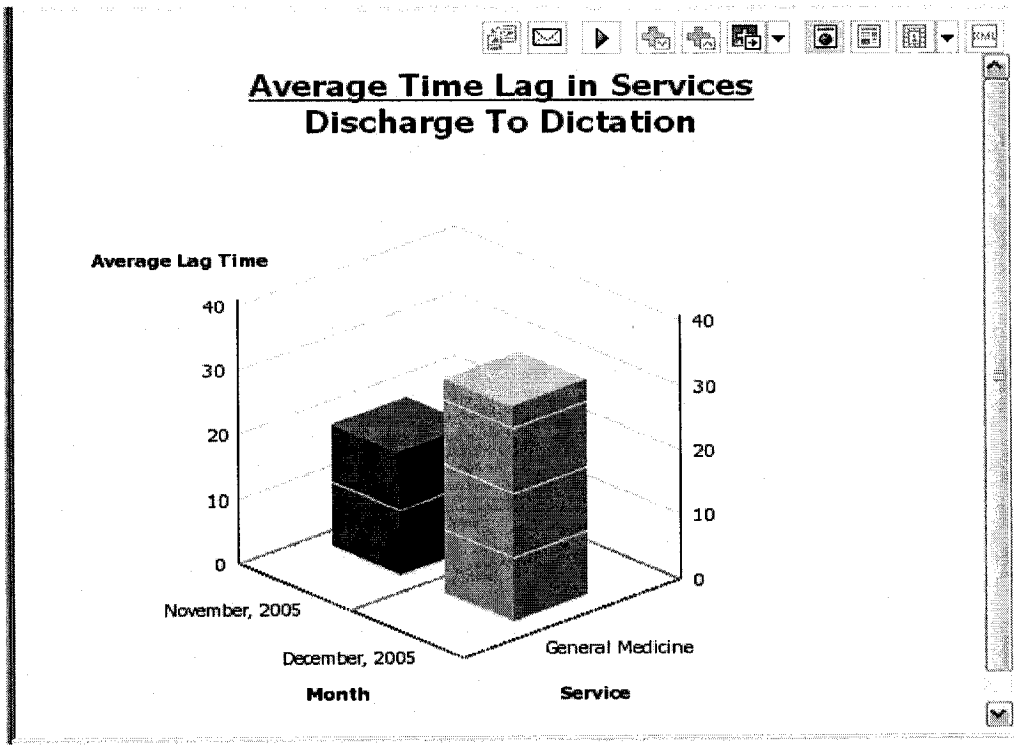


Figure 31 Report "Average Time Lag in Services- Discharge To Dictation"