A Software Method for Managing Event Logs to Improve Quality and Dependability of Business Processes

MEHMET S. UNLUTURK         KAAN KURTEL
Department of Software Engineering
Izmir University of Economics
Sakarya Cad. No.156 Balcova 35330 Izmir
TURKEY
suleyman.unluturk@ieu.edu.tr http://www.ieu.edu.tr

Abstract: - In this paper, we propose a logging service on .NET environment that utilizes the business process management domain to improve business fault tolerance, fault avoidance and alert managers to undesirable effects. The purpose of logging business processes states or exceptions are a repository for actions, events, and communications for a better understanding of the business mechanism. A practical software method called diagnostics logging control (DLC) has been implemented for this business mechanism. Furthermore, a browser-based viewer has also been developed to present the diagnostics information of the overall integrated business system.

Key-Words: - Business process logging, event log, diagnostics logging control, diagnostics log viewer.

1   Introduction
In the 1990s, organizations started to move from function- to process-oriented operational structures. The goal of this transformation was to increase customer and market orientation, as well as the flexibility of enterprises.

Business processes are defined as “a sequence of activities that carries out a complete business goal” [1]. Based on this definition, a business goal is essential to understanding how companies operate at an organizational or inter-organizational level. Business goals also play an important role in the design and the realization of flexible information systems. These information systems provide the technical basis for the rapid creation of new functionality for new products and adapting existing functionality to cater to new marketing requirements [2]. The Internet has influenced and transformed the primary activities of business. The distinct characteristics of the Internet age influence the dynamics of the following: the increasing complexity of business systems, the fact that they are highly sensitive, and the difficulty in meeting stakeholder’s expectations. Moreover, the business environment is now more heterogeneous than ever and consists of increasingly more diverse elements.

These dynamics are considered to create the high-performance business systems, which often need to be error-prone, reliable, robust and fault-tolerant applications in order to ensure business quality and dependability, therefore, require better monitoring processes by business executives to enable the appropriate decisions to be made. For these reasons, companies need to measure a variety of tasks for creating value, improving the customer satisfaction, and competitive advantage. In this context, modern business operations need to consider the necessity of logging and tracing of the processes for better managerial decisions, and visualization of possible re-engineering facilities and quality of services.

Business event management and logging are a subject that has gained a lot of research attention recently. A good discussion on function-to process-oriented governance is found in [1] [2] and [3] (although these works treat quality management models, strategy and organizational issues). There are a few published studies on the details of building event management and process based infrastructure that already dominated the primary information technology companies. Microsoft, IBM and SUN introduce various Internet-related services and give supportive background information commonly for business and software developers. Event based and logging content model were presented in [4] [5] [6] and [7]. Luckham in [8] defines a set of technologies to process large amounts of events, utilizing them to monitor, steer and optimize the business in real time. Erl in [9] identifies the principles of service-oriented architecture (SOA) and design standards.

In this paper, we propose a logging service that utilizes a business process management domain to improve business fault tolerance and fault avoidance, furthermore to warn managers of undesirable situations. These are essential to prevent future economic loss and to develop re-engineering strategies, and also the most important way of creating sustainable competitive advantage. This paper’s main contribution describes the design of a practical logging control (DLC) methodology.
as a .NET component. Designing DLC is logical in the sense that it is developed and debugged only once and then can be used multiple times within the business system. In this architecture, a browser-based viewer (DLV) has also been developed to present filterable event history and statistical view of the overall integrated system. Furthermore, performance of the business system can be examined through these statistics.

The second section explains the architecture for the logging service proposed in this paper. The third section explains how to assess a software application developed for a hospital information systems (HIS) using DLC. The fourth section presents the requirements specification we wish to develop and the fifth section presents diagnostics logging control design and code examples. The final section contains the conclusion and looks at possible future work.

2 The Architecture
Traditionally, engineers use a set of activities, methods, practices, and tools to measure product’s quality. Similarly, in an industrial product, a business process could be measured using the same basic engineering methods that include the following three major steps: First, planning the process and stating the principles, second collecting data, and finally analyzing data. Thus, process measurement to improve direct, control, and coordinate to process or work performed to develop a product or perform a service. Likewise, the responsibility of a process management is defined as being to gather “relevant runtime information, comparing metrics to historical or target values and disseminating the information to interested parties in a timely fashion” [10].

As quality management models and existing practices show, business process management collects and analyses information about an ongoing event processes to during operational activities. The basic idea behind a business-level event logging is to improve the business processes in a more systematic and automated fashion [6]. For this reason, event logging is an effective methodology that provides relevant information for business process management. The aim of logging is:
1. Visualization of what each process is doing when
2. Measuring and estimating the process performance
3. Determining the causes of problems and preventing possible risks
4. Indicating the location of a problem
5. Labeling and classifying system problems
6. Investigating a particular sequence of events
7. Storing historical records of business commitment
8. Controlling confidential or privileged processes
9. Logging forensic records

A log is defined as records of the special events that have occurred within a system or network and records generally related to computer security within an organization [11]. This paper offers two types of logging data contexts:

- Logging business process states (BPS). For example, records of personnel, locations, times, and methods of hospital activities.
- Logging exceptions, such as run-time errors.

In this study, a robust logging-viewer design methodology has been developed to improve the reliability and consistency of a business system. Figure 1 presents an integrated architecture of a logging system.

![Architecture for logging services](image)

**Fig. 1:** Architecture for logging services.

1. The initial stage is to define a target of logging. This stage is generally related to general business policy. Furthermore, business analysts should decide which key process factors are needed to develop and extend the quality of business for better customer services.

2. The second stage is determining business process states and potential exceptional considerations. In this stage, business analysts and software architect define conceptual design cooperatively to ensure multiple states and events, and that all information and services controlled by the business policy. In effect, business analysts and software architects determine which events or exceptions to log.

3. The third stage is adding try/catch blocks in the .NET application that calls Diagnostics Logging Control (DLC) component. This component, which has been designed and developed as a .NET component, is logical in the sense that it is developed and debugged only once and is then used multiple times. Software architects finalize a physical design while wrapping their code with a DLC component for each state and for events that are determined by the second stage. Thus, each .NET client application in the business system has its own individual DLC to trace its own events. In this stage, there could be n .NET clients. These .NET applications report their own errors and states to their own DLCs, which then calculate their statistics and send them to a single underlying database server.
The first three stages are also related to conceptual business modeling that represents a series of relationships between business analysts and architects and are recommended as a means to a collaborative model in SOA projects [9].

4. DLCs store their statistics in a central database, facilitates information analysis [12].

5. The last stage involves information presentation. A browser-based .NET viewer is also developed to display data collection by multiple DLCs and displays the statistics to the user for further analysis. Called Diagnostics Log Viewer (DLV), this service, which is an ASP.NET application, is used to display the statistics of the overall system for the end users. By looking at the DLV, one can see how a system is performing over time. This browser-based viewer does not only show filterable event history, but also a statistical view of the overall integrated system. Furthermore, performance of the business system can be examined through these statistics generated by each DLC.

3 The Software Assessment

The best way to assess software robustness and performance while running is to employ a logging mechanism where states and errors of software are traced. A log entry in this logging system is simply a formatted message that contains important information that can be used during analysis. A well-formed log entry includes the following information:

- **Logging Level**: Logging is broken into five levels [13] with the most critical events logged at Level-4, and fewer critical events logged at levels 3, 2, 1 and 0. Table 1 contains a list of logging levels for logging business processes and run-time exceptions.

<table>
<thead>
<tr>
<th>Level</th>
<th>Logging business process states and run-time exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level-0</td>
<td>Events are heartbeat messages that are sent to target at regular intervals.</td>
</tr>
<tr>
<td>Level-1</td>
<td>Events consist of non-critical communication messages such as warning messages.</td>
</tr>
<tr>
<td>Level-2</td>
<td>Events represent recoverable errors, which mean the system recovers from errors without user involvement.</td>
</tr>
<tr>
<td>Level-3</td>
<td>Events hold critical events, which require user attention.</td>
</tr>
<tr>
<td>Level-4</td>
<td>Events contain link failures and recoveries.</td>
</tr>
</tbody>
</table>

- **Statistic Name**: Each logged message at each logging level is recorded with a statistic name. These names are used in generation of statistics for each application, for example, FAIL and RECOVER are the statistic names for level-4 events. These statistics indicate when the application fails and recovers successfully.

- **Message**: The message section of the log entry holds the actual message. This message is either the error or the state of the application. These verbal messages help developers to debug applications in debug mode and to investigate performance in release mode.

Based on the description of the behavior of a log entry, the question now arises of how to capture the actual behavior of a software application, i.e., on how to specify the trace of states and state transitions that the software has passed through. In DLC, both logging levels and statistic names are used to order the logged entries and accurately represent the essence of software inner workings.

The basic idea of using logging entries for representing software states is simple: each state transition of a software application is represented by a log entry. The log entries are inserted into DLC which simultaneously calculates the statistics. When the DLC runs for the first time, it takes the current time as the statistic start-up and link-up. At the end of each minute, as long as the software application is functioning, DLC adds one minute to the link-up time. Hence, \( \text{Tot_Run_Time} \) is calculated as the time difference between current time and statistic start time. The total run time statistic value helps the designer to determine the length of time the DLC has been running.

\[
\text{Tot_Run_Time} = \text{Current Time} - \text{Statistic Start Time}
\]  

\( \text{Up_Time} \) is the indicator that shows the period the application has been functioning as percentages.

\[
\text{Up_Time} = \frac{\text{Linkup Time}}{\text{Tot_Run_Time}}
\]  

**Mean Time to Recover** shows the average failure recovery time in days, hours, minutes and seconds. The lower its value, the better the business system is. It is calculated as

\[
\text{Mean Time To Recover} = \frac{\text{Current Time} - \text{Linkup Time}}{\text{# of Recoveries}}
\]

**Mean Time to Failure** indicates the average length of time between failures. It is given as

\[
\text{Mean Time To Failure} = \frac{\text{Linkup Time} - \text{Statistic Start Time}}{\text{# of Failures}}
\]

**Mean Time to Error** shows the average length of time between errors and is given as

\[
\text{Mean Time To Error} = \frac{\text{Linkup Time} - \text{Statistic Start Time}}{\text{# of Errors}}
\]

These metrics are displayed within the browser based application called Diagnostics Log Viewer (DLV). Each DLC calculates and stores these statistics in the database. The diagnostics log viewer reads and displays the statistics from the database on a screen (see Figure 2-4).
Figure 2 shows Level 4 messages, which are link failure messages and there are 5 tabs, each representing a different level of log messages.

In Figure 3, DLC logged 5 levels of events for 4 days, 10 hours, 33 minutes and 16 seconds (given as Total Time). 1.173% percent of total time, software application controlling the DLC was functioning. Mean time to failure was 9 minutes and 22 seconds. When a failure occurred, mean time to recovery was 13 hours, 9 minutes and 52 seconds. There were a total of 303 errors within the time frame and mean time to error equaled 14 seconds. Of these, good events were given as 99.431 %. This was calculated as:

$$\text{Good Events} = \frac{\text{Total Events} - \# \text{ of Failures} - \# \text{ of Errors}}{\text{Total Events}}$$  \hspace{1cm} (6)

In this step, we present an event-logging example from the hospital information system software application. In the software, DLC was utilized to log nurse location information as Level-1 messages (Figure 4). Examining the location event highlighted in bold rectangle (log# 59), it can be seen that a nurse with ID 120 was in room 2002, a patient room. The nurse arrived at this room at 4:52:57 AM and was last seen in the hall at 4:52:53 AM. Level-1 message is a detailed record of the nurse’s name (from identity), current task (from room type), room location, and period of occupation. In this hospital system, DLC is not only used to log link failures as Level-4 messages (Figure 2) but also used to trace the nurses within the hospital through Level-1 messages. Therefore, DLC-DLV combination can be utilized in all .NET software applications that require logging errors, and tracing mobile personnel.

### 4 Requirements Specification

The requirements specification aims at restricting the logging system and defining what functionality the logging system should offer. This document is a contract between the developer and the customer of the system and therefore, forms the developer’s view of what customer wants. DLC component details are given in Table 2. The other functional and non-functional requirements are given below:

1. **Log Instance Capacity:** Multiple logs may be kept for each .NET application. Design should be ready to handle up to 100 logs.

2. **Event Capacity:** To maintain responsiveness, events are broken into 5 levels, with the most critical events logged at level 4, and fewer critical events logged at levels 3, 2, 1, or 0. At any given level, for any given log, only 100 most recent events need to be stored. So background cleanup tasks can remove extraneous events.

3. **Event Handling Capacity:** The system must be able to log at an average rate of 10 events per second, with a peak of 20 events per second every 10th second. This should be maintainable for 10 simultaneous logs.

4. **User Interface Response Time:** User interface responses should be within 10 seconds for a screen repaint, and within 20 seconds for initial load.

5. **Reliability and Fault Tolerance:** As with any third party interface application, a critical feature of the software is to provide easily readable interactions between the software to facilitate compatibility testing, problem resolution, and to instill customer satisfaction in the reliability of the installed software, the local dealer, and the systems involved. Clear and accurate logs and reliability statistics are critical to this goal.
6. Database Unavailable: The product must contain tracking and display of link reliability, including event history of link failure and recovery, errors, and missed messages. Counts of each, including a means of reset must be available.

7. Application Failure: All badge locations can be requested, so that locations can be quickly recovered after a failure of the Locator system, the interface, or the link between.

8. Statutory and Regulatory Requirements [14][15]:
   - HIPAA: Service logs may show messages with patient information. Therefore, applications that log these types of messages should only make the logs available to logins with service permission. The restriction is not a part of the logging service or viewer itself.
   - UL1069: PC based applications do not perform fundamental nurse call functions, so do not fall under the jurisdiction of UL1069.

   Table 2: DLC Component Detail.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Up time</td>
<td>See Equation #2</td>
</tr>
<tr>
<td>- Failure Count</td>
<td># of FAIL logged events since DLC is started</td>
</tr>
<tr>
<td></td>
<td>• FAIL events logged only on transition from Active</td>
</tr>
<tr>
<td></td>
<td>• Default to Failed status upon install (No FAIL event logged)</td>
</tr>
<tr>
<td></td>
<td>Save AppUp timestamp every minute: Upon .NET Application boot up, find AppUp, and log FAIL using its timestamp. “FAIL: Application restarted sometime within this minute”</td>
</tr>
<tr>
<td>- Error Count</td>
<td>Anything with “Error”, “error” or “ERROR” and level 3</td>
</tr>
<tr>
<td>- Event Count</td>
<td>Anything Level 1-4</td>
</tr>
<tr>
<td>- Error Details</td>
<td>Count of each Error, by type</td>
</tr>
<tr>
<td></td>
<td>Add types dynamically as they are logged</td>
</tr>
<tr>
<td>- Unusual Events</td>
<td>Count of each level 2 event, by statistic field:</td>
</tr>
<tr>
<td></td>
<td>Add types dynamically as they are logged</td>
</tr>
<tr>
<td>- Total Run Time</td>
<td>See Equation #1</td>
</tr>
<tr>
<td>- MTTF</td>
<td>See Equation #4</td>
</tr>
<tr>
<td>- MTTR</td>
<td>See Equation #3</td>
</tr>
<tr>
<td>- MTTE</td>
<td>See Equation #5</td>
</tr>
<tr>
<td>Events Grid on DLV</td>
<td>• Shows each event in chronological order</td>
</tr>
<tr>
<td></td>
<td>• Most recent event at the top</td>
</tr>
<tr>
<td></td>
<td>• Auto-scrolls to the up upon any refresh or launch</td>
</tr>
<tr>
<td></td>
<td>• Auto-selects top event on any refresh or launch</td>
</tr>
<tr>
<td></td>
<td>• 100 of each level</td>
</tr>
<tr>
<td>Details label on DLV</td>
<td>Shows the summary text plus the logged detail field</td>
</tr>
</tbody>
</table>

This requirements specification contains a total functional specification of the logging system we wish to develop for .NET applications, without any attention to the implementation environment. In the construction phase, which is the next section, we will talk about the DLC’s both implementation language and database management system.

6 Design of DLC

The design model for DLC is given in Figure 5. This model enables us to reduce the complexity of the logging system. In our model, developer logs business process events or exceptions via the Add Log Message interface (see Figure 5). Add Log Message will tell the active object Log Message Receiver that a Log Event Item has been submitted. Log Message Receiver will use the entity objects Level-0, Level-1, Level-2, Level-3 and Level-4 together with the level value received by Add Log Message interface, to find out what type of log event item has been handed in. The current statistics of the received object of this type will be calculated inside the active object called Statistics Calculator. Using the entity object Log Message Base, Log Message Receiver will keep track of how many events of the current type the developer has logged in so far. Statistics and log events are sent to Database Writer so that they can be persisted and presented over the internet by DLV.

In this section, we carry out the design and implementation of the logging system. This logging system is based on the output model from the previous section (see Figure 5). The final product provides reliable and readable logging that is very critical in integrated systems. Developers utilize this logging in debugging problems during development, monitoring field reliability, and pinpointing which solutions can be applied when field problems arise.
7 Conclusion
Implementing a logging mechanism increases the reliability and the dependability of an integrated system. This logging mechanism provides information about what software components are doing, including the data processed, and about business process states or exceptions ran into while the .NET software applications are running. This information can be utilized in tracking the applications during a test procedure and in resolving where software exceptions are taking place in the integrated system. As the applications run, major components properly format and log details of their processes into the database for further analysis. Since several machines may be involved in logging, logging is very important in a complex client-server environment to find out the execution path between machines.

The logging mechanism should write enough information into the database for better statistical analysis. Within the enormous amount of information, it might get harder to isolate important entries. We implement several logging levels inside the log entries to solve this issue. At the present time, there are many logging tool sets available in the software market. However, they are incapable of calculating statistical data of the overall integrated system for further analysis.

In this study, we have developed a novel software tool set that is designed to trace the performance of software applications. One of the major components in this methodology is the diagnostics log controller (DLC), designed as a .NET component in such a way that it can be used multiple times within integrated business systems. Errors and events within each .NET software application are documented by each DLC that resides in these software applications. Each DLC independently calculates and stores its own statistics in the database. Another major component in this tool set is the diagnostic log viewer (DLV). It has been developed as a web-based application to display these statistics on any networked PC. By examining the statistics, a user can monitor and evaluate the performance of any .NET applications in production mode. Furthermore, developers can use such a reporting system in debug mode to catch logical or run-time errors within these .NET applications.

Managing an event based business logging system is a long-term effort. This study focuses exclusively on the logging service part. There are several interesting possibilities for further research with using DLC and DLV methods, and we are particularly interested in investigating process mining related concepts concerning Hospital Information Systems.

References: