Propuesta para la optimización del monitoreo de sistemas informáticos complejos mediante el empleo de tecnología digital del habla.

Rafael Alexándero Urdaneta[1]

Resumen

Hoy en día la creciente complejidad de los sistemas informáticos empleados tanto en la industria como en el ámbito de la medicina y aún en ámbitos cotidianos, requieren a menudo de otros sistemas para monitorear y controlar su correcto funcionamiento. Estos sistemas de monitoreo y control a su vez, requieren de cierto nivel de análisis e interpretación de los indicadores suministrados para posteriormente tomar las decisiones correspondientes al contexto presentado. Este análisis demanda de un proceso cognoscitivo humano, basado en conocimientos adquiridos previamente mediante la experiencia o por un flujo de decisiones predefinido mediante un manual de operaciones en el común de los casos.

Los sistemas de habla y de reconocimiento de voz tienden a ser altamente efectivos al reducir los tiempos de análisis y permitir sintetizar e interpretar de manera precisa una situación de emergencia o de alta criticidad.

En este trabajo se desarrolla una propuesta de arquitectura que presenta un mecanismo de monitoreo de sistemas informáticos industriales con un intérprete de voz que advierte sobre situaciones críticas predefinidas utilizando herramientas informáticas de licencias de uso gratuitas.

Palabras Clave: Sistemas complejos, arquitectura SOA, Ingeniería del Software,

Fecha de recepción: julio 2013 | Fecha de aceptación: octubre 2013

[1] Universidad de Palermo, Facultad de Ingeniería.

software industrial.

Abstract

Today the increasing complexity of the computer systems used in industry, medicine field and even in everyday life often requires other systems to monitor and control its correct functioning. These monitoring and control systems in turn, require some analysis and interpretation of the indicators provided to later make decisions for the context presented. This analysis needs a human cognitive process based on knowledge previously acquired by experience or a predefined decision flow, generally through an operation manual.

Speech and voice recognition systems tend to be highly effective in reducing analysis time and allowing accurately summarizing and interpreting an emergency or high criticality.

This paper develops a proposed architecture that provides a mechanism to monitor industrial computer systems with voice interpreter that warns of predefined critical situations using free-license tools.

Keywords: Complex systems, SOA architecture, software engineering, industrial software.

1. Introducción

Cada día la tecnología incorpora nuevas herramientas a nuestra sociedad produciendo, en consecuencia, cambios en la forma de pensar. Esta evolución se refleja en una manera distinta de modelar los procesos. Desde los inicios de la electrónica y los primeros pasos de la informática, existe una carrera por el desarrollo de mecanismos tecnológicos que permitan realizar una mayor cantidad de actividades de manera automatizada, con mayor seguridad y menor costo. Pero no sólo estos factores han sido suficientes para considerar que el aporte o la innovación sean positivos o beneficiosos; otros factores entran en juego como lo son la velocidad de implementación, alcance, flexibilidad, licenciamiento.

El presente trabajo estudia y propone, una herramienta informática, basada en tecnología WEB cliente servidor, mediante la cual se podrá visualizar y notificar en tiempo real el estado de salud general de un sistema informático empresarial basado en una arquitectura SOA(Service Oriented Architecture). Con la información brindada por la herramienta, se podrían tomar acciones correctivas y preventivas para el funcionamiento estable del sistema. Este proyecto fue aprobado para desarrollarse e implementarse en la empresa Techint Argentina y México. Actualmente se encuentra en funcionamiento en una versión beta, en los ambientes de desarrollo, QAS (Quality Assurance Software) y Producción.

Debido al volumen de información que se maneja a diario (12000 mensajes por minuto por planta en ambiente productivo), se ha hecho necesario el desarrollo de sistemas más robustos; gracias a los avances tecnológicos, se puede contar con hardware con mayor capacidad, velocidad de procesamiento y mayor capacidad de almacenamiento de datos; pero a la vez, se hacen cada vez más complejos los procesos de desarrollo y mantenimiento. Esta complejidad se alimenta de elementos como: servidores de aplicaciones, bases de datos, librerías, componentes, entornos de trabajo (frameworks), control de versionado; y factores como: la distribución del sistema, las capas que conforman la arquitectura donde será implementado, los niveles de seguridad, entre otros.

Se han realizado diversos estudios experimentales sobre los problemas asociados con el uso de los entornos informáticos complejos de uso general que tienen un gran número de funcionalidades, tales como los sistemas operativos y, a menor escala, los paquetes integrados (ej. hojas de cálculo, procesadores de texto). Como lo indica Mehlenbacher B. (1992), estos estudios empíricos (que habitualmente han estudiado, entre otros dominios, el sistema operativo Unix) demuestran que los usuarios tienen un conocimiento incompleto del sistema, utilizando de forma habitual sólo una pequeña parte de las utilidades de las que dispone. El hecho de que muchos usuarios utilicen la computadora únicamente como una herramienta que facilita la realización de sus tareas hace que no suelan desear aprender nada que no

les resulte imprescindible para conseguir sus objetivos. Los usuarios conocen tan sólo un conjunto de operaciones básicas, con las que son capaces de realizar sus tareas habituales de forma satisfactoria, pero de manera que hay partes del sistema que les resultan completamente desconocidas. Esto impide que aprendan otras utilidades más sofisticadas u otras formas de trabajo que les permitirían conseguir un mejor uso de la aplicación.

Si a este sistema informático, se le agrega un entorno empresarial, donde se interrelacionan diversos subsistemas aplicativos, sistemas operativos, distribución geográfica y arquitectura orientada a servicios (SOA), se estará en presencia de un sistema informático complejo y por ende se hace necesario el uso de herramientas adecuadas de control y monitoreo de su estado de salud.

La implementación de un sistema como el que aquí se describe, podría brindar información oportuna y significativa a distintas áreas relacionadas al desarrollo y mantenimiento de sistemas informáticos complejos en tiempo real como los descritos por Silverman L. (2006), desde cualquier ubicación geográfica, sin necesidad de distribución e instalación de paquetes ni pago de licencias; utilizando un punto centralizado de acceso directamente desde un explorador WEB. Como señala Peters, D.K. Parnas, la combinación de herramientas informáticas basadas en tecnología WEB como AJAX, DWR, Java, Spring, entre otros, permitirán redefinir los esquemas de diseño, desarrollo y distribución de las aplicaciones de monitoreo de sistemas existentes en el mercado, como el System Center Operations Manager de Microsoft o el Enterprise Manager de Oracle, convirtiéndose en nuevas tendencias que abren caminos de posibilidades que antes estaban fuera de nuestro alcance.

Arquitectura propuesta

Las siguientes herramientas serán utilizadas en la propuesta:

• Java como lenguaje core del sistema: este es el lenguaje base del proyecto, los dístintos módulos y la interconexión de los frameworks que intervienen están orquestados desde java. El sistema por completo está compilado en un EAR de java, que contiene 2 módulos principales: el model y el controller (JAR) y un módulo web: la vista (WAR).

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

• Javascript como lenguaje intermedio entre backend y frontend: mediante este lenguaje, se realizan las negociaciones entre el controller y la vista.

• Spring como Framework de apoyo: inyecta funcionalidades de scheduler mediante quartz y facilita la injección de dependencias para la inicialización de los componentes java (beans)

http://www.springsource.org/spring-community-download

• Oracle como gestor de Base de datos: esta es la encargada de almacenar los datos que contiene la información de estados e indicadores a monitorear (OLTP de la empresa)

http://www.oracle.com/technetwork/products/expressedition/downloads/index.html

• Tibco GI como interfaz de cliente: Este framework nos permite de manera simple la creación de los componentes de frontEnd web para el cliente.

• DWR como framework para empleo de ajax: Mediante este, implementamos un canal de comunicación directo entre el backend y el frontend, permitiendo actualizar los indicadores en la capa cliente sin necesidad de refrescar manualmente el browser. Será el encargado también de crear las clases proxies java en tempo ejecución haciendo transparente la transición de la capa del core java a la capa web.

http://directwebremoting.org/dwr/index.html

• Web Speech API como interpretador de sonidos de habla: mediante esta API, obtenemos el reconocimiento de voz para generar las acciones que se desean en base a reglas preestabelecidas.

https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

• jbeep.js como librería de soporte para emisión de sonidos: Esta librería nos permite de manera genérica, reproducir los sonidos deseados en el browser cliente con un nivel de calidad aceptable.

https://code.google.com/p/jbeep/

El proyecto será desarrollado siguiendo el patrón de diseño MVC (Model View Controller) y distribuido en tres capas básicas como lo indica la figura 1:

Fig. 1 Capas de Diseño.

Los datos correspondientes a los cambios dentro del negocio son almacenados en la base de datos mediante componentes observadores (Patrón Observer). Estos cambios son relevados por el sistema de monitoreo que utilizará el mecanismo de quartz de Spring Framework para realizarlo periódicamente. El tiempo de periodicidad deberá ser parametrizado considerando aspectos como la frecuencia

promedio con la que ocurren los cambios y la criticidad de los mismos. En la medida que los cambios son más frecuentes y la criticidad de los elementos son mayores entonces la frecuencia deberá ser mayor.

En la capa de negocio se encontrarán los factorys de los objetos monitoreados y las reglas que determinen los cambios de estado y las criticidades correspondientes a dichos estados.

Una vez determinados los cambios y las criticidades se procederá con la publicación asincrónica hacia la capa de presentación de los cambios suscitados. Estos pueden ser implementados mediante texto plano y/o indicadores visuales en forma de led's u otros recursos gráficos disponibles dentro de la tecnología empleada.

La transmisión de la información hacia la capa de cliente se realizará mediante ajax asincrónico(ver fig. 2), el cual permite establecer un canal de comunicación directo desde la capa de negocio enviando los datos en segundo plano y actualizando los estados en el cliente de manera automática sin necesidad de actualizar (refrescar) el browser.

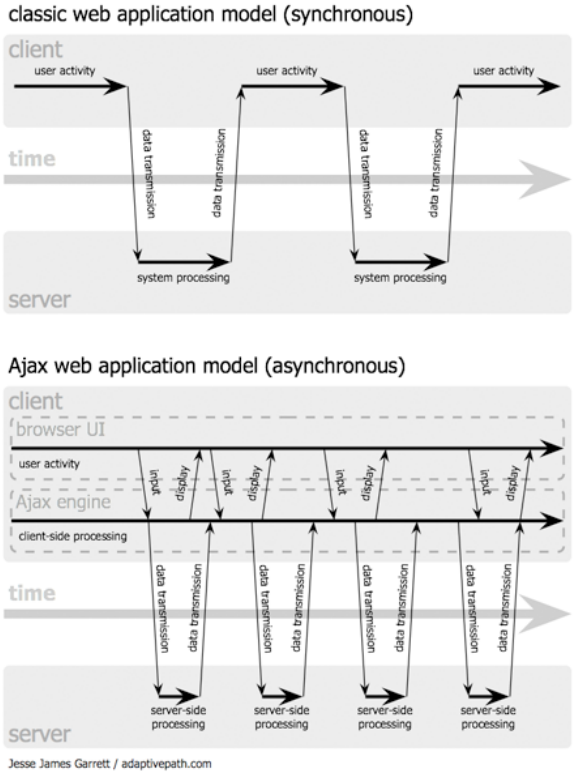

Fig. 2 Diagrama de Transmisión Ajax.

Es en esta etapa de la transmisión de datos se pueden implementar distintas estrategias para la notificación de los eventos. Estas estrategias incluyen:

notificaciones emergentes, cambios visuales mediante gráficos y sonidos emergentes como se muestra en la Fig. 3.

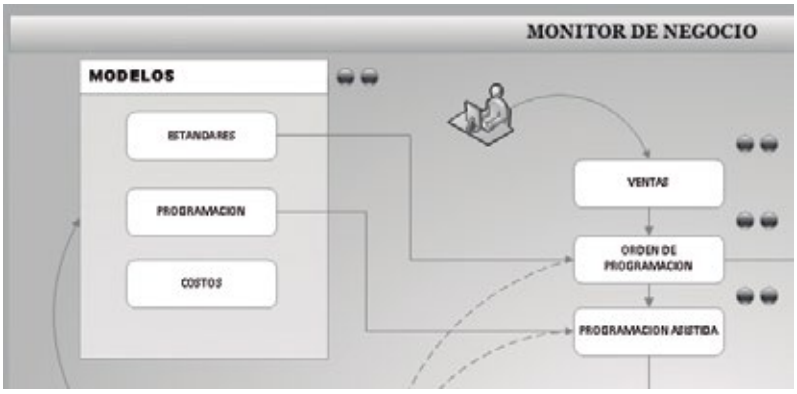

Fig. 3 Monitor de Negocio.

Entre la capa de negocio y la capa de presentación se dispone del espacio ideal para implementar estas estrategias. En cuanto a los cambios de estado se utilizarán funciones javascript para determinar el estado actual del elemento en la capa cliente, comparar y enviar el nuevo estado como se muestra en el ejemplo de la Fig. 4:

```
notificar.js
1
2  if (proveedor[property].indexOf(estadoAnterior) == -1)
3                          notificar = "true";
4
5                      ox.style.backgroundColor = ox.style.backgroundColor;
6                      if ((proveedor[property] == 'UP')
7                          && (ox.innerHTML != '<IMG SRC="JSX/images/icons/16_verde.jpg" >'))
```

Fig. 4 Ejemplo código Javascript para comparar elemento Capa-Cliente.

Para la estrategia de notificación emergente se empleará jquery notification plugin (http://caolanmcmahon.com/files/jquery.notify.js/examples/index.html)

En la Fig. 5 se ejemplifica como podrá ser invocado una vez determinado el cambio de estado y la criticidad del evento en particular.

```
notificar.js
notificar.js
1  function notificar (criticidad, evento, titulo, icono){
2
3      var fecha = new Date();
4      var fechaActual = fecha.getFullYear() + '-' + (fecha.getMonth() + 1) + '-' + fecha.getDate() + ' '
5          fecha.getHours() + ':' + fecha.getMinutes() + ':' + fecha.getSeconds();
6
7          $.notify({text:evento, title:titulo, icon:icono});
```

Fig. 5 Código de Javascript. Código de notificación

Como resultado se obtendrá la notificación emergente como elemento auxiliar y visual para el alerta del sistema de monitoreo.

Fig. 6 Notificaciones emergentes

La tercera estrategia a utilizar y que indefectiblemente puede ser utilizada en paralelo a las otras 2 anteriores (indicadores visuales y notificaciones emergentes), es la notificación auditiva. El objetivo de esta propuesta es demostrar que la implementación de esta estrategia puede convertirse en una herramienta valiosa dada su efectiva manera de llegar al consumidor final, que en este caso está representado por el analista de operaciones o un referente funcional de centro de cómputo.

Para esta implementación previamente se deberán definir los distintos sonidos que se deseen emitir, recordando prudentemente calcular los tiempos de espera y las criticidades de cada caso. Para ello se definirán 3 niveles de criticidad:

Información: para los cambios de estados que indiquen una mejoría o un cambio de estado esperado.

Warning: en los casos donde amerite la atención del operador sin que necesariamente requiera una acción determinada correctiva o de mantenimiento.

Error: como el nivel de criticidad más alto. Representará el tipo de alerta más importante. Deberá ser activado ante la presencia de un cambio de estado que amerite la pronta participación del operador y/o la inmediata activación de algún plan operativo para volver el componente a su estado normal de desempeño.

El segundo elemento sonoro a almacenar deberá ser tipo de componente observado. Este podrá ser definido por su naturaleza (dispositivo electrónico, componente de software, módulo del sistema), o incluso por su función (conector de interface, adaptador, web services, etc.).

El tercer elemento será el nombre del componente. Siendo este de mucha importancia dado que permitirá al escucha ubicarse inmediatamente dentro de un contexto acotado de elementos involucrados. El nombre del componente dependerá de la estructura del sistema monitoreado y de los elementos que lo incluyen; debiendo ser un identificador que sea representativo y unívoco.

El cuarto elemento será el estado en particular: “activo”, “inactivo”, “suspendido”, etc. Este elemento es fundamental para determinar la criticidad del mensaje, dado que dependiendo del estado final del componente monitoreado se podrá definir el nivel de criticidad de la alerta. Por ejemplo: Un conector del sistema industrial que pasa de un estado “suspendido” a un estado “Activo” debería representar una alerta del tipo “Información” dado su nuevo estado. Estos criterios serán relativos a la realidad funcional del componente que se defina.

En este punto se podrá conformar morfológicamente una oración mediante la combinación de: Tipo Alerta + Tipo Componente + Nombre Componente + Nuevo Estado como lo indica la siguiente ilustración:

notificar.js

```
notificar.js
1  if (notificar == "true") {
2                       var criticidad = getCriticidad(tipoComponente,proveedor[property]);
3                       notify(tipoComponente, aplicacion, nodo, estadoAnterior, proveedor[property], criticidad);
4                   }
5
6                   var callback = function(ele) {
7                       return function() {
8                           ele.style.backgroundColor = ele.style.backgroundColor;
9                       };
```

Fig. 7 Código Javascript. Definición de criticidad

La función para emitir los sonidos que representan la oración deseada, podrá ser armada dinámicamente y reutilizada dependiendo de la implementación que se desee realizar. Así pues, determinados los elementos, se invocan secuencialmente en el orden morfológico:

notificar.js

```
1   setTimeout(function(){emitirSonidoTipoAlerta(archivoSonidoTipoAlerta)},1);
2       setTimeout(function(){emitirSonidoComponente(archivoSonidoComponente)},4000);
3       setTimeout(function(){emitirSonidoAplicacion(archivoSonidoAplicacion)},6000);
4       setTimeout(function(){emitirSonidoEstadoNuevo(archivoSonidoEstadoNuevo)},7000);
5
6       notificar(iconoLog,evento,titulo,icono);
7
8   }
9
10
11  function emitirSonidoTipoAlerta(archivoSonido){
12      jBeep(archivoSonido);
13  }
```

Fig. 8 Código Javascript. Emisión de sonido de alerta.

Conclusiones y trabajo a futuro:

Este proyecto aunque aún se encuentra en versión beta, ha logrado transformarse en una herramienta que forma parte de las aplicaciones frecuentes y activas de todos los que de alguna manera tienen parte de responsabilidad dentro de la dirección de sistemas de la empresa.

Por la arquitectura propuesta, se obtiene un producto flexible y escalable. Se facilita la incorporación de nuevas funcionalidades con bajo costo de recursos tecnológicos y con corto tiempo de desarrollo.

El modelo ha demostrado ventajas en función a:

- Disminución del tiempo de interpretación de alertas mediante la sintetización auditiva de los eventos.

- Disminución del tiempo de minutos a segundos, para emitir diagnósticos e informar el estado de un componente del sistema o del estado general del mismo.

- Flexibilización de la incorporación de nuevos componentes al Sistema de Monitoreo, con un tiempo aproximado de 2 horas.

- El tiempo de distribución del sistema es inmediato, solo ingresar vía WEB. Anteriormente era necesario la instalación de una herramienta en el PC del cliente.

- La disposición de información en forma gráfica y en tiempo real. Esto facilita la síntesis de la situación actual del estado de salud del Sistema.

Se proyecta la incorporación de funcionalidades adicionales para cubrir necesidades de monitoreo de procesos de migración y carga inicial (MCI) de los nuevos sistemas que se implementarán en Monterrey, abriendo la posibilidad de incorporar un módulo de visualización para la MCI de las diferentes plantas en los diferentes ambientes.

Bibliografía

Apache Software DWR, Reverse Ajax: http://directwebremoting.org/dwr/documentation/reverse-ajax/index.html
Dojo Foundation (Julio 2009). General Interface: Developer Guide: http://www.generalinterface.org/docs/display/GIDOCS/General+Interface+Developer+Guide

Duffy, T.M., Palmer, J., Mehlenbacher B., (1992). OnLine Help Design and Evaluation. Ablex Publishing Corporation. Norwood, New Jersey, pág. 107.

Fuentes, Juan Mariano. Manual de AJAX Basado en "AJAX, Fundamentos y Habitaciones". 2da Edición (2009) : http://www.elrincondeajax.com/manual-ajax/

Peters, D.K. Parnas, D.L. Fac. of Eng. & Appl. Sci., Memorial Univ. of Newfoundland, St. John's, Nfld. Requirements-based monitors for real-time systems: http://ieeexplore.ieee.org/search/srchabstract.jsp?tp=&arnumber=988496

Silverman, L.Facility Robotics (2006), Inc., Roswell, GA. Real time Data Center Intelligent Power Alarm Monitoring using the Web: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4076021