Projektdokumentation

tr@nsnet
internet business solutions

transnet internet services GmbH

Programmierung eines Systems zur schnellen
Benachrichtigung der Mitarbeiter über auftretende Ereignisse

- Projektdefinition
- Projektplanung
- Erarbeiten von Lösungsansätzen
- Umsetzung
- Ergebnis
- Detailbeschreibung aller Scripte

Autor: Florian Schießl

Prüflingsnummer: 155 20696
Ident-Nummer: 155 1685231

Datum: 15. März 2013

Inhaltsverzeichnis

1 Projektdefinition

Eine schnelle und effektive Benachrichtigung der Mitarbeiter über neue Arbeitsaufträge (Tickets) ist für kürzere Reaktionszeiten unerlässlich. Die Mitarbeiter sollen das neue Ticket sofort bemerken und darauf reagieren können. Um dies zu bewerkstelligen, soll ein visueller Alarm in Form einer USB LED Tisch Ampel eingeschaltet werden, sowie ein Alarm ertönen. Hierfür wird eine Software benötigt, welche überprüft, ob neue Tickets vorhanden sind und falls ja, den Alarm aktiviert. Andere Ereignisse können auch den Alarm auslösen. Beispiele hierfür wären der Ausfall des Internets, sowie ein Problem, welches von unserem Überwachungssystem (Nagios) gemeldet wird. Nagios ist ein System, welches mehrere Server und deren Dienste überwacht. Fällt ein Dienst oder Server aus, so wird dies auf der Weboberfläche von Nagios angezeigt.

Abb. 1 – LED Tisch Ampel

1.1 Projektumfeld

Es handelt sich hierbei um ein firmeneigenes Projekt, welches für das Büro unserer Firma bestimmt ist und auch in diesem durchgeführt wurde. Ein Einsatz der erstellten Software in dem jeweiligen Homeoffice unserer Mitarbeiter wäre möglich, wird aber derzeit noch nicht angewandt.

1.2 Projektziel

Es soll eine leicht konfigurier- und wartbare Software erstellt werden, welche den oben genannten Ansprüchen gerecht wird.

2 Projektplanung

2.1 Ist-Zustand

Aktuell existieren zwei Systeme: Das Ticketsystem und Nagios.

Im Ticketsystem werden alle Arbeitsaufträge gesammelt, bearbeitet und gespeichert. Es unterscheidet zwischen mehreren Ticketzuständen. Für dieses Projekt ist jedoch nur der Zustand 'new' relevant, da alle eingehenden Tickets automatisch diesen Zustand erhalten. Die Mitarbeiter bemerken erst durch das Erscheinen eines neuen Tickets in der Kategorie 'new', dass ein neuer Arbeitsauftrag eingegangen ist. Da jedoch das Ticketsystem nur im Browser verfügbar ist, der Auto-Refresh sich auf eine Minute beläuft und der Tab im Browser meistens aufgrund anderer Tätigkeiten nicht im Vordergrund des Mitarbeiters ist, kann es oft mehrere Minuten dauern, bis ein neuer Auftrag wahrgenommen wird.

Bei Nagios ist das vorherige Problem nicht so stark ausgeprägt, da bei jedem Mitarbeiter ein Browser-Plugin installiert ist, welches anzeigt, ob ein Problem vorhanden ist. Allerdings kann auch dieses übersehen werden, z. B. wenn der Browser minimiert oder geschlossen ist.

Der Internetzugang ist für jedes IT-Unternehmen unerlässlich. Aktuell wird der Internetausfall zwar recht schnell bemerkt, um jedoch die Downtime zu reduzieren, ist auch hier eine sofortige Benachrichtigung nötig.

2.2 Soll-Analyse

Wie im vorherigem Punkt beschrieben, können unterschiedliche Arten von Alarmen auftreten. Die Ampel soll für jeden Alarm ein individuelles Signal zeigen. So soll sie zum Beispiel für ein neues Ticket gelb blinken. Zusätzlich soll zu bestimmten Alarmen ein Ton abgespielt werden, um diese noch besser bemerkbar zu machen. Ein Alarm soll erneut ausgelöst werden, wenn zum Beispiel im Ticketsystem ein weiteres neues Ticket vorhanden ist. In diesem Fall soll der akustische Alarm erneut ausgelöst und das Blinken der Ampel aufrecht erhalten werden.

3 Erarbeiten von Lösungsansätzen

3.1 Ansteuern der Ampel

Bei der Ampel handelt es sich um eine über USB mit dem Computer verbundene Ampel mit drei Farben (Rot, Gelb und Grün). Die Farben können vollständig unabhängig voneinander geschaltet werden. Um für jeden Alarm ein eindeutiges Signal zu zeigen, ergeben sich mehrere Möglichkeiten, wie z. B. das Kombinieren von Farben, abwechselndes Blinken von Farben bis zu kompletten Abläufen von nacheinander aufleuchtenden Farben.
Wir haben uns jedoch dafür entschieden, jedem Alarm jeweils eine Farbe, eine Blinkfrequenz und eine Priorität zuzuordnen. Liegen mehrere Alarme gleichzeitig vor, die die selbe Farbe verwenden, so wird der Alarm mit der höchsten Priorität angezeigt. Die Alarme mit einer niedrigeren Priorität werden hierbei nicht gelöscht, sondern bleiben im Hintergrund erhalten und werden wieder angezeigt, sobald der Alarm mit der jeweils höheren Priorität erloschen ist.

Diese Vorgehensweise bietet mehrere Vorteile:

- Die Ampel kann auf einen Blick über mehrere Alarme gleichzeitig informieren.
- Die Mitarbeiter müssen sich keine komplizierten Farbkombinationen oder Abläufe merken, um zu erkennen, um welchen Alarm es sich handelt.
- Die wichtigsten Alarme werden zuerst erkannt.

3.2 Akustischer Alarm

Bestimmte Alarme sind wichtiger als andere, deshalb wurde beschlossen, dass für dringende Alarme auch ein akustischer Alarm ertönen soll. Der akustische Alarm wird durch das einfache Abspielen einer Audiodatei bewerkstelligt.

3.3 Mögliche Herangehensweisen

Es standen zwei mögliche Software-Architekturen zur Diskussion.

3.3.1 Alles in einem Script

Diese Option wurde zunächst aufgrund ihrer Einfachheit bevorzugt. Sie hätte bedeutet, dass sowohl das Ansteuern der Lampe, als auch die einzelnen Checks in einem Script abgelaufen wären.
Die Vorteile dieser Option wären:

- Die einzelnen Teile des Scriptes hätten das Signal der Ampel an die anderen Alarme anpassen können. So würde zum Beispiel ein Alarm wie das Wegfallen des Ticketsystems nicht gemeldet, falls der Internet-Check ergeben hätte, dass das Internet ausgefallen ist.
- Es wäre keine Abstimmung der einzelnen Konfigurationen notwendig, wie z. B. das Bekanntmachen eines einzelnes Alarms und die Netzwerkkonfiguration.

Die Nachteile wären:

- Das System wäre nicht konfigurierbar und deswegen auch schwer anpassbar.
- Das Script wäre unübersichtlich geworden, und deswegen nicht mehr leicht zu warten und erweiterbar.
- Checks, die auf einem anderen Server ausgeführt werden müssen, wären nicht möglich.

3.3.2 Server / Client Modell

Diese Option bedeutet, dass es einen Server gibt, welcher die Ampel und die akustischen Alarme verwaltet. Für jeden Check wird ein einzelnes Script erstellt, welches die Client-Rolle übernimmt. Die Clients melden einen Alarm über das Netzwerk an den Server, welcher dann die Ampel ansteuert und den akustischen Alarm entsprechend seiner Konfiguration auslöst.
Die Vorteile dieser Option wären:

- Die Checks wären nicht auf einen Server gebunden.
- Neue Checks wären auf einfache Weise erweiterbar. Es könnte einfach ein neues Script erstellt werden.
- Die Checks wären nicht voneinander abhängig.
- Das Script wäre leicht konfigurierbar und deswegen anpassbar auf neue Bedürfnisse.

Die Nachteile wären:

- Es bestünde einmalig mehr Aufwand bei der Programmierung
- Die Konfiguration würde relativ umfangreich ausfallen.
- Es kann nicht zu 100% garantiert werden, dass ein Alarm den Server wirklich erreicht.

Wir haben uns letztendlich für das Server / Client Modell entschieden, da es unseren Anforderungen am besten entspricht.

4 Umsetzung

Im Nachfolgenden werden die einzelnen Teilprogramme der Software beschrieben, die in diesem Projekt erstellt wurden. Das Zusammenspiel der einzelnen Programme ist in dieser Grafik beschrieben. In der Grafik ist der Nagios- und Internet-Check nicht gezeigt, da das Prinzip das Selbe wie beim Ticket-Check ist.

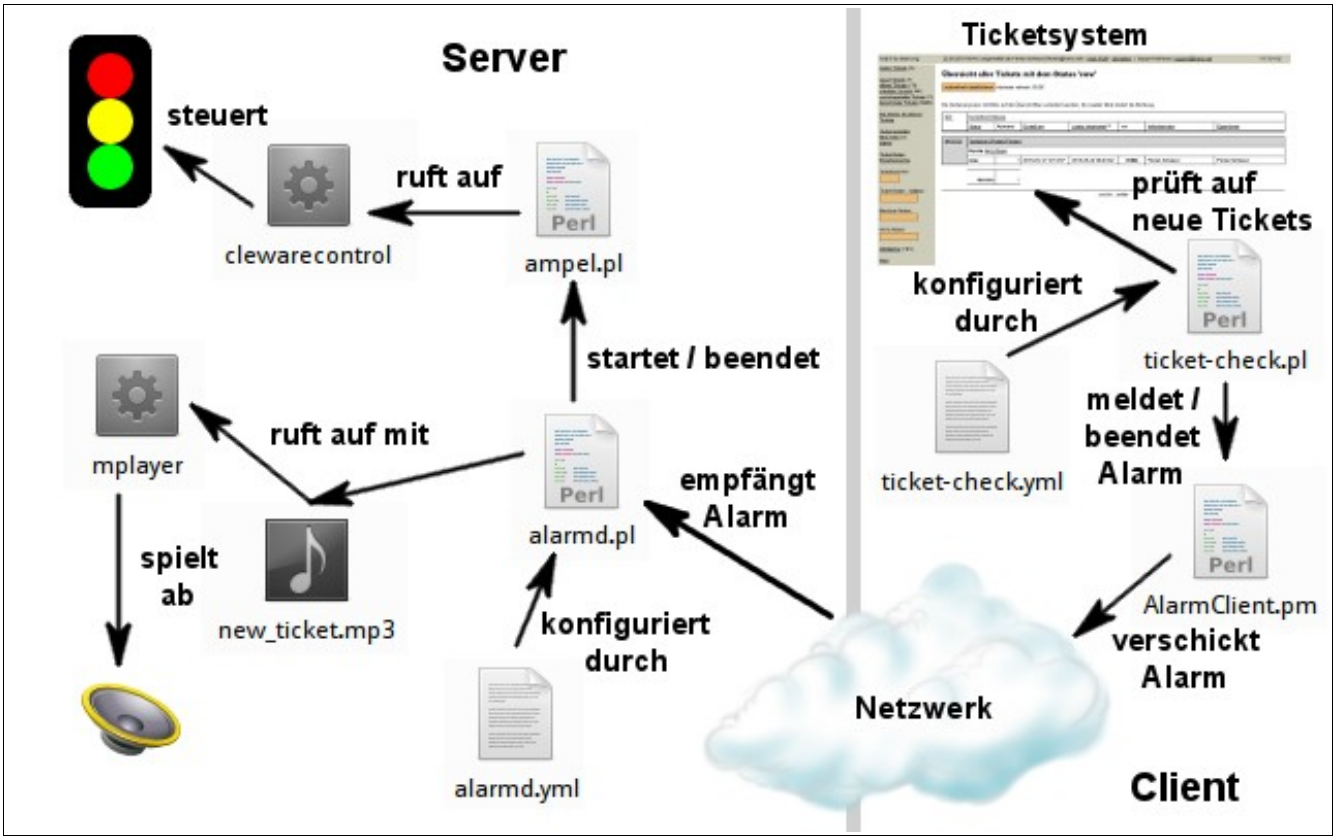

Abb. 2 – Ablauf im Client / Server Modell

4.1 Ampel-Steuerung (ampel.pl)

Die Ansteuerung der Ampel erfolgt über das von Folkert van Heusden geschriebene Programm 'clewarecontrol'. Dieses unterliegt der GNU General Public License und ist somit Open Source.
Dieses Programm kann von der Website http://www.vanheusden.com/clewarecontrol/ heruntergeladen werden. Nach dem Herunterladen muss es zunächst noch kompiliert werden. Anschließend steht auf dem System der Befehl 'clewarecontrol' zur Verfügung. Das Script ampel.pl wurde darauf ausgelegt, mithilfe des Programms 'clewarecontrol' eine einzelne Farbe der Ampel so lange einzuschalten oder blinken zu lassen, bis es beendet wird. Diese Funktion wird von dem Server (alarmd.pl) genutzt. Dieses Script muss als Administrator gestartet werden, da das Programm 'clewarecontrol' root-Rechte benötigt, um auf die USB-Ampel zugreifen zu können.

4.2 Alarm-Server (alarmd.pl)

Der Alarm-Server ist das Herzstück der Software. Er lauscht auf einem konfigurierten Netzwerk-Port. Die Clients verbinden sich mit dem Server und übermitteln den Namen eines Alarms und ob dieser gerade auftritt oder bereits vorbei ist. Der Server erwartet hierfür außerdem ein Passwort.
Die Konfiguration wird über eine Konfigurationsdatei im YAML-Format vorgenommen. Die Wahl fiel auf YAML, da es leicht verständlich und lesbar ist. Es kann außerdem die Informationen in dem benötigten Zusammenhang abbilden. Bei der Konfiguration der anderen Teilprogramme wurde ebenfalls YAML verwendet.
Für die Kommunikation über das Netzwerk wurde das Perl Modul 'IO::Socket' verwendet, da dieses bereits in anderen Projekten zum Einsatz gekommen ist und sich dort bewährt hat.
Wenn ein Client einen Alarm meldet, wird anhand der Konfiguration entschieden, wie die Ampel leuchten bzw. blinken soll und der akustische Alarm abgespielt, sofern dieser konfiguriert wurde. Meldet ein Client, dass ein Alarm vorbei ist, so wird das entsprechende Blinken bzw. Leuchten der Ampel eingestellt. Ist kein Alarm mehr vorhanden, leuchtet lediglich die grüne Lampe.

4.3 Alarm-Client (AlarmClient.pm)

Der Alarm-Client bildet die Schnittstelle zwischen Server und Check-Script. Es handelt sich hierbei um ein Perl Modul, welches von den einzelnen Check-Scripten eingebunden wird. Es abstrahiert die Schnittstelle zu dem Server. Ein Client muss nur eine Funktion des Moduls aufrufen um einen Alarm auslösen oder aufheben zu können. Dadurch wird die Entwicklung weiterer Check-Scripte in Zukunft enorm vereinfacht.

4.4 Nagios-Check (nagios-check.pl)

Der Nagios-Check überprüft die 'Service Detail'-Webseite unseres Nagios auf Hosts mit dem Status 'DOWN' oder 'UNREACHABLE', und Services mit dem Status 'CRITICAL', 'UNKNOWN' oder 'WARNING'.
Nagios selbst verfügt über keine API. Es gibt zwar ein Projekt, welches versucht, eine API für Nagios nachzurüsten, allerdings kam dieses hierfür nicht in Frage, da am Nagios-Server möglichst wenig verändert werden sollte.
Für das Abrufen der Website wird das Perl-Modul 'WWW::Mechanize' verwendet. Ursprünglich war angedacht, das Modul 'LWP::UserAgent' zu verwenden, dieses schnitt allerdings bei einem von circa 20 Versuchen den unteren Teil der Seite ab, was zu einem Fehlalarm führte. Da 'WWW::Mechanize' kompatibel zu 'LWP::UserAgent' ist, musste an dem Script nicht viel verändert werden. Der Nagios-Check speichert die Anzahl der Services bzw. Hosts mit jeweiligem Status zwischen und löst den Alarm erneut aus, falls sich die Anzahl erhöhen sollte. Erst wenn keine Hosts oder Services mehr einen der oben genannten Status inne haben, wird der entsprechende Alarm beendet. Der Nagios-Check kann je nach Konfiguration auch mehrere Nagios-Instanzen überprüfen. Der Check wurde für Nagios 3.0.6 entwickelt und erfolgreich damit getestet.

4.5 Ticket-Check (ticket-check.pl)

Bei dem Ticket-Check handelt es sich um das Script, welches ein oder mehrere 'treE' Ticketsysteme auf neue Tickets überprüft und immer, wenn ein neues Ticket ankommt, einen Alarm an den Server sendet. Der Alarm wird aufgehoben, sobald kein neues Ticket mehr vorhanden ist. Da auch das Ticketsystem nicht über eine API verfügt, muss das Script sich bei der Weboberfläche anmelden und den HTML-Code der Seite für neue Tickets überprüfen. Dies geschieht ebenfalls über das Perl-Modul 'WWW::Mechanize', welches auch schon beim Nagios-Check verwendet wird. Für den Ticket-Check wurde im Ticketsystem ein neuer User angelegt.

Siehe Auch: treE - http://www.deam.org/tree/

4.6 Internet-Check (internet-check.pl)

Durch den Internet-Check werden die in der Konfiguration vorhandenen Hosts angepingt. Sollte keiner der Hosts mehr erreichbar sein, wird ein Alarm ausgelöst. Der Alarm wird aufgehoben, sobald ein Host wieder erreichbar ist. Zum Pingen wurde das Perl-Modul 'Net::Ping' verwendet. Es benötigt root-Rechte, um ICMP Pings abzusetzen, weshalb der Internet Check auch als Administrator ausgeführt werden muss.

4.7 Start-Script (startup.pl)

Um das komplette System auf einmal starten und beenden zu können, ist dieses Script erstellt worden. Es startet zunächst den Alarm-Server und danach alle anderen Check-Scripte. Sobald dieses Script beendet wird, werden automatisch alle anderen Scripte ebenfalls beendet.

4.8 Generierung der Audiodateien

Für den akustischen Alarm über die Lautsprecher waren Audiodateien notwendig, welche von dem Alarm-Server abgespielt werden können. Diese wurden mit dem Open Source Programm Audacity generiert. Es handelt sich dabei um einfache Tonabfolgen abwechselnder Frequenzen.

4.9 Übersicht Audiodateien

Nachfolgend eine Veranschaulichung der Audiodateien:

start.mp3

Frequenz (Hz)	Zeit (ms)		
1300			200
1100		200	
900	200		

stop.mp3

Frequenz (Hz)	Zeit (ms)		
1300	200		
1100		200	
900			200

error.mp3

Frequenz (Hz)	Zeit (ms)						
500	200		200		200		200
0		10		10		10	

new_ticket.mp3

Frequenz (Hz)	Zeit (ms)					
4000		200		200		200
3000	200		200		200	

offline.mp3

Frequenz (Hz)	Zeit (ms)	
400		200
200	200	

Abb. 3 – Übersicht der Audiodateien

5 Ergebnis

5.1 Aktuelle Konfiguration

Im Moment wird die Software auf einem extra dafür vorgesehen Computer betrieben. Der erste Mitarbeiter, der das Büro betritt schaltet diesen ein und der letzte, der das Büro verlässt, schaltet ihn wieder aus.

Die Software wurde wie folgt konfiguriert:

- Ticket-Check
  - Neues Ticket
    - Farbe: Gelb
    - Blinkfrequenz: 0,1 Sekunden an, 0,9 Sekunden aus
    - Priorität: 2
    - Neues Ticket: new_ticket.mp3
  - Ticketsystem nicht erreichbar
    - Farbe: Gelb
    - Frequenz: 2,4 Hz
    - Priorität: 3
- Internet-Check
  - Internet ausgefallen
    - Farbe: Grün
    - Frequenz: 2,4 Hz
    - Priorität: 3
    - Internet ausgefallen: offline.mp3
    - Internet wiederhergestellt: start.mp3

- Nagios-Check
  - Warning erkannt
    - Farbe: Gelb
    - Frequenz: leuchten
    - Priorität: 1
  - Unknown erkannt
    - Farbe: Gelb
    - Frequenz: leuchten
    - Priorität: 1
  - Critical erkannt
    - Farbe: Rot
    - Frequenz: leuchten
    - Priorität: 1
  - Host down erkannt
    - Farbe: Rot
    - Frequenz: 1,2 Hz
    - Priorität: 2
  - Nagios nicht erreichbar
    - Farbe: Rot
    - Frequenz: 2,4 Hz
    - Priorität: 3

5.2 Fazit

Die Software wurde erstellt und funktioniert gemäß den Anforderungen. Durch die Konfiguration mit separaten Dateien im YAML-Format wird eine einfache Konfigurierbarkeit gewährleistet. Die Entscheidung für das Server / Client Modell hat sich als richtig erwiesen, da durch das Auslagern des Ticket-Checks auf den Server mit dem Ticketsystem Bandbreite eingespart werden könnte. Selbiges gilt auch für den Nagios-Check. Durch das Vorhandensein eines Perl Moduls für die Kommunikation mit dem Server, gestaltet sich das Erweitern um neue Checks sehr einfach.
Ob sich die Reaktionszeit der Mitarbeiter verbessert hat, wird sich in den nächsten Wochen zeigen, es ist aber davon auszugehen.

6 Detailbeschreibung der Scripte

6.1 Plain Old Documentation

Die Detailbeschreibung der Scripte wurde mit der Auszeichnungssprache „Plain Old Documentation“ vorgenommen. Hierbei handelt es sich um die gängige Praxis, Perl Scripte zu dokumentieren. Diese wird meist unten an das eigentliche Script angehängt oder befindet sich gar im Code selbst, wodurch der Code sehr nah mit der Dokumentation verbunden wird. Dies bietet auch den Vorteil, dass beim Kopieren der Software die Dokumentation ebenfalls mitkopiert wird und deswegen immer direkt mit dem Script verfügbar ist.

Die Plain Old Documentation zu dieser Software kann, wie die Software selbst auch, über mein GitHub Repository eingesehen werden. Eine als PDF gerenderte Form der Plain Old Documentation ist in dem Unterordner 'doku' verfügbar.

GitHub Repository - https://github.com/iKoze/Abschlussprojekt
POD als PDF - https://github.com/iKoze/Abschlussprojekt/tree/master/Projekt/doku