Displaimer Teknisk Dokumentation: Schemaläggare

Teo Klestrup Röijezon <teo@nullable.se>

23 maj 2017

Sammanfattning

Schemaläggaren är komponenten i systemet som kommunicerar med displayenheten, och bestämmer vad som ska visas när. Detta dokument behandlar först det externa gränssnittet, och sedan implementationen.

Dokumentet är riktat mot en någorlunda teknisk person som antingen skriver en egen displayenhet, vill använda API:t för att schemalägga meddelanden, eller som ska underhålla schemaläggaren i framtiden. Dessutom är kapitlen om beroenden och uppstart relevanta för någon som ska underhålla servern i produktion.

Innehåll

1 Gränssnitt **2**
- 1.1 Beroenden . . . 2
- 1.2 Uppstart . . . 2
- 1.3 Mot Displayenheten . . . 2
- 1.4 Mot Systemet . . . 3
  - 1.4.1 Direkt Databasåtkomst . . . 3

2 Implementation **5**
- 2.1 Koncept . . . 5
  - 2.1.1 Prioritering av Meddelanden . . . 5
  - 2.1.2 Hantering av Upprepade Meddelanden . . . 5
  - 2.1.3 Automatisk Uppdatering . . . 5
- 2.2 Organisation . . . 5
  - 2.2.1 Models . . . 5
  - 2.2.2 Actors . . . 6
  - 2.2.3 Services . . . 6

3 Kända Brister **6**
3.1 Säkerhet 6
3.2 Översättning och Internationella Marknader 6
3.3 System med Flera Displayenheter 7

1 Gränssnitt

1.1 Beroenden

Schemaläggaren kräver att du har en Java 8 (eller nyare) JVM[1], och Sodium installerade. För att kunna kompilera behöver du också ha en JDK[2], SBT, och Node.js installerade. Andra beroenden tillfredställs automatiskt av SBT.

Du måste även skapa en tom PostgreSQL-databas vid namn ”wilcd”, som den nuvarande användaren har åtkomst till. Om databasen heter något annat, eller finns på en annan server, uppdatera ”wilcd-ui/conf/application.conf”.

1.2 Uppstart

Schemaläggaren är en komponent i webbapplikationen, och båda startas tillsammans. För att starta webbapplikationen i utvecklingsläget, kör följande kommando från mappen ”wilcd-ui”:

```
$ sbt run
```

För att skapa ett Debian-paket, kör följande:

```
$ sbt debian:packageBin
```

Notera att du måste öppna webbplatsen (`http://localhost:9000/`) för att starta schemaläggaren i utvecklingsläget, men det krävs inte för Debian-paketet.

1.3 Mot Displayenheten

Schemaläggaren lyssnar på TCP-porten 9797, och använder ett textbaserat format, kodat enligt ISO 8859-1. Varje rad är ett paket, där första tecknet är indikerar paketets typ, och resten är paketets innehåll. För närvarande är följande typer definierade:

[1] Java Virtual Machine
[2] Java Developer Kit

Typ	Innehåll
M	Ett nytt meddelande att visa på displayenheten
T	Den nuvarande tiden[3], används för att kalibrera displayenhetens klocka

All mottagen text ignoreras, och det är tillåtet att ansluta flera gånger. Alla meddelanden skickas till alla klienter.

1.4 Mot Systemet

Det rekommenderade gränssnittet är via schemaläggarens implementation av MessageUpdater, med följande viktiga metoder:

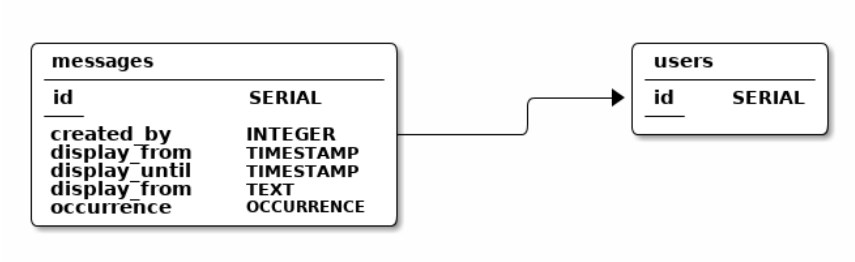
Metod	Beskrivning
scheduleMessage	Planerar att ett meddelande ska visas
deleteMessage	Raderar ett ntUML packages on your system.
getMessage	Hämta det aktuella meddelandet som visas
getNextMessage	Nästa meddelande som ska visas[4]

Om detta gränssnitt används så hålls schemaläggaren automatiskt uppdaterad.

Prioriteringen av meddelanden är implementationsdefinierad.

1.4.1 Direkt Databasåtkomst

Om större kontroll behövs så kan man skriva till databasen direkt. Databasen initieras automatiskt när schemaläggaren startas. Följande utdrag ur databasen används av schemaläggaren:



[3] I den lokala tidszonen, för närvarande alltid Europe/Stockholm.

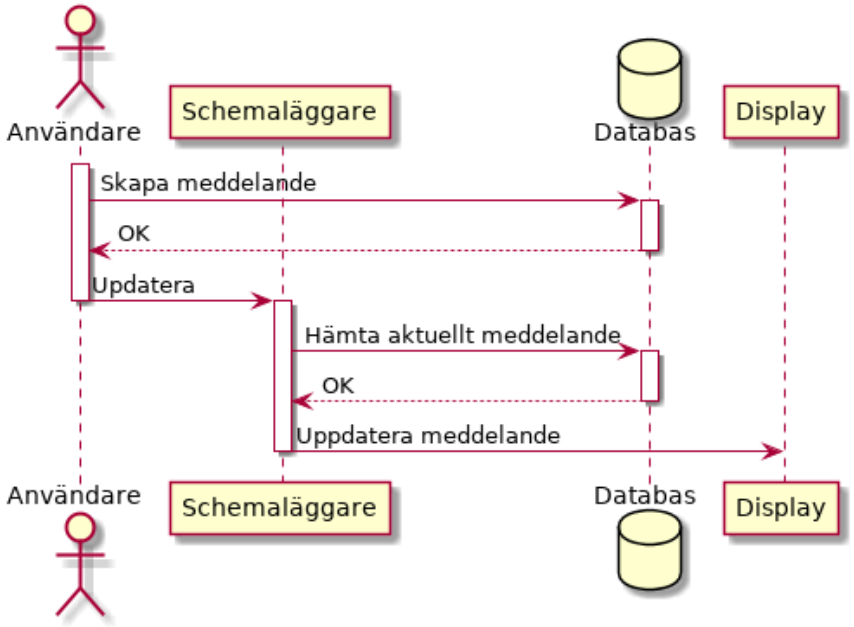
[4] Det finns ingen garanti för att detta meddelande någonsin kommer visas, använd bara detta för att gissa när displayen behöver uppdateras

Där OCCURRENCE är någon av Once, Daily, Weekly, Monthly, eller Yearly. Alla TIMESTAMPs lagras i UTC.

Efter att databasens innehåll ändrats så **måste** du skicka meddelandet ”DbMessageFetcher.Refresh” till instansen av DbMessageFetcher (injiceras av Guice som en ActorRef med namnet ”db-message-fetcher”). Se exempel nedan:

```
import actors.DbMessageFetcher
import akka.actor.ActorRef
import javax.inject.{Inject, Named}
class MyClass @Inject() (
  @Named("db-message-fetcher") dbMessageFetcher: ActorRef) {
  def doStuff(): Unit = {
    // Lek runt i databasen
    dbMessageFetcher ! DbMessageFetcher.Refresh
  }
}
```

Vid åtkomst från Scala rekommenderas det att återanvända tabelldefinitionerna i paketet ”models”, i stället för att skriva SQL-frågor för hand.



2 Implementation

2.1 Koncept

2.1.1 Prioritering av Meddelanden

Meddelanden prioriteras genom att ta alla meddelanden, filtrera bort alla som inte är aktuella (display_from har inte hänt än, eller display_until har redan hänt), och sedan väljs meddelandet med högst display_from.

2.1.2 Hantering av Upprepade Meddelanden

Innan meddelanden hämtas så gör vi först en sökning efter meddelanden där occurrence inte är Once, och där display_from redan har passerat. Dessa meddelanden kopieras framåt i tiden (display_from och display_until ökas båda med t.ex. 1 dag för Daily), och för det gamla meddelandet ändras sedan occurrence till Once. För att undvika att meddelanden dubbleras så låses dessa rader under operationen.

2.1.3 Automatisk Uppdatering

Schemaläggaren försöker att automatiskt uppdatera displayen när det aktuella meddelandet ändras. Detta sker genom att titta på det nuvarande meddelandets display_until, och nästa meddelandes (dvs meddelandet efter det nuvarande enligt 2.1.1) display_from, och schemalägga en Refresh för den tidigare av de två händelserna.

Därför måste en manuell Refresh skickas om nästa ändring planeras om till att ske tidigare. Det behövs tekniskt sett inte ifall ändringen i stället senareläggs, men det rekommenderas ändå att skicka meddelandet för att undvika buggar.

2.2 Organisation

Schemaläggarens kod ligger tillsammans med webbplatsen i paketen ”actors”, ”models”, och ”services”.

2.2.1 Models

Models innehåller den interna koden som kommunicerar med databasen.

2.2.2 Actors

Actors innehåller bakgrundstjänsterna.

DbMessageFetcher Väljer det aktuella meddelandet och skickar det vidare till TcpDisplayUpdater. Försöker förutsäga när displayen ska uppdateras, se 2.1.3.

TcpDisplayUpdater Kommunicerar med displayenheten, se 1.3 för mer detaljer. Vid anslutning skickar den den aktuella tiden och det senaste meddelandet, och sedan skickar den det nya meddelandet varje gång det ändras.

2.2.3 Services

Services innehåller det externa gränssnittet, som webbplatsen kommunicerar med. Se 1.4 för användningsinformation.

MessageUpdaterDatabase Den aktuella implementationen av MessageUpdater som arbetar mot modellen.

MessageUpdaterNoop Minimal MessageUpdater som kan användas vid tester utan att behöva en PostgreSQL-databas. Den bahandlas inte i detta dokument.

3 Kända Brister

3.1 Säkerhet

Ingen kryptering eller auktorisering på applikationslagret sker mellan displayenheten och schemaläggaren. I stället förutsätts det att displayenheten är ansluten till schemaläggaren via ett isolerat och krypterat nätverk.

Dessutom sker det idag ingen auktorisering av displayenheten, utan det förutsätts att extern åtkomst blockeras via en brandvägg.

3.2 Översättning och Internationella Marknader

All text skickas över standarden ISO 8859-1 som tyvärr inte täcker hela Unicode, men som är enklare att implementera eftersom alla tecken har en fast bredd. Skulle produkten användas utanför norden eller den engelsktalande världen så skulle en en övergång till exempelvis UTF-8 eller UCS-16 vara nödvändig.

3.3 System med Flera Displayenheter

Just nu krävs det en isolerad schemaläggare per displayenhet, om de inte alltid ska visa samma innehåll.