Detta är den svenska titeln

JULIUS RECEP COLLIANDER CELIK

Civilingenjör Informationsteknik
Datum: 12 april 2018
Handledare: Patric Dahlqvist
Examinator: Anders Västberg
Uppdragsgivare: LS Elektronik AB
Engelsk titel: This is the English translation of the title
Skolan för elektroteknik och datavetenskap

Sammanfattning

Svensk sammanfattning.

Abstract

English abstract.

Innehåll

Kapitel 1

Introduktion

Examensarbetet handlade om att undersöka möjligheten att skapa en modell för att beskriva och annotera redigerbar data, och sedan automatiskt generera ett användargränsnitt.

1.1 Mimer SoftRadio

Arbetet utfördes hos LS Elektronik AB *(LSE)*, som är ett tekniskt företag, som utvecklar och tillverkar elektroniska produkter [**Ehne**]. LSE erbjuder bland annat ett radiosystem som heter Mimer SoftRadio vilket kan användas för att ansluta ett flertal annars inkompatibla radioenheter i ett och samma system, samt fjärstyra radioenheterna från en persondator med ett klientprogram. I resten av rapporten kommer datorn med klientprogrammet kallas operatörsdator, där användaren kallas operatör.

Mimer SoftRadio var ett program med väldigt många möjliga inställningar. I många fall var dessa inställningar för komplexa för de vanliga operatörerna att redigera själva, så därför brukade vissa kunder låsa redigeringsmöjligheterna, och bara tillåta vissa administratörer att ställa in alla inställningar på rätt sätt. Det fanns också kundfall där flera operatörer använde samma dator, vid olika tidpunkter. Ett vanligt kundfall var då att en operatör jobbade dagtid med att leda och organisera dagsarbete, medan en annan operatör tog över nattskiftet för att övervaka många fler radioenheter.

För att förenkla dessa två kundfall påbörjade LSE utvecklingen av funktionalitet som skulle erbjuda användare att spara uppsättningar av inställningar i olika *profiler*. Det skulle gå att enkelt byta mellan flera

förinställda konfigurationer av Mimer SoftRadio. För att konfigurera dessa profiler skapades ett administratörsprogram, som skulle kunna fjärkonfigurera profilerna hos operatörsdatorerna. Fjärstyrningen skulle underlätta administratörer att ställa in profiler på flera datorer samtidigt, som sannolikt skulle innehålla liknande inställningar.

1.2 Problembeskrivning

Systemet för att konfigurera profiler byggdes på att alla operatörsdatorer fungerade som en server, där de exponerade ett API över en TCP-port. Administratörsprogrammet skulle erbjuda ett användargränsnitt för att konfigurera profilinställningar, för att sedan kommunicera ändringarna till operatörsdatorerna. I resten av rapporten kan operatörsdatorerna och administratörsprogrammet kallas server respektive klient.

Kommunikationsprotokollet var ett egetskapat som byggde på att skicka JSON-Objekt via TCP. För en beskrivning av JSON, se kapitel 1.3.1. För en mer utförlig beskrivning av kommunikationsprotokollet se XXX. Problemet som LSE hade inför utvecklandet av användarprofilerna var skapandet av ett användargränsnitt på administratörsprogrammet. Olika operatörsdatorer, hos olika kunder, kunde ha olika versioner av Mimer, med olika funktionalitet tillgänglig, och därmed olika uppsättningar konfigurerbara inställningar.

Det vore orimligt kostsamt för LSE att skapa ett administratörsprogram per version av Mimer, då både Mimer ändrades med tiden, samt att olika kunder köpte till extra funktionalitet. Samtidigt behövde användargränssnittet på administratörsprogrammet anpassas så att det skulle vara tydligt vad en administratör kunde konfigurera. Helst skulle ett administratörsprogram fungera bra med framtida versioner av Mimer, utan några eller utan stora justeringar av programmet. LSE ville helt enkelt att servern kommunicerade tillgängligga inställningar, till klienten så att klienten sen skulle kunna anpassa sitt användargränsnitt, och det skulle ske på ett tillräckligt generellt sätt att administratörsprogrammet var framtidssäkert.

1.3 Bakgrund

Introducera kapitlena här.

1.3.1 JSON och webkommunikation baserat på JSON objekt

JavaScript Object Notation *(JSON)* är ett texbaserat dataformat för att utbyta data mellan webbtjänster. Till skillnad mot andra alternativ, som exempelvis XML, är det både läsbart för människor och datorer, samtidigt som det är väldigt kompakt, vilket är en anledning till att det är ett av de mest populäraste dataformaten för datautbyte mellan webbtjänster. [**Pezoa2016**]

JSON erbjuder stöd för några enkla datastrukturer: textsträngar *(string)* (`"hej värld"`), tal *(number)* (`4`), ett tomt värde *(null)* (**`null`**), samt booleska värden *(booleans)* (**`false`**). JSON erbjuder dessutom stöd för två komplexa datatyper vilket är vektorer *(array)*, en ordnad lista av JSON-värden:

```
["hej värld", 4, null, false]
```

Figur 1.1: Exempel på JSON-array

samt objekt *(object)*, vilket är en oordnad mängd av namn-värde-par:

```
{
  "firstName": "Erik",
  "lastName": "Andersson",
  "age": 30
}
```

Figur 1.2: Exempel på JSON-object

Med hjälp av att rekursivt använda *array* eller *object* går det att representera komplexa datastrukturer med hjälp av JSON. Det finns inga begränsningar i hur komplexa datastrukturer kan representeras.

1.3.2 JSON i webbkomunikation

På grund av att JSON är kompakt, enkelt läsbart och har brett stöd hos många språk och implementationer, har JSON blivit väldigt utbrett bland webbtjänster. En hypotetisk förfrågan till en webbtjänst skulle kunna se ut som följande:

```
{
  "country": "Sweden",
  "city": "Stockholm"
}
```

Figur 1.3: Exempel på förfrågan till webbserver

där en klient förfrågar om de nuvarande väderförhållandena i Stockholm i Sverige. Svaret från webbservern skulle kunna se ut som följande

```
{
  "timestamp": "06/01/2018 10:45:08",
  "country": "Sweden",
  "city": "Stockholm",
  "weather": "Snowing",
  "temperature": -3
}
```

Figur 1.4: Exempel på svar på förfrågan från webbserver

där webbservern svarar att temperaturen är minus tre grader celsius och att det snöar. Exemplet visar hur simpelt JSON som dataformat är att förstå, vilket skulle kunna vara en delvis förklaring för popularite-ten.

1.3.3 JSON Schema

Trots att JSON är det populäraste dataformatet för datautbyte mellan webbtjänster saknas det ett väletablerat standariserat ramverk för metadata-definition [**Pezoa2016**]. En väldigt lovande formell standard är JSON Schema, vilket är ett ramverk som fortfarande utvecklas av Internet Engineering Task Force *(IETF)*. IETF beskriver själva JSON Schema som *"JSON Schema asserts what a JSON document must look like, ways to extract information from it, and how to interact with it."* [**A.Wright**].

JSON Schema är ett ramverk för att förklara hur JSON värden kan se ut. JSON Schema specifiserar regler som kan användas för att antingen bestämma om befintliga JSON värden är giltiga, eller för att i förväg beskriva hur gilltiga värden får se ut.

Objektet i Figur 1.2 skulle kunna valideras enligt följande JSON Schema:

```
{
  "type": "object",
  "required": ["firstName", "age"],
  "properties": {
    "firstName": { "type": "string" },
    "lastName": { "type": "string" },
    "age": { "type": "integer" }
  }
}
```

Figur 1.5: Exempel på simpelt JSON Schema

Användningsområden för detta är bland annat:

1. Validering av data.
2. Annotering av data.
3. Beskrivning av REST APIer.
4. Automatisk generering av kompatibel kod, för att hantera JSON värden beskrivna med JSON Schema.
5. Automatisk generering av API-dokumentation.
6. Automatisk generering av användargränsnitt.

Att använda JSON Schema för användningsområdena 1-3 är trivialt. Det går att utveckla program som kan hantera alla oändligt möjliga permutationer av JSON Schema. Det som däremot inte är trivialt är hur användningsområdena 4-6 skulle kunna generaliseras så pass mycket att ett program eller algoritm skulle kunna hantera vilket giltigt JSON Schema som helst. Användningsområde fyra och fem omfattas inte av den här rapporten, och varför användningsområde sex inte är trivialt diskuteras mer i kapitel XXX.

1.4 Frågeställning

Hur skulle man kunna använda en delmängd av JSON Schema för att dynamiskt anpassa ett användargränsnitt mot olika versioner av apier som erbjuder olika funkionalitet, på ett långsiktigt sätt?

Kapitel 2

Metodval

OBS! Detta kapitel kan ignoreras. Det är nästan bara egna anteckningar

Dela upp i "hur man brukar göra" och "hur jag faktiskt gör".
qualative data
textual analysis

1. Föreslå ett eget JSON Schema
2. Skapa en JSON Schema genererare. ????
3. Skapa en JSON Schema parser för Delphi. / Undersök befintliga JSON Schema parsers för Delphi.
4. Skapa en direkt mappning mellan JSON Schema och en eller flera datatyp(er) i Delphi
5. Create a dynamic interface based on parsed JSON Schemas.

2.1 Avgränsningar

JSON Schema kan användas till:

- validering av data
- annotering av data
- automatisk generering av kompatibel kod
- beskrivning av REST APIer

- automatisk generering av API-dokumentation

Det kanske går att använda för att automatisera tester?? Det skulle kunna gå att testa att datan ett api ger stämmer överens mot ett jsonSchema, eller att ett api klarar av att hantera all tillåten data.

Dessutom finns det exempel på JSON Schema som automatiskt genereras utifrån kod XXXXX.

Detta projekt intresserar sig enbart för att försöka använda JSON Schema för att annotera data som kan redigeras. Det vill säga beskriva vilken data som kan redigeras, samt hur den kan redigeras. Därför kommer inte all funktionalitet av en JSON Parser implementeras, då det är utanför intresseområdet av rapporten.

Utöver funktionalitet kommer JSON parsern bara förstå en förenklad delmängd av JSON Schema, då vissa egenskaper av JSON Schema inte är eftertraktade. Ett exempel på ej eftertraktade egenskaper nyckelordet `multipleOf` är att kunna specificera att en `number` eller `integer` ska vara en multiple av en annan siffra.

Modellen som rapporten föreslår för att annotera JSON data kommer inte nödvändigtvis vara en strikt delmängd av JSON Schema. Om ej implementerade egenskaper behövs, kan modellen utökas för att inkludera egenskaper som saknas i JSON Schema.

Kapitel 3

Resultat

3.1 Vad saknas i JSON Schema

Hur hanterar man olika valideringsfel?

Föreslå kanske att JSON Schemat som föreslogs i rapport X ska användas.

Kapitel 4

Diskussion och Slutsats

Bilaga A

Extra Material som Bilaga