Danmarks Tekniske Universitet & Visma Consulting, Hellerup

Projekt rapport

Optimering af fartøjs kommunikationsmodul konfiguration
Optimizing fishing vessel communication module configuration

Studerende: Ibrahim Nemli

Studienummer: s093477

DTU Vejleder: Carsten Witt

Virksomhedsvejledere: Søren Lund, Mikael Grønfeldt & Klaus Mogensen

Virksomhed: Visma Consulting, Hellerup

Afsluttende eksamensnummer: IMM-B.Eng-2012-25

Afleveringsdato: 28-01-2013

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45 25 33 51, Fax +45 45 88 26 73
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

This is a report about optimizing and making a user-friendly interaction with a configuration for a system called communication unit which is a part of vTrack. vTrack is a software program developed and maintained by Visma Consulting at Hellerup in the department of Fishery Solutions[1]. The main objectives of this software are to track and monitor fishing vessels for the country concerned.

The communication unit starts up by reading a configuration file (written in xml) which is quite long and messy. From the information in this file it initializes some processes (as threads) and starts to perform different kind of tasks as specified in the configuration.

The aim in this project is to get rid of this xml file by mapping the data into a database and then making a graphical environment to maintain the configuration in a convenient way.

The project has been achieved by splitting the project into sub projects:

Subproject 1 – database modelling
In this subproject a database architecture has been developed which makes is possible to map the data in the configuration file to database.

Subproject 2 – integration and unit testing
After developing the database the communication unit has to be integrated with the database so it reads the configuration from it and initializes the correct processes as it did with the xml file. That no functionalities have been corrupted during this switchover has to be validated by performing unit tests for the developed source code.

Subproject 3 – environment
Finally as the last part of the project an environment (a web application) is needed to make it possible to interact with the data stored in the database and manipulate it regarding to the CRUD principles (create, read, update, delete).
Those changes should immediate take effect on communication unit without restarting it from scratch. This is also an optimizing since any changes on the configuration file (the xml file) caused a manual restart of the communication unit.

Subproject 4 – xml parser
This subproject came up while discussing at the company. This is a small program which is able to parse the configuration file and convert it to SQL statements. Furthermore this program gives you the opportunity to save the statements into a SQL file and run it directly on a database.

This project has been made in collaboration with Visma Consulting at Hellerup and Technical University of Denmark (DTU).

The rest of this report has been writing in Danish language.

[1] http://www.fisherysolution.com/

Forord

Jeg er en IT-diplomingeniør studerende fra DTU, Danmarks Tekniske Universitet, der skal skrive et eksamensprojekt (bachelor), med det formål at opnå en B. Eng grad i IT.

I dette projekt arbejdes der med et såkaldt Vessel Monitoring System (VMS) som er i stand til at kunne overvåge fiskefartøjer i det land, det er installeret i.
VMS'en er opbygget af forskellige enheder og en af disse enheder er "kommunikationsenheden", som vedtager al kommunikation mellem de eksterne datakilder, f.eks. de overvågende fiskefartøjer, og systemet selv. Denne enhed startes op ved at indlæse konfigurationsparametre fra en fil, således at den kan oprette forbindelser til de korrekte datakilder. Ulempen ved denne løsning er, at der kommunikationsenheden skal genstartes når der modificeres i filen, hvilket giver kommunikationsnedetid. En anden ulempe er, at filen er efterhånden så stor og kompleks, at den er svær at overskue.

Denne opgave har til formål at udvikle en arkitektur der vil gøre det muligt, at kunne persistere data fra konfigurationsfilen i en database. Dernæst skal kommunikationsenheden ændres, så den læser konfigurationen fra databasen. Så snart der sker ændringer i konfigurationsparametrene, skal disse indlæses af kommunikationsenheden. Endelig skal der udvikles et grafisk brugerinterface hvori de gemte data i databasen kan modificeres på en brugervenlig måde. Dette vil sørge for at kommunikationsenheden ikke genstartes hver gang der ændres i parametrene manuelt.

Projekt udføres i et samarbejde med vejledere fra DTU og Visma Consulting, Hellerup.

Ibrahim Nemli – s093477
Hellerup, januar 2013

Indholdsfortegnelse

1 Projekt indledning og problemformulering

I forbindelse med den internationale fiskeriindustri er der udviklet it-systemer, som gør det muligt for fiskefartøjer rundt om i verden at bl.a. opsamle informationer omkring fartøjets nuværende position, hastighed og retning.
Om bord på skibene er der monteret en transceiver, som via en satellitforbindelse sender informationerne til landets lokale fiskerimyndighed, som herefter kan overvåge fartøjerne ved hjælp af de data de får ind.
Visma Consulting har udviklet sådan et Vessel Monitoring System (VMS), kaldet vTrack. Systemet benyttes af 13 fiskerimyndigheder verden over.
vTrack består af en kommunikationsenhed, som varetager al kommunikation med satellitudbyderne, en præsentationsenhed, som er en web-baseret overvågningsapplikation til brugerne, og en database, hvori indkomne data fortolkes og persisteres.

Kommunikationsenheden består af en række moduler. Der findes bl.a. moduler for hver satellitudbydere som anvendes (fx Thrane-Thrane, CLS, Orbcom m.fl.), moduler for kommunikation via mailservere og moduler til at kommunikere med andre fiskerimyndigheder. Ved en kombination af disse moduler sættes der en konfiguration op for den enkelte kunde (fiskerimyndighed).
Hvert modul har en række konfigurationsparametre, således at man kan tilpasse modulets funktionalitet til den enkelte kunde. Eksempelvis er der nogle moduler, som har behov for at få en liste over de fartøjer, som der skal hentes positioner fra. Den anvendte konfiguration med de enkelte moduler og dens tilhørende data, gemmes i en XML-fil.
Databasen benytter også en række konfigurationsparametre for at kunne fortolke data. En række af disse parametre er identiske med konfigurationsparametrene i kommunikationsenheden. Dette giver konflikter i forbindelse med redundant data og adskillelse af forudbestemte retningslinjer, da man har mulighed for at læse data fra 2 kilder (fra fil og fra database). Derfor er der et behov for at kommunikationsenheden og databasen læser parametre fra samme kilde.

Opgaven går derfor ud på at fjerne denne redundans ved at migrere alle data, som i den nuværende løsning er placeret i XML-filen, over i en databasestruktur, som udarbejdes til formålet. Migreringen af konfigurationsdata skal foregå således at implementeringen af kommunikationsenheden påvirkes mindst muligt. Der skal udvikles en række testscenarier (unit tests), som kan vise at funktionaliteten bevares.

Somme tider bliver det også aktuelt for brugerne at kunne ændre i de forskellige parametre i de forskellige moduler i konfigurationen. Bl.a. i situationer hvor der skal opdateres i en parameter i et bestemt modul. Derfor skal der udvikles et miljø, hvor man har mulighed for at tilpasse konfigurationen. Adgang til konfigurationsændringerne skal tage hensyn til vTracks rollebaserede rettighedssystem.
Opgaven skal implementeres på en sådan måde, at der efterfølgende kan tilføjes nye moduler i kommunikationsenheden samt tilføjes og fjernes konfigurationsparametre, uden at der skal foretages programmeringsmæssige ændringer af hverken indlæsning af kommunikationsparametre eller det miljø hvori brugerne kan ændre værdierne.

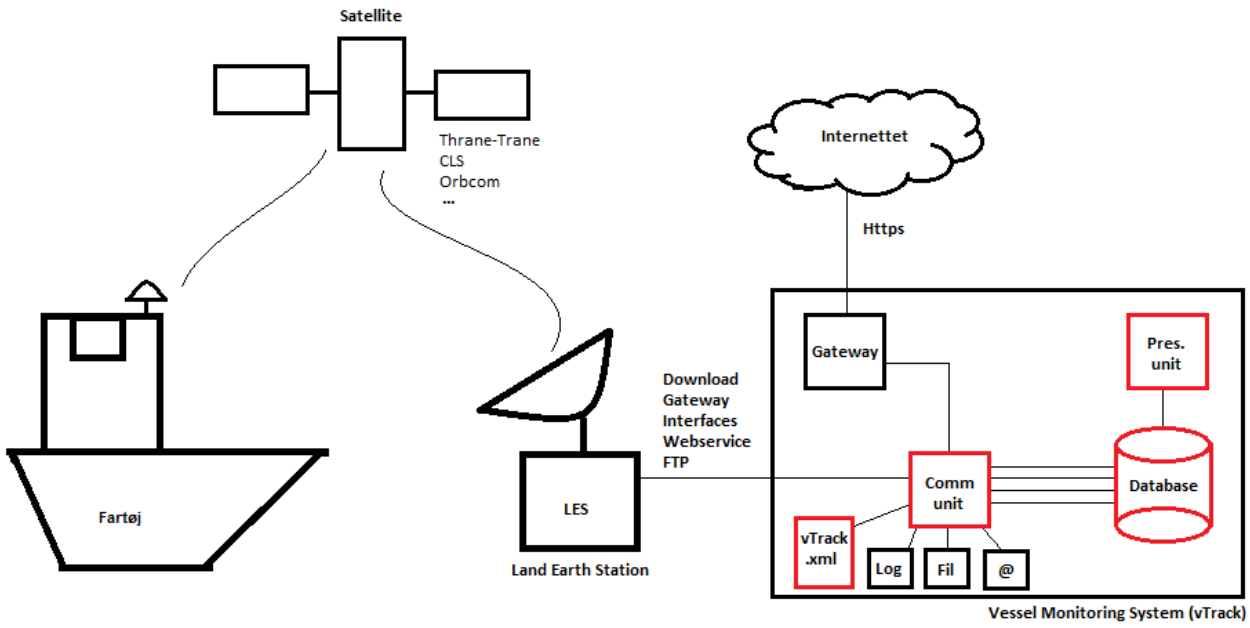
2 Vision

For brugere af vTrack har dette projekt til formål at få fjernet den nuværende konfigurationsfil (baseret xml) permanent. Løsningen er at konfigurationen til kommunikationsenheden aflæses fra en database og der udvikles et miljø, med det formål at kunne tilpasse konfigurationsparametrene på en brugervenlig måde.

Ved dette opnås en optimering dels ved at kommunikationsenheden ikke genstartes hver gang når der ændres i konfigurationsparametrene og man får et større abstraktionsniveau over konfiguration idet at man kan ændre konfigurationen i et grafisk miljø, end en editor med xml syntaks.

3 vTrack systemarkitektur

Figuren nedenfor (Figur 1) giver et overblik over hvordan vTrack er sat sammen i forbindelse med at overføre data fra/til et fartøj gennem en satellit forbindelse og til/fra vTrack som beskrevet i afsnit 1.
Comm unit (kommunikationsenheden) initialiseres ved at indlæse konfigurationen i vTrack.xml filen, hvorved den opretter forbindelser til forskellige komponenter som vist på figuren.
Kommunikationsenheden er også i stand til at kunne sende mails, skive log filer og gemme data i filer i tilfælde af, at en forbindelse skulle gå tabt til en ekstern enhed.
De forskellige enheder der er markeret med rødt viser de komponenter som dette projekt skal tage hånd om og som skal udvikles videre på.



Figur 1 - vTrack systemarkitektur

4 Foranalyse

Inden jeg går i gang med planlægningen af projektet og laver en kravspecifikation foretages der en kort analyse af kommunikationsenheden og præsentationsenheden, således jeg har en grundlæggende forståelse af disse.
Denne foranalyse vil også gøre det nemmere for mig at kunne identificere problemerne og hvad der skal laves. På baggrund af denne viden, kan jeg nemmere estimere tidbehovet der er for projektets opgaver og se hvor meget indsats der skal til, for at kunne færdiggøre opgaven succesfuldt.

4.1 Kommunikationsenheden

Kommunikationsenheden er et selvstændigt (Java) projekt og er uafhængigt af de øvrige enheder som vTrack består af, se evt. Figur 1 (afhængighederne til de øvrige enheder startes når konfigurationen indlæses og forbindelser til eksterne enheder etableres).
Som sagt starter denne enhed op ved at indlæse en konfigurationsfil. Jeg vil først gå i dybden med denne fil og derefter finde ud af hvordan denne anvendes ved at kigge i koden og analysere designet af den nuværende implementation.

4.1.1 Analyse af vTrack.xml

Konfigurationsfilen til kommunikationsenheden er en xml fil (filen der anvendes hedder vTrack.xml). Inden jeg går i gang med selve analysen af filen, vil jeg kort forklare om xml filer (overordnet) og derefter gå i gang med selve analysen. Derved kan vi etablerer en forståelse for hvad xml er og hvorfor det anvendes.

Xml står for er eXtensible Markup Language og er udviklet af W3C[2]. Xml er designet for at kunne strukturere, persistere og transportere data mellem forskellige systemer. Data i xml filer gemmes i plain text (ren tekst) således det er nemmere at udveksle data mellem forskellige applikationer. En af de gode fordele ved xml er, at man selv skal definere sine tags og på den måde kan man selv bestemme hvordan man vil strukturerer sine data.

Med simple regler for hvordan man bygger xml filer op, kan man relativt hurtigt lære xml. Dette er måske en af grundene til hvorfor xml er så populært som det er i dag.
Xml er faktisk så et populært værktøj i softwareverdenen, at der er opfundet nye sprog på baggrund af xml (fx XHTML og WSDL for web services) og måske derfor stadig anvendt i flere forskellige projekter verden over.

Xml er i vTrack blevet brugt til at gemme data til at konfigurere kommunikationsenheden ved opstart. Denne etablerer forbindelser til eksterne enheder og starter prædefinerede opgaver op, alt efter de parametre der er specificeret i filen. Dette er selvfølgelig varierende fra kunde til kunde alt efter deres specifikationer og krav.

For at finde ud af hvordan vTrack.xml filen er opbygget har jeg kigget i xml filen. I denne fil har jeg analyseret mig frem til følgende træstruktur som ses på Figur 2.
Parentesen til højre er antal af forekomster (multiplicitet) af de forskellige xml tags.
(vTrack.xml filen som anvendes af kommunikationsenheden ligger i ”Comm unit” projektet under mappen ”conf”).

[2] http://www.w3.org

Conf (1)
- Group (1)
  - ParameterCollection (0..*)
    - Parameter (0..*)
      - Name (1)
      - Value (1)
  - Group (1)
    - ModuleCollection (0..*)
      - TimedModule (0..*)
        - Name (1)
        - Description (1)
        - ClassName (1)
        - Frequence (1)
        - Offset (1)
        - Parameter (0..*)
          - Name (1)
          - Value (1)
        - Username (0..1)
        - Password (0..1)
        - Server (0..1)
        - Port (0..1)
        - Command (0..*)
          - Name(1)
          - Subject (0..1)
          - Message(1)

Figur 2 - vTrack.xml træstruktur

Jeg opdagede at vTrack.xml filen kan have referencer til andre xml filer bl.a. efter den sidste `TimedModule` tag findes der en reference til vTrack_listeners.xml. Denne xml fil har følgende struktur:

ListenerModule (0..*)
- Name (1)
- Description (1)
- ClassName (1)
- Port (1)
- Parameter (0..*)
  - Name (1)
  - Value (1)

Figur 3 - vTrack_listeners.xml træstruktur

Det er også muligt at finde referencer til andre xml filer herunder vTrack_inmarsat.xml og vTrack_argos.xml. Disse referencer er angivet midt i de pågældende moduler (primært under `TimedModule` tag).

Inmarsat og argos har samme xml struktur og er opbygget således:

Parameter (0..*)
- Name (1)
- Værdi (0..1)
- HexValue (0..1) (kun vTrack_inmarsat.xml)

Figur 4 - vTrack_inmarsat.xml og vTrack_argos.xml træstruktur

Alt i alt anvendes der 4 xml filer som kommunikationsenheden indlæser.
Der findes også et XSD skema til vTrack. Dette skema består af et sæt "regler" for hvordan vtrack.xml filen skal være angivet formelt, således man har en gyldig xml fil, der kan anvendes af programmet. Vi ikke vil komme ind på dette, da det ikke er relevant for projektet lige nu, men vi vil komme ind på det senere i rapporten.

Som man kan se, så består konfigurationsfilen primært af en samling af moduler (forholdsvis af timed modules og listener modules) og en samling af parametre (ParameterCollection) (de øvrige tags vil vi komme ind på senere i rapporten). Udover den strukturelle analyse som vi lige har foretaget os, vil jeg nu gå endnu dybere med konfigurationsfilen, for at få en dybere forståelse for, hvilket formål disse moduler foretager. Dette gøres ved at analysere det data som disse moduler består af, med det håb om at lære noget nyt og måske noget relevant indenfor projektets scope.

Som det ses på Figur 2 og Figur 3, så er det et tag (`description`) som giver kort beskrivelse af modulerne. Ved at skimme disse igennem sammen de øvrige tags, kan jeg nu lave denne liste:

Der findes moduler til…

- … at oprette logfiler og forbindelse til database.
- … at hente data fra forskellig satellit udbydere (Inmarsat, Argos, Orbcomm, Faria Watchdos, CLS America m.fl.).
- ... at læse jobs fra databasen.
- … at etablere http og https gateway forbindelser.
- … at etablere FTP (File Transfer Protocol) forbindelser.
- … at overvåge forbrug af hukommelse (Java kan godt bruge store mængder af hukommelse).
- … at slette gamle filer.
- … at oprette (server) socket forbindelser for at lytte på kommandoer og data (listener module).

Der er et af disse moduler som er vigtigt at tage hensyn til allerede fra nu af og dette er det modul som opretter logfiler og forbindelse til databasen.
Uden at have initialiseret dette modul, så vil det være umuligt at kunne læse konfigurationen til kommunikationsenheden fra databasen, da kommunikationsenheden ikke ved hvilken database den skal forbinde sig til ved opstart. Derfor kan vi ikke gemme dette modul i databasen. Disse data bliver nødt til at findes i en ekstern kilde (evt. i en properties fil) som indlæses før den restende konfiguration indlæses!

4.1.2 Opstart af kommunikationsenheden

jeg vil nu gå ind i kildekoden for kommunikationsenheden for at analysere hvordan konfigurationsfilen starter kommunikationsenheden op, hvordan den behandler data fra xml filen og hvordan de forskellige moduler og tråde initialiseres.
Denne analyse vil give et indblik af hvordan den nuværende implementation er og vil være til inspiration for hvordan det nye system kan se ud.

Indlæsningen af konfigurationen til kommunikationsenheden starter ved `init()` metoden i klassen `VtrackServer`. I denne metode oprettes der et `Configuration` objekt som er rygraden i forbindelse med indlæsningen af data fra xml filen. Den klasse har ansvaret for at parse xml filen til objekter, som kan anvendes til at oprette de rigtige tråde.

Først indlæses xml filen og konverteres et `ByteArrayInputStream` objekt som herefter bliver parset om til den interne datarepræsentation, der er genereret af det xml framework der benyttes i vTrack (xmlBeans). Jeg vil ikke gå ind i detaljerne omkring dette framework, da dette er uden for projektets scope. Hvis du er interesseret kan du læse mere omkring dette på nedenstående hjemmeside[3].
Under denne proces tages der også hensyn til de øvrige xml filer som der bliver refereret til, således disse også bliver parset.

Når denne proces er færdig, kan vi hente konfigurationen ind som et objekt (med `getConf()` metoden). Hernæst henter vi en gruppe (java objekt med kaldet `getGroup()`) og indlæser denne. Indlæsning af en gruppe består i at indlæse parametre fra `ParameterCollection` tagget og moduler fra `ModuleCollection` tagget. Da `group` objektet (se evt. Figur 2) har en refleksiv association til sig selv, kan `group` objektet igen indlæse sig selv (rekursivt) ved at hente `group` (dette bliver endnu tydeligere når vi analyserer xsd skemaet til vTrack.xml filen senere i rapporten).

Tags der er barn til `ParameterCollection` bliver indlæst ved at hente navne og værdier fra de forskellige parametre tags og gemmes i et objekt kaldet `PropertyHelper`, som nedarver fra `HashTable<object, object>`.

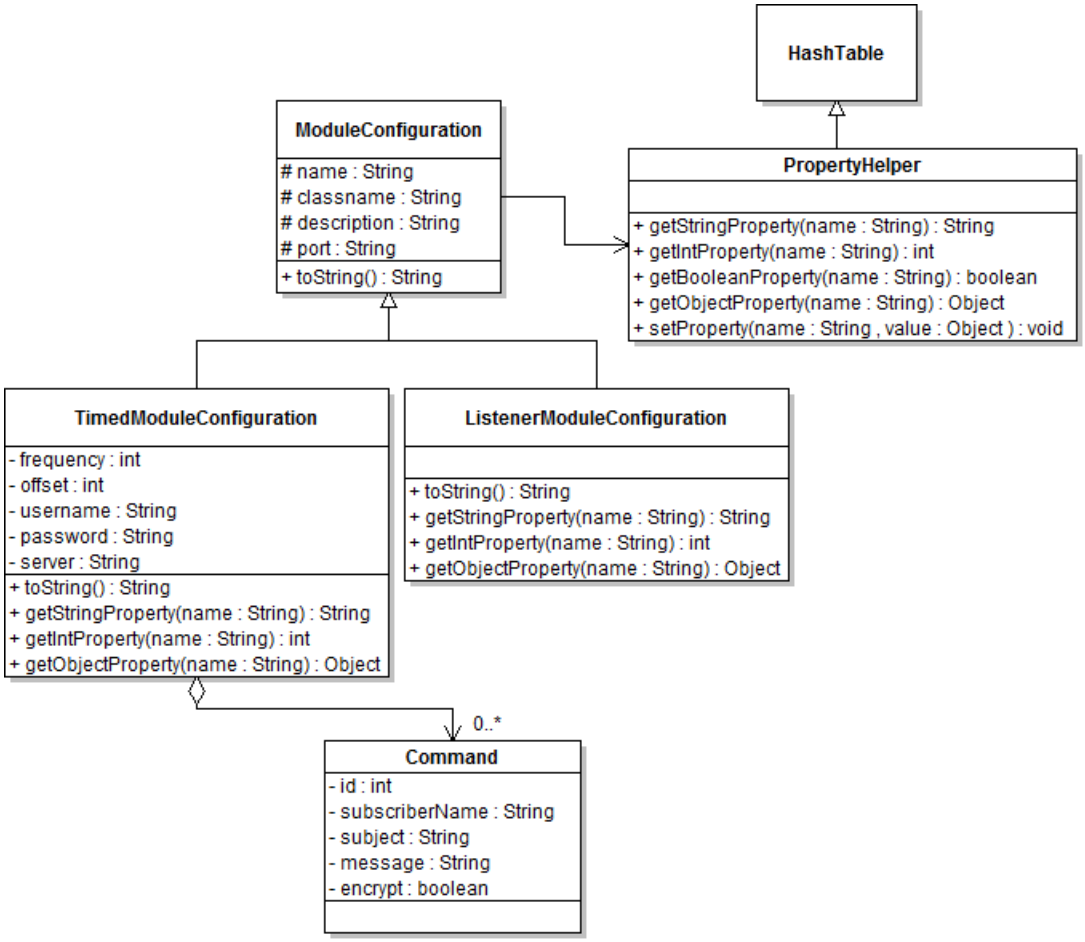
Dernæst indlæses modulerne. Indlæsningen af modulerne i `ModuleCollection`-tagget består af 2 trin.
I det første trin indlæses `TimedModule` objekterne (se metoden `readTimedModule(timedModule)`).
Det der sker her er, at `TimedModule` objektet bruges til at oprette et `TimedModuleConfiguration` objekt (se Figur 5, dette objekt tager `TimedModule` som parameter i konstruktøren). Herefter tjekkes der om modulet er ”aktivt” dvs. om modulet skal initialiseres eller ej. I xml filen er det nemlig muligt at definer hvilke moduler der skal være aktive og hvile der ikke skal (dette felt angives som xml attribut i `TimedModule` og i `ListenerModulet` tagget). Dette har en betydning om hvorvidt modulet skal køres af kommunikationsenheden eller ej.

I step 2 indlæses `ListenerModule` objekterne (se metoden `readListenerModule(listenerModule)`).
Dette sker stort set på samme måde som før med timed modules. `ListenerModule` objektet gives som parameter til konstruktøren til et `ListenerModuleConfiguration` objekt. Hvis dette modul er aktivt oprettes der nyt `Listener` objekt (tråd).

[3] http://xmlbeans.apache.org/

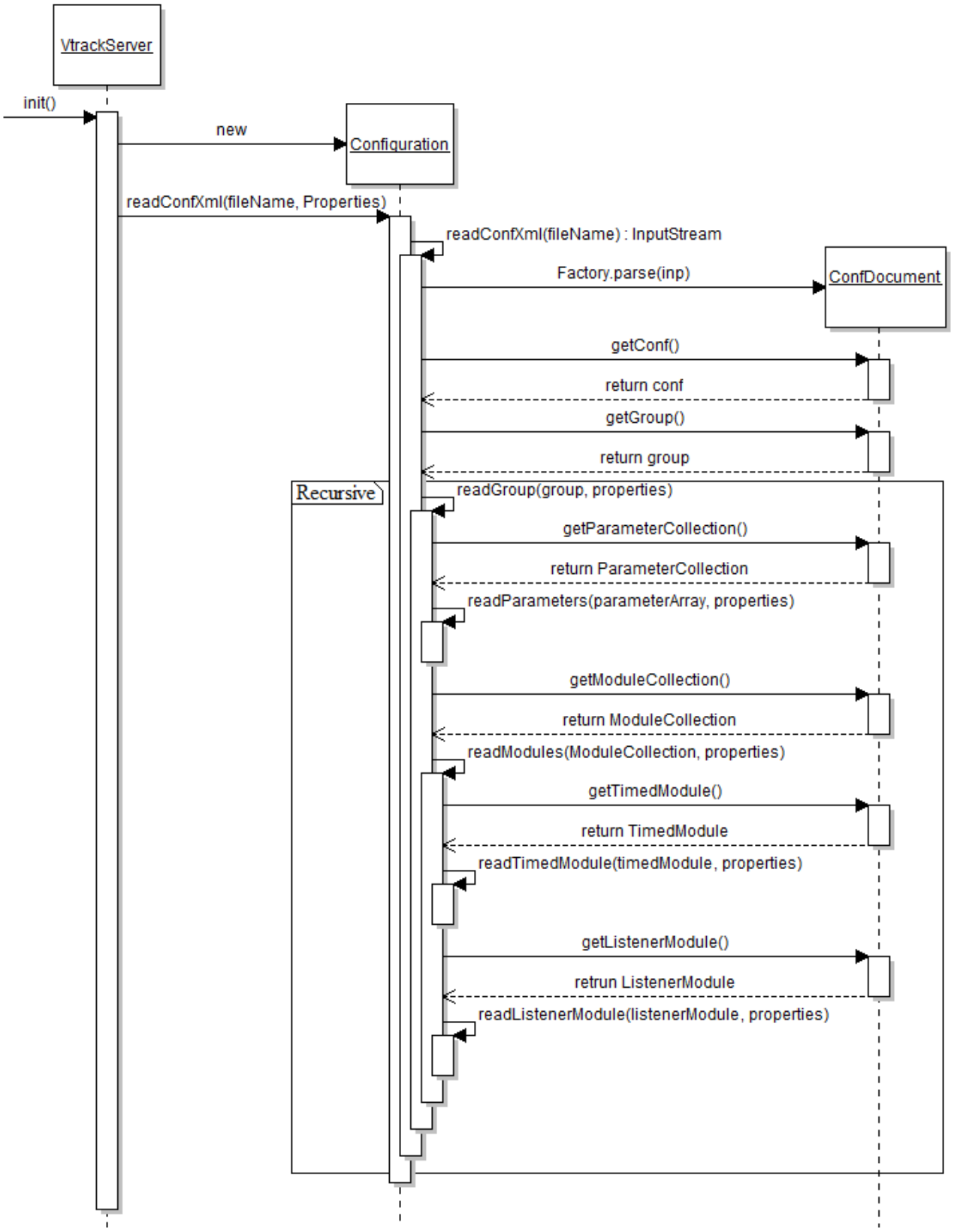
Objekterne der oprettes på baggrund af de forskellige moduler kører i deres egen tråd. Disse tråde er tidsskeduleret, så de kører engang imellem på skift, indtil `stop` variablen sættes til `true`. Så vil den pågældende tråd afsluttes.

Diagrammet nedenfor (Figur 5) viser den interne arkitektur over de objekter der oprettes med data der er læst fra xml filen. Disse objekter anvendes til at oprette tråde der er specificeret af modulet. Dette gøres ved at initialiser typen/klassen der er angivet i variablen `classname` som et objekt (disse klasser/objekter er implementeret af kommunikationsenheden), hvor den så herefter vil køre i sin egen tråd.
Klasserne `TimedModuleConfiguration` og `ListenerModuleConfiguration` har konstruktører som tager imod de passende objekter der er genereret af xmlBeans.
(Klasser vist på figuren nedenfor ligger i dk.vtrack.comunit.conf package).



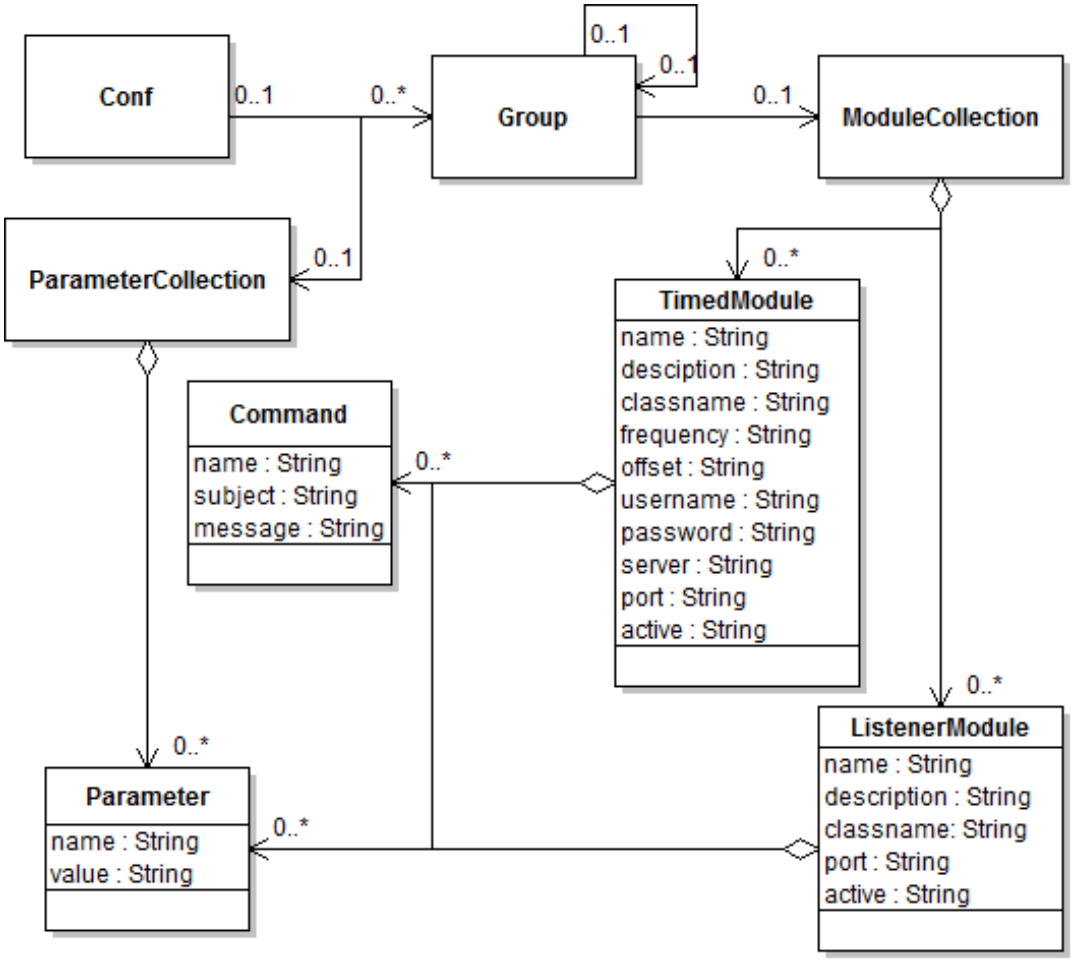
Figur 5 - Konfigurationsarkitektur

Forneden findes der et sekvensdiagram der giver et overblik over indlæsningen af konfigurationsparametrene til kommunikationsenheden som beskrevet overfor.



Figur 6 - Sekvensdiagram over gammel implementation

Figuren nedenfor (Figur 7) viser den interne data repræsentation (domænet af xml data) som er genereret af xmlBeans, og som bruges til at oprette de forskellige objekter vist på Figur 5.



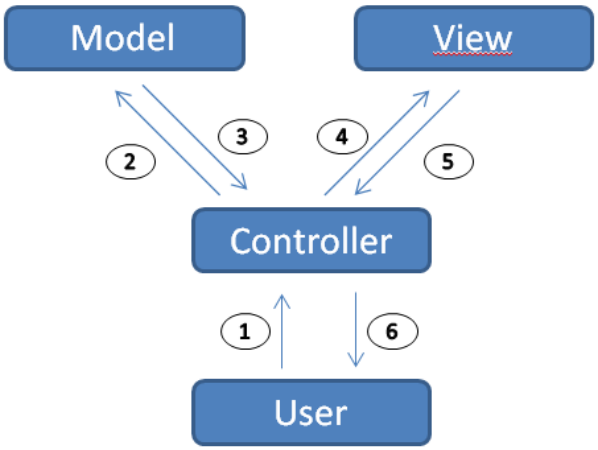
Figur 7 - Xml data repræsentation (xmlBeans)

4.2 Præsentationsenheden

Præsentationsenheden er en webapplikation hvor brugerne kan se og overvåge fartøjer med det data der er modtaget gennem kommunikationsenheden. Man kan også foretage yderligere administrative opgaver i præsentationsenheden, som vi ikke vil gå i dybden med. I dette afsnit laves der en beskrivelse af hvordan systemet er bygget op, således jeg kan få en forståelse for hvordan præsentationsenheden kan udvides med Miljøet som skal udvikles i forhold til de principper, teknologier og standarder der anvendes i dette projekt.

Præsentationsenheden er opbygget gennem et framework kaldet Spring. Spring består af mange moduler såsom fx Data access/integration med databaser, web, AOP (Aspektorienteret programmering), dependency injection, og test[4].
Til dette projekt anvendes Spring's web framework. Web frameworket er designet efter MVC (Model-View-Controller) design pattern.

Dette design pattern er baseret på interaktioner mellem brugeren og systemet. Det går ud på at brugeren får vist et view (en grafisk brugergrænseflade) efter brugerens handling (punkt 1 på Figur 8). Dette vil starte en proces i controlleren, som nu vil hente data fra modellen (punkt 2 og 3). Dette gives det videre til viewet (punkt 4) gennem controlleren, som kan sætte data ind i de forskellige grafiske komponenter som viewet består af. Når dette er klart, vises viewet til brugeren (punkt 6).



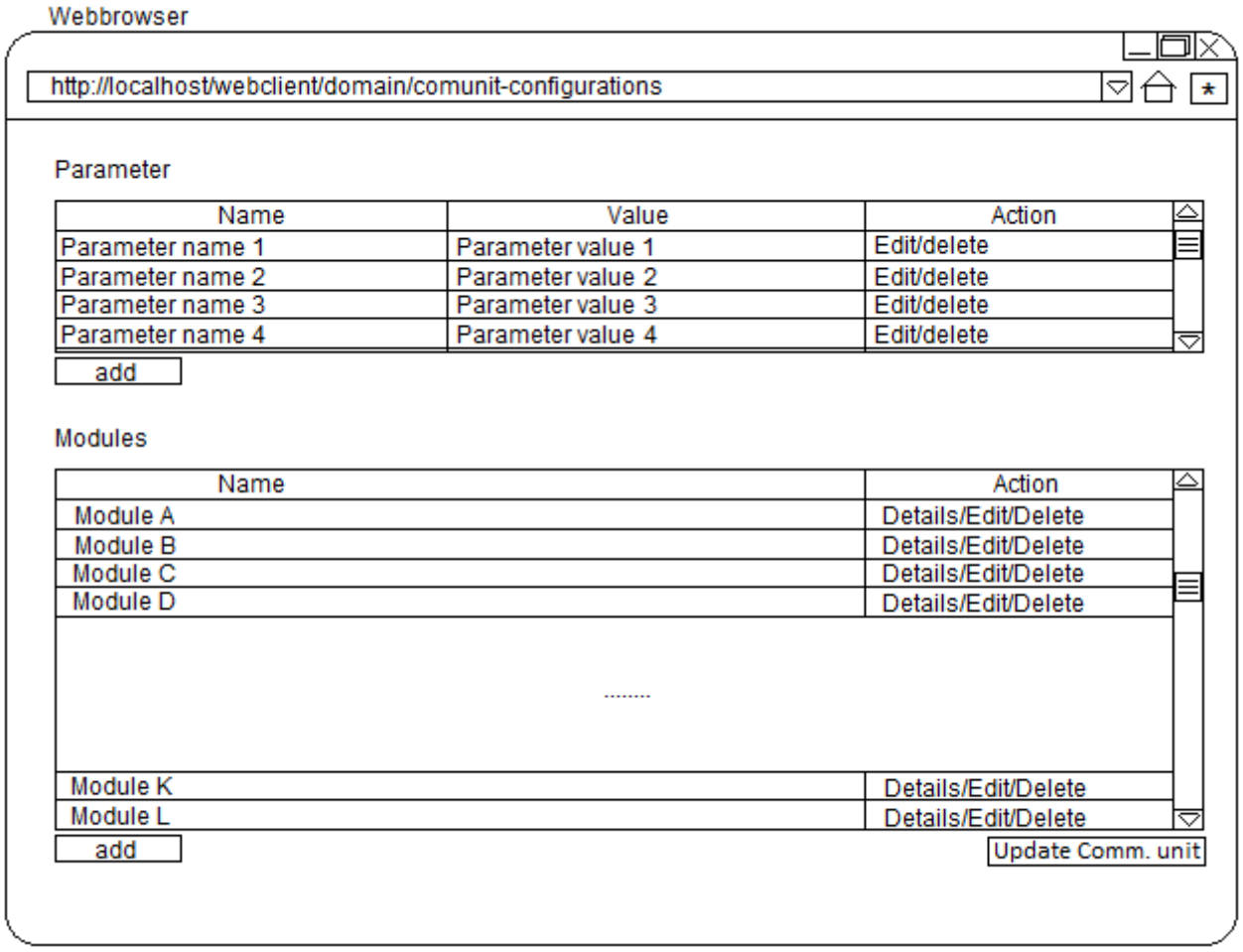
Figur 8 - MVC design pattern

Der er mange frameworks som anvender Model-View-Controller arkitekturen for at separere modellen (data laget) med controlleren (forretnings laget) fra viewet (brugergrænsefladen). Dette mønster er generelt opfattet som "good practice" fra et objekt orienteret synsvinkel, siden det giver mulighed for at dele koden op i logiske dele og genbrug af koden[5].

Spring har sin egen implementation af denne arkitektur som jeg ikke vil gå ind på nu. Jeg vil nu lave et sidespring og skifte emne ved at lave et indledende udkast for hvordan skærmbilledet kommer til at se ud på Miljøet.

[4] http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html
[5] http://en.wikipedia.org/wiki/Web_application_framework

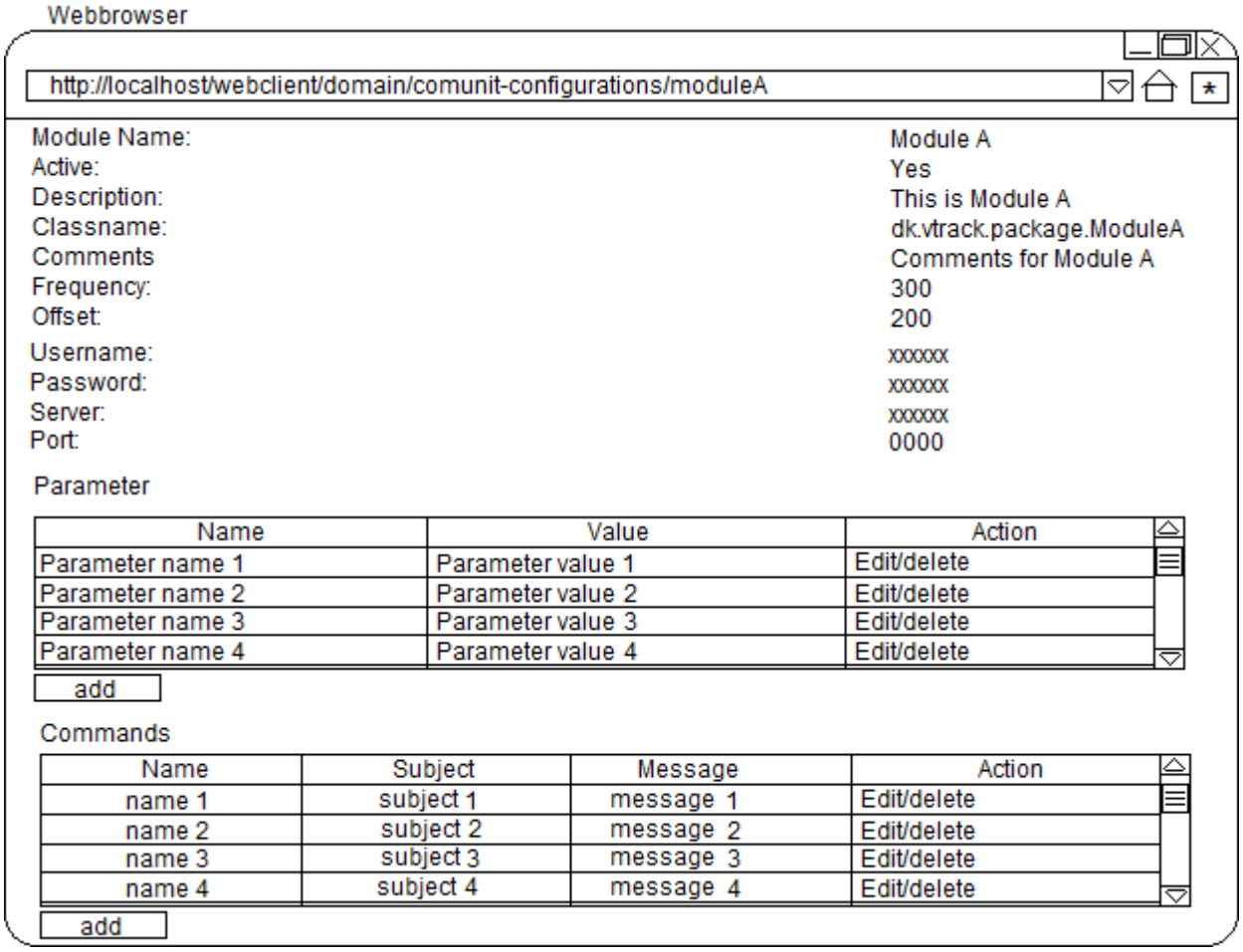
Som man kan se fra analysen i forrige afsnit, så er det modulerne der udgør kerne elementerne i en konfiguration.
For at give en helheldsoplevelse af konfigurationen, så har jeg valgt at lave 2 sider. På den første side kan man få en oversigt at alle modulerne og parametrene (fra `ParameterCollection` tagget, se Figur 2). Et eksempel på dette er vist på Figur 9.
Knappen ”Update Comm. unit” anvendes til at signalere at kommunikationsenheden skal opdateres, med de nye værdier som er ændret gennem Miljøet.



Figur 9 - Skærmbillede 1 af Miljøet (parametre og moduler)

Ud fra hvert modul (på Figur 9) har man have mulighed for at kunne se alle modulets detaljer. Dette gøres ved at klikke på knappen ”Details”. Denne handling vil fører brugeren videre til et skærmbillede vist på Figur 10. Her kan man se alle modulets tilhørende data (Parametre, kommandoer og hosts). Dette er selvfølgelig afhængigt af modultypen (listener module/timed module) og hvilke data der er tilknyttet til modulet.

Skærmbilledet i figuren nedenfor viser detaljerne for at timed module.



Figur 10 - Skærmbillede 2 af Miljøet (Modul detaljer – Timed module)

4.3 Konklusion

Det her er et optimeringsprojekt for et system som er fuldt funktionelt den dag i dag. Derfor har det været nødvendigt at lave en foranalyse af projektet for at få en fuld forståelse af hvad der skal laves og for at videre udvikle projektet til den ønskede tilstand.
Vi har i dette afsnit realiseret dette. Vi har kigget på konfigurationsfilen og hvordan denne anvendes af kommunikationsenheden. Derefter har vi tag et kig på præsentationsenheden og hvilken arkitektur denne er opbygget af og vi har lavet et par udkast til hvordan vi gerne vil vise konfigurationsdata til brugeren.

Jeg har nu opsamlet ny viden omkring systemet og kender til endnu flere detaljer omkring systemet og har etableret en god forståelse af problemstillingen.
Denne viden kan bruges til at lave en mere specifik og målrettet planlægning, analysering og specificering af det nye system og af de software implementeringer som projektet kommer til at består af.

5 Features

Den nye løsning til vTrack består af 4 hoveddele, nemlig de dele der vedrører databasen, kommunikationsenheden og det nye miljø. Midt i projekt perioden diskuterer vi også om at lave en xml parser der er i stand til at kunne oversætte de nuværende konfigurationsfiler om til SQL kald, således de kan indsættes i databasen. Derfor består løsningen til dette projekt også af denne del, alt i alt 4 hoveddele. Her er nogle nøgle features disse dele kommer til at have.

5.1 Database

Databasen har til formål at persistere data på en struktureret måde. Databasen anvendes i denne opgave i forbindelse med at persistere data fra vTrack.xml filen, ud fra den database arkitektur der udvikles til formålet.
Hvilken database og hvilken Query sprog der anvendes, henvises der til afsnit 21 Teknologier.

5.2 Kommunikationsenheden

Kommunikationsenheden er en af de vigtige komponenter i vTrack. Denne enhed fungerer som motoren af vTrack forhold til, at det er den som henter data, hvorimod de andre enheder laver en repræsentation af dette. Den sørger for al kommunikation etableres mellem de eksterne datakilder og at der kan modtages og afsendes data.
Denne enhed har hidtil anvendt en konfigurationsfil til at etablere forbindelser til de eksterne enheder. Dette skal nu laves om, således den læser sin konfiguration fra databasen og etablere de samme forbindelser som den altid har gjort.

5.3 Miljø

Med det nye miljø bliver det endnu nemmere for brugerne at kunne tilpasse konfigurationsparametrene i forbindelse med opsætningen kommunikationsenheden.
Det nye miljø implementeres efter CRUD principperne (Create, Read, Update, Delete), som er de mest grundlæggende funktioner der anvendes i forbindelse med data som gemmes permanent.
Så snart der sker ændringer i konfigurationen gennem dette miljø vil disse blive tilpasset af kommunikationsenheden, når brugeren giver et signal (gennem ”update” knappen på Figur 9).

Af sikkerhedsmæssige årsager skal der tages hensyn til vTracks rollebaseret rettighedssystem, således at kun autoriserede personer har adgang til at kunne modificere de nødvendige data.

Det nye miljø integreres med præsentationsenheden i vTrack, så miljøet bliver tilgængeligt for dens brugere og forudbestemte layouts, templates m.v. anvendes (bl.a. stylesheeting).

5.4 vTrack Xml Parser

vTrack xml parser er et særskilt program der udvikles til det formål at kunne oversætte den nuværende konfigurationsfil til kommunikationsenheden om til SQL kald. Programmet vil derfor gøre det muligt for vores nuværende kunder at installere den nye version af kommunikationsenheden på en hurtigere og en mere effektiv måde, end en manuel oversættelse af konfigurationsfilen.

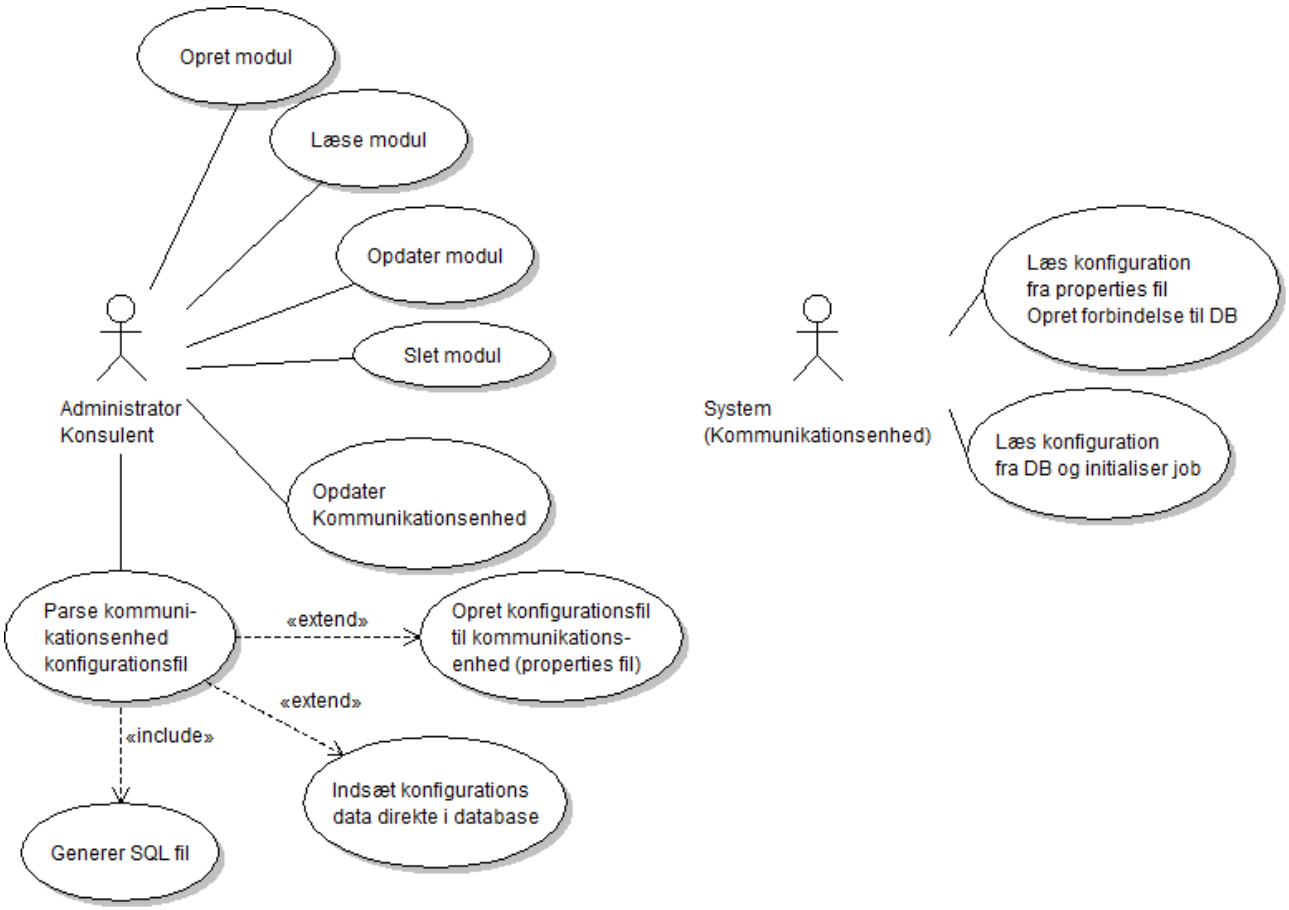
6 Kravspecifikation

Nedenfor er der sat en række funktionelle og non-funktionelle krav, som skal overholdes og implementeres i forbindelse med projektet.

6.1 Funktionelle krav

- Databasescriptet skal fysisk gøre det muligt at kunne implementere den modellerede databasestruktur i den anvendte database (Oracle database). Modellen skal tage hensyn til den nuværende database implementation der anvendes af vTrack.
- Man bør kunne ændre konfigurationsparametrene efter CRUD principperne (Create, Read, Update, Delete).
- Kommunikationsenheden skal kunne opdateres når der kommer opdateringer til konfigurationen via Miljøet som brugeren interagerer med.
- Xml parseren skal kunne oversætte data fra konfigurationsfilen om til SQL kald, således disse kan indsættes i databasen.
- Xml parseren skal kunne etablere forbindelse til database, således systemet kan indsætte den genereded data fra xml filen direkte i en database.
- Xml parseren skal give mulighed for brugeren at kunne konfigurere og generere properties filen som anvendes af kommunikationsenheden (Jævnfør foranalysen af konfigurationsfilen i afsnit 4.1.1).
- Kommunikationsenheden skal kunne læse data fra en properties fil således den kan oprette forbindelse til databasen hvor resten af konfigurationen er gemt. Resten af konfiguration indlæses herfra og de rigtige tråde (job) initialiseres.

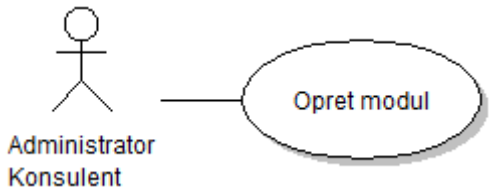
Figur 11 nedenfor viser de overstående krav som et use-case diagram. Bemærk at konfigurationen består af flere forskellige elementer end modul som angivet i figuren (se evt. Figur 2). Disse detaljer kommer vi ind på senere i rapporten. Figuren er forsimplet da formålet med denne figur er, at jeg gerne vil vise at man skal kunne udfører de forskellige CRUD principper på alle elementer som en konfiguration består af. Grunden til at moduler er valgt, er fordi disse er kerne elementerne i en konfiguration.



Figur 11 - Use Case diagram

Nedenfor findes der use-case beskrivelser som forklarer yderligere detaljer om de krav der er til programmet og til figuren overfor. Beskrivelserne tager udgangspunkt i succes scenariet, men alternative flows beskrives også.

De første 4 use case beskrivelser nedenfor tager kun udgangspunkt i CRUD funktionerne på modul elementet i konfigurationen. De samme use cases er også gældende for de andre elementer som konfigurationen består af.



Use case id: U01.
Use case navn: Oprettelse af modul til kommunikationsenhed konfiguration.

Primær aktør: Administrator/konsulent.

Prækonditioner: Der er 2 prækonditioner der kan være gældende før denne use case udføres:
1 – Der er udviklet (programmeret) et nyt modul i kommunikationsenheden som skal gøres aktivt, således modulet kører når kommunikationsenheden starter op eller når der gives besked gennem Miljøet.
2 – Der findes allerede en konfiguration databasen som kommunikationsenheden anvender. Det nye modul oprettes som en del til denne konfiguration.

Postkonditioner: Modulet oprettes som en del af konfigurationen og er gemt i databasen og er klar til at blive anvendt af kommunikationsenheden enten når der genstartes eller når brugeren opdater kommunikationsenheden gennem Miljøet (se use case id: U05).

Basic flow:

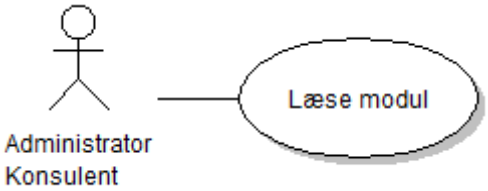
1. Aktøren logger ind og finder konfigurationen hvor modulet skal indsættes.
2. Aktøren indsætter data til at oprette det nye modul.
3. Systemet valider data der er angivet (sanity test).
4. Systemet gemmer de givne data i databasen.
5. Systemet opdater listen over moduler der findes for konfigurationen.

Alternative flows:

- Aktøren er logget ind men har ikke de nødvendige rettigheder for at kunne oprette modul (fejlsituation opstået ved punkt 2):
    1. Systemet viser en fejlside i stedet for siden hvor modulet oprettes.
    2. Aktørens rettigheder ændres således at aktøren får rettigheder til at oprette moduler.
    3. Basic flow udføres forfra.

- Brugeren har angivet ugyldigt data (fejlsituation opstået ved punkt 3):
    1. Systemet afviser de givne data, da de ikke består sanitetstesten.
    2. Systemet viser en fejlmeddelelse til aktøren og giver henvisninger til hvad der bør gøres.
    3. Aktøren retter fejlen mht. meddelelsen og udfører basic flow fra punkt 2.

- Systemet kan ikke indsætte data i databasen (fejlsituation opstået ved punkt 4):
    1. Systemet viser fejlen til aktøren (fejlmeddelelse fra database).
    2. Aktøren retter fejlen mht. meddelelsen og udfører basic flow forfra.



Use case id: U02.
Use case navn: Aflæsning af modul fra kommunikationsenhed konfiguration.

Primær aktør: Administrator/konsulent.

Prækonditioner: Der findes mindst en konfiguration med tilhørende parametre og moduler til kommunikationsenheden som er gemt i databasen.

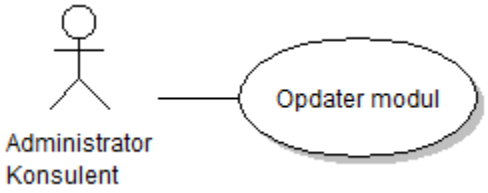
Post konditioner: Konfigurationen aflæses via Miljøet af aktøren.

Basic flow:

1. Aktøren logger ind og finder det modul der skal aflæses for en konfiguration.
2. Aktøren klikker på en knappen ”Details” (som på Figur 9) og får vist alle detaljerne omkring modulet og dens tilhørende værdier (som fx på Figur 10).

Alternative flow:

- Aktøren er logget ind men har ikke de nødvendige rettigheder for at kunne læse modul (fejlsituation opstået ved punkt 2):
    1. Systemet viser en fejlside i stedet for siden hvor modulets detaljer vises.
    2. Aktørens rettigheder ændres således at aktøren får rettigheder til at læse moduler.
    3. Basic flow udføres forfra.



Use case id: U03.
Use case navn: Opdatering af et modul til kommunikationsenhed konfiguration.

Primær aktør: Administrator/konsulent.

Prækonditioner: Der findes en konfiguration i databasen med diverse moduler hvor mindst et af dem skal opdateres, da der er evt. kommet kodemæssige ændringer for kommunikationsenheden eller andre forhold som der skal tages hensyn til (bl.a. fra kundens side).

Postkonditioner: De nye data omkring modulet er angivet og er gemt i databasen. De nye opdateringer er klar til at blive anvendt af kommunikationsenheden enten når der genstartes eller når brugeren opdater kommunikationsenheden gennem Miljøet (se use case id: U05).

Basic flow:

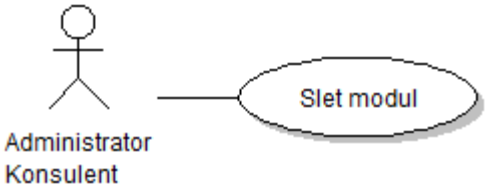
1. Aktøren logger ind og finder modulet som skal opdateres.
2. Aktøren indsætter de nye data for modulet og gemmer værdierne.
3. Systemet valider data der er angivet (sanity test).
4. Systemet gemmer de givne data i databasen.
5. Systemet opdater listen over moduler der findes for konfigurationen.

Alternative flows:

- Aktøren er logget ind men har ikke de nødvendige rettigheder for at kunne opdatere modulet (fejlsituation opstået ved punkt 2):
    1. Systemet viser en fejlside i stedet for siden hvor modulet opdateres.
    2. Aktørens rettigheder ændres således at aktøren får rettigheder til at opdater moduler.
    3. Basic flow udføres forfra.

- Brugeren har angivet ugyldigt data (fejlsituation opstået ved punkt 3):
    1. Systemet afviser de givne data, da de ikke består sanitetstesten.
    2. Systemet viser en fejlmeddelelse til aktøren og giver henvisninger til hvad der bør gøres.
    3. Aktøren retter fejlen mht. meddelelsen og udfører basic flow fra punkt 2.

- Systemet kan ikke indsætte/opdater data i databasen (fejlsituation opstået ved punkt 4):
    1. Systemet viser fejlen til aktøren (fejlmeddelelse fra database).
    2. Aktøren retter fejlen mht. meddelelsen og udfører basic flow forfra.



Use case id: U04.
Use case navn: Sletning af et modul fra kommunikationsenhed konfiguration.

Primær aktør: Administrator/konsulent.

Prækonditioner: Et af modulerne (implementation) i kommunikationsenheden er slettet og dette skal angives i konfigurationen, således der ikke opstår fejl.

Postkonditioner: Modulet og dens tilhørende værdier er slettet fra databasen og anvendes ikke længere af kommunikationsenheden.

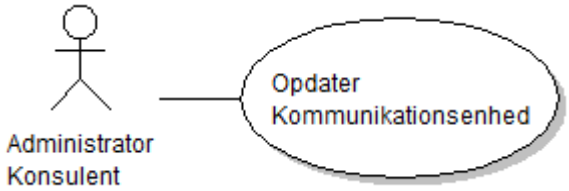
Basic flow:

1. Aktøren logger ind og finder det modul som skal slettes.
2. Aktøren klikker på en knap som sletter modulet og dens tilhørende data/værdier (som fx på Figur 9).
3. Systemet sletter modulet og tilhørende værdier fra databasen.
4. Systemet opdater listen over moduler der findes for konfigurationen.

Alternative flow:

- Aktøren er logget ind men har ikke de nødvendige rettigheder for at kunne slette modulet (fejlsituation opstået ved punkt 2):
    1. Systemet viser en fejlside i stedet for at slette modulet.
    2. Aktørens rettigheder ændres således at aktøren får rettigheder til at slette moduler.
    3. Basic flow udføres forfra.

- Systemet kan ikke slette data i databasen (fejlsituation opstået ved punkt 2):
    1. Systemet viser fejlen til aktøren (fejlmeddelelse fra database).
    2. Aktøren retter fejlen mht. meddelelsen og udfører basic flow forfra.



Use case id: U05.
Use case navn: Opdatering af kommunikationsenheden.

Primær aktør: Administrator/konsulent.

Prækonditioner: Der er 4 prækonditioner der kan være gældende før denne use case udføres (mindst ét af dem skal være gældende):
1 – Der er fortaget ændringer (opdateringer) i konfigurationen således at de aktive moduler i kommunikationsenheden skal opdateres med nye værdier.
2 – Der er oprettet mindst et nyt modul eller mindst et deaktiveret modul er sat aktiveret som skal træde i kraft/ startes op (som ny tråd) af kommunikationsenheden.
3 – Et eller flere aktive moduler i kommunikationsenheden skal stoppes, da disse er enten blevet slettet eller sat som deaktiveret i konfigurationen.
4 – Der er foretaget ændringer i konfigurationen (ændringer i parametre fra `ParameterCollection` tagget (se Figur 2)) som kræver fuld genstart af kommunikationsenheden (jeg vil senere i rapporten komme ind på hvorfor dette kræver en fuld genstart).

Postkonditioner: Ændringerne træder i kraft og de gældende moduler opdateres, startes og stoppes med hensyn til de nyeste værdier i konfigurationen. I tilfælde af punkt 4 i prækonditioner genstartes alle tråde i kommunikationsenheden fra bunden.

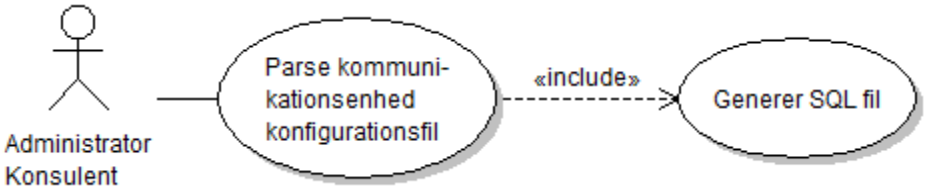
Basic flow:

1. Aktøren logger ind og finder knappen som opdaterer kommunikationsenheden (se Figur 9).
2. Brugeren klikker på knappen.
3. Systemet indsætter en række i databasen, som indikation på at kommunikationsenheden skal opdateres.
4. Denne rækker indlæses af kommunikationsenheden.
5. Kommunikationsenheden sørger for at opdatere, starte og stoppe de moduler og parametre der gør sig gældende. En fuld genstart kan muligvis fremkomme i tilfælde af prækondition 4.

Alternative flow:

- Aktøren er logget ind men har ikke de nødvendige rettigheder for at kunne opdatere kommunikationsenheden (fejlsituation opstået ved punkt 2):
    1. Systemet viser en fejlside i stedet for at opdater kommunikationsenheden.
    2. Aktørens rettigheder ændres således at aktøren får rettigheder til at opdater kommunikationsenheden.
    3. Basic flow udføres forfra.

- Systemet kan ikke indsætte række i databasen (fejlsituation opstået ved punkt 3):
    1. Systemet viser fejlen til aktøren (fejlbesked fra database).
    2. Aktøren retter fejlen mht. meddelelsen og udfører basic flow forfra.



Use case id: U06.
Use case navn: Parsing af konfigurationsfil til kommunikationsenheden.
Bemærkninger: Denne use case inkluder use casen for generering af SQL fil.

Primær aktør: Administrator/konsulent.

Prækonditioner: Der er 2 prækonditioner der skal gøre sig gældende før denne use case kan udføres:
1 – Der findes en gyldig og valid konfigurationsfil til kommunikationsenheden som har været brugt hidtil.
2 – Databasearkitekturen der er udviklet til at kunne gemme indholdet af konfigurationsfilen er implementeret og er sat i drift i en (Oracle) database.

Postkonditioner: Der genereres en SQL fil der indeholder data fra konfigurationsfilen og kan anvendes til at indsætte i databasen.

Basic flow:

1. Aktøren angiver stien til konfigurationsfilen.
2. Systemet tjekker for valid XML.
3. Systemet tjekker for gyldig XML.
4. Systemet oversætter (parser) den givne fil.
5. Systemet opretter en SQL fil og indsætter data fra den oversatte fil.
6. Aktøren indsætter data fra SQL filen i databasen.

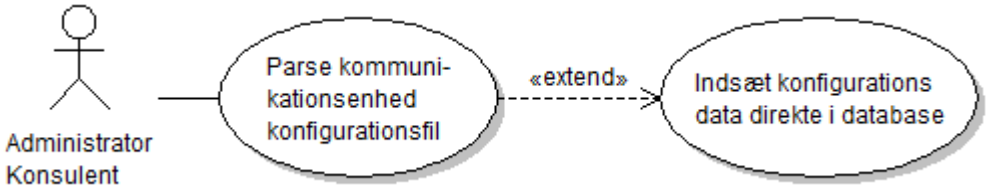
Alternative flows:

- Den givne fil er ikke valid (fejlsituation opstået ved punkt 2):
    1. Programmet stopper med oversættelsen og beder aktøren om at kontrollere XML filen.
    2. Aktøren retter fejlen.
    3. Basic flow udføres forfra.

- Den givne fil er ikke gyldig (fejlsituation opstået ved punkt 3):
    1. Programmet stopper med oversættelsen og beder aktøren om at kontrollere XML filen.
    2. Aktøren retter fejlen.
    3. Basic flow udføres forfra.

- Oversættelsen fejler (fejlsituation opstået ved punkt 4):
    1. Programmet stopper med oversættelsen og viser fejlen til aktøren.
    2. Aktøren retter fejlen.
    3. Basic flow udføres forfra.

- Systemet kan ikke oprette SQL fil (fejlsituation opstået ved punkt 5):
    1. Processen stoppes og der vises en fejlmeddelelse til aktøren.
    2. Aktøren retter fejlen.
    3. Basic flow udføres forfra.



Use case id: U07.
Use case navn: Indsættelse af data fra konfigurationsfil til database.
Bemærkninger: Denne use case er en udvidelse af use case U06.

Primær aktør: Administrator/konsulent.

Prækonditioner: Samme forhold som U06 samt at databasen skal være aktiv (må ikke være slukket).

Postkonditioner: Data fra konfigurationsfilen indsættes automatisk i databasen af systemet. Således punkt 6 i basic flow for use case U06 ikke behøver at blive foretaget af aktøren.

Basic flow:

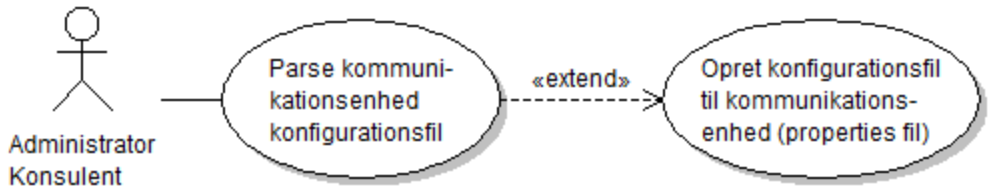
- Efter systemet har oversæt den givne fil i punkt 5 i use case U06, skal data indsættes i databasen, dette sker ved:
    1. Aktøren indtaster oplysninger om den database hvor der skal etableres forbindelse til.
    2. Systemet tjekker om der er en gyldig forbindelse til databasen.
    3. Systemet indsætter data i databasen.

Alternative flows:

- Aktøren mangler at angive data omkring databasen (fejlsituation opstået ved punkt 2):
    1. Systemet viser en fejlmeddelelse til aktøren om at der er nogle felter er mangler at blive udfyldt.
    2. Aktøren indsætter manglende data.
    3. Basic flow udføres forfra.

- Der findes ingen gyldig database forbindelse (fejlsituation opstået ved punkt 2):
    1. Systemet viser fejlmeddelelse til aktøren samt henvisninger til hvad der bør gøres.
    2. Basic flow udføres forfra.

- Systemet fejler ved indsættelse af data (fejlsituation opstået ved punkt 3):
    1. Der oprettes en SQL fil der indeholder de kald som fejlede. For hvert kald er der en beskrivelse af fejlen og hvad der bør gøres for at rette fejlen.
    2. Aktøren retter disse fejl og indsætter disse kald manuelt.



Use case id: U08.
Use case navn: Konfiguration af properties fil.
Bemærkninger: Denne use case er en udvidelse af use case U06.

Primær aktør: Administrator/konsulent.

Prækonditioner: Samme forhold som U06, dog er prækondition nummer 2 ikke et must.

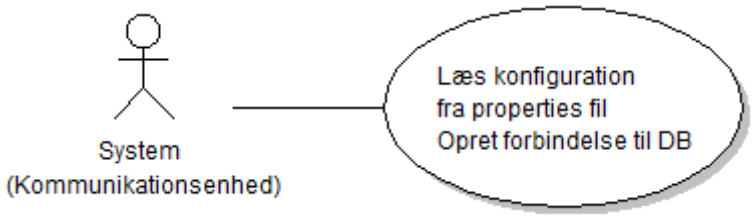
Postkonditioner: Der oprettes en properties fil som kan anvendes af den nye version af kommunikationsenheden til at etablere forbindelse til database og oprette logfiler.

Basic flow:

- Efter systemet har oversæt den givne fil i punkt 5 i use case U06 og evt. udført use case U07, konfigureres en properties fil som kan anvendes af kommunikationsenheden:
    1. Systemet viser både parametre (fra `ParameterCollection` tagget, se Figur 2) og moduler (fra `ModuleCollection` tagget, se Figur 2) som systemet har indlæst af konfigurationsfilen.
    2. Aktøren vælger de parametre og moduler som properties filen skal bestå af.
    3. Systemet opretter en properties fil med de parametre og moduler som brugeren har valgt.

Alternative flow:

- Systemet kan ikke oprette properties fil (fejlsituation opstået ved punkt 3):
    1. Processen stoppes og der vises en fejlmeddelelse til aktøren samt hvad der anbefales at gøre.
    2. Aktøren retter fejlen.
    3. Basic flow udføres forfra.



Use case id: U09.
Use case navn: Indlæsning af properties fil.

Primær aktør: System (kommunikationsenheden).

Prækonditioner: Der findes en properties fil som indeholder data til at kunne oprette det modul der opretter logfiler og etabler forbindelse til databasen som bl.a. indeholder resten af konfigurationen til kommunikationsenheden.

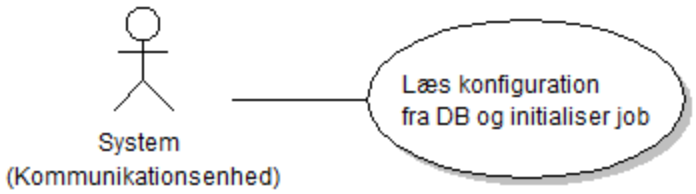
Postkonditioner: Der er oprettet logfiler og etableret forbindelse til databasen således den resterende konfiguration til kommunikationsenheden kan indlæses.

Basic flow:

1. Systemet indlæser properties filen og bearbejder disse data.
2. Systemet opretter det modul (timed module objekt) som opretter log fil og etablerer database forbindelse med passende data (parametre og host).
3. Modulet oprettes og initialiseres af kommunikationsenheden (som en aktiv tråd).

Alternative flow:

- Properties filen findes ikke eller kan ikke indlæses:
    1. Systemet stopper omgående og skriver til logfilen/konsollen om at properties fil mangler.
    2. Fejlen rettes og kommunikationsenheden genstartes manuelt.



Use case id: U10.
Use case navn: Indlæsning af kommunikationsenhed konfiguration fra database.

Primær aktør: System (kommunikationsenhed).

Prækonditioner: Der er 2 prækonditioner der skal gøre sig gældende før denne use case kan udføres:
1 – Use case U09 skal være udført, således kommunikationsenheden har en gyldig database forbindelse.
2 – Der skal findes mindst en konfiguration i databasen som kommunikationsenheden kan anvende. Evt. skal/kan use case U06 anvendes.

Postkonditioner: Kommunikationsenheden har oprettet og kører de samme tråde (jobs) ligesom den gjorde, da den gamle konfigurationsfil (xml fil) blev anvendt.

Basic flow:

1. Systemet anvender database forbindelsen til at indlæse resten af konfigurationen:
    a. Systemet indlæser parametre fra database (`ParameterCollection`).
    b. Systemet indlæser aktive timed modules fra database samt tilhørende værdier (parametre, host og commands).
    c. Systemet indlæser aktive listener modules fra database samt tilhørende værdier (parametre).
2. Systemet opretter og initialiserer parametrene (`ParameterCollection`).
3. Systemet opretter og initialiserer timed modulerne (som tråde).
4. Systemet opretter og initialiserer listener modulerne (som tråde).
5. Systemet frigiver database forbindelsen.

6.2 Non-funktionelle krav

Ud over de funktioneller krav er der listet en række non-funktionelle krav til projektet.
Her er disse:

- Database strukturen bør optimeres således der er minimal redundans (dette løses ved normalisering).
- Kommunikationsenheden skal påvirkes mindst muligt under og efter den nye løsning er implementeret.
- Det skal undgås at kommunikationsenheden skal genstartes manuelt når der ændres i konfigurationsparametrene.
- Adgangen til det nye miljø skal være rollebaseret med hensyn til vTracks rettighedssystem.
- Der må ikke være yderligere programmeringsmæssige ændringer af hverken indlæsning af kommunikationsparametrene (i kommunikationsenheden) eller det Miljø hvori brugerne kan ændre værdierne. Med andre ord betyder dette at løsningen skal kunne håndtere nye udvidelser/nye moduler i kommunikationsenheden uden af dette skal gå på kompromis med Miljøet eller kommunikationsenheden.
- Det nye Miljø der skal udvikles mht. tilpasningen af konfigurationsparametrene skal kunne afvikles på en Apache Tomcat Server.

7 Produkt karakteristika

I dette afsnit vil jeg gå mere i dybden med nogle af de non-funktionelle krav til projektet. Underafsnittene nedenfor tager udgangspunkt i FURPS+ modellen. Der er dog nogle af overskrifterne fra denne model som jeg ikke har valgt at tage med (fx funktionalitet).

Sikkerhed
I henhold til vore kunders database bør vi få adgang til disse i forbindelse med opdateringen af vTrack databasen med de udvidelser dette projekt bidrager med.

Der forventes ikke et behov for at tage hensyn til nogen sikkerhedsmekanismer ved opstart af kommunikationsenheden, da denne skal læse parametre fra databasen og starte tråde op på baggrund af dette. Dette skyldes at der er ingen brugerinteraktion med systemet som muligvis kunne forårsage ravage og at enheden har en bestemt procedure som den skal følge.

Der forventes at produktet anvender det rollebaserede rettighedssystem der findes i vTrack, når brugerne vil ændre på konfigurationsparametrene i det grafiske Miljø.

Pålidelighed
Programmet skal være i stand til at kunne udføre de ønskede funktioner uden særlige problemer. Brugeren skal have følelsen af at computeren/programmet er i gang med at eksekvere deres handlinger ved interaktion.
Yderligere må systemet ikke være sårbart for input fejl. Hvis brugeren indtaster forkerte data bør der komme sigende fejlmeddelelser ud, som brugeren kan handle ud fra.

Brugervenlighed
Ved interaktion bør brugeren altid have oplevelsen af hvad systemet gør nu og det har en betydning for brugeroplevelsen ved anvendelse af det grafiske Miljø.
Systemet bør være så enkelt og overskueligt som muligt, således brugeren ikke er i tvivl om hvordan systemet skal bruges.
Dette gælder også for vTtrack XML Parseren.

Performance
Da hvert modul kører i sin egen tråd og da der i teorien ingen grænser for hvor mange tråde der kan køres af en computer, bør man stadig være opmærksom på området. Man bør dog aldrig nogensinde overbelaste systemet baseret på det hardware computeren stiller til rådighed.
Antallet af moduler/tråde kan have en indflydelse på performance og derfor er responstiden et vigtigt krav.

Databasen skal indeholde så lidt redundans som muligt, således der ikke overføres store mængder af data til og fra applikationen som egentlig ikke kan anvendes fornuftigt.

Platform
Programmet kan køres stort set på alle platforme Windows, Mac OSX, Linux samt andre platforme. Det eneste krav er, at platformen understøtter java og der er installeret en Oracle database samt en server der kan køre Java web applikationer (fx Apache Tomcat).

Der er dog minimums krav som de forskellige programmer kræver:

Java[6]:

- Memory: 64 – 128 MB RAM (afhængig af operativsystem).
- Disk: 58 – 98 MB (afhængig af operativsystem).
- En browser (Mozilla Firefox er anbefalet).

Oracle 10g[7]:

- Memory: 512 MB RAM (anbefalet).
- Disk: 1,5 GB.
- CPU: 200 MHz minimum.

Det har ikke været muligt at finde ud af minimumssystemkravet til Apache Tomcat serveren. Men baseret på min undersøgelse af følgende links [8] og [9] så er ikke noget konkret svar på hvor meget hardware computeren skal have, da dette afhænger af hvad serveren skal kører.

En ren Apache Tomcat server fylder ca. 10-15 mb, kræver ca. 30 MB RAM og en 300 MHz Pentium processor til at kunne kører.

For at kunne køre vTrack, samt de andre applikationer og for at få en nogenlunde tilfredsstillende responstid anbefales der derfor at man anvender et system med følgende hardware (minimum):

- Memory: 1 GB ram.
- CPU: 1 GHz (Intel Core).
- 3-5 GB harddisk.

Vedligeholdelse
Udvidelserne som dette projekt bidrager med, skal være minimalt i henhold til vedligeholdelse af kildekoden. Derfor skal løsningen til projektet være åben for fremtidige udvidelser af vTrack og for nye modul implementeringer, uden at gå på kompromis med opsætningen af konfigurationen til kommunikationsenheden.

Implementering
I henhold til implementering af dette projektet anvendes Oracle 10/11g Express Edition, Java, Spring framework og Apache Tomcat server der er i stand til at kunne køre webapplikationen. Se mere i afsnit 21 Teknologier om de forskellige teknologier og programmer der anvendes i projektet.

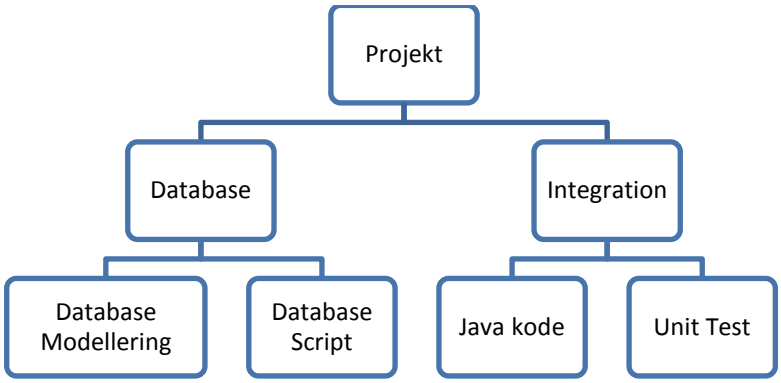
[6] http://www.java.com/en/download/help/sysreq.xml
[7] http://www.dba-oracle.com/t_oracle_10g_pc_hardware_windows_requirements.htm
[8] www.jguru.com/faq/view.jsp?EID=585982
[9] http://tomcat.10.n6.nabble.com/System-requirements-for-running-Apache-Tomcat-on-Windows-Box-td2091878.html

8 Løsningsstrategi

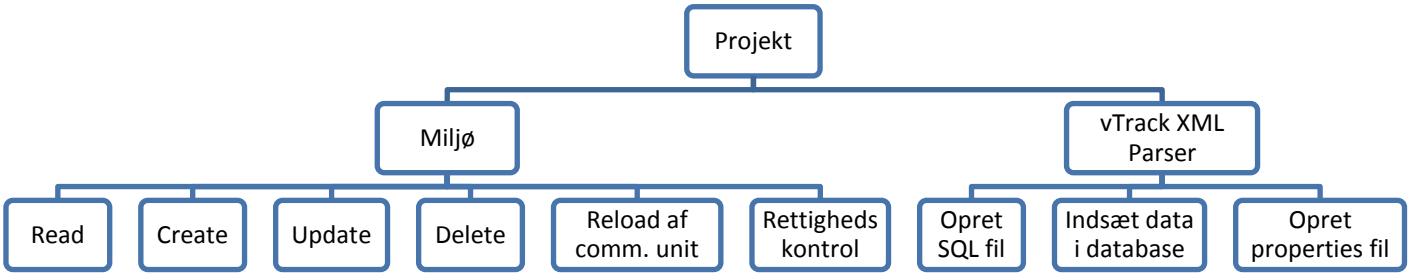
Jeg har valgt at opdele projektet i 4 forskellige hovedaktiviteter som skal udvikles i forbindelse med løsning af den givne problemstilling. Hver af disse dele kan betragtes som selvstændige enheder.
De 4 dele har jeg yderligere opdelt i underprojekter. Ved at gøre dette har jeg bedre mulighed for at estimere disse og derved få et mere troværdigt, samlet tidsestimat af projektet som en helhed. Dette vil også hjælpe mig løbende at samle op på de enkelte underprojekter, med henblik på at omfordele resurser anderledes, hvis et projekt ser ud til at komme bagud med hensyn til tidsplanen (se afsnit 9.1 Tidsplan).

Jeg har valgt en strategi som går ud på først at efterprøve min ide i forsimplet form som prototyper og diagrammer. På denne måde kan jeg forhåbentligt hurtigt vurdere om min ide kan implementeres indenfor tidsplanen, eller om det er nødvendigt at udtænke en ny løsning.

Figur 12 og Figur 13 giver et overblik af disse 4 hovedaktiviteter og deres underopgaver. Af hensyn til pladsen er figuren nedenfor delt i 2 dele (figuren bør ses som et samlet).



Figur 12 – Løsningsstrategi, del 1



Figur 13 – Løsningsstrategi, del 2

Her er en kort beskrivelse af de overordnede opgaver som skal laves:

1. Database:
    - Database modellering: Data fra vTrack.xml filen er analyseret, der skal nu laves en databasestruktur som kan mappe dette.
    - Databasescript: Der udvikles et databasescript som realiserer den overstående databasestruktur.

2. Integration:
    - Der skal udvikles ny software, som er i stand til at kunne bruge data fra databasen i stedet for xml filen og samtidig sørge for at den nuværende funktionalitet bevares.
    - Unit test: Der laves test til validering af den nuværende funktionalitet af vTrack bevares i den nye løsning med database og properties fil.

3. Miljø:
    - Der skal udvikles et miljø, hvori man kan tilpasse konfigurationsparametrene efter behov og som er designet efter CRUD principperne.
    - Miljøet skal samtidig gøre det muligt for kommunikationsenheden at reloade konfigurationen når der sker ændringer.
    - Miljøet skal understøtte vTracks rollebaseret rettighedssystem.

4. vTrack XML Parser:
    - På baggrund af den gamle konfigurationsfil skal parseren kunne oprette en SQL fil.
    - Brugeren skal have mulighed for at indsætte data i databasen direkte fra programmet.
    - Brugeren skal have lov til at konfigurere og oprette properties filen som kan anvendes af den nye version af kommunikationsenheden.

9 Projekt planlægning

Projektet har en varighed over 18 aktive uger (minus onsdage, hvor jeg har et andet kursus) og strækker sig over en tidshorisont på ca. 5 kalender måneder (fra mandag den 3. september 2012 til mandag den 28. januar 2013). Der afholdes ferie, fridage og vintereksamen m.m. i overensstemmelse med undervisningssemesteret på DTU for efteråret 2012. Se mere på følgende link[10]. Projektet har et omfang på 20 ECTS points som svarer til 560 timers arbejde.

Der planlægges at de forskellige aktiviteter som er omfattet i projektet, udvikles i den rækkefølge som de står i afsnit 8 Løsningsstrategi. Nedenfor findes der en tabel der viser en detaljeret aktivitetsoversigt, tidsestimering for de forskellige aktiviteter og en endelige status over de forskellige opgaver.

Bemærk at opgave 7 og 8 er diverse hængepartier der er opstået under projektets forløb og er derfor ikke med her i tabellen nedenfor, men kan findes i status rapporterne (se mere afsnit 11 Status rapporter). Grunden til at opgave 9 (Xml parser) står længere nede end opgave 7 og 8 er fordi denne opgave blev planlagt senere i projektet.

For at vise progressionen under projektets forløb, har jeg valgt at tage felt med som indiker den procentvise færdighedsgrad ifølge tidsplanen (dette kommer til at give mere mening når Figur 15 vises).
Dette felt hedder ”Opfyldt ifølge tidsplan” og skal fortolkes på denne måde:

- Hvis den er på 100 % så betyder dette at færdiggørelsen af opgaven følger tidsplanen (præcist).
- Hvis den er over 100 % så betyder dette at opgaven er mere færdigt end det, den burde i forhold tidspunktet.
- Hvis den er mindre en 100 % så viser den hvor stor en andel opgaven der færdigt i forhold til det, opgaven burde være til tidspunktet.

Der anbefales at tage et kig på statusrapporterne på Cd-rommen for at se udviklingen af projektet (med ca. 14 dages mellemrum).

Resultatet af feltet ”aktuelt brugt tid” for en opgave på niveau x.y er summen af dens underopgaver på et underniveau svarende til x.y.z. Denne stuktur følges hvis der er flere underniveauer.

[10] http://shb.dtu.dk/Default.aspx?documentid=2806&Language=da-DK&lg=&version=2011/2012

1	Database Modellering	Status	Aktuelt forbrugt tid (timer)	Opfyldt ifølge tidsplan(%)	Faktisk på dato(%)
1.1	Analyse af xml filen, således strukturen af denne kendes og som vil være til grundlag for databasestrukturen der skal udvikles til løsning af opgaven	Færdig	4	100 %	100 %
1.2	Analyse af xsd skema for xml filen. Dette kan hjælpe med at finde ud af hvordan man formelt skal angive de forskellige tags/elementer i xml filen.	Færdig	4	100 %	100 %
1.3	Udvikling af E/R diagram, som er integrerbar med den nuværende database arkitektur	Færdig	8	100 %	100 %
1.4	Optimering af modellen. Dette indebærer normalisering således redundante data undgås.	Færdig	6	100 %	100 %
1.5	Rapport skrivning	Færdig	10	100 %	100 %
Tidsestimat: 2 uger	Total:	Færdig	32	100 %	100 %

2	Databasescript	Status	Aktuelt forbrugt tid (timer)	Opfyldt ifølge tidsplan(%)	Faktisk på dato(%)
2.1	Der skal udvikles et databasescript der realiser databasemodellen der er udviklet i den overstående opgave	Færdig	12	100 %	100 %
2.2	Kontroller at den udviklede kode virker og kan integreres med den nuværende databaseløsning	Færdig	4	100 %	100 %
2.3	Test at data fra vTrack.xml og de refererede filer kan mappes i databasen	Færdig	5	100 %	100 %
2.4	Rapport skrivning	Færdig	10	100 %	100 %
Tidsestimat: 1 uge	Total:	Færdig	31	100 %	100 %

3	Integration	Status	Aktuelt forbrugt tid (timer)	Opfyldt ifølge tidsplan(%)	Faktisk på dato(%)
3.1	Forstå koden bag kommunikationsenheden	Færdig	4	100 %	100 %
3.1.1	Udvikling af UML diagrammer (klasse diagrammer + sekvensdiagrammer)	Færdig	4	100 %	100 %
3.2	Finde det sted hvor koden skal placeres, således at den nye løsning påvirker Kommunikationsenheden mindst muligt	Færdig	1	100 %	100 %
3.3	Udvikling af kode	Færdig	34	100 %	100 %
3.3.1	Analyse + Design (OOAD)	Færdig	8	100 %	100 %
3.3.2	Udvikling af diverse SQL forespørgsel til det formål at kunne læse data fra database (select * from) (erstatning af xml fil)	Færdig	3	100 %	100 %
3.3.3	Udvikle java kode til formålet	Færdig	23	100 %	100 %
3.4	Rapport skrivning	Færdig	25	100 %	100 %
Tidsestimat: 3 uger	Total:	Færdig	64	100 %	100 %

4	Unit test	Status	Aktuelt forbrugt tid (timer)	Opfyldt ifølge tidsplan(%)	Faktisk på dato(%)
4.1	Opstilling af test scenarier til opgave 3	Færdig	10	100 %	100 %
4.2	Udvikling af unit test kode til de opstillede test scenarier	Færdig	10	100 %	100 %
4.3	Rapport skrivning	Startes	10	100 %	100 %
Tidsestimat: 2 uger	Total:	Færdig	30	100 %	100 %

5 Miljø	Status	Aktuelt forbrugt tid (timer)	Opfyldt ifølge tidsplan(%)	Faktisk på dato(%)
5.1 Analysering af præsentationsenheden og finde ud af hvordan den virker.	Færdig	8	100 %	100 %
5.2 Udviklingen af de forskellige funktioner af CRUD paradigmet	Færdig	48	100 %	100 %
5.2.1 Læse (read)	Færdig	12	100 %	100 %
5.2.1.1 Udvikling af "select * from" forespørgsler	Færdig	5	100 %	100 %
5.2.1.2 Udvikling af Java kode (Spring)	Færdig	7	100 %	100 %
5.2.2 Opdater (update)	Færdig	12	100 %	100 %
5.2.2.1 Udvikling af "update" forespørgsler	Færdig	5	100 %	100 %
5.2.2.2 Udvikling af Java kode (Spring)	Færdig	7	100 %	100 %
5.2.3 Oprette (Create)	Færdig	12	100 %	100 %
5.2.3.1 Udvikling af "insert into" forespørgsler	Færdig	5	100 %	100 %
5.2.3.2 Udvikling af Java kode (spring)	Færdig	7	100 %	100 %
5.2.4 Slette (Delete)	Færdig	12	100 %	100 %
5.2.4.1 Udvikling af "delete from" forespørgsler	Færdig	5	100 %	100 %
5.2.4.2 Udvikling af Java kode (Spring)	Færdig	7	100 %	100 %

5.3 Implementering af genstart funktionalitet for kommunikationsenheden	Ikke færdig	15	80 %	80 %
5.4 Implementering af rettighedskontrol	Færdig	4	100 %	100 %
5.5 Rapport skrivning	Færdig	25	100 %	100 %
Tidsestimat: 10 uger Total:	Færdig	100	96 %	96 %

6 Dokumentation	Status	Aktuelt forbrugt tid (timer)	Opfyldt ifølge tidsplan(%)	Faktisk på dato(%)
6.1 Rapport skrivning samt sidste kode kommenteres	Færdig	55	100 %	100 %
Tidsestimat: 2 uger Total:	Færdig	55	100 %	100 %

9 Xml Parser	Status	Aktuelt forbrugt tid (timer)	Opfyldt ifølge tidsplan(%)	Faktisk på dato(%)
9.1 Kode til xml validering (valid xml)	Færdig	3	100 %	100 %
9.2 Kode til validering af gyldig xml (xml vs. xsd skema)	Færdig	3	100 %	100 %
9.3 Parse xml data til java objekter	Færdig	3	100 %	100 %
9.4 Generering af SQL kode	Færdig	7	100 %	100 %
9.5 GUI	Færdig	19	100 %	100 %
9.6 Forretningslogik	Færdig	7	100 %	100 %
9.7 Refaktorering	Færdig	7	100 %	100 %

9.8 Rapport skrivning	Færdig	15	100 %	100 %
Tidsestimat: 12 uger Total:	Færdig	64	100 %	100 %

Tabel 1 - Aktivitetsoversigt og tidsestimering

Ifølge tidsestimatet burde projektet ca. vare 22 uger, men som sagt er dette er kun et estimat og derfor er det ikke sikkert om der vil afviges fra praksis når projektet udføres. For at kunne nå det hele er jeg nødt til at lave nogle af opgaverne parallelt, dette ses i Figur 15.

De 22 uger kommer fra at jeg har estimeret xml parser opgaven (opgave 9) til at vare ca. 2 uger. Grunden til at tidsestimatet til xml parseren er sat til 12 uger i tabellen overfor er pga. jeg havde en tidsperiode på 12 uger, hvor jeg kunne gøre opgaven færdig i og det var et projekt som blev udviklet sideløbende med de andre aktiviteter.

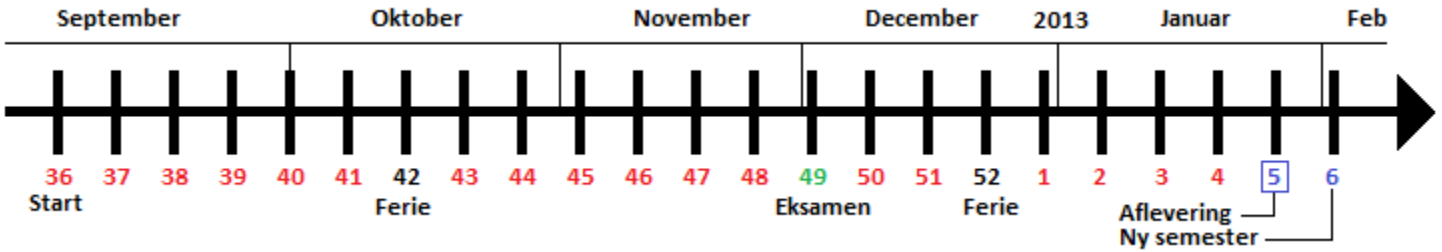
Den procentviseske færdighedsgrad (”Faktisk på data(%)”) er beregnet på et gennemsnit af opgaverne på niveau x.y. Dette betyder samtidig også at alle underopgaver for en opgave vægtes ligeligt.

Bemærk at opgave 5 ikke er færdig (opgaven er 96 % færdig, se tabel overfor). Jeg blev ikke færdig med opgave 5.3 implementeringen af genstart funktionalitet til kommunikationsenheden (se også Figur 15). Vi kommer ind på dette senere i rapporten (afsnit 16.6).

Timerne i feltet ”Aktuelt forbrugt tid” er et cirka tal der angiver hvor meget tid der er brugt på opgaven i det omfang hvor man ikke har gået i stå. Den reelle tid der er brugt i projektet kan ses i status rapporterne og i timeregnskabet i Appendix E.

9.1 Tidsplan

Jeg forventer at starte på projektet mandag den 3. september 2012 (uge 36), og aflevere projektet mandag den 28. januar 2013.

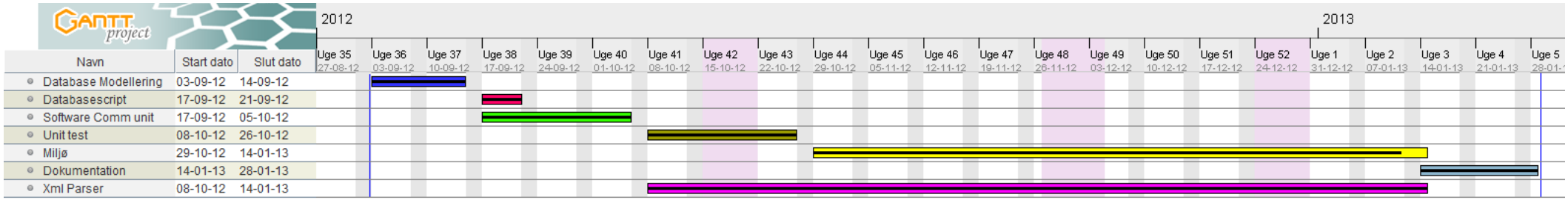


Figur 14 - Projekt tidsplan

Uger markeret med rødt på Figur 14 er de uger hvor det er planlagt at arbejde med projektet. Under eksamen (markeret med grønt) forventer jeg at ikke at arbejde med projektet. Projektet afleveres mandag i uge 5 2013. Alt i alt burde projektet varer 18 uger i et tidsrum på 5 kalender-måneder.

Her er et Gantt diagram (Figur 15) der viser hvornår de forskellige aktiviteter (de 7 hovedopgaver) vil finde sted. Uger der er markeret med rødt (uge 42, 48-49 og 51) antyder ferie eller eksamen, altså tidsperioder hvor der ikke arbejdes på projektet.

Der er sat code-freeze den 14. januar 2013, hvor der ikke vil udvikles flere funktioner til projektet. Fra denne dato vil jeg kun vil fokuseres på at samle og gøre rapporten og projektet færdigt.



Figur 15 - Gantt diagram

10 Arbejdsgang

Arbejdet udføres primært selvstændigt med vejledning af følgende personer beskrevet i afsnit 19 Vejledere.
Virksommen anvender SCRUM som udviklingsmetode. Derfor har jeg valgt at bruge denne udviklingsmetode, dog i en forsimplet udgave.
Der defineres ingen roller (da jeg er alene om projektet) og der udvikles heller ikke Backlog items (use-case diagrammet og beskrivelserne fra afsnit 6.1 Funktionelle krav anvendes).
Hvert sprint har en varighed på max 14 dage og min. 7 dage. Hvert Sprint afsluttes med en status rapport (se næste afsnit). Her diskuteres projektets status og hvad der skal laves i næste sprint.
Jeg forventer at arbejde med projektet primært hos virksomheden, dog kan møder foregå på DTU, hos DTU-vejlederen.

11 Status rapporter

For at kunne se fremskridtet i projektet har jeg ca. hvert 14. dage afleveret en status rapport til mine vejledere. Dette har været vigtigt så mine vejledere kunne følge med i udviklingen i projektet og derved havde mulighed for at gribe ind, i tilfælde af noget skulle gå galt eller når der var behov for hjælp.
Hver rapport indeholder et timeregnskab, et resume og en oversigt af hvilke opgaver der er lavet i perioden og hvilke opgaver der skal laves i den næste periode. Til sidst i rapporten findes en konklusion.
Løbet af projekt perioden er der skrevet 9 status rapporter. Alle rapporter findes i den afleverede Cd-rom (se mere i appendiks C).

12 Afgrænsning

For at have et succesfuldt projekt kan det muligvis være nødvendigt at afgrænse projektet af en eller anden grund (fx mangel er ressourcer eller forkert estimering af projektets indhold).
Hvis der afviges fra tidsplanen afgrænses projektet med nogle af de funktionelle krav der finder sted i Miljøet. Dette aftales nærmere afhængig af projektets fremdrift mellem vejlederne. Der laves yderligere dokumentations i rapporten hvis der sker noget angående.

13 Intro

Jeg har valgt at dele rapporten i 4 dele, hvor hver del omhandler løsningsbeskrivelserne til de 4 underprojekter fra Figur 12 og Figur 13.

Del 1 omhandler løsningen til databasearkitekturen som er til grundlag for persistering af konfigurationen.

I del 2 kommer vi ind på løsningen til kommunikationsenheden, således den bliver i stand til at kunne læse konfigurationen fra databasen, som bruges til at etablere forbindelser til de eksterne datakilder.

Derefter vil i del 3 komme ind på Miljøet og en løsningsbeskrivelse på denne. Vi vil også i dette afsnit komme dybere ind på frameworket Spring og hvordan denne anvendes i dette projekt.

Endelige i del 4 vil vi finde løsningen på xml parseren. Således kan vi omsætte konfigurationen om til SQL sætninger og udfører disse i databasen.

14 Del 1 – Database

Dette er det første delprojekt som jeg har lavet i forbindelse med projektet. I dette afsnit vil jeg gå mere dybden med databasen og databasemodellen som skal udvikles. Afsnit her starter med en analyse af xsd skemaet der anvendes i projektet. På baggrund at dette vil jeg undersøge hvordan data gemmes og hvordan de er struktureret. Herefter udvikles der en databasearkitektur, der er i stand til at kunne gemme konfigurationen til kommunikationsenheden i databasen. Til sidst udvikles SQL kode til formålet og der udføres diverse tests.
Jeg ligger først ud med at lave en kort beskrivelse af hvad et xsd skema er og hvad det bruges til.

14.1 Xsd skema

Der findes xml som er syntaksmæssig ”velformuleret” (valid xml) og der findes xml filer som er ”gyldig” xml. Ved at overholde de simple xml regler som fx kun et rod element, start og slut tags og korrekt indlejring af tags, vil man have xml dokumenter som er velformuleret, men et velformuleret xml er nødvendigvis ikke gyldig xml. Derfor har det været nødvendigt for Visma Consulting at udvikle sådan et xsd skema, som kan bruges til at validere xml filer med og derved have gyldigt xml data som kan anvendes i vTrack.

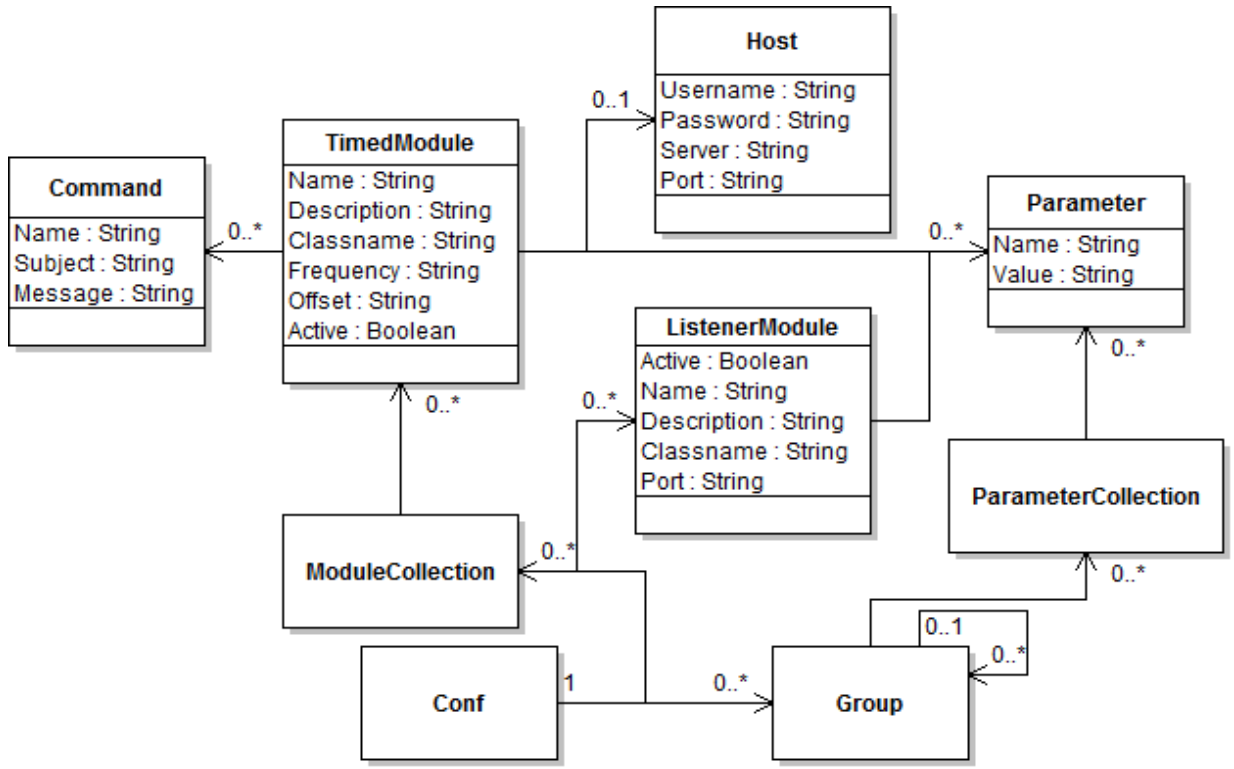
Disse xsd skemaer beskriver strukturen af hvordan et xml dokument skal sættes op. I disse filer er det fx muligt at finde ud af, hvilke elementer og attributter der kan fremkomme, hvilke elementer der er børn elementer og se hvilke datatyper der anvendes i xml filen.
En anden fordel ved xsd er at der anvendes xml syntaks og derfor er det relativt nemt at gå til.

(Xsd filen der analyseres i det næste afsnit hedder configuration.xsd og ligger i mappen schema i Comm unit projektet.)

14.2 Analyse af configuration.xsd

For at finde ud af hvordan den formelle struktur af vTrack.xml filen er opbygget har jeg analyseret configuration.xsd filen. Denne fil indeholder et sæt ”regler” for hvordan vTrack.xml filen skal angives. Dette har været relevant i forbindelser med at finde ud af hvilke entiteter der findes i domænet og deres tilhørende attributter.

Ved analysen har jeg haft objekt orienteret tankegang og har prøvet at lave et klassediagram over xsd skemaet. Resultatet findes i Figur 16 nedenfor:



Figur 16 - Diagram over configuration.xsd

Associationerne mellem 2 objektet betyder at der er et parent-child forhold til hinanden i xml filen. Multipliciteten mellem objekterne kommer fra, at der er xsd filen er specificeret følgende occurrence indicators: `minOccurs="0"` og `maxOccurs="unbounded"`.

Rod elementet i vTrack.xml fil er `Conf` tagget, som kan indeholde elementer af datatypen `Group`.
`Group` kan have en samling af `ModuleCollection` og `ParameterCollection` tags.
`ParameterCollection` tagget indeholder en række `Parameter` objekter.
`Parameter` objekterne indeholder 2 værdier nemlig et navn og en værdi (key/value par). Dette anvendes i forbindelse med initialiseringen af de forskellige variabler for modulerne i kommunikationsenheden.
`Group` har samtidig også en refleksiv association til sig selv.

`ModuleCollection` tagget kan kun indeholde 2 typer af moduler, henholdsvis `TimedModule` og `ListenerModule`. Begge disse moduler indeholder en række parametre. `TimedModule` tagget kan yderligere have en række `Commands` tag og eventuelt en enkelt `Host` tag.
`Host` objektet indeholder oplysninger om en server, hvor der skal etableres forbindelse til samt et username og password (til serveren).
`Commands` er en række kommandoer som specificer nogle opgaver som kommunikationsenheden skal udfører.

Xsd skemaet findes i den vedlagte cd-rom (configuration.xsd). Se mere i Appendix C.

14.3 Databasemodellen

Det gik hurtigt op for mig at der var nogle strukturelle (designmæssige) unødvendigheder i xml filen som kan undgås at have med i databasen modellen. Fx det med at have `Conf`, `Group`, `ParameterCollection` og `ModuleCollection`, da de ikke indeholder data der er relevante for konfigurationen.

Ud fra min analyse og den viden jeg har tilegnet mig fra xml og xsd filerne (jf. afsnit 4.1.1 og 14.2) har jeg inddelt data i 3 hoved kategorier (entiteter), som jeg skal tage hensyn til når jeg skal udvikle database modellen:

1. ParameterCollection
    - Indeholder en liste af parametre med variable navne og tilhørende værdier. Disse vil fra nu af kaldes for "globale" parametre i resten af rapporten, med mindre andet er angivet. Disse parametre er "synlige" for alle moduler (ligesom vi kender det med globale parametre i programmering).
2. TimedModule
    - Et modul der specificer en opgave for kommunikationsenheden. Denne opgave køres med en bestemt frekvens (tidsskeduleret). Parameternavne og dens tilhørende værdier bruges til at initialisere opgaven med bestemte værdier. Til nogle af modulerne kan der være specificeret at modulet skal oprette forbinde til en server med et givet password og username. Disse moduler kan også godt have Commands.
3. ListenerModule
    - Indeholder data således modulet kan oprette en serversocket forbindelse specificeret med en port. Herefter vil modulet afvente indkomne kald på denne. Disse moduler kan også indeholder parametre hvis det er nødvendigt for forbindelsen.

Den endelige databasemodel som jeg er kommet frem til er lavet over flere iterationer (modellen nedenfor er 5. udkast) og i samarbejde med Klaus Mogensen (fra virksomheden). Hvordan jeg er kommet frem til denne model og hvilke overvejelser og ændringer der har været løbende, henvises der til Appendix A (Databasemodel udvikling). Den endelige model ses på Figur 19 - Final database model.

Mens jeg udviklede databasemodellen var der en ting som jeg lage mærke til. I xml filer kan man kommentere linjer ud ved at bruge følgende syntaks: `<!-- -->` og dette var brugt meget i xml filen. Her er et par eksempler fra vTrack.xml filen:

```
<!-- YO --> <Parameter><Name>gw.selfsigned.connection.url</Name><Value>http://localhost:61616/queue</Value></Parameter>
<!-- YO --> <Parameter><Name>gw.signed.connection.url</Name><Value>http://localhost:61617/queue</Value></Parameter>
```

Figur 17 - vTrack.xml eksemplel 1

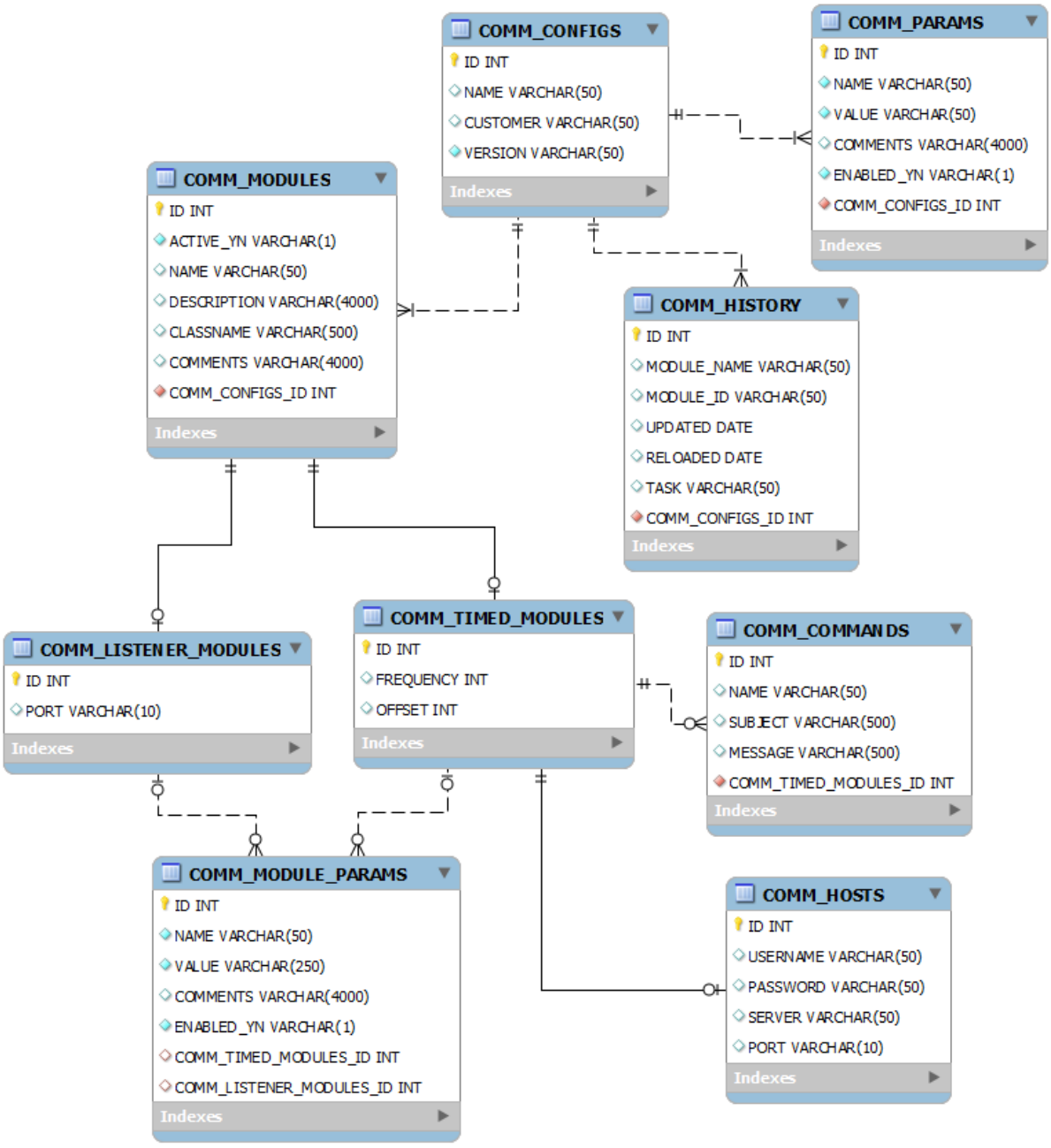
Og

```
<!-- Parameter><Name>mail.imap.ssl.enable</Name><Value>true</Value></Parameter --> <!-- Enable ssl -->
<!-- Parameter><Name>mail.imap.starttls.enable</Name><Value>true</Value></Parameter --> <!-- Enable STARTTLS -->
```

Figur 18 - vTrack.xml eksempel 2

Som udvikler kan man godt lide at bruge kommentar, da man på denne måde hurtigt kan slå en linje til og fra, uden at slette noget kode og som måske kan bruges senere (fx som i Figur 18). Nogle gange kan man også skrive en reminder til sig selv eller til sine kollegaer ved siden af tagget (fx i Figur 17).

Nu da vi går over til en ny løsning med database syntes jeg at det var lidt synd, at vi nu ikke kunne bruge denne feature. Derfor har jeg i nogle af tabellerne nedenfor gjort det muligt, at man kan skrive en kommentar (`COMMENTS`) (ligesom fig. 17) og har gjort det muligt at sætte den aktiveret (`ENABLED_YN`) (ligesom fig. 18) (se `COMM_PARAMS` og `COMM_MODULE_PARAMS` tabellerne).



Figur 19 - Final database model

`COMM_CONFIGS` er "hovedtabellen" og er den tabel som har ansvaret for at "holde styr" på konfigurationen. Her har jeg gjort det muligt at man kan angive navnet på konfigurationen, hvem kunden er og hvilken version af konfigurationen kunden kører med (dette var ikke muligt med xml filen).
Denne tabel har også en unique constraint som sørger for at konfigurationsnavnene skal være unikke. Med andre ord betyder dette, at man ikke kan have 2 konfigurationer med samme navn.

Der er 3 tabeller som har foreign key til denne tabel, dette er COMM_MODULES, COMM_HISTORY og COMM_PARAMS tabellerne. Disse tabeller indeholder henholdsvis data fra ParameterCollection og ModuleCollection taggene fra xml filen.
COMM_HISTORY tabellen er en tabel som kom på senere i projektet. Det er en tabel som er beregnet til at holde styr på hvilke moduler der ændret (gennem Miljøet) og hvornår disse ændringer har trådt i kraft af kommunikationsenheden. Feltet "task" angiver hvilken opgave kommunikationsenheden skal påtage sig ved det gældende modul. Vi vil komme tilbage til denne tabel senere i rapporten, hvor jeg vil gå mere i dybden og beskrive hvad den bruges til.

Da mange af værdierne i ListenerModule og TimedModule tagget var ens, har jeg lavet 3 tabeller, 1 master og 2 details tabeller. Master tabellen (COMM_MODULES) indeholder værdier der er ens for de 2 tags hvorimod de adskilte værdier findes i details tabellerne (COMM_LISTENER_MODULES og COMM_TIMED_MODULES).
Master tabellen har også en unique constraint som sørger for, at man til én konfiguration ikke kan have moduler med ens navne (comm_configs_id og name skal være unikke).
Begge disse modultyper kan have Parameter tags (se Figur 2 og Figur 3). Disse data gemmes i COMM_MODULE_PARAMS tabellen. Yderligere kan COMM_TIMED_MODULES tabellen have referencer til COMM_HOSTS og COMM_COMMANDS tabellerne som vist i figuren overfor.

Da nogle af associationerne mellem tabellerne skal være i forholdet 1-til-1, har jeg valgt at lave id'et på nogle af tabellerne både som primary key og foreign key. Fx er id'et i tabellen COMM_HOSTS både primary key og foreign key til id'et i COMM_TIMED_MODULES tabellen. Det samme er gældende for id'et tabellerne COMM_LISTENER_MODULES/COMM_TIMED_MODULES til id'et i COMM_MODULES. På denne måde har jeg "sparet" et felt for foreign keys.

Ved 1-til-mange relationer har jeg oprettet en foreign key i tabellen som peger på en anden tabels primary key. Der skal dog lige bemærkes at der findes en constraint på COMM_MODULE_PARAMS tabellen som sørger for at kun én af de 2 foreign keys kan sættes (exlusive or) (felterne comm_timed_modules_id og comm_listener_modules_id er foreign keys i comm_module_params tabellen). Dette er lavet således man kan adskille mellem hvilke parametre der til tilhører hvilket modul. Se mere i SQL koden (Appendix C (Database create comm unit tables.sql)).

Der forventes at tags i ParameterCollections i xml filen (se Figur 2) gemmes i tabellen COMM_PARAMS hvorimod parametre der findes i ListenerModule og TimedModule tagget gemmes i COMM_MODULE_PARAMS. Dette er gjort for at adskille tingene fra hinanden, selvom de godt kunne ligge i den samme tabel.

Der skal dog lige bemærkes at det program (MySQL Workbench) jeg har brugt til at lave Figur 19 med og de første 4 figurer i Appendix A, ikke understøttede nogle af datatyperne. Alle VARCHAR bør ændres til VARCHAR2 og INT bør ændres til NUMBER(15) (de 15 angiver antallet af cifre som feltet kan indeholde)!
De gule nøgler er indikation på primary key for tabellen,
turkis diamant: værdien kan ikke være null,
rød diamant: foreign key som ikke kan være null,
punkteret rød diamant: værdien er foreign key til en anden tabel (kan være null),
punkteret turkis diamant: værdien kan være null.

14.4 SQL kode

Nu da jeg havde udviklet databasemodellen, blev det nemmere for mig at udvikle SQL koden til at oprette tabellerne vist på Figur 19 i databasen. Den endelige SQL kode kan findes på den vedlagte Cd-rom, se mere i Appendix C (Database create comm unit tables.sql).
I dette afsnit vil jeg lave en kort beskrive SQL koden og hvordan den er opbygget. I mit eksempel tager udgangspunkt i tabellen `COMM_CONFIGS`.

```
1  -- ------------------------------------------------------------------------------
2  -- COMM_CONFIGS TABLE - 1
3
4  CREATE TABLE COMM_CONFIGS
5  (
6    ID NUMBER(15) NOT NULL ,
7    NAME VARCHAR2(50 BYTE) NULL,
8    CUSTOMER VARCHAR2(50 BYTE) NULL,
9    VERSION VARCHAR2(50 BYTE) NOT NULL
10 );
11
12 -- PRIMARY KEY FOR COMM_CONFIGS
13 ALTER TABLE COMM_CONFIGS
14   ADD CONSTRAINT COMM_CONFIGS_PK PRIMARY KEY (ID);
15
16 -- ADD UNIQUE CONSTRAINT - NAMES MUST BE UNIQUE
17 ALTER TABLE COMM_CONFIGS
18   ADD CONSTRAINT UNIQUE_CONFIG_NAMES UNIQUE (NAME);
19
20 COMMENT ON TABLE COMM_CONFIGS IS 'Head table for the configuration to the comm. unit';
21
22 -- SEQUENCE FOR COMM_CONFIGS TABLE
23 CREATE SEQUENCE COMM_CONFIGS_SEQ
24   INCREMENT BY 1
25   START WITH 1
26   MAXVALUE 9999999999999999999999999999
27   MINVALUE 1
28   NOCYCLE
29   CACHE 15
30   NOORDER;
```

Figur 20 - SQL kode eksempel

Ved linje 1 og 2 inden tabelen oprettes er der en linje som adskiller oprettelsen mellem 2 tabeller og nedenunder denne linje, er der en kommentar der angiver navnet på tabellen efterfulgt af et tal. Dette tal er en indikation på hvilken rækkefølge tabellen skal oprettes. I dette eksempel skal `COMM_CONFIGS` tabellen oprettes som den først tabel. Rækkefølgen er vigtig, da man kan risiker at oprette en tabel som har en foreign key eller en anden reference til en tabel som endnu ikke er oprettet i databasen (derved kan der opstå foreign key constraint fejl).

Ved linje 3 – 10 oprettes en tabel i databasen med navnet `COMM_CONFIG`. Denne tabel har 4 værdier, et id, navn, kunde og version. Bemærk at version ikke kan være null, da den laveste version nummer anbefales at være 1.0.0 hos vores kunder ved start. På denne måde kan man holde en forudbestemt syntaks og vide hvordan denne skal sættes op, ved fremtidig vedligeholdelse af konfigurationen.

Ved linjerne mellem 12-14 manipuler jeg tabellen således at vi opretter en constraint, hvor jeg sætter en primary key til tabellen. Dette er feltet Id.

I dette eksempel er der ingen foreign key constraint, men for de relevante tabeller vil dette være specificeret under denne linje.

Efter denne linje har jeg typisk valgt at oprette der indexes for foreign key(s), således databasen kan finde data hurtigere og på en mere effektiv måde. Dette vil kunne ses i koden, men da der er ingen foreign keys i denne tabel oprettes den ikke. Oracle opretter indexes automatisk på primary keys ved oprettelse af tabellerne.

For hver tabel findes er en kommentar. Koden for oprettelse af kommentar for tabellen ses på linje 20 i den overstående figur.

Ved linjerne mellem 23 og 30 oprettes der en sequence som er specifikt for Oracle databaser. Dette er en måde at få variabler til at inkrementere automatisk ved indsættelse af en ny række. Dette bruges i forbindelse med Id'erne for tabellerne. Så man kan oprette nye rækker i tabellen uden at tage hensyn til værdien af Id kolonnen.
Når man skal oprette en ny række i tabellen skal man bruge funktionen `COMM_CONFIG_SEQ.NEXTVAL` ved Id feltet. I MySQL (som det jeg er vant til) bruges koden: auto_increment.

Den resterende SQL kode til oprettelse af tabeller er opbygget stort set på samme måde som beskrevet i dette afsnit og derfor vil jeg ikke gå i dybden med denne.

Der findes også SQL filer til at slette de oprettede tabeller og sekvenser, til at tømme tabellen, give adgang til kommunikationsenheden til at skrive og læse fra databasen, samt en SQL fil indsætte data i databasen. Jeg vil ikke komme ind på disse filer i dette afsnit. Der findes en "Readme.txt" i samme mappe som scriptet for oprettelse af tabellerne på cd-rommen. Denne tekst fil giver en kort beskrivelse af alle disse filer. De kan bruges i tilfælde noget går galt (se mere i Appendix C).

14.5 Navne konventioner

I forbindelse med navngivning af de forskellige tabeller i vTrack databasen anvendes der navne konventioner for de forskellige tabeller for dette projekt. Dette gør det nemmere at finde rundt i arkitekturen, da databasen indeholder mange tabeller og at Visma også har et andet produkt (vCatch) som anvender den samme database.

Jeg har fået vide fra min vejleder Klaus Mogensen at jeg skulle bruge præfixet "COMM_" i forbindelse med navngivningen af database tabellerne.

Udover dette har jeg anvendt Oracles navne konventioner[11]. Fx slutter tabelnavne altid med flertals s.
Primary keys er indikeret ved at tilføje endelsen _PK på tabelnavne (se Figur 20 linje 14).
Foreign keys er indikeret ved at tilføje præfikset FK_ efterfulgt af variable navnet.
Da nogle af foreign keys der har den samme navn (Fx i tabellerne `COMM_PARAMS` og `COMM_MODULES`) er der tilføjet nummerering i endelsen af navnet.

Indexes starter med præfikset IDX_ efterfult af navnet af på den kolonne som skal indekseres.

[11] http://ss64.com/ora/syntax-naming.html

Jeg har udover disse konventioner også anvendt forkortelser fx parameters er forkortet til params og configuration er forkortet til config. Dette skyldes antallet af karakter som tabelnavnene må indeholde.

14.6 Test

Der er udført 3 tests mht. til databasen og for at validere koden virker som den skal.

Den første test har til formål i at finde ud af SQL scriptet der opretter tabellerne på Figur 19 kan køres på en Oracle database (SQL scriptet ”Database create comm unit tables.sql”) uden at det skaber problemer. Her blev der testet på 2 databaser. Henholdsvis på en lokal database og vtra10 som er en intern database i virksomheden.
Koden blev kørt i Oracle SQL developer og outputtet stammer herfra (den faktiske output findes i Appendix B).

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
1.1	Køre udviklet SQL script på en lokal database.	Se ”Database create comm unit tables.sql” filen	Tabeller, constraints, indexes og sequences oprettes succesfuldt	Se appendix B2	Ok
1.2	Køre udviklet SQL script på vtra10 databasen.	Se ”Database create comm unit tables.sql” filen	Tabeller, constraints, indexes og sequences oprettes succesfuldt	Se appendix B2	Ok

Tabel 2 - Database test

Fra denne test kan vi konkludere at SQL koden opretter tabellerne og de tilhørende constraints bliver oprettet succesfuldt ud fra den databasemodel der er lavet i forbindelse med projektet.

Den anden test som jeg har udført var integrationstesten. Denne test går ud på at finde ud af om data i vTrack.xml filen og de andre xml filer kan mappes i databasen. Jeg har lavet en forsimplet test ved at tage forskellige xml tags henholdsvis fra en lokal vTrack.xml fil og en fra vores kunde (USA) og har lavet to nye xml filer på baggrund af dette (Simple local vtrack.xml og Simple USA vtrack.xml – se Appendix C. De originale filer og Simple USA vtrack.xml er ikke vedlagt på cd-rom pga. forretningshemmeligheder og erklæringer). På baggrund af dette, har jeg lavet SQL kode der kan mappe dette i databasen. SQL koden findes i den vedlagte CD-rom (Simple local vtrack test.sql, dog ikke for USA).
For hver af de originale filer har jeg taget 2 parametre fra `ParameterCollection` tagget, 2 fra `TimedModules` og 2 fra `ListernerModule`. Alle tags er valgt tilfældigt!
Hvis dette kan mappes, så kan jeg være sikker på at resten af xml filen også kan mappes, da de er opbygget på samme måde. Da mange af modulerne indeholder gentagelser af samme tags har jeg også valgt at slette disse og udvælge nogle få.
Pga. forretningshemmeligheder og fortrolighed er tags som Username og Password ændret. Dette har dog ingen betydning for denne test. Testen er kun kørt på den lokale database.

Tabellen nedenfor viser resultatet fra de 2 scripts jeg har lavet og som jeg har kørt på Oracle SQL Developer.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
2.1	Oprette en ny konfiguration i database baseret på Simple local vtrack test.sql filen.	Se SQL kode i vedlagt cd rom (Simple local vtrack test.sql)	Insert statements bliver udført succesfuldt	Se appendix B3	Ok
2.2	Kan de oversatte værdier i `ParameterCollection` tagget fra Simple local vtrack.xml fil til SQL indsættes i databasen	Se XML og SQL kode i vedlagt cd rom	Insert statements bliver udført succesfuldt	Se appendix B3	Ok
2.3	Kan de oversatte værdier i `ModuleCollection` tagget fra Simple local vtrack.xml fil til SQL indsættes i databasen	Se XML og SQL kode i vedlagt cd rom	Insert statements bliver udført succesfuldt	Se appendix B3	Ok
2.4	Oprette en ny konfiguration i database baseret på Simple USA vtrack test.sql filen.	SQL kode i Simple USA vtrack test.sql	Insert statements bliver udført succesfuldt	Se appendix B3	Ok
2.5	Kan de oversatte værdier i `ParameterCollection` tagget fra Simple USA vTrack.xml fil til SQL indsættes i databasen	XML og SQL kode for Simple USA vTrack.xml og Simple USA vtrack test.sql	Insert statements bliver udført succesfuldt	Se appendix B3	Ok
2.6	Kan de oversatte værdier i `ModuleCollection` tagget fra Simple USA vTrack.xml fil til SQL indsættes i databasen	XML og SQL kode for Simple USA vTrack.xml og Simple USA vtrack test.sql	Insert statements bliver udført succesfuldt	Se appendix B3	Ok

Tabel 3 - Integrationstest fra xml til database

Det har været nødvendigt at lave en forsimplet test pga. de ressourcer jeg har til rådighed og fordi nogle af konfigurationerne indeholder omkring 800-900 linjer xml kode.

Ud fra denne test kan jeg være sikker på at databasen understøtter arkitekturen af xml filerne. Uanset hvilke data der findes hos vores kunders vtrack.xml fil, så kan de mappes (dog begrænset af størrelsen af værdien/data), selvom jeg kun har lavet en forsimplet test i dette projekt. Dog er

Der er nogle af kommentarerne i xml filen der ikke kan mappes, men dette har ikke nogen relevans for funktionaliteten af projektet.
Fx indeholder `Server` tagget en kommentar i den første `TimedModule` i Simple USA vtrack.xml som ikke kunne mappes i databasen. Dette skyldes at vi ikke har taget højde for det, da jeg designede databasemodellen. Denne problemstilling undlades i projektet.

I den sidste test vil vi teste de unikke constraints jeg har oprettet for tabellerne `COMM_CONFIGS` og `COMM_MODULES`. Jeg vil også i denne test, teste constrainten i tabellen `COMM_MODULE_PARAMS` (det med at en række i denne tabel kun kan have én foreign key).
Denne test tager udgangspunk i at Simple local vtrack test.sql filen er kørt på databasen, da jeg vil prøve at oprette "dubletter" fra data i dette script.
Koden der er udviklet til denne test findes på cd-rommen (Constraint test.sql, se Appendix C).
Testen er kun kørt på den lokale database.

Tabellen nedenfor viser test resultaterne:

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
3.1	Oprette 2 konfigurationer med ens navne.	Se SQL kode (Constraint test.sql) på cd-rom	Fejlbesked om at dette ikke kan oprettes.	Se appendix B4	Ok
3.2	Oprette 2 ens moduler for en konfiguration.	Se SQL kode (Constraint test.sql) på cd-rom	Fejlbesked om at dette ikke kan oprettes.	Se appendix B4	Ok
3.3	Oprette en parameter "delt" af 2 moduler (timed module og listener module).	Se SQL kode (Constraint test.sql) på cd-rom	Fejlbesked om at dette ikke kan oprettes.	Se appendix B4	Ok

Tabel 4 - Constraint test på database

Ud fra denne test kan vi sige at de specifikke constraints som jeg har oprettet virker på databasen. Vi har ikke udført en detaljeret test på de øvrige constrains fx dem med foreign keys. Jeg ikke fundet grund til at lave test på disse og derfor undlades de.

14.7 Andre overvejelser

Dette afsnit omhandler andre overvejelser jeg har haft i projektet og som er relevante indenfor database modellering. Disse overvejelser kan evt. laves i forbindelse med videre udvikling af databasen og projektet.

14.7.1 Historik

Historik er godt med hensyn til at se hvem der har foretaget hvad og hvornår og for at finde ud af hvorfor programmet reager som den gør, hvis det pludselig skulle opstå anomalier.

Her kunne der fx laves en undertabel for hver tabel på Figur 19 med de samme parametre og udvide den med yderligere 3 kolonner med følgende parametre:

- Operation,
- Tidsstempel og
- Bruger

Således kunne man spore hver ændring der sker løbende i kommunikationsenheden. Man ville også have mulighed for at kunne gå tilbage og bruge de forrige indstillinger, hvis programmet skulle begynde at reagere uforventet.

14.7.2 Lås

Som regel vil der kun være een bruger der retter i konfigurationen, har dette emne ikke været særlig relevant. Hvis der kommer flere brugere på og skal til at ændre i konfigurationen parallelt, kan man vælge at have låse på. På denne måde kan man undgå at transaktionerne interferer med hinanden, så der opstår inkonsistent data.
Man kan vælge mellem 3 typer at låse:

1 - Pessimistisk låsning:
Databasen låses for andre brugere ligeså snart der er én der ændrer data i databasen. Låsen fjernes så snart brugeren er færdig med sine ændringer og er gemt (committed) i databasen.

2 - Optimistisk låsning:
Her kan man have flere brugere der kan være i gang med at ændre data i databasen, men så snart der er en der gemmer data vil de andre brugere få besked på, at det data de arbejder med er forældet.

3 – 2 fase låsning:
Som navnet antyder så består denne låsning af 2 faser, growing phase og shrinking phase. Denne type lås minder meget om hvordan semaforer virker.
Transaktionen starter ved at man i den første fase låser alt det data man har brug, således ingen kan tilgå de samme data og så snart man er færdig (shrinking phase) med sine ændringer frigiver man låsen, som man har lavet i fase 1.
Denne type lås er god imod problemer som lost update. Der skal dog bemærkes at ved brug af denne type lås, kan der opstå situationer som deadlock.

14.7.3 Fremtidige forbedringer

Da jeg kiggede på det stykke arbejde jeg har lavet igennem opdagede jeg nogle fejl og mangler. Dette har jeg ikke haft tid til at rette og derfor noteret i dette afsnit med håb på, at en anden person i virksomheden vil tage hånd om. I dette afsnit kigger vi kun på de forbedringer der kunne laves i forbindelse med tabellerne i databasen. I de øvrige afsnit vil vi komme ind på andre forbedringer man kan foretage sig i de pågældende delprojekter.

Man kunne have en constraint i tabellen `COMM_MODULE_PARAM` som sørgerede for at navn og modul id'et er unikt, dvs. unikke parametre til hvert modul.
Når disse felter indlæses af kommunikationsenheden vil dette gemmes i en hashtabel (`PropertyHelper` objekt, se Figur 5). Denne datastruktur vil ved indsættelse af to ens nøgler overskrive den eksisterende værdi. Dette kan resultere eventuelle fejl i kommunikationsenheden.
Det samme gælder for `COMM_PARAMS` tabellen. Unikke "globale" parametre til konfigurationen.

Man kan flytte tabellen `COMM_MODULE_PARAM` væk fra de 2 details tabeller og sætte den sammen med master tabellen (`COMM_MODULES`). Dette ville medføre at vi gik fra to foreign keys (`comm_timed_modules_id` og `comm_listener_modules_id`) til en foreign key (denne kunne evt. hedde `comm_module_id`)
Dette ville også sørge for at constraint'en vi har oprettet for denne tabel ikke behøves længere.

`Value` feltet i `COMM_PARAMS` tabellen kan være for lille, den bør måske sættes til 250 bytes ligesom i `comm_module_params` tabellen.

14.8 Konklusion

Jeg har i dette afsnit analyseret xsd filen som bruges i vTrack og har på baggrund af denne analyse og analysen i afsnit 4.1.1 udviklet en databasearkitektur som er i stand til at kunne mappe data fra xml filen til databasen (se Figur 19). Dette har vi valideret ved at udfører integrationstest.
Ved at databasemodellen kan mappe alt data fra xml filen har vi samtidig accepteret at modellen er generisk nok til, at der ikke er behov for at lave kodemæssige ændringer i strukturen, som var et af kravene til projektet.

Ud over dette har vi kigget på nogle andre overvejelser der kan laves i forbindelse med projektet.
Her har vi taget højde for at der typisk vil være éen bruger der anvender konfigurationen og derfor er der ikke behov for at have låse med. Og at man evt. kunne have undertabeller til at spore ændringer i konfigurationen. Vi har også kigget på andre forbedringer man kan lave ved fremtidig udvikling.

Som beskrevet i foranalysen fandt vi ud af, at vi ikke kunne komme helt af med xml filen, da den indeholder oplysninger om hvilken database den skal etablere forbindelse til. Denne problemstilling har ikke været mulig at løse i dette delprojekt, men forventes at blive løst i de senere delprojekter.

15 Del 2 – Integration og Unit Test

Det andet delprojekt har til formål at få udviklet funktionalitet til kommunikationsenheden således at den bliver i stand til at kunne etablere en database forbindelse, som bruges til at indlæse resten konfigurationen med.
Jeg vil ud fra min viden som jeg har tilegnet mig i foranalysen bruge dette som inspiration til at designe det nye system og på denne måde initialisere de rigtige tråde, på samme måde som før med xml filen.

Herefter udføres der en række unit tests for at validere at funktionaliteten bevares uændret i forhold til den gamle løsning.

15.1 Udvidelser af kommunikationsenheden

Nu da jeg har fundet ud af hvilke data der anvendes, hvilke objekter der oprettes og hvordan det hele er skuet sammen, kan jeg bruge denne viden til at udvikle det nye system.

Der er 2 overordnede problemstillinger som jeg skal have løst i denne del af projektet er følgende:

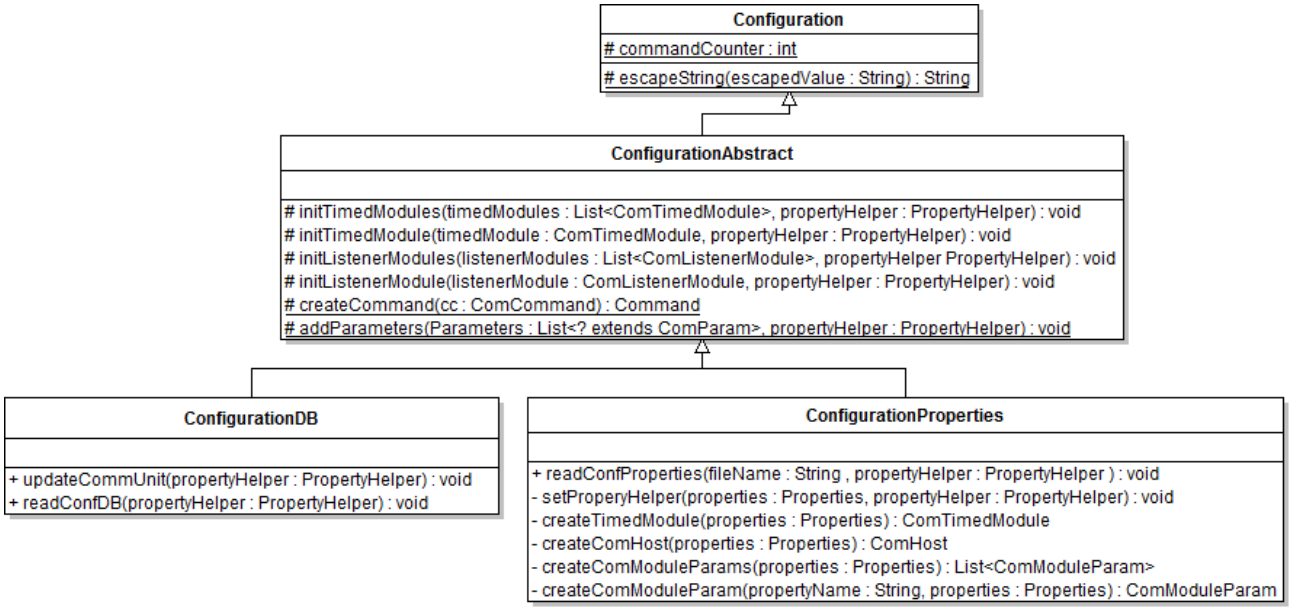
- Etablering af database forbindelse som bruges til at indlæse konfigurationsparametrene (implementering af use case U09 og U10).
- På baggrund af data fra databasen skal der oprettes nøjagtig de samme objekter/tråde som man kunne med xml filen og dette skal valideres med unit test.

15.2 Design og implementering

I Afsnit 4.1.2 fandt vi ud af, at det var klassen `Configuration` som er rygraden med hensyn til indlæsning og initialisering af modulerne specificeret i konfigurationsfilen. Derfor har jeg valgt et design som nedarver fra denne klasse, således jeg har alle de funktioner som denne klasse stiller til rådighed, plus dem som jeg bidrage med i projektet (anvendelse af strategi design pattern).
Jeg opdagede at `Configuration` klassen havde både til opgave at oversætte xml filen og starte modulerne op.
Dette har jeg valgt at lave om på, således at der er klasser der ansvaret for at initialisere moduler og klasser som har ansvaret for at læse fra kilderne (hhv. fra properties fil og database).

Klassen der nedarver fra `Configuration`, hedder `ConfigurationAbstract` (som er en abstrakt klasse). Denne klasse har ansvaret for initialiseringen af de forskellige moduler og parametre som bliver indlæst.
Da vi skal læse fra 2 kilder, har jeg yderligere lavet nedarvning fra `ConfigurationAbstract` klassen. Dette er klasserne `ConfigurationDB` og `ConfigurationProperties`. Disse klasser har ansvaret for at læse fra hver sin kilde og oprette de objekter der anvendes til at initialiser modulerne/trådene med.

Den færdige arkitektur af konfigurationsklasserne er afbilledet på Figur 21.



Figur 21 - Ny konfigurationsarkitektur

Før jeg startede på implementering af disse udvidelser til kommunikationsenheden, havde klassen `Configuration` mangle flere metoder end dem som vist på figuren. Da disse ikke anvendes længere har jeg valgt ikke at tage dem med i figuren (disse metoder er noteret med annotationen @Deprecated i kildekoden).

`commandCounter` variablen bruges til at tælle antallet af initialiserede `Command` objekter (se Figur 5) og `escapeString(String)` metoden bruges til at formatere værdierne i `parameter` tagget i xml filen til string værdier (nogle af `value`-tags i xml filen findes i hexadecimal format).
Disse nedarves og anvendes stadig i den nye implementation og derfor er de ikke slettet. Måske kunne man senere i fremtiden flytte dem ned i `ConfigurationAbstract` klassen, se mere i afsnit 15.7 Fremtidige forbedringer.

Hvis man kigger på sekvensdiagrammet på Figur 6, vil man se af `Configuration` klassen har 2 metoder `readTimedModule(timedModule)` og `readListenerModule(listenerModule)`. Der er disse 2 metoder som har ansvaret for at initialisere modulerne som tråde. Disse metoder er lavet om og ligger i `ConfigurationAbstract` klassen sammen med metoderne `initTimedModule(ComTimedModule)` og `initListenerModule(ComListenerModule)` (hvis man har en liste af `ComTimedModule` eller `ComListenerModule` objekter anvendes metoderne `initTimedModules(List<ComTimedModule>)` og `initListenerModules(List<ComListenerModule>)`).
Logikken i disse metoder er stort set den samme som den gamle implementation. Modulerne tjekkes dog ikke for hvorvidt om de er om de er aktive eller ej, da SQL forespørgslerne sørger for dette (med where-clausen `enabled_yn = 'Y'` – se mere i afsnit 15.4).

Koden nedenfor (Figur 22) viser et eksempel på hvordan modulerne initialiseres.
For at gøre det mere overskueligt har jeg udeladt koden i catch-clausen. Hvis modulet ikke kan oprettes af en eller anden grund skrives det i logfilen.

```
1  protected void initTimedModule(ComTimedModule timedModule, PropertyHelper propertyHelper) {
2      TimedModuleConfiguration timedModuleConfiguration = new TimedModuleConfiguration(timedModule, propertyHelper);
3      String name = timedModuleConfiguration.getName();
4      String className = timedModuleConfiguration.getClassName();
5      try {
6          dk.vtrack.comunit.server.TimedModule instance = (dk.vtrack.comunit.server.TimedModule)Class.forName(className).newInstance();
7          instance.init(timedModuleConfiguration);
8          propertyHelper.put(name, instance);
9      } catch (ClassNotFoundException e){
10         //Handle Exception
11     } catch (Exception e) {
12         //Handle Exception
13     }
14 }
15
16 protected void initListenerModule(ComListenerModule listenerModule, PropertyHelper propertyHelper) throws Exception {
17     ListenerModuleConfiguration listenerModuleConfiguration = new ListenerModuleConfiguration(listenerModule, propertyHelper);
18     new Listener(listenerModuleConfiguration);//start listener
19 }
```

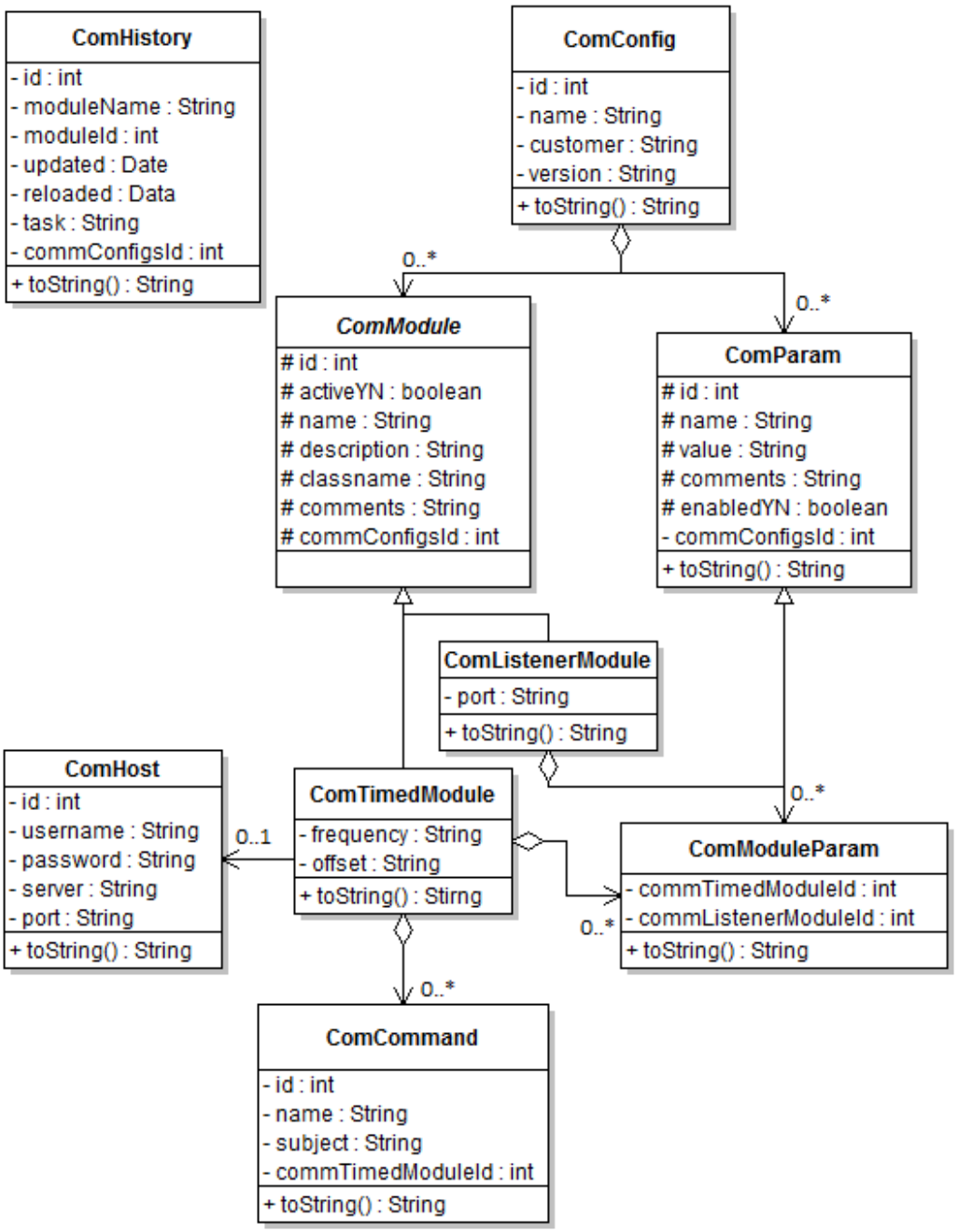
Figur 22 - Initialisering af moduler

Som tidligere nævnt så havde xmlBeans (det framework der oversætter xml filer til java objekter) genereret nogle klasser (se Figur 7), således data kunne repræsenteres som java objekter og det var disse objekter som blev brugt til at oprette objekterne på Figur 5. På denne måde kunne modulerne/trådene oprettes (det er objekterne `TimedModuleConfiguration` og `ListenerModuleConfiguration`, se også Figur 22, linje 2 og 17).

Se samme objekter oprettes stadig, men fordi vi går væk fra xml og fra xmlBeans, og fordi vi nu skal til at anvende nye datakilder, kan vi ikke længere bruge denne arkitektur. Så derfor blev jeg nødt til at oprette disse klasser selv, således data kan repræsenter (domænemodel).
Figuren nedenfor viser arkitekturen over dette (Figur 23).

For at opsummere så betyder dette, at når vi indlæser data enten fra properties filen eller fra databasen så bliver der oprettet objekter som på figuren nedenfor og disse objekter anvendes til at initialise modulerne i kommunikationsenheden (se kode eksemplet på Figur 22).
Initialiseringen af modulerne sker ved at man opretter et objekt af typen `TimedModuleConfiguration` eller `ListenerModuleConfiguration` afhængig af modul typen og fra det objekt hentes variablen `classname` og dette initialiseres med `init()` metoden (hver Timed modules implementationer implementer interfacet `TimedModule` og alle Listener modules implementationer implementer interfacet `ListenerModule`, fra disse interfaces kaldes `init()` metoden).

Denne omlægning har også den betydning at konstruktøren i klasserne `TimedModuleConfiguration` og `ListenerModuleConfiguration` skal ændres således de kan tage imod et objekt i parameteren af typen `ComTimedModule` eller `ComListenerModule` og anvende data fra disse objekter præcis på samme måde som i den gamle implementation.
De nye implementeringer af konstruktørerne er stort set de samme som den gamle og derfor vil jeg ikke gå i dybden med denne.



Figur 23 - Ny konfiguration domæne

Når man kigger på dette diagram, så vil man opdage at det ligner utrolig meget den samme som database modellen på Figur 19. Derfor syntes jeg ikke at der er behov for forklaring af detaljerne af denne figur.

`ComConfig` objektet er det objekt som indeholder konfigurationen. Denne har en liste af `ComModule` (abstract klasse) og `ComParam` objekter.
Fra `ComModule` objektet nedarver de 2 modul typer som vi kender (`ComListenerModule` og `ComTimedModule`). Hver disse har en liste af `ComModuleParam` objekter.
`ComTimedModule` har yderligere en liste af `ComCommand` objekter og et `ComHost` objekt.

Den mest bemærkelsesværdige forskel er at klassen `ComHistory` ingen associationer har med de øvrige klasser. Dette har der ikke været behov for, da denne klasse anvendes særskilt af de øvrige klasser (bruges ved genstart af kommunikationsenheden, dette kommer vi ind på i del 3 - Miljø).

Bemærk at `ComModuleParam` objektet (objekter som er beregnet til at indeholde parametre som er "lokalte" for modulet, altså fra `COMM_MODULE_PARAMS` tabellen) nedarver fra `ComParam` (dette objekt er beregnet til at indeholde data for "globale" parametre, altså fra `COMM_PARAMS` tabellen).
Alle objekter indeholder de samme attributter som findes i tabellerne i databasen, udover dette har de forskellige klasser getters og setters metoder og en `toString()` metode.

Implementation af klasserne overfor findes i pakken dk.vtrack.comunit.conf.domain.

15.3 Properties filen

Vi har nu en domænemodel som kan indeholde konfigurationsdata (som java objekter). Disse objekter bruges til at oprette `Timed-/ListenerModuleConfiguration` objekterne, som bruges til at starte trådene op. Det eneste vi mangler at diskutere er hvordan data aflæses fra data kilderne (properties fil og database). Inden vi går gang med denne diskussion, vil jeg kort komme ind på den properties filen skal anvendes i dette projekt.

I den nuværende implementation er det vtrack.xml filen der indeholder oplysninger om hvilken database der skal etableres forbindelse til. Derfor er kommunikationsenheden nødt til at kende denne information før den resterende konfiguration kan indlæses.

Til løsning af denne problemstilling havde vi 2 potentielle løsningsforslag:

1. Skrumpe den nuværende xml fil, så den kun indeholde informationer om databasen. Den resterende konfiguration er gemt i databasen og indlæses herfra.
2. Flytte database informationer til en properties fil. Således at kommunikationsenheden først læser data herfra og derefter etabler en database forbindelse som så bruges til at kunne indlæse resten konfigurationen med.

Vi har valgt at bruge den anden løsning, dels fordi vi gerne vil af med xml filen (som også er en del af visionen med dette projekt) og det framework der anvendes (xmlBeans). En anden fordel ved properties filer er, at det er en del af Java standard API'et og derfor behøver vi ikke noget 3. parts framework til at løse denne problemstilling.

Properties filer er primært anvendt af Java relaterede teknologier som bl.a. bruges til at gemme konfigurationsparametre for et program. Hver parameter gemmes som et par (key/value-par) og er derfor lynhurtigt nemt at lære og oprette et, som kan bruges i et program[12].

Det modul der opretter forbindelse til databasen hedder LOGGER, så derfor har jeg først og fremmest været nødt til at mappe alt data fra dette modul til properties filen.
Men dette har ikke været nok. LOGGER modulet implementer klassen `LogMessageWriter`, og når man kigger kildekoden igennem for den klasse, kan man se at der er nogle interne afhængigheder mellem dette modul og parametrene fra `ParameterCollection`-tagget. Derfor skal nogle af parametrene fra `ParameterCollection`-tagget også mappes i properties filen.

[12] http://en.wikipedia.org/wiki/.properties

Dette er hovedsageligt parametre angående logfil. Udover Logger modulet etabler en databaseforbindelse har den også ansvaret for at oprette en logfil, hvor exceptions og lignende kan logges. Uden at gå så meget i detaljerne for denne klasse, vil jeg kort nævne hvilke parametre der er tale om:

Paramtrene der er i tale om er følgende:

- file.path.log
- file.name.log
- log.level.trace
- log.level.db
- file.name.sendblocker
- file.path.sendblocker
- file.path.message
- properties.stactrace

Disse parametre bør findes i properties filen og ikke i databasen, da dette vil give redundant data!
Husk at disse værdier gemmes i et `propertyHelper` objekt som rent faktisk nedarver fra en hashtabel. Denne datastruktur vil overskrive værdien, i det tilfælde hvor 2 identiske nøgler indsættes!

15.4 Aflæsning af data

Som det sidste i dette afsnit vil vi lave en diskussion for hvordan jeg læser data fra data kilderne.
Jeg indleder med at beskrive hvordan data indlæses fra properties filen og derefter hvordan data fra databasen indlæses og behandles.

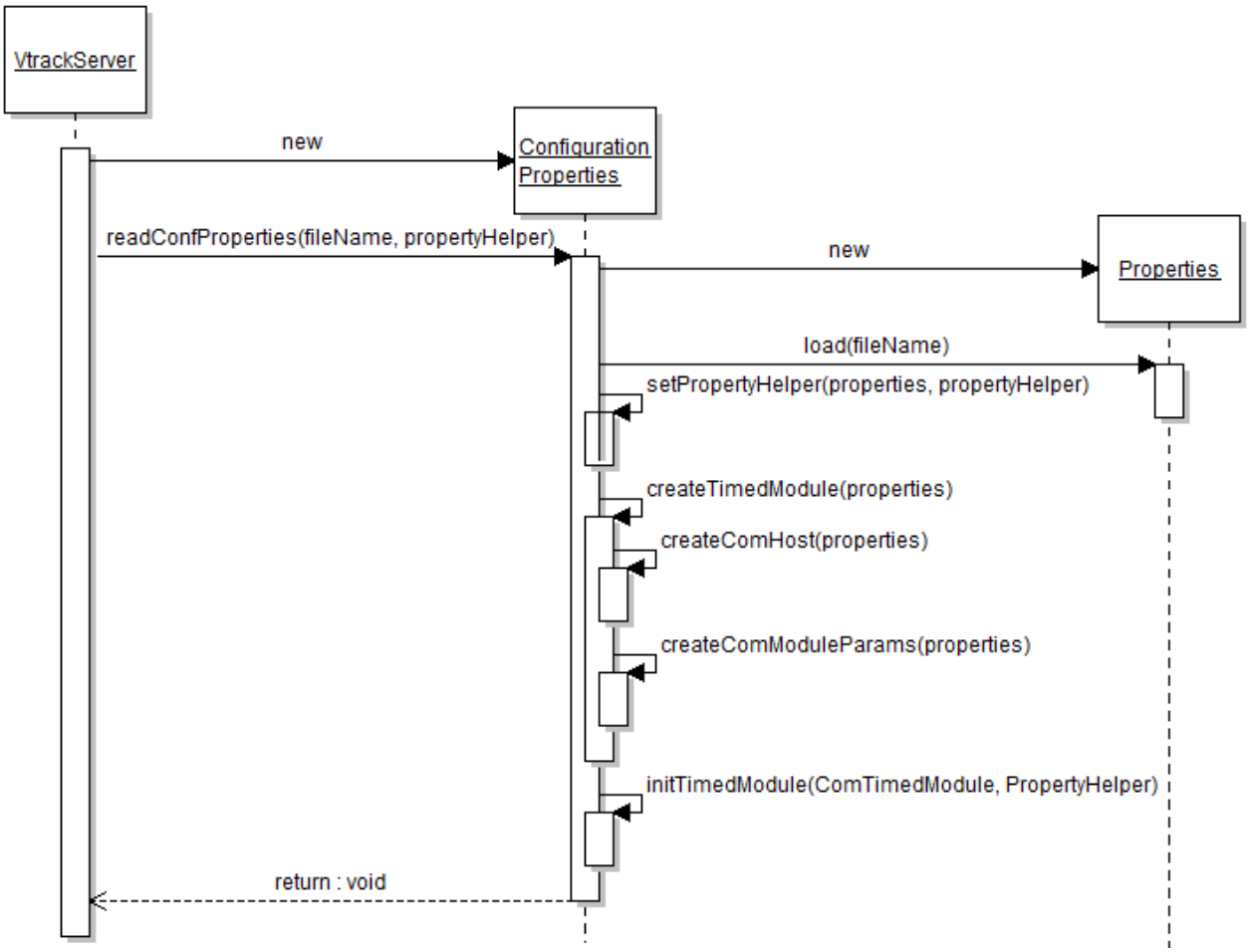
Ansvaret for indlæsning af properties filen er tildelt til klassen `ConfigurationProperties`. I denne klasse anvendes metoden `readConfProperties(fileName, propertyHelper)` til at oprette og initialisere logfiler og forbindelse til databasen.
Som det første i denne metode vil properties filen indlæses. Da properties filer er en del af Java API'et er det derfor relativt simpelt at indlæse denne. Kode eksemplet i figuren nedenfor viser et eksempel på hvordan man indlæser en properties fil i java.
Man giver stien til filen (`fileName`) til en `InputStream` objekt som `Properties` objektet kan læse fra. Hvis filen ikke kan findes eller der sker en fejl under indlæsningen, vil dette blive fanget i catch-clausen. Her er det op til programmøren at tage hensyn til fejlen og lave de nødvendige implementeringer.

```
1   Properties properties = new Properties();
2   //load properties to object
3   try {
4       properties.load(new FileInputStream(fileName));
5   } catch (FileNotFoundException e) {
6       //the properties file was not found
7       //Handle exception
8   } catch (IOException e) {
9       //didn't read properties file successfully
10      //Handle exception
11  }
```

Figur 24 - Indlæsning af properties fil

Når filen er indlæst, kan man bruge metode kaldet `properties.getProperty(key)` til at hente værdien for en nøgle.
Resten af koden i denne klasse sørger for at oprette et `ComTimedModule` objekt med de tilhørende værdier (host og parametre) og opretter LOGGER modul som en tråd.
Der findes også kode til at indsætte data fra properties filen ind i `PropertyHelper` objektet. Parametrene som er listet i afsnittet overfor skal indsættes i dette objekt (Husk at dette objekt indeholder parametre som er "synlige" for alle moduler).

Nedenfor vises et sekvensdiagram over hvordan det hænger sammen.



Figur 25 - Indlæsning af properties filen

Efter indlæsningen af properties filen vil kommunikationsenheden nu have oprettet forbindelse til databasen, hvorfra resten af konfiguration indlæses. Indlæsning af data fra databasen er lidt kompliceret.

Til indlæsningen af data har jeg brugt Spring frameworkets JDBC implementation. Dette var anbefalet af virksomheden og primært fordi det er det, de bruger i deres andre projekter bl.a. også i præsentationsenheden. Derved bliver systemet også nemmere at vedligeholde, når der anvendes ensartede 'basis komponenter' i projekterne.
Brug af Spring frameworket medførte at jeg var nødt til at tilføje nye biblioteker (jar filer) i lib folderen for kommunikationsenheden og tilpasse build path'en.
Fordelen ved Springs JDBC implementation er at den sørger for at oprette forbindelse databasen, udføre SQL kaldet, iterere over result set som man får tilbage fra et databasekald, handle i tilfælde af exceptions

og lukke database forbindelsen[13], alt sammen nogle ting som man kan spare udviklingen for, da disse er noget, som man selv skal sørge for normalt. Dette giver også en mere effektiv og målrettet udvikling idet man næsten kun skal tage hensyn til sin SQL forespørgsel.

Fra frameworket bruger jeg `JdbcTemplate` klassen som er et af det centrale objekter der findes i JDBC frameworket. Dette objekt indeholder metoder som giver mulighed for at udføre de nødvendige metodekald som jeg skal bruge i forbindelse med at hente data fra databasen.
Når data modtages fra databasen skal de mappes til objekter dette foregår ved brug af `RowMapper` interfacet som Spring stiller til rådighed.
Den sørger for at oprette et objekt per række som den opnår ved at iterere over `ResultSet`'et som bliver oprettet ved udførslen af forespørgslen til databasen[14].

Figur 26 viser et kode eksempel på implementeringen hvor der hentes data fra `COMM_CONFIGS` tabellen og mappes ind i `ComConfigs` objektet (domæne objekt). Her er `JdbcTemplate` er anvendt til at udføre SQL kaldet.

For hver tabel som jeg har oprettet i databasen findes der en klasse lignende som vist nedenfor og hver af disse klasser implementer sit eget interface (i dette tilfælde interfacet `ComConfigDao` (Data Access Objekter)).
Et særligt tilfælde er tabellen `COMM_MODULES`, da dette er en master tabel. Her anvendes details tabellerne for at hente modul data. Dette gøres ved at details tabellen laver en "natural join" på master tabellen. På denne måde får vi alle de nødvendige data for at kunne oprette modulet.
Apropos modul data, så findes der en where clause som sørger for at kun aktive moduler indlæses og derved sparer vi lidt kode ved metoderne vist på Figur 22 (se mere i `JdbcComListenerModuleDao` og `JdbcComTimedModuleDao` klasserne)

```
1  public class JdbcComConfigDao extends JdbcComDao implements ComConfigDao{
2
3      @Override
4      public ComConfig getComConfig(String configurationName) {
5          String sql = "SELECT ID, NAME, CUSTOMER, VERSION FROM COMM_CONFIGS " +
6          "WHERE NAME = '" + configurationName +"'";
7
8          List<ComConfig> configs = jdbcTemplate.query(sql, new Object[] {},
9          new RowMapper<ComConfig>() {
10             public ComConfig mapRow(ResultSet rs, int rowNum) throws SQLException {
11                 ComConfig cc = new ComConfig();
12                 cc.setId(rs.getInt("ID"));
13                 cc.setName(rs.getString("NAME"));
14                 cc.setCustomer(rs.getString("CUSTOMER"));
15                 cc.setVersion(rs.getString("VERSION"));
16                 return cc;
17             }
18         });
19         return configs.isEmpty() ? null : configs.get(0);
20     }
21 }
```

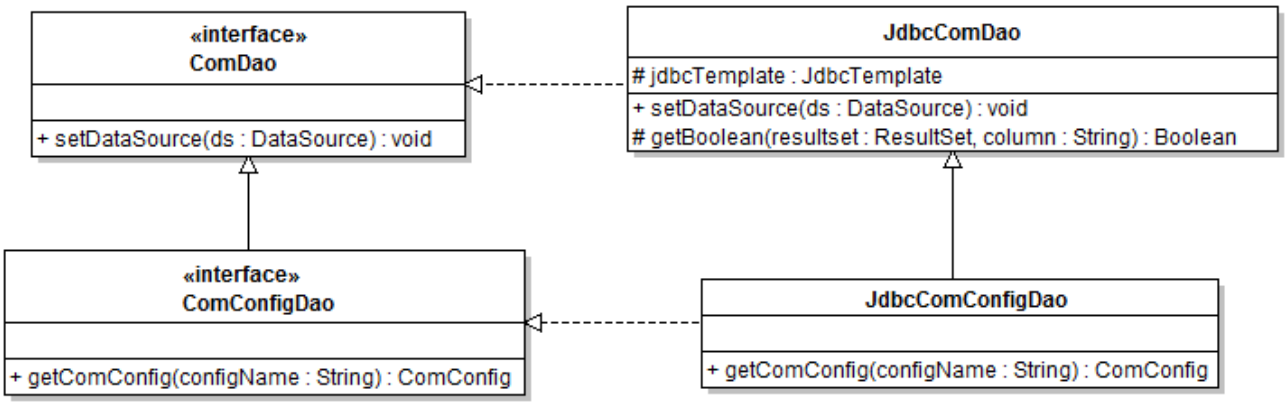
Figur 26 - Indlæsning af data fra database med Spring

[13] http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jdbc.html
[14] http://static.springsource.org/spring/docs/2.5.0/api/org/springframework/jdbc/core/RowMapper.html

For at give et oveordnet arkitektur over hvordan interfaces (Dao) og JDBC klasserne er struktureret vises Figur 27.

Jdbc klasserne nedarver fra `JdbcComDao`. Denne indeholder `jdbcTemplate` variablen og to metoder. `getBoolean()` metoden sørger for at konventer værdier som 'Y' og 'N' til boolean værdier (true og false). `setDataSource(DataSource)` metoden anvendes til at angive hvilken data ressource man skal anvende (dette er forbindelsen til databasen).

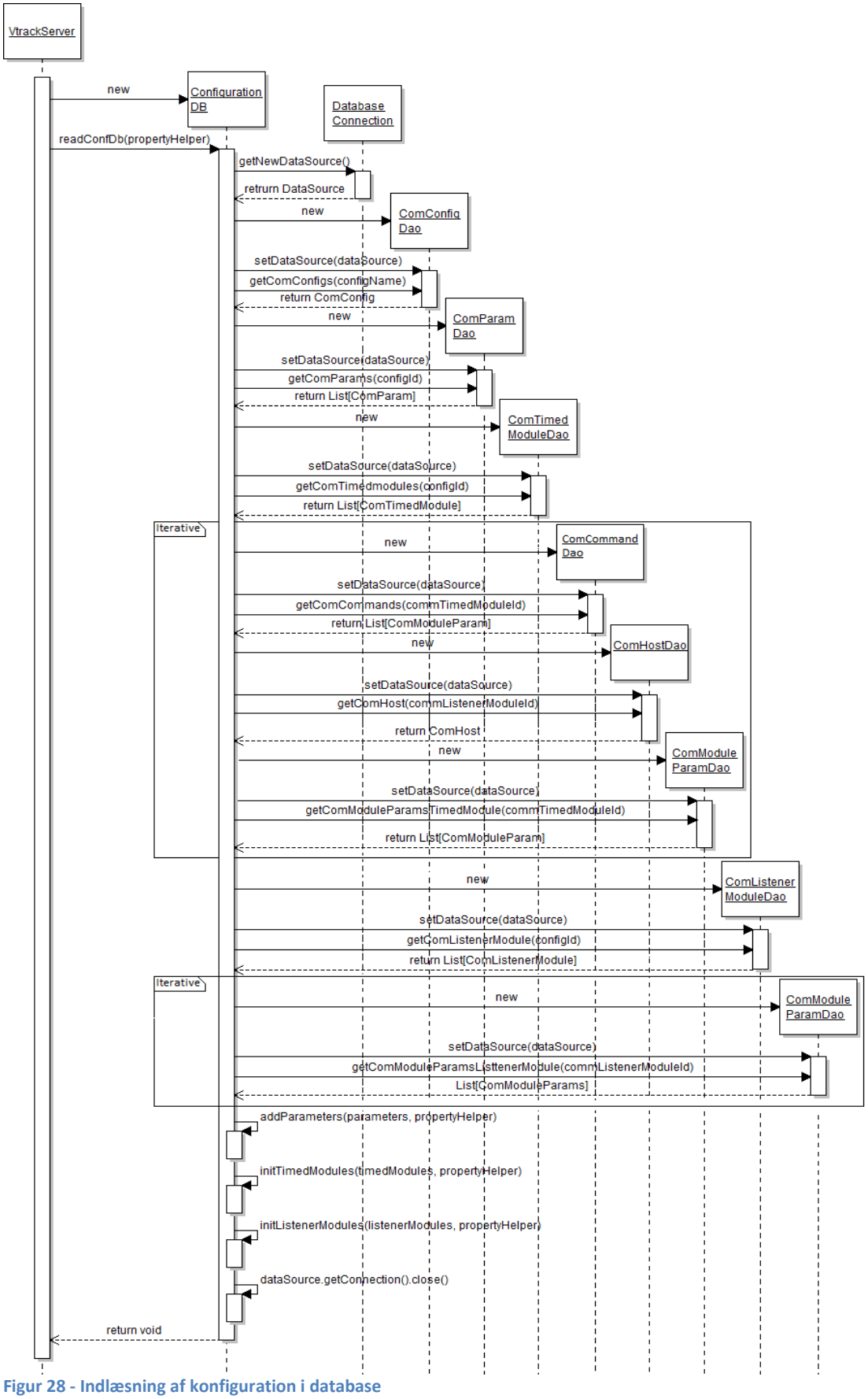
Figuren neden er forsimplet, da den kun viser eksemplet for det overstående. Det fulde diagram kan findes i Appendix A (A.3).



Figur 27- JDBC og Dao implementeringer

Vi har nu de store brikker på plads, men vi mangler stadig prikken over i'et, således vi har et samlet billede af hvordan den nye version af kommunikationsenheden indlæser konfiguration fra databasen og initialiserer modulerne på baggrund af disse data.
Lige efter metoden `readConfProperties(fileName, PropertyHelper)` returner (se Figur 25), starter `readConfDB(PropertyHelper)` metoden i `configurationDB` klassen (husk at denne har ansvaret for at indlæse data fra databasen). I denne metode hentes en database forbindelse (ved brug af klassen `DatabaseConnection`), som bruges til at hente resten af konfigurationen med. Først hentes `ComParams` efterfuldt af `ComTimedModule` objekterne med deres tilhørende værdier(`ComCommand`, `ComHost` og `ComModuleParam`) og derefter `ComListenerModule` og deres `ComModuleParam` objekter. Til sidste initialiseres parametrene og modulerne og den anvendte database forbindelse frigives

Nedenfor vises et sekvensdiagram for hvordan dette foregår:



Figur 28 - Indlæsning af konfiguration i database

15.5 Unit test

Kommunikationsenheden er én at de vigtige komponenter i vTrack og derfor er det vigtigt at denne kører stabilt. Nu da jeg er i gang med en stor omlægning af kommunikationsenheden er det vigtigt at den bliver testet grundig igennem.
I dette afsnit vil jeg komme ind på de forskellige tests jeg har lavet, og for hver type af test beskrive dens formål.

Jeg har lavet 4 overordnede test. Hovedformålet med disse test er at validere funktionaliteten af kommunikationsenheden bevares uændret forhold til den gamle løsning. De nedstående afsnit omhandler disse.
Test 2, 3 og 4 tager udgangspunkt i ”Simple local vtrack.xml” og ”Simple local vtrack test.sql” filen (se Appendix C). Dog skal data fra Simple local vtrack test.sql filen findes i en database.

Test 1 – Konfiguration properties test
Nu da vi har flyttet konfigurationen til databaseforbindelsen til en properties fil, vil jeg i denne test finde ud af om den er i stand til at kunne etablere en databaseforbindelse og sammenligne dette med 2 reference objekter (RO1 og RO2)

Til denne test har jeg oprettet en hashtabel med nøgler og parametre som properties filen skal bestå af og som jeg er sikker på, at er korrekte til at kunne oprette database forbindelsen (se `setUp()` metoden i `TestConfigurationProperties` klassen).

Det aktuelle objekt (AO) som skal testes, bliver initialiseret med værdier fra denne tabel.
RO1 bliver oprettet ved brug af `createTimedModule()` metoden fra `ConfigurationProperties` klassen. Værdierne til dette objekt bliver indlæst fra samme tabel (her testes også om metoden `createTimedModule` kan oprette `ComTimedModule` objektet korrekt).
RO2 bliver oprettet ved brug af samme metode som i RO1, men data til dette objekt indlæses fra properties fil.

Den første del af testen består i at sammenligne det aktuelle objekt med de 2 reference objekter.
Udover denne test, så udføres der endnu en test. Denne test består af 2 dele. I den første del testes der om disse objekter kan etablere en gyldig database forbindelse og den anden del af testen består i at hente data fra databasen (dato fra databasen).

Test resultater af denne test findes i tabellen nedenfor.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
4.1	Er AO lig med RO1	Se data i `setUp()` metoden i `TestConfigurationProperties.java`	Ingen output til konsol	Ingen output til konsol	Ok
4.2	Er AO lig med RO2	Se data i `setUp()` metoden i `TestConfigurationProperties.java` og properties filen vtrack_database.properties	Ingen output til konsol	Ingen output til konsol	Ok
4.3.1	Kan AO oprette gyldig DB forbindelse?	Se data i `setUp()` metoden i samme klasse overfor	Ingen output til konsol	Ingen output til konsol	Ok
4.3.2	Kan AO hente dato fra DB?	Se data i `setUp()` metoden i samme klasse overfor	Ingen output til konsol	Ingen output til konsol	Ok
4.4.1	Kan RO1 oprette gyldig DB forbindelse?	Se data i `setUp()` metoden i samme klasse overfor	Ingen output til konsol	Ingen output til konsol	Ok
4.4.2	Kan RO1 hente dato fra DB?	Se data i `setUp()` metoden i samme klasse overfor	Ingen output til konsol	Ingen output til konsol	Ok
4.5.1	Kan RO2 oprette gyldig DB forbindelse?	Se data i `setUp()` metoden i samme klasse overfor	Ingen output til konsol	Ingen output til konsol	Ok
4.5.1	Kan RO2 hente dato fra DB?	Se data i `setUp()` metoden i samme klasse overfor	Ingen output til konsol	Ingen output til konsol	OK

Tabel 5 - Konfiguration properties test resultater

Test 2 – Parameter test

Denne test har til formål at validere at `Paramter`-tags i xml filen som er børn til `ParameterCollection` tagget er mappet i databasen og at de indeholder præcist samme information som xml filen, når de bliver indlæst fra databasen til java objekter (`ComParam`).
I denne test sammenligner jeg om navne og værdier er ens og om antallet af parametre er ens i databasen og i xml filen.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
5.1	Er antallet af parametre ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok
5.2	Er navne ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok
5.3	Er værdier ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok

Tabel 6 - Parameter test resultater

Test 3 – Timed Module test

I denne test vil jeg finde ud at om de timed modules der findes i xml filen og de moduler der findes i databasen er ens.
Dette gøres ved at hente data fra de 2 kilder og sammenligne om modulerne er ens, og om deres parametre og commands er ens. Antallet af moduler i de 2 kilder skal også være ens!
Test resultaterne findes i tabellen nedenfor (Tabel 7).

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
6.1	Er antallet af moduler ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok
6.2	Er modulerne ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok
6.3	Er modulernes parametre ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok
6.4	Er modulernes commands ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok

Tabel 7 - Timed Module test resulater

Test 4 – Listener Module test
I denne test vil jeg finde ud af om de listener modules der findes i xml filen og de moduler der findes i databasen er ens.
Dette gøres ved at hente data fra de 2 kilder og sammenligne om modulerne er ens, og om deres parametre er ens. Antallet af moduler i de 2 kilder skal også være ens!
Test resultaterne findes i tabellen nedenfor (Tabel 8).

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
7.1	Er antallet af moduler ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok
7.2	Er modulerne ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok
7.3	Er modulernes parametre ens?	Data fra xml og database	Ingen output til konsol	Ingen output til konsol	Ok

Tabel 8 - Listener Module test resultater

15.6 Test Konklusion

Denne test har været meget vigtig i den forstand at kommunikationsenheden er en af vigtige komponenter i vTrack. Husk også at et af formålene med dette projekt er at sikre sig at funktionaliteten af den nye version af kommunikationsenheden bibeholdes fra den gamle version og at den fortsat kører stabilt og uændret forhold den gamle version.

Kilderne hhv. xml filen og database/properties filen bruges til at oprette objekter som ses på Figur 5 og disse objekter bruges til at oprette trådende med. Hvis jeg kan være sikker på, at jeg kan oprette disse objekter og de er identiske med dem som man kunne med xml filen, så kan jeg også være sikker på at resten af funktionaliteten må være korrekt. Derfor har jeg af denne grund ikke udført test på tråd niveau, hvilket også ville være meget mere komplekst.

Overordnet kan man sige at data i Properties filen kan oprette et korrekt objekt af instansen `ComTimedModule` som kan bruges til at oprette en database forbindelse med. Og med denne forbindelse kan vi hente den resterende konfiguration. Vi kan hente parametre fra `COMM_PARAMS` tabellen som indeholder værdier der er identiske med de objekter der blev parset (med den gamle kode) fra xml filen og det samme er gældende for listener modulerne og timed modulerne.

Min endelige konklusion er den, at migreringen af xml filen til den nye løsning, er, at funktionaliteten bevares uændret mod den gamle løsning.

Al kildekode til test findes i mappen test, i en package kaldet dk.vtrack.comunit.configuration.

15.7 Fremtidige forbedringer

Det her er et kort afsnit hvor jeg beskriver et par forslag til nogle af de fremtidige forbedringer man kan lave i forbindelse med videreudviklingen af kommunikationsenheden.

De anvendte metoder og variabler i `Configuration` klassen kan flyttes ned i `ConfigurationAbstract` så vi kan slette `Configuration` klassen.

Der er nogle designmæssige (OOA/D) forbedringer der kan laves. Hvis vi kigger på Figur 23, så kan man se at både `ComListenerModule` og `ComTimedModule` har associationer til `ComModuleParam`. Dette kan flyttes til superklassen. Denne forbedring kan også laves ved database løsningen.
Grunden til at jeg ikke rette dette er pga. jeg er færdig med implementation af denne del og skal videre med det næste delprojekt. Det vil også tage mig relativt lang tid at få det rettet, da jeg også skal tilbage til databasen og få det rettet der.

Ved opstart af kommunikationsenheden bør `COMM_HISTORY` tabellen opdateres ved de rækker som indiker at kommunikationsenheden ikke har opdateret. Dette bør gøres af metoden `readConfDB()` i klassen `ConfigurationDB` lige efter den har initialiseret trådene. Dette er indlysende da kommunikationsenheden har startet op med de allernyeste parametre gemt i databasen. Men når man så kigger på tabellen i databasen, vil det se ud som om, at den kører med de gamle data (da den som om ikke har opdateret). Dette emne vil også blive berørt i delprojekt 3 – Miljø.

Kildekoden i `ConfigurationDB` kan refactoreres, da der er noget kode som går igen, dette kan trækkes ud i en separat metode (evt. en privat metode) og derved kan man spare antallet af linjer af kildekode og øge genbrug af kode.

15.8 Konklusion

Dette delprojekt har til formål at aflæse konfigurationen til kommunikationsenheden ved brug af databasen og det har vi etableret ved at oprette en properties fil, således den indeholder informationer omkring hvilken database der skal etableres forbindelse til. Når kommunikationsenheden har læst dette modul ind, er den nu i stand til at kunne læse data fra databasen, hvori resten af konfigurationen indlæses.

Der er udviklet SQL forespørgsler til at kunne hente konfigurationen og resultatet herfra mappes ind til de rigtige objekter ved brug af Spring frameworket.
Disse domæneobjekter bruges til at oprette de rigtige tråde som bruges i kommunikationsenheden.

Et at de vigtige dele i dette delprojekt har været test. Her har jeg sammenlignet objekterne oprettet med data fra xml filen og fra databasen. Fra test resultaterne kan jeg sige at disse data stemmer overens med hinanden og de objekter der oprettes indeholder ens data, således modulerne oprettet korrekt.

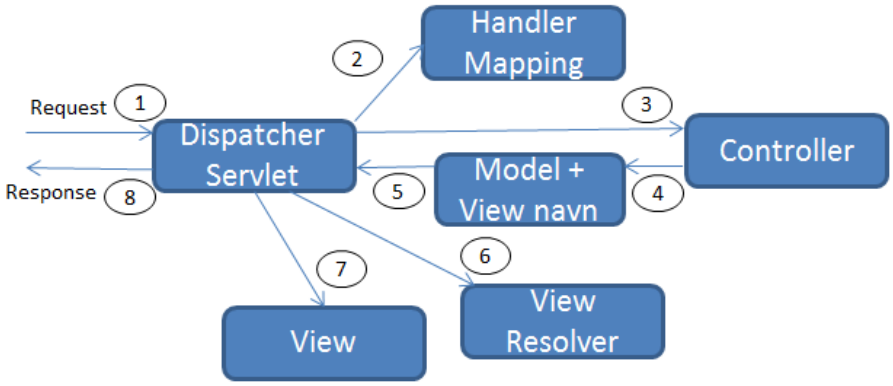
Som den endelige konklusion kan man sige at funktionaliteten er bevaret og det har succesfuldt lykkedes at få integreret databasen med kommunikationsenheden i den nye løsning.
Al kildekode for dette projekt (også for test) er lagt op på subversion som en ny branch (version 9.00).

16 Del 3 – Miljø

Nu da vi har gemt konfigurationsparametrene i en database og derved gjort det sværere at tilgå disse parametre i tilfælde af, konfigurationen skal opdateres. I det tredje delprojekt skal der derfor udvikles et grafisk miljø (en webapplikation), hvori værdierne til konfigurationen af kommunikationsenheden kan vedligeholdes og behandles efter CRUD (Create, Read, Update og Delete) principperne.
Yderligere skal dette Miljø give mulighed for at kunne genstarte kommunikationsenheden og samtidig tage hensyn til vTrack rollebaseret rettighedssystem.
Dette projekt skal integreres sammen med den nuværende implementation af præsentationsenheden (webclient) som er en del af vTrack.
I dette delprojekt arbejdes der med implementeringen af use cases U01 - U05.

16.1 Model view controller i Spring

Som beskrevet i foranalysen, så anvender præsentationsenheden Spring's Web framework. Dette framework er baseret på MVC design pattern.
Figur 29 nedenfor viser hvordan dette er implementeret i Spring. Når brugeren sender et request gennem browseren, går det gennem en dispacher servlet, handler mapping, controller og view resolver, indtil der kan sendes et response tilbage til brugeren. Hver disse komponenter i Spring MVC udfører bestemte opgaver.



Figur 29 - Model-view-controller implementation i Spring web framework

Når brugeren sender et request fra klientsiden vil request'ens første stop være `DispatcherServlet`. Denne virker som en front-end controller og uddeleger ansvar til de øvrige komponenter, således man kan udføre den aktuelle proces, som der bliver requestet om.
Formålet med `DispatcherServleten` er at sende requestet til en af Springs controllere. Controller er en enhed i Spring som kan behandle en request. Men da der er typisk vil være mange controllers i et projekt, så bliver `DispatcherServleten` nødt til at have lidt hjælp, til at afgøre hvilken controller der skal anvendes. Så derfor tager den kontakt til `HandlerMapping` som hjælper den med afgørelsen af, hvor requestet skal stoppe næsten gang.
Når `HandlerMapping` skal til at afgøre hvilken kontroller den skal behandles af, gøres dette bl.a. ved hjælp af denne URL som requestet indeholder.
Controlleren har til ansvar at behandle requestet med de informationer som brugeren har givet. Resultatet af controlleren vil typisk resultere i at der skal vises noget data til brugeren på browseren. Dette kaldes for modellen. Data i modellen gives videre til et view, så det kan blive formateret på en pæn og overskuelig måde, inden det vises til brugeren.

Den sidste ting som controlleren gør, er at identificer navnet på det view som skal vises, dette bliver sammen med modellen og request sendt tilbage til `DispatcherServleten`.
`DispatcherServleten` vil bruge `ViewResolveren` til at slå et view op på baggrund af view navnet som den har fået fra controlleren. Når view'et er fundet vil modellen gives til view'et som vil rendere et output der kan vises til klienten som response på requestet[15]. Dermed afsluttet requestet sin rejse på server siden.

16.2 Model

Jeg har tidligere i rapporten været inde på domænemodellen for konfigurationen i forrige afsnit. I dette delprojekt anvendes nogenlunde den samme arkitektur og derfor vil jeg ikke vise et klassediagram over dette (se evt. Figur 23). Forskellen i denne model forhold til den forrige model, er at klasserne i denne implementation har ingen associationer mellem hinanden, da der ikke har været behov for det og hver klasse repræsenterer en tabel i databasen. En instans at et givet objekt, er en række i den pågældende tabel.
En anden vigtig ændring der findes denne domænemodel, er at klassen `ComHistory` indeholder en enumeration, Task, med værdierne, start, stop reload og restart. Hvad hver disse felter bruges til, vil vi komme ind på senere i rapporten.

De data jeg har behov for at vise sammen med viewet hentes fra databasen og dette gøres igen ved brug af Spring frameworkets JDBC implementation. I dette projekt har jeg dog behov for mange flere kald end implementationen i kommunikationsenheden, da jeg skal implementer alle CRUD funktioner for de enkelte objekter i domænet. Interfacet `RowMapper` anvendes stadigvæk, for at kunne mappe data fra databasen i de rigtige felter for et objekt. Alle disse JDBC klasser er tilpasset en lille smugle, så de kunne integreres med projektet for præsentationsenheden.
Alle klasser i domænemodelen har en JDBC implementation som implementer et DAO (Data access objekt) interface. Alle database kald man har behov for, udføres derfor gennem disse interfaces. Set fra dette perspektiv, så er der storset ingen forskel modellen i præsentationsenheden og i kommunikationsenheden, samt hvordan data indlæses fra databasen.

En lille bemærkelsesværdig detalje er, at funktioner som anvendes til at oprette, opdatere og slette en række fra databasen returner som regel ikke noget svar tilbage fra databasen. For at være sikre på, at disse metoder er opfyldt som forventet, returnerer disse derfor et `FunctionResponse<String>` objekt. Med dette objekt kan man tjekke om de overnævnte funktioner er opfyldt succesfuldt eller ej.

Der findes en JDBC klasse sammen med et interface (`ComUpdateComuniDao`) som anvendes til at give kommunikationsenheden besked på, at den skal opdater modulerne/trådene som kører i kommunikationsenheden. Denne klasse vil jeg komme ind på senere i rapporten.

Et diagram over DAO og JDBC implementationerne findes i bilaget A.4

16.3 View

Viewet bruges til at vise en grafisk brugergrænseflade til brugeren, som man kan interagere med systemet. Jeg har brugt JSP (Java Server Pages) til at implementer de forskellige views der anvendes i miljøet. Disse JSP sider er delt op i 3 mapper, create, edit og view, som inddeler siderne op i logiske enheder. Delete

[15] Craig Walls, 2011, *Spring in Action – 3rd edition*, Manning Shelter Island

funktionen kræver ikke nogen særskilte sider, da denne funktion tilgås gennem JSP siderne der findes i view mappen.
De forskellige JSP sider er bygget op ved brug af en blanding af HTML, javascript og JSTL (Java Standard Tag Library).

Udover Spring anvendes frameworket Apache Tiles[16]. Tiles bruges til at lave nogle JSP sider. Disse sider kan ses som nogle byggestene og en kombination af disse sider bruges til at bygge en komplet side. Man kan side at hver tile indeholder fragmenter af en side og ved runtime samles disse byggestene til en komplet side.
Det gør udviklingen af de separate sider nemt og overskueligt og samtid øger det effektiviteten ved udviklingen. Hvis en side har brug for en tile kan man bruge JSTL 's `<c:import url="URL"/>` tag til at importere tilen og den vil vises som en del af den komplette side. Ved brug af tiles har jeg oprettet en JSP side per entitet fra domænemodellen.

Uden at gå særlig meget i dybden med koden, vil jeg hurtigt forklarer hvordan siderne er bygget op. Koden i JSP siderne i hver mappe (create, edit og view) minder meget om hinanden, og derfor vil jeg ikke gå i så meget i detaljerne med koden.

For hver JSP side i create mappen findes der i toppen javascript kode som bruges til at valider input felterne med, når brugeren gerne vil oprette et nyt element til konfigurationen. Koden sørger også for at tjekke at obligatoriske felter ikke er tomme og for andre felter findes der validering mod længden af tekststrengen og validering af felter som skal være angivet som et heltal og evt. i et bestemt tal område (fx 0 - 10).
Derefter kommer der en form (`<form>`) tag. Denne form er opbygget af en tabel med 2 kolonner. I den første kolonne er der en ledetekst, som giver en kort beskrivelse af input feltet og i den anden kolonne findes selve inputfeltet. Hvert input felt er angivet i sin egen række.
I den sidste række i tabellen findes der 3 knapper, en til at gå tilbage til forrige menu, en anden knap til at gemme informationerne i databasen og en tredje knap til at slette inputfelterne.
Hvis man kigger kildekoden igennem vil man opdage at nogle af felterne er skjulte (`style="display: none;"`) som fx id'er. Dette skyldes at når brugeren fx skal oprette en parameter til et modul, så skal brugeren ikke angive id'et på det modul parameteren skal tilhøre til, dette skal systemet selv finde ud af. Derfor kan man på denne måde overførere data mellem to sider (husk at http er stateless).

JSP siderne i edit mappen er stort set opbygget på samme måde som JSP siderne der findes i create mappen. Javascript koden som anvendes til validering af inputfelterne er de samme og form'en i edit siderne er stort set opbygget på samme måde som i JSP siderne i create mappen. Der kan dog være lidt flere skjule felter, da de elementer som man skal opdatere altid vil have et id, som de nødvendigvis ikke vil have ved oprettelse.
Når siden loades vil de gamle værdier for objektet indsættes i de korresponderende felter. Reset knappen vil nu ikke slette alt indhold på inputfelterne, men indsætte de gamle værdier ind.

Endelig har vi JSP siderne i view mappen. JSP siderne i denne mappe har til formål at vise data fra databasen til brugeren og give dem mulighed for at kunne oprette, slette og opdatere de enkelte elementer af konfigurationen.
Hvert entitet fra modellen har sin egen JSP side, på denne måde kan man tilpasse layoutet på siden som man ønsker at vise det til brugeren. Nogle af disse JSP sider iterer over en liste af objekter og viser dem på en tabel format (fx moduler og parametre) mens andre JSP sider viser data om et enkelt objekt (fx siden hvor detaljerne omkring et modul vises).

[16] http://tiles.apache.org

Man kan tydelig se fordelen ved anvendelse af tiles i dette projekt. Da hver JSP side viser information om en enkel entitet fra modellen, kan man ved brug af en kombination de disse sider, få vist en komplet side som er relevant for brugeren. Fx når brugeren ønsker at se informationer omkring et modul, kan man via JSTL 's import tag vise dens tilhørende værdier (parametre, host, commands) uden yderligere besvær.

16.4 Controller

Controlleren er en af de vigtige komponenter i MVC design pattern, idet den binder modellen sammen view'et, så brugeren kan få vist de rigtige sider.
Jeg har oprettet en java klasse (`ComunitConfigurationController`) der fungerer som controlleren for de JSP sider jeg har oprettet. Denne klasse indeholder logik til at kunne styre hvilken model der skal hentes og hvilket view der skal vises til brugen.

I dette projekt anvendes Spring version 2.5. I denne version introducerede Spring annotation baseret programmering for MVC controllers. Der findes annotationer som fx `@RequestMapping`, `@RequestParam`, `@ModelAttribute`, osv[17]. Nogle af disse annotationer anvendes i dette projekt.

I toppen af klassen er der angivet annotationen `@Controller` (lige før `public class ComunitConfi…`) således Spring kan genkende denne klasse som MVC controller.

Denne klasse anvender en række DAO interfaces, således man kan udføre de SQL kald man har brug for og på denne måde hente data fra databasen. Alle disse felter er annoteret med `@Autowired`. Disse felter vil blive automatisk initialiseret af Spring's dependency injection når disse felter anvendes[18], derfor behøver man ikke selv at initialiser dem.

Når brugeren interagerer med systemet vil der sendes et request til serveren. For at Spring kan vide hvilken metode der skal anvendes til at behandle denne request skal man annoterer sine metoder med annotationen `@RequestMapping("URL")`. Disse metoder kaldes også for request-handling methods. Når URL'en passer til en af metoderne, vil denne pågældende metode køre.

Der er 2 yderligere Spring annotationer som jeg bruger i dette projekt. Disse anvendes som en del af parametrene i metodernes signatur. Disse parametre er annoteret med `@PathVariable` og `@ModelAttribute`.

Med `@PathVariable` kan man angive en værdi som indikerer at argumentet til metoden er bundet til værdien i URL variablen. Hvis request-handling metoden tager hensyn til URLs på følgende format:

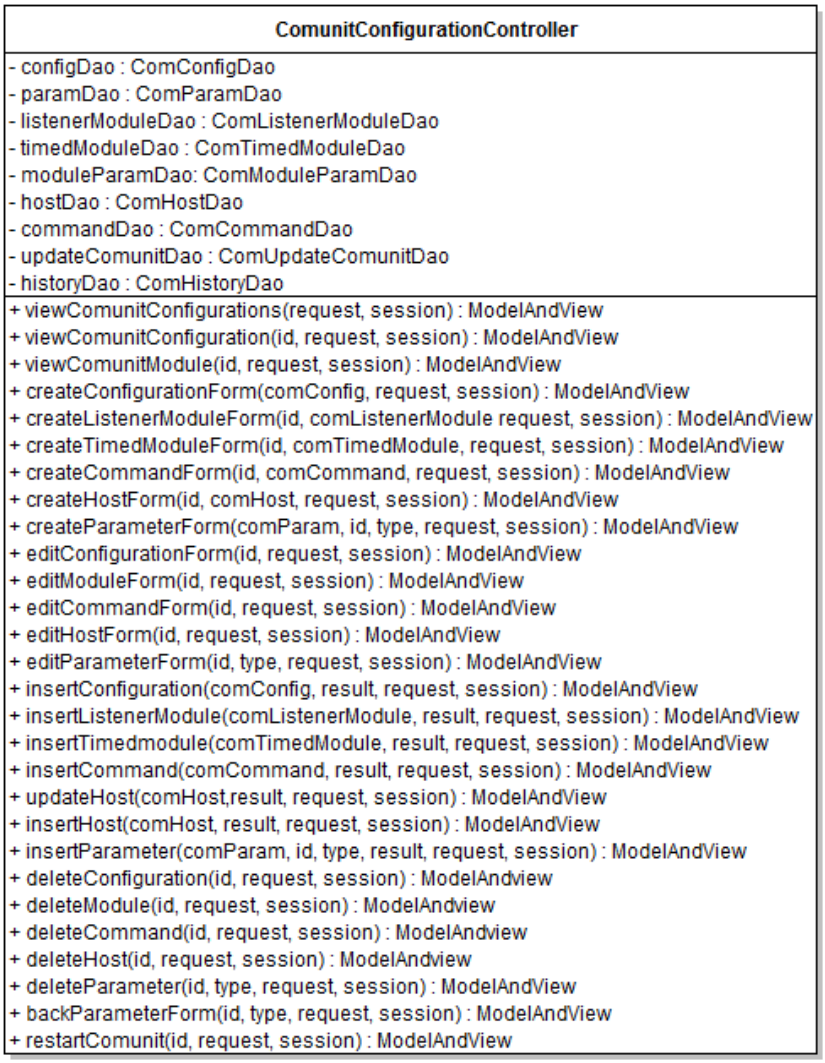
http://www.example.com/users/{userid}

Man kan specificere `@PathVariable` til at hente værdien fra userid. For eksempel hvis brugeren ønsker at se en side med følgende URL: http://www.example.com/users/ibrahim vil denne værdi sættes til ibrahim, når den anvendes af metoden.

`@ModelAttribute` anvendes til at binde værdier fra fx inputfelterne på en JSP side til et objekt som kommer gennem request'et. Herefter kan man arbejde videre med objektet som man har lyst til.

[17] http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html
[18] http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/beans/factory/annotation/Autowired.html

Som sagt har jeg har oprettet en controller klasse der har ansvaret for at behandle alle de indkommende requests som fra brugeren sender gennem Miljøet. Jeg vil hurtigt gennemgå opbyggelsen af denne klasse og fortælle hvilke metoder den indeholder. Det fulde klassediagram kan findes i Appendix A.5, hvorimod Figur 30 viser den vigtigste metoder.



Figur 30 - Controller for Miljøet

De første 3 metoder anvendes til at læse data fra databasen. De bruges henholdsvis til at vise en oversigt af alle de konfigurationer der findes i databasen, detaljer om en enkelt konfiguration (ved brug af id'et på konfiguration) og detaljer for et modul (understøtter begge typer).

De næste 6 metoder anvendes til at kunne vise siden for oprettelse af et element til konfigurationen. Her findes der metoder til at vise viewet for oprettelse af en konfiguration, listener module, timed module, command, host og parameter.

De efterfølgende 5 metoder anvendes til at behandle requests, når brugeren ønsker at opdatere værdierne for de forskellige elementer som konfigurationen består af. Disse metoder vil returnere et view/JSP side som findes i mappen edit.

De næste 7 metoder anvendes til at indsætte data i databasen, når brugeren har klikket på "gem" knappen ved oprettelse eller ved opdatering af et element til en konfiguration.
Disse metoder tager imod et `BindingResult` (`result`) objekt ind som parameter (dette er implementeret af Spring). Dette objekt kan anvendes til at binde værdierne angivet i inputfelterne til et objekt. Det kan også bruges til at tjekke om de angivne værdier indeholder fejl (denne funktion anvendes dog ikke)[19].

De næste 5 metoder anvendes til at slette et element af en konfiguration. Her kan man vælger at slette en hel konfiguration, et modul, command, host eller en parameter.

Endelig til sidst har jeg 2 metoder. Den ene er `backParameterForm()`, denne metode anvendes til at finde ud af, hvilken side der skal vises, når brugeren klikker på tilbage knappen der findes på JSP siden ved oprettelse eller opdatering af en parameter.
`restartComunit()` metoden anvendes til at give besked til kommunikationsenheden om at trådene skal opdateres forhold til de nyeste værdier i databasen.

Måske har i figuren ovenfor bemærket at metoder der vedrører parameter objektet har en parameter i metode signaturen som hedder "type". Denne værdi anvendes til at adskille hvem parameteren tilhører til, da parameter objektet kan tilhører til forskellige elementer i en konfiguration. Derfor har jeg været nødt til at bruge en ekstra parameter.
Hvis parameteren tilhører et listener modul er denne værdi sat til "`listenermodule`", hvis den tilhører til et timed modul er værdien "`timedmodule`" og hvis det er en global parameter er værdien "`configuration`". Dette kan være vigtigt, når man fx skal oprette en parameter til et listener module og at man ikke kommer til at oprette parameteren som en global parameter (i `COMM_PARAMS` tabellen) da JSP siden der anvendes til oprettelse og opdatering af parameter objektet er den samme.

Hver metode returner et `ModelAndView` objekt. Dette objekt indeholder både modellen og view'et som skal vises til brugeren. Når metoden returner vil Spring selv finde ud af, at sende et response til brugeren.

16.5 Rettighedskontrol

Hver metode i controller klassen anvender interfacet `AccessControlService`. Dette interface tilbyder funktioner for tildeling, tilbagekalde og tjekke privilegier for en bruger.
Her anvendes `hasPrivilege()` metoden. Denne metode tager imod en `Role`, `Privilege` og en `key` som parametre og på baggrund af dette, valides adgangen til en side.
Rollen for en bruger hentes gennem http-sessionen som bliver sat ved login. Til feltet `Privilege` angiver man hvilken privilegium man ønsker at tjekke for. Her har man mulighed for at tjekke for følgende privilegier `READ, UPDATE, DELETE, CREATE` (disse findes som en enumeration).
Nøglen bruges til at tjekke om brugeren har de nødvendige rettighed til at se den pågældende side.

Da de sider jeg har oprettet er nye, blev jeg nødt til at oprette en nøgle i databasen. Med denne nøgle kan man tildele afgang til brugerne, for at tilgå de nye websider for vedlige vedligeholdelse af konfigurationen til kommunikationsenheden. Jeg har oprettet en fil "Access control webclient.sql" (se appendix c). Denne fil indeholder SQL insert statements for tildeling af adgang til brugerne med rollen `ADMIN_VTRACK_ROLE` til at tilgå de nye websider jeg har oprettet.

[19] http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/validation/BindingResult.html

```
1   @RequestMapping(MAPPING_COMUNIT_CONFIGURATION_VIEW_BY_ID)
2   public ModelAndView viewComunitConfiguration(@PathVariable("id") int id, ServletRequest request, HttpSession session) {
3       //Shows view if permitted
4       if(accessControlService.hasPrivilege(
5           Role.valueOf(SessionValuesUtil.getLoggedInRole(session)),
6           Privilege.READ,
7           key2)) {
8               //get all modules for the configuration
9               List<ComModule> modules = new ArrayList<ComModule>();
10              modules.addAll(timedModuleDao.getComTimedModules(id));
11              modules.addAll(listenerModuleDao.getComListenerModules(id));
12
13              return getEnrichedView("comunit-module-view-form", request, session)
14              .addObject("id", id) //id of the configuration
15              .addObject("parameterType", "configuration") //used when creating parameter (Global parameters)
16              .addObject("modules", modules) //modules in the configuration
17              .addObject("parameters", paramDao.getComParams(id)); //"Global" parameters for the configuration
18      }else {
19          return getErrorView();
20      }
21  }
```

Figur 31 - Controller kode eksempel

På Figur 31 vises et kodeeksempel på implementering af en request-handling metode, som har ansvaret for at vise detaljerne for en konfiguration. Konfigurationen der skal vises er specificeret gennem URL'en og med annotationen `@PathVariable("id")` kan man hente og bruge id'et på den konfiguration der skal vises. I selve metoden tjekkes der først om brugeren har de nødvendige rettigheder til at se siden (linje 4 - 7). Hvis ikke, så returneres `ErrorView'et` (linje 19) som er en general fejlside der anvendes i vTrack.
I det tilfælde brugeren har de nødvendige rettigheder, vil programmet hente modulerne og de globale parametre som konfigurationen består af(modellen i MVC). Dette vil sammen med navnet på view'et (`"comunit-module-view-form"`) returneres tilbage. Herefter vil Spring klarer resten, så brugeren vil få et response lignende på Figur 9.

16.6 Genstart af kommunikationsenheden

Genstart funktionaliteten af kommunikationsenheden blev ikke 100 % færdigt (er estimeret til at være 80 % færdigt jf. afsnit 9 Projekt planlægning). Derfor vil jeg i dette afsnit diskuter hvad der er færdigt og hvad der mangler at blive lavet for at kommunikationsenheden kan genstartes som ønsket.

I forbindelse med genstart af kommunikationsenheden har vi ønsket at kunne lave en partiel og fuld genstart af modulerne i kommunikationsenheden.
Ved en partiel genstart forstås der, at findes nogle aktive moduler i kommunikationsenheden som enten skal stoppes eller genstartes. Men det skal samtidig også være muligt, at kunne starte en ny tråd/modul op uden yderligere konfiguration, så snart modulet bliver oprettet gennem Miljøet.
For eksempel i situationer hvor et modul fra Miljøet slettes eller sættes i deaktiveret tilstand skal det gældende modul stoppes af kommunikationsenheden. Og omvendt hvis der tilføjes et nyt modul eller et modul sættet i aktiveret tilstand, skal modulet startes op som en ny tråd i kommunikationsenheden. I det tilfælde hvor der for eksempel tilføjes en ny parameter til et modul eller en eksisterende parameter opdateres, vil dette medføre at modulet skal genstartes, før de nye værdier kan træde i kraft af kommunikationsenheden.

Grunden til at der er behov for en fuld genstart af kommunikationsenheden er, at hver implementation af trådende har en `init()` metode. Denne metode bliver kun kørt en gang ved opstart og derefter kommer man aldrig i denne metode, med mindre end man genstarter modulet fra bunden. Herefter vil trådende kun køre koden i `execute()` metoden hver gang tråden får lov til at køre af cpu'en. I nogle af implementeringerne af de forskelle moduler anvendes værdierne fra disse "globale" parametre i `init()` metoden og hvis trådende ikke genstartes fra bunden, så vil opdateringerne aldrig nogen sinde ikke træde i kraft og opdateringen vil ikke virke som ønsket.

For at kunne styr på ændringerne der sker gennem Miljøet for modulerne, har jeg oprettet en tabel `COMM_HISTORY`. Denne tabel har følgende felter:

- id (number),
- module_name (varchar2)
- module_id (varchar2)
- updated (date)
- reloaded (date)
- task (varchar2)
- comm_configs_id (number)

Task feltet er nok den mest bemærkelsesværdige felt at tage hensyn til. Her angiver man hvilken opgave kommunikationsenheden skal foretage sig, for det pågældende modul.
Man bør til feltet task anvende værdierne: 'start', 'stop', 'restart' og 'reload' (bemærk disse værdier er fuldstændig magen til i enumerationen, `Task` i klassen `ComHistory`), da disse felter vil fortolkes af kommunikationsenheden på følgende måde:

- start - starter et modul op.
- stop - stopper et modul.
- restart - genstarter et modul.
- reload - fuld genstart af alle moduler.

I første tanke kunne man godt forestille sig at module_name (navnet på modulet) var unødvendigt da man har feltet module_id (id'et på modulet). Men sådan er det ikke. Module_name er vigtigt i den forståelse af, at kommunikationsenheden har navngivet alle tråde med modulets navne og man kan kun få fat i en tråd (som man fx ønsker at stoppe), ved at bruge navnet på modulet. Man kan ved brug af metoden `ActiveThreadCount.getProcess(moduleName)` hente en tråd (modul) og herefter kalde metoden `stopProcess()` til at stoppe tråden.
I det tilfælde hvor et modul er slettet fra databasen og modulet skal stoppes af kommunikationsenheden, kan man via id'et ikke spore modulets navn, derfor bruges værdien i module_name.

Hvis et modul skal genstartes eller startes op, skal man kende til id'et på modulet, så man kan hente de nye informationer om modulet og dens korresponderende data fra databasen. Derved kan man initialisere objektet og få den startet som en ny tråd i kommunikationsenheden.

Feltet updated angiver datoen for hvornår de seneste ændringer er sket for modulet gennem Miljøet, derfor bør dette felt kun angives af Miljøet.
Reloaded angiver datoen for hvornår kommunikationsenheden har udført opgaven (task'en) på modulet. Dette felt bør derfor kun indsættes af kommunikationsenheden, efter den har udført handlingen på modulet.

Når kommunikationsenheden starter op fra bunden manuelt, bør den tjekke at der ikke findes nogen række i `COMM_HISTORY` tabellen hvor reloaded er `null`. Dette giver ikke mening, da kommunikationsenheden netop har indlæst de allernyeste værdier for konfigurationen i databasen. Dette mangler at bliver implementeret i `readConfDB()` metoden i `ConfigurationDB` klassen. Gør dette ved brug af metoden `updateComHistory()` i interfacet `ComHistoryDao`, se evt. metoden `updateCommUnit()` som eksempel.

comm_configs_id angiver id'et på konfigurationen. Husk at man kan have flere konfigurationer gemt i databasen. Dog vil en kommunikationsenhed anvende en konfiguration, men man kan gemme flere konfigurationer i database, i det tilfælde kunden har mere en kommunikationsenhed i deres version af vTrack.

Der bør ske en fuld genstart af kommunikationsenheden når værdierne i `COMM_PARAMS` tabellen ændres (husk at denne tabel indeholder data fra `ParameterCollection` tagget i xml filen (”globale parametre”)).
I dette tilfælde skal der oprettes en række i `COMM_HISTORY` tabelen hvor feltet task er sat til værdien 'reload' og man behøver ikke at angive noget module_id eller module_name, man angiver kun updated, comm_configs_id og 'reload' til feltet task.
Det er lige meget hvor mange rækker der indsættes med denne opgave i `COMM_HISTORY` tabelen, der skal bare være mindst en række, så kommunikationsenheden kan finde ud af, at den skal genstarte alle modulerne på ny.
Hvis der findes andre opgaver til kommunikationsenheden udover reload tasken, så fordi den laver en fuld genstart vil disse også blevet taget i betragtning, således moduler der skal stoppes, stoppes, moduler der skal startes, startes.
Efter denne genstart vil kommunikationsenheden tage hensyn til rækkerne hvor reloaded er `null` og indsætte tidspunktet i databasen, hvor den har genstartet.

Men dette er ikke gældende for de øvrige værdier for feltet task og indsættelse af en række i denne tabel kan derfor være tricky. Forestil dig at man opretter et modul, hvortil man tilføjer nogle parametre, og som man senere vælger at sætte deaktiveret.
Hvis man indsætter i `COMM_HISTORY` tabellen lige efter man har udført disse handlinger, vil der være 3 rækker i databasen, som fortæller at kommunikationsenheden skal starte og derefter genstarter og til sidst stoppe tråden. Dette kan give problemer for kommunikationsenheden (er dog ikke testet). Derfor anbefales der, at der kun findes een række per modul som angiver hvilken opgave kommunikationsenheden skal udfører. Logik og kode til dette mangler! Her bør der ligges strategi for hvordan problemet løses (dette er ikke gjort) og derefter implementer løsningen.

Når man har lavet sine ændringer i Miljøet skal man have dem opdateret af kommunikationsenheden. Dette gøres ved at klikke på knappen ”Restart Comm. Unit” som er afbilledet på Figur 9.

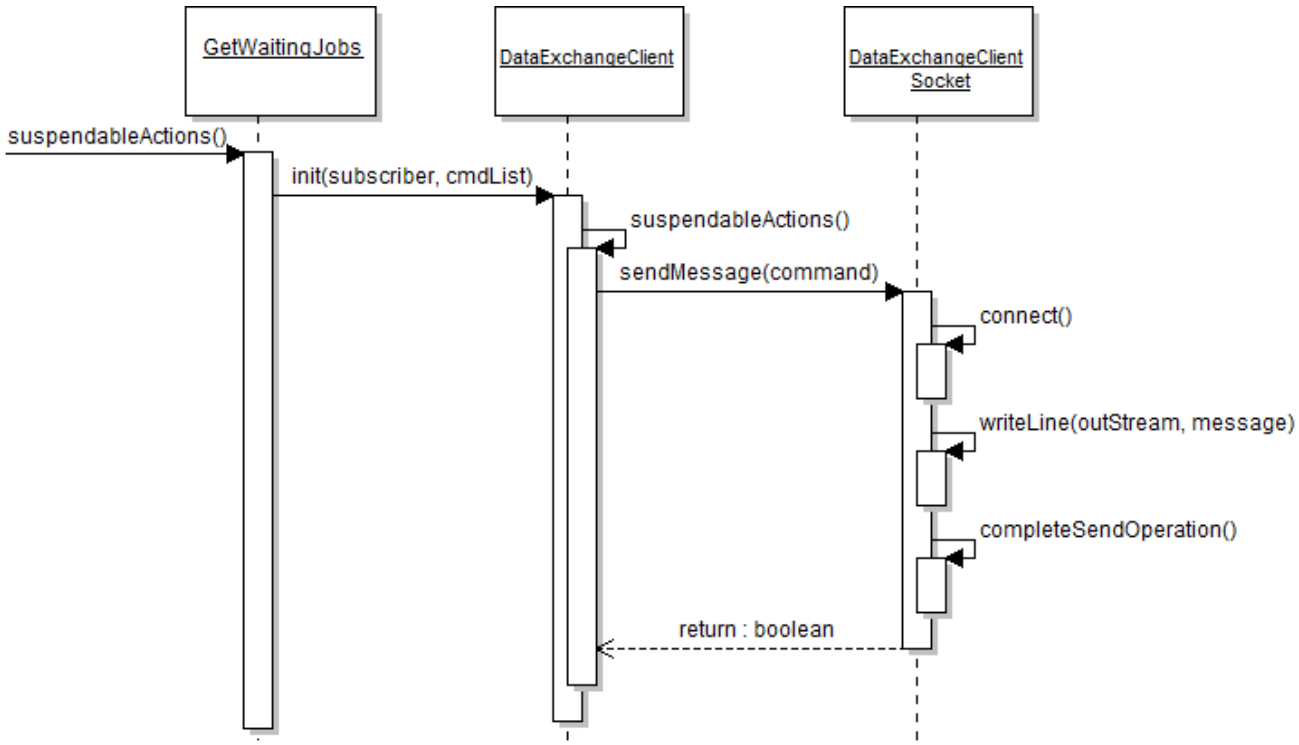
Der findes nogle værdier i databasen som kommunikationsenheden skal læse fra en gang imellem, når der gives besked på det.
I databasen findes der en tabel som hedder `FMC_OUT`. Denne tabel bruges til forskelige formål, men en af de ting som den bruges til, er at give besked til kommunikationsenheden om at den nu skal læse fra databasen for at opdatere sine værdier.
Til det formål sættet feltet `FMC_SUBSCRIBER_NAME` til ’COMM’ og `MESSAGE` til ’load param’ i tabellen `FMC_OUT`. Kommunikationsenheden læser fra `FMC_OUT` en gang imellem (ca. en gang i minuttet) og når den ser en række med disse værdier, så ved at kommunikationsenheden at den skal opdatere værdierne fra databasen.

Det modul der sørger for at læse fra tabellen `FMC_OUT` hedder `DATABASE`. Dette modul implementer klassen `GetWaitingJobs`. Når den læser en række fra databasen som ovenfor vil den initialiser objektet `DataExchangeClient` (denne klasse implementer interfacet `SenderModule` og derfor initialiseres den gennem dette interface)
I denne klasse vil metoden `suspendableActions()` kører hvor den vil prøve at sende en kommando/ besked (med værdien 'load param') ved at oprette en socket forbindelse til en port og IP-adresse som er angivet for subscriberen. Herefter sendes beskeden med metoden `writeLine()` og forbindelsen afsluttes. Et sekvensdiagram over dette ses på Figur 32.

Alle eksisterende subscribers kan findes i tabellen `FMC_SUBSCRIBERS` og i denne tabel kan man specificer hvilken IP-adresse og port den skal oprette forbindelse til.

Når man analyserer dette nærmere (for konsulenter i Visma, se linket[20]) vil man opdage at den etablere en socket forbindelse til sig selv.
Listener modulet med navnet `COMMAND_SERVER` anvendes til dette formål. Dette modul opretter en server socket forbindelse (som lytter på samme port for subscriberen 'COMM') og lytter på indkommende data.

Modulet `COMMAND_SERVER` er implementeret af klassen `CommandServer` har ansvaret for at bearbejde data der kommer ind fra socket forbindelsen.
Denne klasse implementer interfacet `ListenerModule` og bliver initialiseret ligesom alle de øvrige moduler. Dette betyder også at modulet bliver kørt af en sin tråd køres, på skift med de andre tråde.
Når denne tråd for lov til at køre (se `execute()` metoden) bliver den indkommende besked kørt gennem en række if-sætninger, for at se hvilken case den passer til. Dette er en måde at se om den indkommende data er understøttet af programmet eller ej. Hvis et af disse cases bliver sande, vil koden i if-sætningen kører. Dette er selvfølgelig afhængigt af hvilken data der modtages.



Figur 32 – Behandling af load param

[20] http://wiki.siriusit.com/confluence/display/vTrack/Socket+communication+%28load+param%29

Taget inspiration af denne undersøgelse, har jeg udvidet klassen `CommandServer`, således den kan tage hensyn til opdatering af trådende i kommunikationsenheden. Derfor har jeg oprettet en ny if-sætning der tjekker for at der er modtaget værdien 'update config' gennem socket forbindelsen (i stedet for 'load param'). Dette vil kalde `putMessage()` metoden i klassen `VtrackServer`. Herfra kaldes metoden `updateCommUnit()` i klassen `ConfigurationDB` (se evt. Figur 21).
Denne metode indeholder logik til at kunne indlæses data fra `COMM_HISTORY` tabellen som diskuteret overfor og på baggrund af dette udføre de opdateringer som der er specificeret i tabellen.
Først vil den tjekke om den skal lave en fuld genstart ej. I det tilfælde der skal laves en fuld genstart, stoppes alle trådene og modulerne fra databasen genindlæses og initialiseret som plejet. Til sidste opdateres `COMM_HISTORY` tabellen i de felter hvor reloaded er `null` for at indikere at kommunikationsenheden har indlæst de nyeste værdier fra databasen og har oprettet tråde på baggrund af disse.
Hvis ikke der skal laves en fuld genstart, vil den finde ud af hvilke moduler der stoppes, genstartes og startes (se metoderne `getModulesToStop()`, `getModulesToRestart()` og `getModulesToStart()` i `JdbcComHistory`). På baggrund af disse informationer opfyldes de ønskede opdateringer af kommunikationsenheden.
Tabellen `COMM_HISTORY` vil også blive opdatereret ved at indsætte 'reload' feltet.

For at kunne sætte denne handling i gang, efter brugeren har klikket på knappen "Update Comm Unit" gennem Miljøet, bliver man derfor nødt til at indsætte en række i `FMC_OUT` tabellen. Dette SQL kald kan laves således:

```
INSERT INTO FMC_OUT (fmc_subscriber_name, command, message, done_retries,
last_try_utc_date, next_try_utc_date)
SELECT 'COMM', 'SEND', 'update config', 0, NULL, fmc_util.local_to_utc(sysdate)
FROM dual
```

Dette kald er implementeret i klassen `JdbcComUpdateComunitDao` (se `restartComunit()` metoden i controller klassen Figur 30).

Jeg har prøvet at udfører nogle simple tests på kommunikationsenheden hvor jeg manuelt har indsat rækker i `COMM_HISTORY` tabellen som signalerede at nogle af modulerne skulle genstarte, stoppe og starte. Jeg har testet ved at oprette en socket forbindelse til kommunikationsenheden og sende beskeden 'update config' (se klassen `TCPClient` i kommunikationsenheden) og jeg har også prøvet det gennem databasen, hvor jeg satte en række ind i FMC_OUT med værdierne som bekrevet overfor.
Ud fra de tests jeg har lavet ser det ud til genstart funktionaliteten. Jeg har også testet en fuld genstart og det ser også ud til at virke.

For at opsummere status på genstart funktionen af kommunikationsenheden og for opgaven, kan man sige at når der sker ændringer i konfigurationsparametrene gennem Miljøet, skal der indsættes en række i `COMM_HISTORY` tabellen. Af informationer herfra kan kommunikationsenheden finde ud af, hvad den skal foretage sig på de enkelte moduler. Hertil mangler der logik til at kunne indsætte data i `COMM_HISTORY` tabellen, da disse rækker skal indsættes gennem Miljøet!
Opdateringen af kommunikationsenheden bliver triggeret ved at man indsætter en række i `FMC_OUT` tabellen med de værdier som beskrevet overfor.
Kommunikationsenheden indlæser fra denne tabel og finder ud af at kommunikationsenheden skal opdateres. Der oprettes en socket forbindelse til sig selv hvor værdien 'update config' sendes. Dette vil kalde på metoden `updateCommUnit()` i klassen `ConfigurationDB` som vil tage hensyn til informationerne i `COMM_HISTORY` tabellen fra databasen og udfører de opdatering der skal til for

trådene/modulerne. Kommunikationsenheden vil derefter opdatere `COMM_HISTORY` tabellen, ved at indsætte feltet i updated.

16.7 Test

Jeg har lavet 2 overordnet test for Miljøet. I den første test har jeg lavet diverse test på inputfelterne ved oprettelse og opdatering af et element til konfigurationen. I den anden test har jeg testet CRUD funktionerne på alle de forskellige elementer som en konfiguration kan bestå af.

Test 1 – Input test i Miljø

I denne test vil jeg finde ud af om koden til validering af inputfelterne fungere korrekt ved oprettelse og opdatering af hvert element som en konfiguration kan består af. Jeg har udført testen på følgende måde:

- Indsættelse af tomme værdier i alle inputfelter.
- Indsættelse af data i alle felter på nær et enkelt felt, gentaget for alle felter.
- Limit test hvor jeg har sat værdier i inputfelterne op til 25 tilfældige tegn.
- På felter der forventer et positivt heltal har jeg test med talværdier som er:
    - Negativt heltal.
    - Negativt komma tal (dansk + engelsk komma).
    - Positivt komma tal (dansk + engelsk komma).
- For tal felter som skal være indenfor en grænse (fx 0 og 100) har jeg lavet tests på øvre og nedre grænse med værdier som er ±1, i dette tilfælde ville værdierne være: -1, 0 1, 99, 100 og 101 der skulle testes.

Da denne test er ret lang har jeg valgt at tage et kort udsnit af test resultaterne med i rapporten. De fulde test resultater kan findes i Appendix B.5. Tabellen nedenfor viser resultaterne for oprettelse og opdatering af parameter.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
9.1	Input validering ved oprettelse/opdatering af parameter	Tomme værdier i alle input felter	Fejl om at navn og value skal angives	Name cannot be empty	Ok
9.2	Input validering ved oprettelse/opdatering af parameter	Name: Value: a	Fejl om at navn skal angives	Name cannot be empty	Ok
9.3	Input validering ved oprettelse/opdatering af parameter	Name: a Value:	Fejl om at value skal angives	Value cannot be empty	Ok
9.4	Limit test ved oprettelse/opdatering af parameter	Name og value på 25 random tegn	Værdierne indsættes i database og der vises en ny række	Data er indsat og en ny række vises	Ok

Tabel 9 - Input test ved oprettelse og opdatering af parameter

Test 2 – CRUD test i Miljø

I denne test har jeg udført tests for at validere CRUD funktionerne virker som de skal i miljøet.
Jeg har i denne test oprettet forskellige elementer til en konfiguration, dem har jeg prøvet at opdatere og slette, for at se om de blev korrekt fjernet fra databasen og fra Miljøet.
Når man fx sletter et modul skal dens resterende tilhørende værdier også slettes fra databasen og når man slettet en konfiguration skal de resterende konfigurationsparametre også slettes fra databasen.

Her er et kortfattet redegørelse over testens forløb:
Der oprettes en konfiguration med to parametre (P1 og P2), to timed modules (TM1 og TM2) og to listener modules (LM1 og LM2).
Til TM1 oprettes en host (TM1H), 2 parametre (TM1P1 og TM1P2) og 2 commands (TM1C1 og TM1C2).
For TM2 oprettes en host (TM2H), 1 parameter (TM2P1) og 1 command (TM2C1).
Til LM1 modulet oprettes 2 parametre (LM1P1 og LM1P2) og for LM2 oprettes en enkel parameter (LM2P1).

Derefter testes der om update og delete funktionerne virker som ønsket på de oprettede elementer.
Til P1 testes update og delete funktionen, det samme gøres for TM1H, TM1P1 og TM1C1.
Dernæst testes update og delete funktionen på selve modulet (TM1), for at se om de resterende værdier er slettet dvs. den tilhørende parameter TM1P2 og command TM1C2.

Til LM1 testes update og delete på LM1P1 og dernæst på selve modulet, for at se om de resterende værdier er slettet dvs. den tilhørende parameter LM1P2.

Update og delete funktionerne testes også på selve konfigurationen, for at se om de resterende værdier er slettet (P1, TM2, TM2H, TM2P1, TM2C1, LM2 og LM2P1).
Alle værdier der anvendes er fiktive og har ingen betydning for testen!

På denne måde testes der for alle CRUD funktioner og man kan bl.a. tjekke for om de tilhørende konfigurationsparametre til en konfiguration slettes, når brugeren ønsker at slette en hel konfiguration.

Denne test er rimelig omfattende så derfor har jeg valgt at tage en lille del af testen med i rapporten. Den fulde test og testresultaterne kan findes i Appendix B.6.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
14.1	Oprettelse af konfiguration	Name: Config Customer: Test Version: 1.0.0	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.4	Oprettelse af TM1	Name: TM1 Description: TM1 Classname: Dk.vtrack.TM1 Comments: Test TM1 Frequency: 30 Offset: 30 Active: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.6	Oprettelse af TM1P1	Name: Tm1p1 Value: Tm1p1 Comments: Test Enabled: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok

14.23	Update af TM1P1	Name: Tm1p1U Value: Tm1p1U Comments: TestU Enabled: No	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.24	Slet af TM1P1		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok
14.27	Update af TM1	Name: TM1U Description: TM1U Classname: Dk.vtrack.TM1U Comments: Test TM1U Frequency: 35 Offset: 35 Active: No	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.28	Slet af TM1		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok
14.32	Update af konfiguration	Name: Config2 Customer: Test Version: 2.0.0	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.33	Slet af konfiguration		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok

Tabel 10 - CRUD test i Miljø

16.8 Test konklusion

I denne afsnit har jeg lavet en test på de funktionelle krav til programmet. Jeg har testet de forskellige inputfelter ved oprettelse og opdatering af de forskellige elementer som en konfiguration kan bestå af. Her kan vi se at valideringen fungerer korrekt, således der kan indtastes og persisteres gyldigt data i databasen.

Derefter har jeg testet CRUD (Create, Read, Update, Delete) funktionerne for hvert enkelt element i en konfiguration. Jeg startede med at oprette forskellige elementer hvor jeg så herefter prøvede at opdater og slette dem enkeltvist. Fra resultaterne herfra kan jeg konkluderer at dette virker som ønsket.

Bemærk at der i dette afsnit ikke beskrevet om test af genstart funktionaliteten af kommunikationsenheden. Dette skyldes at der ikke er lavet noget test dedikeret i forbindelse med dette, ud over den uformelle test der er beskrevet i afsnit 16.6.
Grunden til dette er også at denne funktionalitet ikke virker 100 %.

Jeg har ikke haft mulighed for at lave test på rettighedskontrolmekanismen på de sider jeg har oprettet, da jeg ikke har kendt til en anden bruger med en anden rolle end den jeg har brugt (”hhr” som har rollen ADMIN_VTRACK). Derfor lader jeg denne test til at blive til testet og valideret af en anden person fra virksomheden.

16.9 Fremtidig udvikling

I dette afsnit vil jeg diskuterer om nogle af de ideer man kan implementere som forbedringer til programmet, hvis man ønsker at arbejde videre med projektet ved en fremtidig udvikling.

Jeg har lavet en undersøgelse af den nuværende implementation af ca. 20 moduler for kommunikationsenheden for at se hvilke parametre der er obligatoriske, i den forstand at hvis disse felter ikke er angivet vil modulet ikke virker efter hensigten eller vil kaste exceptions, og hvilke parametre der er valfrie, dvs. dem som ikke vil kaste en exception, hvis de mangler at blive angivet.
Man kan bruge denne viden til at sikre sig, at man ikke kommer til at slette parametre som er obligatoriske for modulet i forbindelse med vedligeholdelse af konfigurationen.
Der findes et Excel dokument på cd-rommen (”Analyse af vTrack parametre.xlsx”) som indeholder analysen. Analyse af hvert modul findes i sit et eget worksheet.

En anden forbedring som man kan lave og som er nice-to-have er at Miljøet tager hensyn til sprog konfigurationen i vTrack så ledeteksterne bliver på det valgte sprog. Lige nu er dette hardcoded på engelsk.

En anden forbedring er udviklingen af de sidste funktioner der mangler ved genstart af kommunikationsenheden. Dette har vi været inde på tidligere rapporten, så dette emne vil jeg ikke komme ind på igen (se afsnit 16.6 Genstart af kommunikationsenheden).
Man kunne også til `COMM_HISTORY` tabellen oprette en constraint således at den tjekkede for at man ikke kunne indsætte andre værdier til task feltet forskelligt fra ’start’, ’stop’, ’restart’ og ’reload’.
Evt. kan man bruge koden: `CHECK(TASK IN ('start', 'stop', 'restart', 'reload'))` ved oprettelse af denne tabel.

16.10 Konklusion

I dette delprojekt har jeg udviklet et miljø (en webapplikation) hvori konfigurationsparametrene til kommunikationsenheden kunne vedligeholdes med hensyn til CRUD principperne. Dette har vi realiseret ved at udvide præsentationsenheden for vTrack, med nye JSP og java implementeringer.
Præsentationsenheden er opbygget gennem Springs web framework og dette har jeg benyttet i dette projekt således de bestemte retningslinjer for projektet overholdes. Derved gøres det også nemmere for den næste i virksomheden som skal arbejde videre med projektet.

Miljøet skal ikke kunne tilgås af alle vTrack brugere og skal kun være begrænset til brugere med administrator rettigheden og dette tages der også hensyn til, med de nøgler jeg har oprettet i databasen.

Dette produkt har release ca. en gang om året og derfor er det vigtigt at afgrænse de ønskede funktioner, således de releases man udgiver til kunden virker stabile og pålidelige. Vi fik kommunikationsenheden til at læse fra `COMM_HISTORY` tabellen, men der blev ikke udviklet kode til at kunne indsætte rækker i denne tabel, derfor virker genstart af kommunikationsenheden ikke fuldt funktionelt. Vi har i rapporten diskuteret om hvad der skal gøres for at færdiggøre dette.

Til de dele som der er udviklet i forbindelse med dette projekt har jeg udført 2 omfattende tests, henholdsvis på inputfelterne i webapplikationen og test af CRUD funktionerne på de forskellige elementer som konfigurationen består af.
På baggrund af disse test kan jeg sige at vedligeholdelsen af disse konfigurationsparametre virker korrekt.

17 Del 4 – vTrack XML Parser

Dette er det sidste delprojekt jeg har arbejdet med i projektperioden. I dette delprojekt udvikles der et program der er i stand til at kunne konventer konfigurationsfilen (vtrack.xml og de øvrige filer) til SQL sætninger, som er tilpasset til de tabeller jeg har oprettet i databasen.
Jeg vil i dette afsnit beskrive nærmere omkring hvordan parseren er opbygget og dens funktioner.

vTrack XML Parser programmet er et separat java program og er uafhængig af de øvrige enheder i vTrack. Ideerne til dette projekt er kommet løbende, idet jeg havde en tidsperiode på 12 uger, for at gøre denne opgave færdig, se tidsplanen. I første omgang var det vigtigt at kunne oversætte data fra xml filen til java objekter som derefter oversættes til SQL. Senere blev det vigtigt at man kunne validere filen mod gyldig og valid xml, da oversættelsen muligvis kunne fejle eller oversættelsen blev forkert. Når dette var på plads kom der yderligere udvidelser til programmet til fordel for brugeren. Den første udvidelse var indsættelse af data i databasen direkte fra programmet. Når dette var færdigt, udvidede jeg programmet endnu engang, så man kunne konfigurere og oprette properties filen, til den nye version af kommunikationsenheden.

Dette afsnit koncentrer sig om at implementer use case U06, U07 og U08

17.1 JAXB frameworket

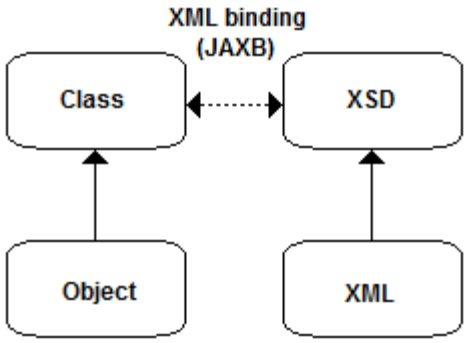
Som det første hovedproblem jeg havde i forbindelse med denne opgaver var at kunne processer data fra xml filen til en data repræsentation af java objekter, så det var nemmere at arbejde med, når jeg skulle oversætte til SQL.
Der findes forskellige parsers til java og de 2 mest populære er DOM (Document Object Parser) og SAX (Simple API for XML) parsers.
DOM parses tilbyder et interface som er af de simpleste forstå og anvende, men ulempen ved denne parser er at den gemmer alt xml data i hukommelsen og man skal processer data i en træstruktur. Af disse grunde anses DOM parsers for at være langsomme og har et højt forbrug af hukommelse. SAX parsers anvender en anden strategi. Her udvikler man callback metoder, som henter data fra xml filen efter eget ønske. Derfor bruger den mindre hukommelse og er som regel hurtigere end DOM parsers, da den ikke skal tage hensyn til hele dokumentet. Denne type af parsers kaldes også for event-baseret parser.
Ulempen for begge er at man ikke kan tilgå data i dokumentet i en tilfældig rækkefølge. Derfor kan det være svært at processes data med disse parsers. Derfor blev jeg nødt til at udtænke en anden løsning.

Ud over disse 2 populære xml parsers findes de der også en række forskellige frameworks til java som kan anvendes til at processer data fra xml dokumenter. For eksempel findes der xmlBeans, DOM4J[21] , JDOM[22] , JAXB[23] med flere.
Til dette projekt har jeg anvendt JAXB frameworket, da det samtidig også anvendes i vCatch som er et andet produkt i vores produkt suite.

JAXB står for Java Architecture for XML Binding og giver mulighed for at processer xml data uden at have kendskab til xml eller hvordan xml processeres i Java.
JAXB giver også mulighed for at tilgå data i en non-sekventiel rækkefølge som man for eksempel ikke kan i de overnævnte parsere.

[21] http://dom4j.sourceforge.net/
[22] http://www.jdom.org/
[23] http://jaxb.java.net/

En anden fordel ved JAXB er, at den simplificerer adgangen til data i et xml dokument for java programmet, ved at præsentere xml dokumentet i java format.
Det første trin i denne proces er at binde xsd skemaet for xml til et set af java klasser som repræsenterer et skema (xsd filer). Dette betyder at man genererer et set af java klasser som repræsenterer skemaet. Det der sker her er at JAXB programmet tilbyder et "binding compiler" værktøj som binder skemaet til java objekter (hvordan denne compiler er opbygget er selvfølgeligt implementationsspecifikt)[24]. Dette kaldes for xml binding. De oprettede/genererede klasser vil bliver annoteret som med annotationer af JAXB frameworket[25]. Figuren forneden viser hvordan det hænger sammen.



Figur 33 - Xml binding med JAXB

Ved udvikling af dette projekt fandt jeg et JAXB plugin til Eclipse. Dette plugin findes på cd-rommen, så man kan installere det på sin egen maskine, vis man har lyst (se mere i Appendix C).
Ved brug af dette plugin kan Eclipse automatisk genererer en java arkitektur (domæne model) ved brug at et xsd skema.

De klasser der bliver genereret har stort set den samme arkitektur som vist på Figur 7, da jeg anvender den samme xsd skema (configuration.xsd) til dette projekt. Derfor vil jeg ikke gå i dybden med denne arkitektur. Yderligere tilbyder JAXB marshalling og unmarhalling af xml filer.

Marshalling betyder at man "pakker" java objekterne ind og genererer et xml dokument som indeholder data fra disse objekter og opfylder kravene opsat i xsd skemaet.
Unmarshalling er det omvendte af marshalling. Her pakker man data fra xml filen ud og sætter det ind i de java objekter som er genereret. Dette er også den vigtigste feature som jeg har brugt.

```
1  public CommConfig parseXML(String xmlPath){
2      CommConfig cc = new CommConfig();
3      try {
4          JAXBContext jaxbContent = JAXBContext.newInstance("com.siriusit.vtrack.configuration.xml");
5          Unmarshaller unmarshallar = jaxbContent.createUnmarshaller();
6          Conf conf = (Conf) unmarshallar.unmarshal(new File(xmlPath));
7          Group group = conf.getGroup();
8          cc = readGroup(group, cc);
9      } catch (JAXBException e) {
10         //handle exception
11     }
12 }
```

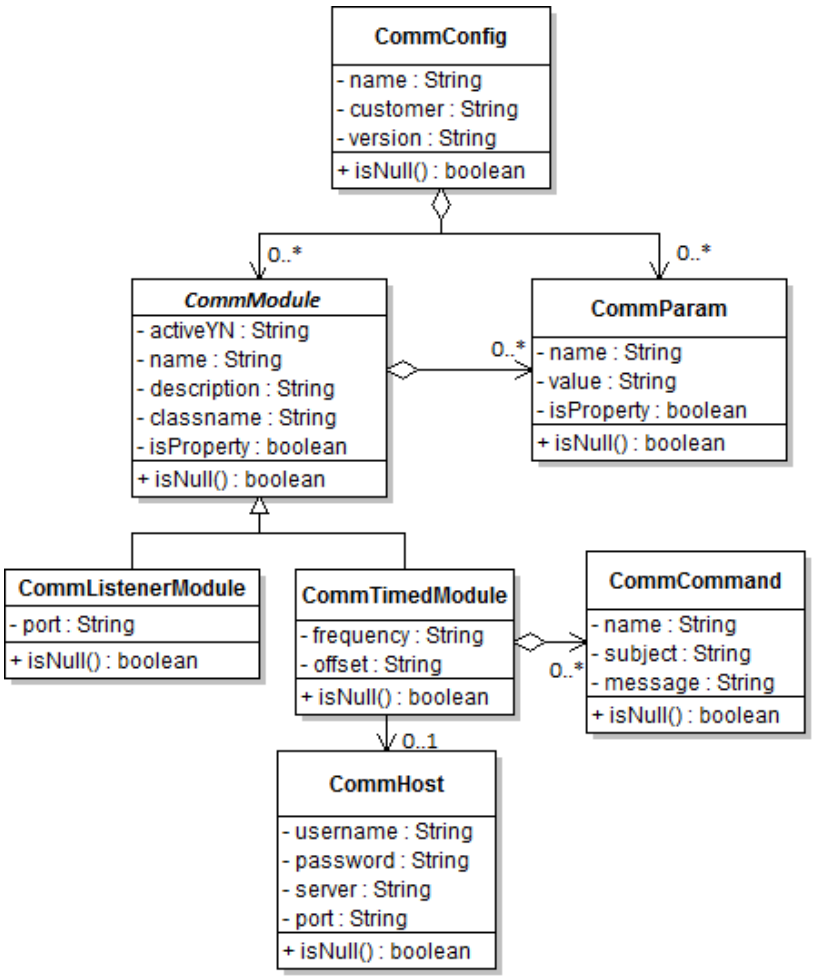
Figur 34 - Unmarshalling med JAXB og java

[24] http://www.oracle.com/technetwork/articles/javase/index-140168.html
[25] https://www.youtube.com/watch?v=4J_ytgQ96Kg

Figuren ovenfor viser et kodeeksempel hvor jeg oversætter en given xml fil til et `Conf` objekt (genereret af JAXB). Fra dette objekt hentes de resterende objekter og gemmes ind i et `CommConfig` objekt som skal returneres af metoden. Dette er et objekt som jeg selv har oprettet og er en del af domænemodellen for vTrack xml parseren.
Resten af koden i denne klasse minder meget om det der ses på Figur 6. Fra et `Conf` objekt hentes `Group`. `Group` objektet indlæses rekursivt hvor man henter parametrene fra `ParameterCollection` tagget og modulerne fra `ModuleCollection` taget sammen med deres tilhørende data.
Logikken er stort set den samme som vises på Figur 6 og derfor vil jeg ikke gå i dybden med den.

Data fra xml filen ligges ind i et domæne model som jeg har oprettet. Arkitekturen for modellen findes på Figur 35. Den minder meget om dem som anvendes i de forrige delprojekter (Figur 23), men jeg har lavet den optimering, at i stedet for `CommTimedModule` og `CommListenerModule` har en direkte association til `CommModuleParam` objektet har jeg nu sørget for at `CommModule` (super klassen) har associationen til `CommParam` objektet. Ved denne omlægning opbevares stadigvæk alle de nødvendige data fra xml filen. Jeg har heller ikke taget hensyn til om parametrene er lokale til modulet eller globale for modulerne, dette har ikke nogen betydning i dette tilfælde.

Hvert objekt har en `isNull()` metode. Denne metode bruges til at tjekke at vi ikke kommer til at indsætte rækker i tabellerne med tomme værdier.
De viste klasser nedenfor har konstruktører som tager imod objekterne genereret af JAXB, så man kan oprette instanser af de viste objekter.



Figur 35 - Domænemodel

17.2 Validering af xml

Det blev hurtigt indlysende at det var vigtigt at kunne validere xml dokumentet mod valid og gyldig xml. Uden dette så ville oversættelsen fejle og der kunne muligvis være situationer hvor data ikke var angivet korrekt, hvilket også ville medføre at oversættelsen vil bliver forkert.

Figur 36 viser et kodeeksempel med 2 metoder `isWellformed()` og `isValid()` hvor denne funktionalitet er implementeret.
Valideringen udføres ved det at oprette et `DocumentBuilder` objekt og lade xml filen blive parset af dette objekt (se linje 5 og 21). Dette skal helst ske uden der opstår fejl (exceptions).

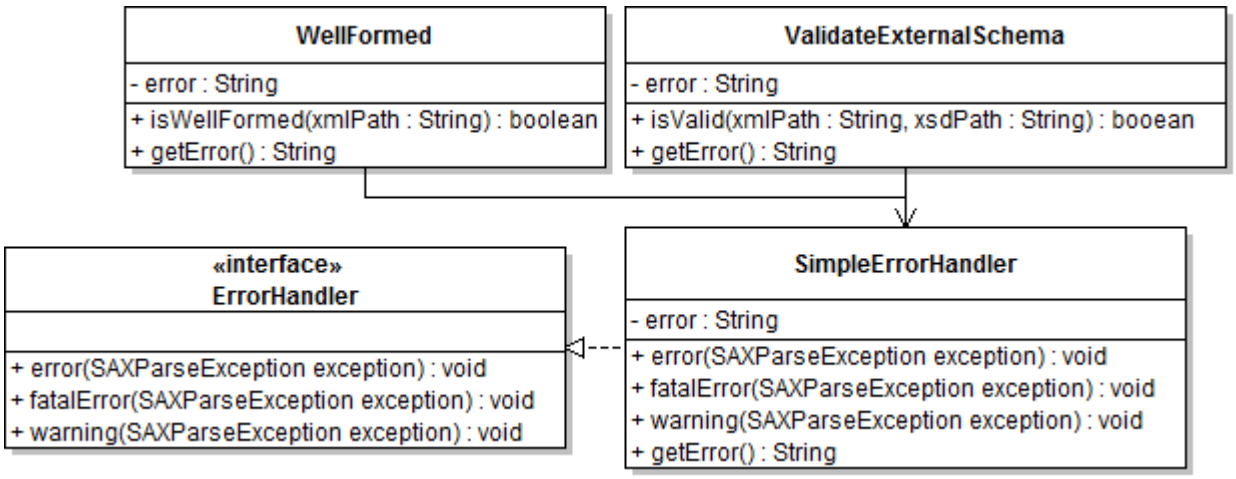
Man kan også yderligere specificere `DocumentBuilder` objektet således at man angiver et xsd skema som xml filen skal validering op mod (se linjerne 17+18). Hvis dette parses uden fejl, betyder dette af xml filen er valid og dette kan bruges af programmet.

Hvis de 2 nedstående test bestås vil programmet nu parse dokumentet om til java objekter, som beskrevet i afsnittet overfor.

```
1  public boolean isWellformed(String xmlPath) {
2      try {
3          DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
4          DocumentBuilder builder = factory.newDocumentBuilder();
5          builder.parse(new InputSource(xmlPath));
6          return true;
7      } catch (Exception e) {
8          //handle all exceptions
9          return false;
10     }
11 }
12
13 public boolean isValid(String xsdPath, String xmlPath) {
14     try {
15         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
16
17         SchemaFactory schemaFactory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
18         factory.setSchema(schemaFactory.newSchema(new Source[] {new StreamSource(xsdPath)}));
19
20         DocumentBuilder builder = factory.newDocumentBuilder();
21         builder.parse(new InputSource(xmlPath));
22         return true;
23     } catch (Exception e) {
24         //handle all exceptions
25         return false;
26     }
27 }
```

Figur 36 - Validering mod gyldig og valid xml

Koden til metoden `isWellformed()` ligger i klassen `WellFormed` og `isValid()` metoden ligger i klassen `ValidateExternalSchema`. Hvis der opstår en fejl i valideringen har her klasse en `error` string som bruges til at vise yderligere detaljer omkring fejlen der har opstået.
Hver disse klasser har en association til klassen `SimpleErrorHandler` som implementer interfacet `ErrorHandler`. Gennem denne implementation kan selv tilpasse (customize) de forskellige fejl der kan ske under valideringen.
Ved oprettelse af `DocumentBuilder` objektet (gennem Factory klassen) skal man dog lige huske at anvende `setErrorHandler()` metoden, hvor man angiver klassen der implementer interfacet (ej vist på Figur 36)[26].
Koden for validering findes i pakken: dk.vtrack.comunit.parser.validation og består af 3 klasser som vist på Figur 37.



Figur 37 - Klassediagram xml validering

[26] http://docs.oracle.com/javase/1.5.0/docs/api/org/xml/sax/ErrorHandler.html

17.3 Oversættelse og fil oprettelse

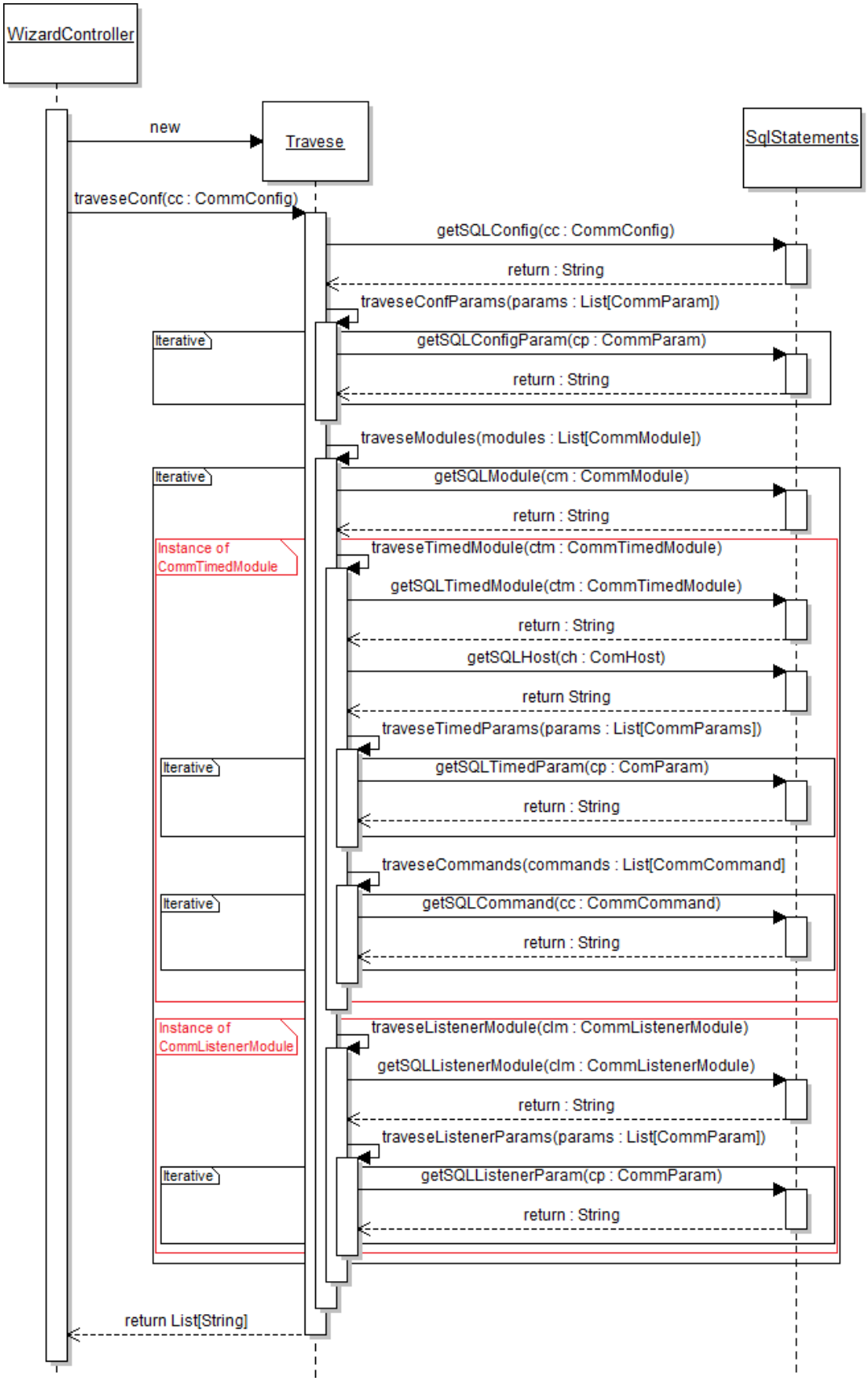
Vi har valideret xml dokumentet og fået den oversat til java objekter, vi skal nu lave dette om til SQL sætninger som passer ind til den databasearkitektur som jeg har oprettet tidligere i projektet. Rækkefølgen af kaldene skal også tilpasses, så man undgår fejl som for eksempel foreign key constraint fejl.

Funktionaliteten til at oprette de nødvendigt SQL sætninger findes klassen `SqlStatements`. Denne indeholder statiske metoder som man kan kalde og som vil returner koden for SQL forespørgslen. Men dette er ikke nok, vi skal huske på at strukturer disse kald i den rigtige rækkefølge. Dette sørger `Travese` klassen for. Denne klasse tager imod et `CommConfig` objekt og løber den igennem. Derved sørger den for at SQL kode genereringen arrangeres på en måde, at den giver mening når det skal indsættes i databasen.

Som du måske har bemærket så indeholder klasserne `CommModule` og `CommParam` variablen `isProperty`. Dette felt indikerer om modulet/parameteren er valgt som en parameter til properties filen. Hvis dette er valgt som en del af properties filen, skal de som regel ikke oprettes som et SQL kald, da dette vil give redundante data (da data både vil findes i databasen og properties filen), men hvis man ønsker dette, stiller programmet også denne funktionalitet til rådighed. Yderligere angående dette kommer vi ind på, senere i rapporten.
Sekvensen af gennemløbet for oversættelsen kan ses i sekvensdiagrammet nedenfor (Figur 38).

Jeg opretter først SQL sætningen for et `CommConfig` objekt og derefter for de globale parametre. Dernæst oprettes SQL sætningerne for modulerne med `traveseModules()` metoden. Her tjekkes der om modulet er en instans af et `commTimedModule` objekt eller `CommListenerModule` objekt. Afhængig af typen vil deres tilhørende værdier oversættes til SQL i korrekt ordner.
Alle disse kald returner en liste af strings (som indeholder SQL kaldene) som gives videre til et `Writer` objekt. Dette objekt anvendes til at oprette filer og skrive af data i dem, ved at tage imod en liste af string som argument til metoden. Disse skrives på filen ved at løbe elementerne igennem (ej vist på Figur 38, se evt. Figur 41).

SQL syntaksen er lidt forskellig afhængig af hvilken platform (database) man udvikler til. Derfor har jeg valgt at lave kaldene så de passer ind til en Oracle databaser, da det er den vi bruger i virksomheden og det den som er installeret hos vore kunder. Det vil også være ude af projektets scope at begynde at implementere database kald der tager hensyn til andre platforme end Oracle. Koden i `SqlStatements` returner derfor kun SQL kode med hensyn til Oracle databaser.

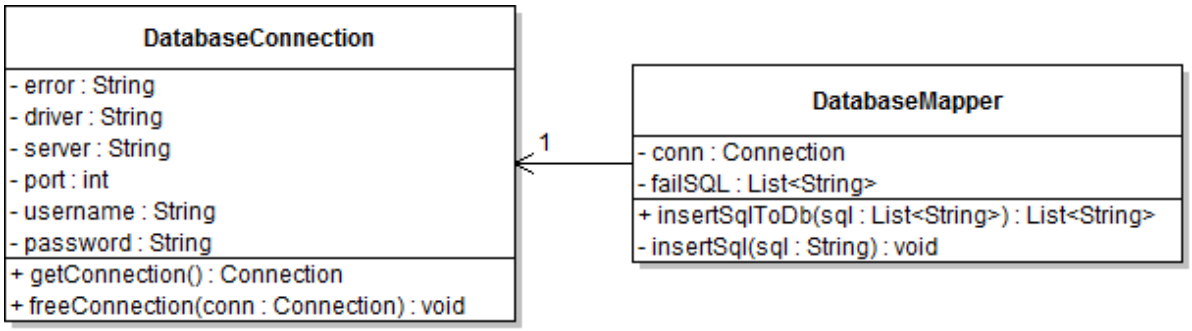


Figur 38 - Oversættelse fra java til SQL

17.4 Database

Funktionaliteten af xml parseren var stort set færdig på dette tidspunkt, med hensyn til det den skulle kunne og var derfor fuld funktionelt.
Efter dette var på plads tænke jeg på at give brugeren mulighed for at kunne indsætte data direkte i databasen, i stedet for at oprette en SQL fil som brugeren manuelt skal indsætte.
Rækkefølgen af SQL sætningerne var på plads, så det skulle der ikke bekymres om. Derfor lavede jeg en klasse til at etablere og frigive forbindelse til en database (`DatabaseConnection`) og en anden klasse som indsætter data ind ved brug af database forbindelsen (`DatabaseMapper`). På Figur 39 vises klasserne som jeg har oprettet.
`DatabaseMapper` tager imod en liste af strings (genereret af `Travese` klassen) og løber disse igennem en efter en og indsætter disse i databasen (med `insertSql()` metoden). I det tilfælde der opstår fejl, af en eller anden grund gemmes disse kald i en liste sammen med fejlårsagen (`failSQL`). Hvis listen ikke er tom vises der en fejlmeddelelse til brugeren om at der var nogle af SQL kaldene mislykkedes at blive indsat i databasen. Brugeren kan nu vælge om man ønsker at gemme disse kald i en separat fil eller ej. Hvis brugeren gerne vil have filen, vil programmet oprette denne, men brugeren skal selv sørge for at rette fejlene og indsætte disse kald manuelt.

Fordi vi etabler forbindelse til en oracle database har det været nødvendigt at anvende Oracles JDBC (Java DataBase Connecter) driver til dette projekt (ojdbc6.jar).



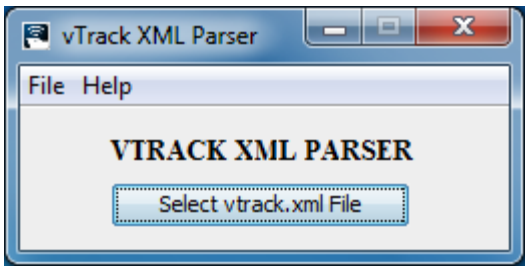
Figur 39 - Database connection

17.5 Properties fil konfiguration

Properties filen som anvendes af den nye version af kommunikationsenheden blev oprettet på baggrund af data fra vtrack.xml filen, og denne fil indlæses af dette program, så hvorfor ikke give brugeren mulighed for at konfigurere denne og oprette properties filen herfra? Ja, dette var tanken bag denne udvidelse for properties fil konfigurationen.
Fra delprojekt 1 så jeg, at det var modulerne og parametrene fra `ParameterCollection` tagget som man skulle angive til properties filen. Derfor har jeg tilføjet en værdi, `isProperty`, til `CommModule` og `CommParam` objekterne. Dette felt indiker at modulet eller parameteren skal tilføjes i properties filen.
Jeg udviklede en klasse `PropertiesConfigurationDialog` som giver mulighed for at vælge hvilke moduler og parametre man ønsker at tilføje til filen. Dette gøres ved at klikke på en check box ud fra værdierne.
Værdierne herfra tilføjes en liste (`ArrayList<String>`) som gives videre til `Writer` klassen og som kan oprette properties filen med de givne data.

17.6 GUI og arkitektur

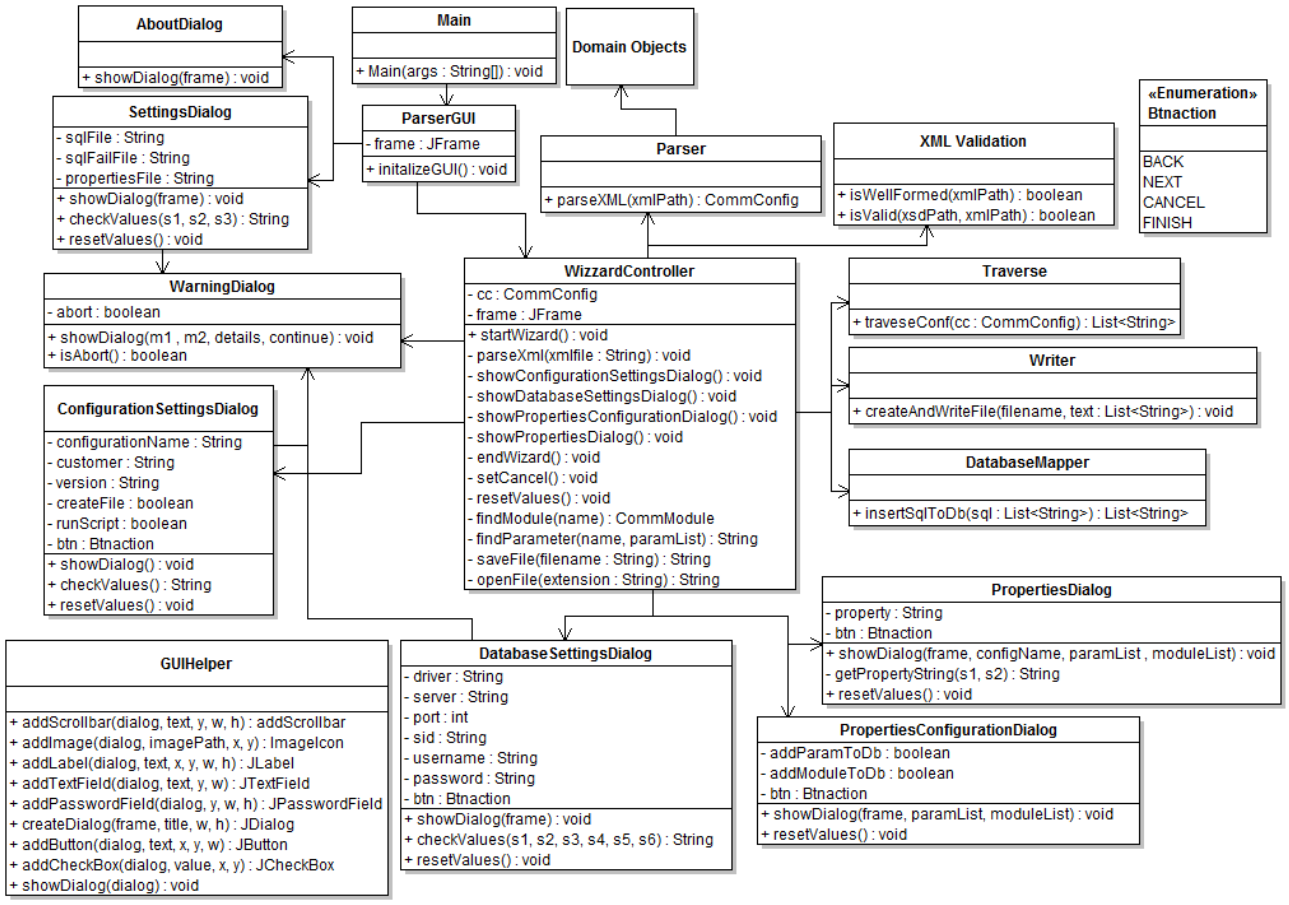
For at gøre interaktionen nemmere mellem programmet og brugeren, har jeg valgt at lave menuerne i en grafisk brugergrænseflade. Hertil anvendes Java Swing og AWT (Abstract Window Toolkit) objekter til at oprette og vise de forskellige grafiske komponenter på skærmen. Når programmet starter op, vil der vises et skærmbillede som vist på figuren nedenfor (Figur 40).



Figur 40 - Hovedmenu

Hovedmenuen er ret simpelt og består af en menubar (`JMenuBar`) med 2 menuer (”File” og ”Help”) og en knap (`JButton`).

Jeg vil ikke gå så meget i dybden med koden omkring de grafiske menuer som programmet består, men jeg vil forklare hvad og hvordan de forskellige klasser anvendes. Et klassediagram over programmets generelle arkitektur ses på Figur 41 nedenfor.



Figur 41 - GUI arkitektur

Programmet starter i `Main` klassen som opretter en instans af objektet `ParserGUI` og kalder på metoden `initializeGUI()`. Denne metode vil vise et skærmbillede som overfor (Figur 40).
I menuen "Help" findes der en undermenu kaldet "About…", når brugeren klikker på denne, vil metoden `showDialog()` i `AboutDialog` kører. Denne vil oprette en `JDialog` hvor der er sat en figur(`ImageIcon`) diverse labels på(`JLabel`), så man kan se yderligere detaljer omkring programmet som fx version, og udgivelsesdato.
De resterende `Dialog` klasser vist på Figur 41 er opbygget næsten på samme måde. Afhængig af funktionaliteten af klassen oprettes der fx tekstfelter (`JTextField`) checkbokse (`JCheckBox`), knapper (`JButton`) eller noget fjerde.
For at gøre implementeringen af de forskellige GUI komponenter nemmere har jeg lavet en hjælpeklasse (`GUIHelper`). Denne klasse indeholder en række metoder som kan oprette de grafiske komponenter på en nemmere måde. Derfor kan man med et enkelt metodekald (på en linje) for eksempel få lavet en knap, i stedet flere linjes kode. Dette sparer en del kode, da knapper og lignende objekter oprettes og anvendes tit. Koden for oprettelse disse minder desuden også meget om det samme, så for at undgå at gentagelser i koden en hel masse gange, har jeg lavet denne klasse.

I menuen "File" findes der to undermenuer, "Exit" og "Settings". Et klik på "Exit" knappen vil afslutte programmet.
Hvis der klikkes på "Settings" knappen vil `showDialog()` metoden i `SettingsDialog` køre. Dette vil åbne et nyt et skærmbillede. I dette skærmbillede har man mulighed for at specificere filnavnene på de tre filer som kan genereres af programmet. De tre filer er, SQL filen som indeholder alle de genererede kald, SQL filen som indeholder kald, der fejlede ved indsættelse i databasen og navnet på properties filen som anvendes af den nye version af kommunikationsenheden. Disse 3 felter er per default givet af programmet, men hvis man ønsker at ændre i dem, giver denne dialog mulighed for det. I det tilfælde må felterne ikke være tomme. Her sørger `checkValues()` metoden for tjekke værdierne er gyldige. Hvis tjekket fejler, vil der vises en fejlmeddelelse ved brug af klassen `WarningDialog`.

Denne klasse anvendes til at vise fejlmeddelelser ud til brugeren i tilfælde der opstår fejl i programmet. Menuen består af 3 knapper, "continue", "abort" og "details". "Details" knappen vil udvide menuen, så man kan se yderligere detaljer omkring fejlen. "abort" knappen vil stoppe den nuværende handling og gå tilbage til forrige menu, mens knappen "continue" bruges til at give brugeren mulighed for at fortsætte, med det, man er i gang med.
Det er metoden `showDialog()` i `WarningDialog` der viser denne menu. Metoden tager imod 3 string værdier og en boolean værdi. m1 og m2 er henholdsvis 2 korte sætninger (spredt over 2 linjer på menuen) som siger at der er opstået en fejl. Værdien i details stringen anvendes til at vise yderligere detaljer om fejlen, hvis brugeren klikker på "details" knappen.
Hvis man kan forudse at fejlen som vises til brugeren ikke vil give yderligere konsekvenser senere i processen, kan man sætte (boolean) værdien continue til true. Denne giver brugeren mulighed for at forsætte med processen, da "continue" knappen vil være aktiv. Ellers skal dette felt sættes til false og knappen vil være deaktiveret. Programmet giver altid brugeren mulighed for at anvende knappen "Abort" til at afslutte processen.

Så snart man klikker på knappen "Select vtrack.xml File" som vist på Figur 40, vil klassen `WizardController` tage i aktion med `startWizard()` metoden. Her vil den som det første bede brugeren om at navigere hen til den fil som skal parses (med metoden `openFile()`). Når filen åbnes, valideres den med metoderne `isValid()` og `isWellformed()` som beskrevet i forrige afsnit. Hvis filen består disse 2 tests vil metoden `parseXML()` starte, som finder sted i `Parser` klassen. Den har til formål at parse xml filen til et `CommConfig` objekt.

Når filen er oversat har vi næsten alle de informationer vi skal bruge til at kunne indsætte data i databasen, men der er nogle få informationer som ikke findes i xml filen, som brugeren bliver nødt til at angive manuelt når konfigurationen skal indsættes i databasen. Dette er data som skal ind i `COMM_CONFIGS` tabellen. Husk det er denne er "hovedtabellen" og det er den tabel som holder styr på resten af konfigurationen. Brugeren vil derfor få vist et skærmbillede, hvor det er muligt at indsætte data hertil.

vTrack XML Parser programmet består af en 4 trins wizard som guider brugeren trin for trin ved oversættelse af xml filen. I bunden af hver disse vinduer, findes der 4 knapper, back, next, cancel og finish. Back knappen tilbringer brugeren til forrige menu, next knappen anvendes til at føre brugeren til næste menu, cancel knappen anvendes til at afslutte processen og finish knappen afslutter wizarden og oversætter xml filen og genererer de ønsker filer.
Til disse events bruges enumarationen `Btnaction` så `WizardControlleren` kan følge med i hvad der skal foretages, når der skiftes fra menu til menu (de 4 metoder mellem `showConfigurationSettings()` og `showPropertiesDialog()` sørger for dette, se klassen `WizardController` på Figur 41).

I første trin angives de manglende data (konfigurationsnavn, kunde og version). Ansvaret for dette er tildelt til klassen `ConfigurationSettingsDialog`. Den sørger for at disse værdier angives. Ydere kan man i denne dialog specificere om man gerne vil oprette SQL filen og indsætte SQL scriptet i databasen (findes som checkbokse på skærmbilledet).
Når brugeren klikker på et af de nedstående knappen i dialogen kan `WizardControlleren` ved hjælp af værdien på `Btnaction` finde ud af hvilken menu man skal videre til. Metoden `showConfigurationSettingsDialog()` i `WizardController` bruges til at "kontroller" denne handling og vise den rigtige dialog efterfølgende.

`showDatabaseSettingsDialog()` metoden i `WizardController` vil kalde `showDialog()` metoden i `DatabaseSettingsDialog`. Dette er den anden menu i wizarden, i denne menu angives informationer til at kunne etabler forbindelse databasen, hvor man ønsker at gemme konfigurationen.
Inden menuen vises vil programmet søge efter LOGGER modulet i `CommConfig` objektet sammen med en række bestemte parametre fra dette modul. Hvis dette modul findes vil programmet indsætte de fundne værdier i menuen. Husk (fra delprojekt 2) at det var dette modul som anvendes til at etablerer database forbindelse, så dette modul må indeholde data til at kunne oprette database forbindelsen.
Denne menu vises kun hvis man i den forrige menu har valgt at indsætte SQL scriptet i databasen, ellers springes dette trin over og man kommer videre til trin 3 i wizarden.

I trin 3 vil `showPropertiesConfigurationDialog()` metoden tage i aktion, i denne metode bliver `showDialog()` metoden kaldt i klassen `PropertiesConfigurationDialog`. I denne menu angiver man hvilke moduler og parametre som man ønsker at indsætte i properties filen.
Parametrene og modulerne vil bliver listet op i en tabel format, så brugeren vil have mulighed for at vælge hvilke parametre og moduler man ønsker at tilføje til properties filen.

Som det sidste trin i wizard'en vil `showPropertiesDialog()` metoden køre. Denne menu bruges til at vise den konfigurerede properties fil. Dette gøres gennem metoden `showDialog()` i `PropertiesDialog` klassen. Man kan ikke ændre indholdet af properties filen i denne menu (kan gøres efter filen er genereret)! Da dette er det sidste trin i wizarden har man nu mulighed for at klikke på finish knappen hvor de ønskede handlinger vil tage plads (gøres af `endWizard()` metoden).
Ved hjælp af `traveseConf()` metoden i klassen `Travese` bliver `ComConfig` objektet oversættes til SQL kald i den korrekte rækkefølge, derefter vil `Writer` klassen oprette og skrive en fil med disse data i. `DatabaseMapper` klassen tages i brug, hvis brugeren har valgt at indsætte SQL kaldene i databasen.
Når disse ting er færdige vil `WizardControlleren` afslutte og vende tilbage til hovedmenuen.

17.7 Test

Jeg har til dette produkt udført black-box test nogle af de vigtigste funktionelle krav som var stillet op til dette delprojekt. De nedstående afsnit omhandler disse.

Valideringstest

I denne test vil jeg teste om valideringen virker. Her laves der to tests. I den ene test vil jeg parse en fil som ikke er gyldig (se filen test1.xml på cd-rom) og i den anden test vil vi parse en anden xml fil som er gyldig men ikke valid (se filen test2.xml på cd-rom).

Test Nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
15.1	Virker validering mod ugyldig xml fil?	test1.xml	Programmet viser fejl og stopper	The markup in the document following the root element must be well-formed.	Ok
15.2	Virker validering mod ikke valid xml fil?	test2.xml	Programmet viser fejl og stopper	cvc-elt.1: + unexpected element	Programmet stop-per efter 2 fejl-meddelelser er vist.

Tabel 11 - Valideringstest resultater

I test 15.2 viser programmet at der er en fejl (cvc-elt-1), men dette stopper ikke programmet, da man har mulighed for at klikke på ”continue” knappen, når fejlmeddelelsen vises. Når man klikker på ”continue” knappe fortsætter programmet men der vises omgående endnu en fejl (unexpected element) og først herefter kan man ikke fortsætte.

Parser test

Jeg vil i denne test oversætte ”Simple local vtrack.xml” filen som jeg har oprettet i forrige delprojekt og se og jeg vil få genereret en SQL fil som er magen til ”Simple local vtrack test.sql”, som er en fil jeg havde oprettet i forrige delprojekt.
Den genererede fil (vTrackXML.sql findes på cd-rom) fra programmet sammenlignes manuelt med ”Simple local vtrack test.sql”.

Test Nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
16.1	Er oversættelsen korrekt?	Simple local vtrack.xml	Genereret fil er identisk med Simple local vtrack test.sql	Filene er ens men uden mellemrum og kommentar som i ”Simple local vtrack test.sql” filen	Ok

Tabel 12 - Parser test resultater

Ved den manuelle sammenligning så jeg at der var nogle små ting som ikke passerede i overens med ”Simple local vtrack test.sql”. Der var en host række som skulle indsættes lidt tidligere end angivet i ”Simple local vtrack test.sql”.
Kommentar på et af felterne var ikke genereret (dette understøttes ikke af programmet). Værdierne for Frequence og Offset sat med gåseøjne i ”Simple local vtrack.sql” forhold til det der blev genereret.
Alle disse ting har ikke noget rigtig betydning for oversættelsen og derfor anses testen som bestået.

Database test

Dette er den sidste test som jeg har lavet i forbindelse med dette program.
I denne test vil jeg parse "Simple local vTrack.xml" filen og indsætte dette i databasen gennem programmet. Dette sammenlignes med data der ville indsættes når "Simple local vtrack test.sql" blev kørt af en database.

Test Nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
17.1	Er oversat fil indsæt i database korrekt?	Simple local vtrack.xml	Data identisk når Simple local vtrack test.sql indsættes.	intet	Ok

Tabel 13 - Database test resultater

Der var en kommenter ud fra en `Parameter` tag som ikke blev parset, denne findes i "Simple local vtrack test.sql". Oversættelse af kommentarer understøttes ikke af programmet, så derfor er det i orden at denne ikke oversættes. Resten af filen blev parset og indsat i databasen som den skulle, derfor er resultatet af denne test ok.

17.8 Test konklusion

Jeg har ikke haft særlig meget tid til at kunne teste programmet grundigt og derfor har jeg kun lavet test på de vigtigste dele af programmet (oversættelse og indsættelse af data i databasen/fil). Derfor kan jeg ikke udelukke at der muligvis kan være fejl og bugs i programmets øvrige dele (fx i wizarden).
Med de tests jeg har lavet kan man sige at de vigtigste funktioner virker og funktionsdygtige. Der er fx ikke lavet specifikke test til metoderne i klasserne `Traverse`, `Writer` og `Parser`. Men set ud fra de tests som jeg har lavet i overstående afsnit, ser det ud til, at disse også virker korrekt, i den betragtning af, at disse dele bliver anvendt når jeg kørte de tests som jeg har lavet.

17.9 Fremtidig udvikling

Programmet understøtter de mest nødvendige funktioner som man har brug for, så hvis man ønsker at lave fremtidig udvikling på dette projekt er det mest nice-to-have funktioner som man kan implementere.

En af disse nice-to-have funktioner er oversættelsen af kommentarerne fra xml filen. Det lykkes mig aldrig nogensinde at få læst kommentarerne fra xml filen ind, så de kunne indsættes i tabellerne, da jeg nu havde taget hensyn til dem. Jeg har ikke undersøgt området grundigt pga. tid og jeg ved det rent faktisk ikke om det er muligt om man kan oversætte kommentarerne. Måske skal man bruge et nyt framework eller noget andet. Hertil bør man lave en undersøge for at finde ud af det. Hvis det lykkedes at oversætte kommentarerne, så behøver man nemlig heller ikke selv skrive så meget kommentar til konfigurationsparametrene, når man flytter xml filen over til databasen.

17.10 Konklusion

Vi har i delprojekt udviklet et nyt værktøj til vTrack, som er i stand til at kunne oversætte de nuværende konfigurationsfiler til kommunikationsenheden om til SQL kald. Disse kald kan mappes ind i de tabeller som jeg har oprettet i databasen fra delprojekt 1.
Programmet består af en 4 trins wizard som guider brugeren trin for trin igennem til at oversætte og oprette de ønskede filer (SQL og properties fil).

Dette delprojekt kom til idé, da vi diskuterede hovedopgaven i samarbejde med vejlederne fra virksomheden. Efter dette møde har jeg gået rundt og har haft denne ide i tænkerne. Senere besluttede jeg mig for at udvide projektet med dette delprojekt og startede på at udvikle det, når jeg fandt tid til det. Efterhånden som tiden skred frem, fik jeg lavet de mest grundlægende funktioner og derefter begyndte jeg med at lave de to yderligere udvidelser til programmet. En udvidelse til at kunne indsætte data direkte til databasen og en anden udvidelse til at kunne konfigurerer og oprette properties filen som anvendes af den nye version af kommunikationsenheden.
Projektet er primært lavet sideløbende med de øvrige delprojekter og havde derfor en tidsperiode på 12 uger til at gøre det færdigt.

Udover udviklingen af dette program har jeg udført en hurtig test på de vigtigste dele af programmet og kan på baggrund af denne test, kan man sige at programmet opfylder dens primære funktioner.
Dette program er lavet med det formål at kunne optimer tidsforbruget og mindre indsats når vi skal opgradere vore kunders kommunikationsenhed til den nye version.

Der findes der en håndbog for programmet som fortæller hvordan man anvender vTrack XML Parseren. Du kan finde det i appendix (se appendix D).
På cd-rommen kan du også finde kildekoden for projektet, som du kan importer til Eclipse og der findes også en kørbar jar fil (vTrack XML Parser.jar) som vil starte programmet op. Det krævet dog at du har java installeret på din computer først.

18 Konklusion

Jeg har på en tidsperiode over 5 måneder (sep. 2012 – jan. 2013) arbejdet på et projekt med det formål at opnå en B.Eng. grad indenfor IT.
I dette projekt har jeg arbejdet med et Vessel Monitoring System (VMS), kaldet vTrack. vTrack består at forskellige dele, hvor jeg har arbejdet med kommunikationsenheden, præsentationsenheden og databasen. Kommunikationsenheden har bl.a. til formål at hente data fra forskellige satellitudbydere og eksterne data kilder og persistere disse en database, således disse kan vises til brugeren gennem præsentationsenheden. Kommunikationsenheden starter op ved at indlæse en konfigurationsfil (baseret på xml format), så den ved hvilke eksterne enheder den skal oprette forbinder til.
I det tilfælde hvor konfigurationsfilen skal ændres med nye værdier, er man nødt til at genstarte kommunikationsenheden manuelt, så ændringerne kan træde i kraft. Ulempen ved denne løsning er at dette giver nedetid.
Denne enhed er ca. 10 år gammel og med de udvidelser der er kommet gennem årene, er filen efterhånden blevet så stor og kompleks, at den er svær at overskue, idet man også skal tage hensyn til 4 xml filer.

Dette projekt har til formål at optimere på denne problemstilling. På baggrund af disse krav og af resultaterne fra min foranalyse, kravspecifikationen, use cases, projekt planlægning og min løsningsstrategi blev der udviklet en løsning som har været realiseret ved at udføre 4 mindre delprojekter.

Første delprojekt har bestået af udvikling af en databasearkitektur, som giver mulighed for at kunne mappe alle de informationer der findes i konfigurationsfilen til databasen.

Kommunikationsenheden skal nu afkobles fra den xml baserede konfiguration og integreres med databasen, hvilket af en stor omlægning. Ved denne omlægning er det derfor vigtigt at kommunikationsenheden stadigvæk virker stabilt og pålideligt. Yderligere skal vi være sikre på at funktionaliteten bevares mod den gamle løsning. Dette har vi valideret ved at udføre en række unit tests som beskrevet i Del 2.

Da vi persisterer data fra xml filen i en database, bliver det nu endnu sværere for brugeren at tilgå konfigurationen og lave de ændringer som man kan have behov for. Derfor blev der udviklet et Miljø (en webapplikation) til at kunne interagerer med konfigurationsparametrene gemt i databasen og derved give brugeren mulighed for at kunne foretage de ønskede ændringer i konfigurationen. Miljøet stiller alle CRUD funktioner til rådighed, så man kan foretage sig lige præcist den handling man har lyst til på parametrene. Enhver ændring der foretages i konfigurationen kan medføre automatisk en partiel eller fuld genstart af modulerne i kommunikationsenheden, uden af man selv skal genstarte enheden manuelt.

Som det sidste delprojekt, har jeg udviklet et simpelt program der giver mulighed for at kunne oversætte de nuværende konfigurationsfiler (baseret på xml) om til SQL kald, således disse kan mappes ind til den databasearkitektur som jeg lavet i projektet. Disse data kan anvendes af kommunikationsenheden og vil kan ses på Miljøet lige efter man har persisteret disse data i databasen.
Med dette produkt kan vi spare tid for en manuel oversættelse af konfigurationsfilerne for vore kunder, når de skal til at anvende den nye version af kommunikationsenheden. Yderligere stille dette program forskellige nice-to-have features som medfører at opdateringen til den nye version af kommunikationsenheden kan afvikles hurtigt og pålideligt.

Projektet er i nuværende tilstand er ikke 100 % færdigt, da genstart af kommunikationsenheden ikke virker som den skal. Den primære årsag til dette er, at præsentationsenheden ikke indeholder logik til at kunne indsætte data i en tabel i databasen (`COMM_HISTORY`). Derfor er denne opgave ladt til fremtidig udvikling.

Jeg har samtidig også for hvert delprojekt forklaret hvilke andre forbedringer man kan foretage sig, i en fremtidig udvikling af projektet som en helhed.

Dette projekt har været et optimeringsprojekt, så lads os kigge på, hvilke optimeringer vi har opnået gennem dette projekt:

- Migrering af konfigurationen til databasen har medført at vi nu ikke længere behøver frameworket xmlBeans og det som den stiller til rådighed. Vi behøver heller ikke xml filer eller xsd skemaer til opstart af kommunikationsenheden.

- Miljøet giver mulighed for at kunne ændre i konfigurationsparametrene på en brugervenlig måde, hvilket forøger abstraktionsniveauet af konfigurationen end en lang og kedelig xml fil, som kan virke uoverskueligt og yderligere når man tænker på, at denne havde referencer til andre xml filer.

- Miljøet giver også mulighed for at genstarte kommunikationsenheden partielt eller fuldt efter brugerens ændringer. Her formindsker vi kommunikationsnedetiden, idet vi ikke længere behøver en manuel genstart.

- Xml parseren sparer tid og mindre bestræbelser af produktionen af den nye version af kommunikationsenheden til vores kunder.

Udover de bidrag jeg har foretaget til dette projekt, har dette projekt også foretaget bidraget til mig som jeg kan tage med fra dette projekt. Jeg har bl.a. fået et indblik af 2 nye frameworks til java, Spring og JAXB og jeg har lært JSTL bedre at kende. Jeg har samtidig også forøget mine samarbejdsevner med andre ingeniør, da jeg primært har arbejdet i et ingeniørrelevant arbejdsmiljø under dette projekt. Udover dette har jeg også fået bedre indsigt i Unified Proces (UP) og iterativ udvikling. Alt sammen noget, som jeg kan tage med i fremtidige projekter.

Selvom projektet ikke er 100 % færdigt, må jeg konkludere målene til dette projekt er opfyldt.
Visma Consulting forventer af disse forbedringer vil indgå i den næste version af vTrack produktet.

19 Vejledere

Navne på vejledere på projektet henholdsvis fra DTU og virksomheden, Visma Consulting:

DTU- vejleder:
Carsten Witt (Associate Professor)

Virksomhedsvejledere:
Klaus Mogensen (Databasearkitekt)
Mikael Grønfeldt(Projektleder)
Søren Lund (Software arkitekt) – Primær vejleder.

20 Kode konventioner

vTrack er udviklet i Java derfor udvikles al kildekode efter standard Java konventioner. Dvs. UpperCamelCase for klasser og lowerCamelCase for variabler og metoder. Konstanter defineres med Upper_Camel_Case separeret med understreger. Al kildekode udvikles på engelsk. Metoder og klasser skal have Java Doc.
Den udviklede SQL kode skal også have kommentar de øvrige konventioner angående den udviklede kan findes i afsnit 14.5.

21 Teknologier

Her er en rækker diverse teknologier/programmer der anvendes i forbindelse med projektet:

- **Oracle Database** – 10/11g Express Edition. Databasen der anvendes i forbindelse med projektet.
- **SQL** – Structured Query Language, et programmeringssprog der anvendes i forbindelse med relationelle databaser og til at håndtere data i databasen.
- **Java** – Java er et objekt orienteret programmeringssprog som oprindeligt blev udviklet af Sun, men som er nu overtaget af Oracle. Java anvendes som det primære programmeringssprog i forbindelse med projektet. Der anvendes Java version 1.6 update 21, som var specificeret under installationen af vTrack.
- **Spring** – Et java-baseret framework. Fra dette framework anvendes komponenter til at tilgå databasen og i forbindelse med web-baserede user interfaces. Web frameworket i Spring anvender Model-View-Controller paradigmet.
- **Microsoft Word** – Tekstbehandlingsprogram der anvendes i forbindelse med dokumentation under projektet.
- **PDF** – Portable Document Format. En standart filformat som er udviklet af Adobe og bør kunne anvendes i næsten alle platforme. Pdf anvendes til aflevering de endelige dokumenter.
- **Violet** – En gratis UML editor som er distribueret under GNU General Public licens. Anvendes til at lave UML diagrammer med.

- **MySQL Workbench** – Et udviklingsmiljø hvori der bl.a kan udvikles SQL scripts og E/R diagrammer. Man kan også oprette forbindelser til databaser gennem dette program. Dette program er dog mest egnet til MySQL databaser (ikke Oracle som anvendes).

- **GanttProject v2.5**– Et gratis software produkt som er distribueret under GNU General Public Licens. Bruges i projektet i forbindelse med at lave tidsplaner og tidsestimater for projektet og til at få et overblik over projektets status.

- **Eclipse Helios version 3.6.1** – Primær udviklingsværktøj som anvendes gennem projektet.

- **SVN –** Subversion, bruges til software revisionering og revisionskontrol og anvendes som et repository til at opbevare og dele source kode med andre udviklere. Herunder anvendes følgende programmer:

    - TortoiseSVN, som er et menubaseret værktøj til subversion adgang.

- **JUnit** – Et framework og et plug-in til Eclipse. Bruges i forbindelse for Unit testing af kildekode.

- **JAXB** – Et framework til processering af xml filer i java format. Anvendes i forbindelse med delprojekt 4.

- **CampusNet** – Anvendes til kommunikation mellem den studerende, DTU-vejleder og virksomhedsvejledere. Dokumenter og lignende kan ligges op i fildelingen mht. dokument styring, dokument revisionering og endelig aflevering.

22 Litteraturliste

[1] Visma Consulting A/S, Fishery Solution [besøgt den 23. december 2012].
http://www.fisherysolution.com/

[2] World Wide Web Consortium, W3C [besøgt den 10. september 2012].
http://www.w3.org

[3] xmlBeans, The Apache Software Foundation [besøgt den 24. september 2012].
http://xmlbeans.apache.org/

[4] Introduction to Spring Framework, SpringSource [besøgt den 19. januar 2013].
http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html

[5] Wikipedia contributers, Wikipedia – The Free Encyclopedia [besøgt den 19. januar 2013].
http://en.wikipedia.org/wiki/Web_application_framework

[6] Java, Oracle Corporation [besøgt den 19. januar 2013].
http://www.java.com/en/download/help/sysreq.xml

[7] Oracle 10g PC Windows and hardware requirements, Burleson Consulting [besøgt den 19. januar 2013].
http://www.dba-oracle.com/t_oracle_10g_pc_hardware_windows_requirements.htm

[8] Forum, jGuru [besøgt den 19. januar 2013].
www.jguru.com/faq/view.jsp?EID=585982

[9] Forum - Apache tomcat, The Apache Software Foundation [besøgt den 19. januar 2013].
http://tomcat.10.n6.nabble.com/System-requirements-for-running-Apache-Tomcat-on-Windows-Box-td2091878.html

[10] Studiehåndbog, DTU [besøgt den 19. januar 2013].
http://shb.dtu.dk/Default.aspx?documentid=2806&Language=da-DK&lg=&version=2011/2012

[11] Oracle Naming Conventions, SS64 [besøgt den 19. januar 2013].
http://ss64.com/ora/syntax-naming.html

[12] Wikipedia contributers, Wikipedia – The Free Encyclopedia [besøgt den 19. januar 2013].
http://en.wikipedia.org/wiki/.properties

[13] Data access with JDBC, SpringSource [besøgt den 19. januar 2013].
http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jdbc.html

[14] Javadoc for RowMapper, SpringSource [besøgt den 19. januar 2013].
http://static.springsource.org/spring/docs/2.5.0/api/org/springframework/jdbc/core/RowMapper.html

[15] Craig Walls, 2011, Building web applications with Spring MVC: *Spring in Action – 3rd edition*, Manning Shelter Island, 164-197.

[16] Apache Tiles, Apache[besøgt den 24. januar 2013]
http://tiles.apache.org

[17] Web MVC framework, SpringSource [besøgt den 24. januar 2013]
http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html

[18] Annotation Type Autowired, SpringSource [besøgt den 24. januar 2013]
http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/beans/factory/annotation/Autowired.html

[19] Interface BindingResult, SpringSource [besøgt den 24. januar 2013]
http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/validation/BindingResult.html

[20] Socket communication (load param), vTrack Wiki [besøgt den 24. januar 2013]
http://wiki.siriusit.com/confluence/display/vTrack/Socket+communication+%28load+param%29

[21] dom4j, sourceforge.net [besøgt den 19. januar 2013]
http://dom4j.sourceforge.net/

[22] JDOM, JDOM [besøgt den 19. januar 2013]
http://www.jdom.org/

[23] Project JAXB, java.net projects[besøgt den 19. januar 2013]
http://jaxb.java.net/

[24] Java Architecture for XML Binding (JAXB), Oracle [besøgt den 20. januar 2013]
http://www.oracle.com/technetwork/articles/javase/index-140168.html

[25] JAXB tutorial part 1: XML Binding explained, YouTube [besøgt den 20. januar 2013]
https://www.youtube.com/watch?v=4J_ytgQ96Kg

[26] Interface ErrorHandler, SpringSource [besøgt den 20. januar 2013]
http://docs.oracle.com/javase/1.5.0/docs/api/org/xml/sax/ErrorHandler.html

23 Appendix oversigt

Appendix A – Modellering af designs

A.1 Introduktion

Dette appendix viser det iterative arbejde der har fundet sted, mens jeg modellerede de forskellige diagrammer som jeg har brugt i projektet.
Her forklarer jeg hvilke ændringer og optimeringer der har været undervejs mellem hvert iteration.
Jeg synes ikke at det har været relevant at have dette med i selve rapporten, da det viser en proces.
Hvorimod jeg i rapporten fokuser på de afsluttende resultater som jeg kommer frem til.
Dette er også dokumentation på at der er brugt iterativ udvikling i projektet.

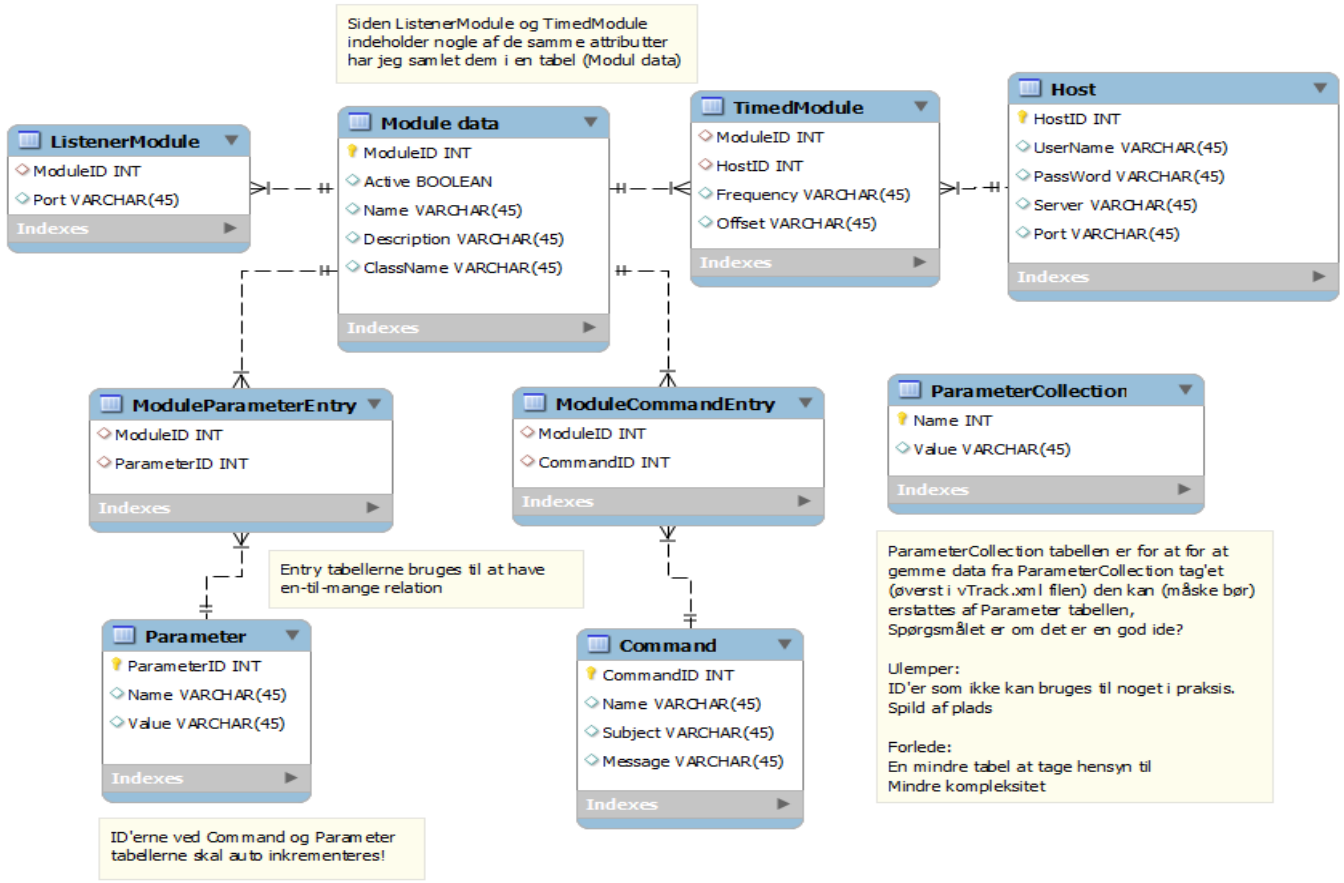
I dette appendix findes der også diagrammer som ikke er taget med i rapporten.

A.2 Databasemodel udvikling

Dette afsnit omhandler den iterative udvikling der har fundet sted i forbindelse med udviklingen af den endelige databasearkitektur. Den endelige database model er udviklet efter 4 iterationer.

A.2.1 1. udkast

På baggrund af de data der gemmes i xml filen har jeg udviklet en model som ses på Figur 1. Jeg har i denne omgang ikke taget hensyn til den nuværende databas arkitektur eller navnekonventioner der anvendes i vTrack. Derfor har jeg lavet en "komplet" løsning, som kan anvendes i forbindelse med enhver database. Ved udvikling af denne model havde jeg ikke taget hensyn til xsd filen der bruges til at validere vtrack.xml filen.
Resultat af dette ses på Figur 1.



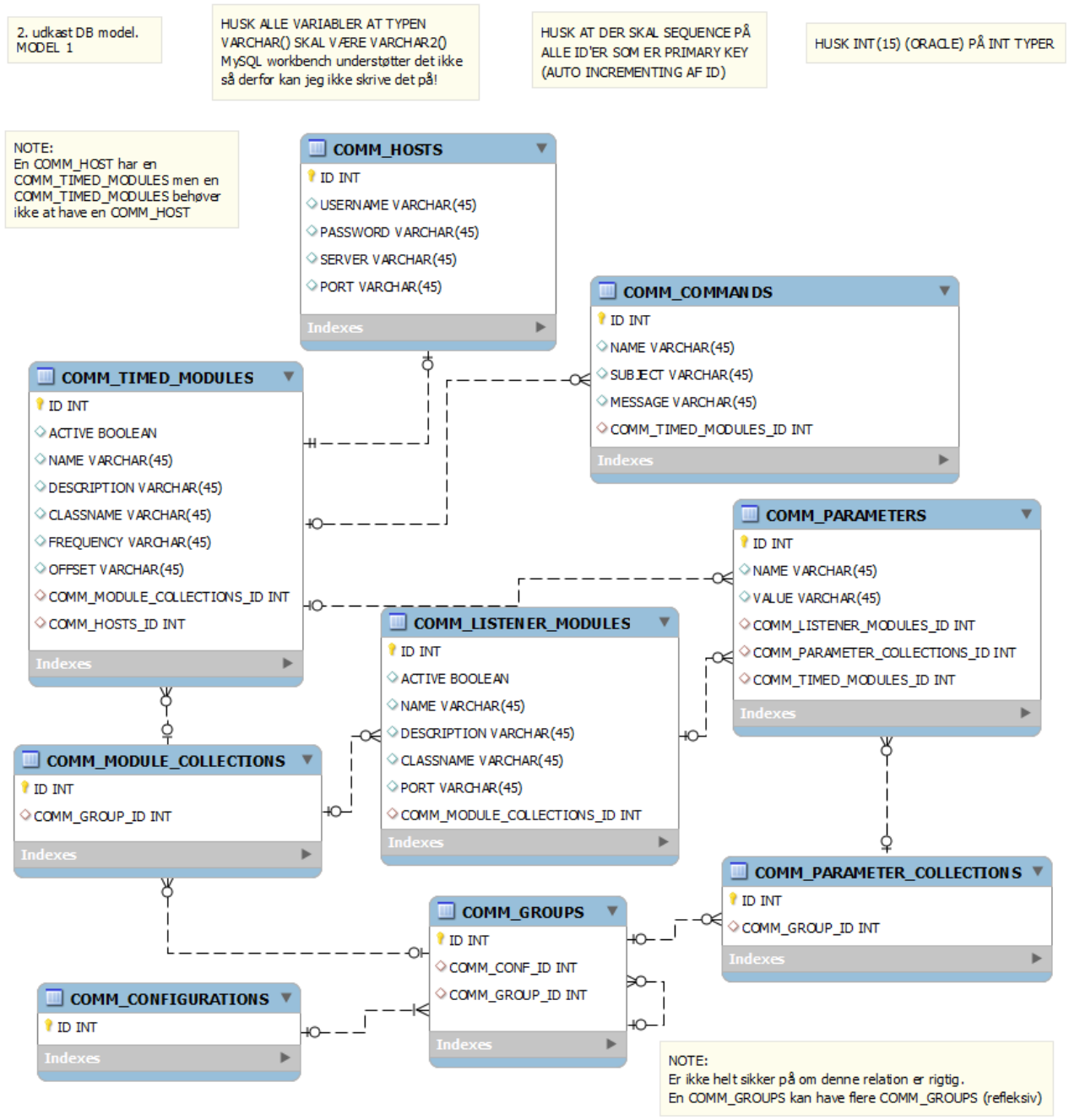
Figur 1 - 1. udkast database model

Jeg har prøvet at samle attributter der er ens for de 2 moduler (listener module og timed module) i en tabel (`Module data`) og efterfølgende har jeg lavet nogle entry tabeller som bruges til at have en til mange relationer.
Der er udtænkt en løsning hvor jeg har tænkt på at persistere tags fra `ParameterCollection` tagget i en separat tabel (`ParamterCollection`).

A.2.2 2. udkast

Efter jeg havde analyseret xsd filen for vtrack.xml, kom jeg frem til en løsning på Figur 2. Her har jeg taget hensyn til alle de xml tags kan fremkomme i konfigurationen. Denne arkitektur minder meget om figur 16 - Diagram over configuration.xsd i rapporten.

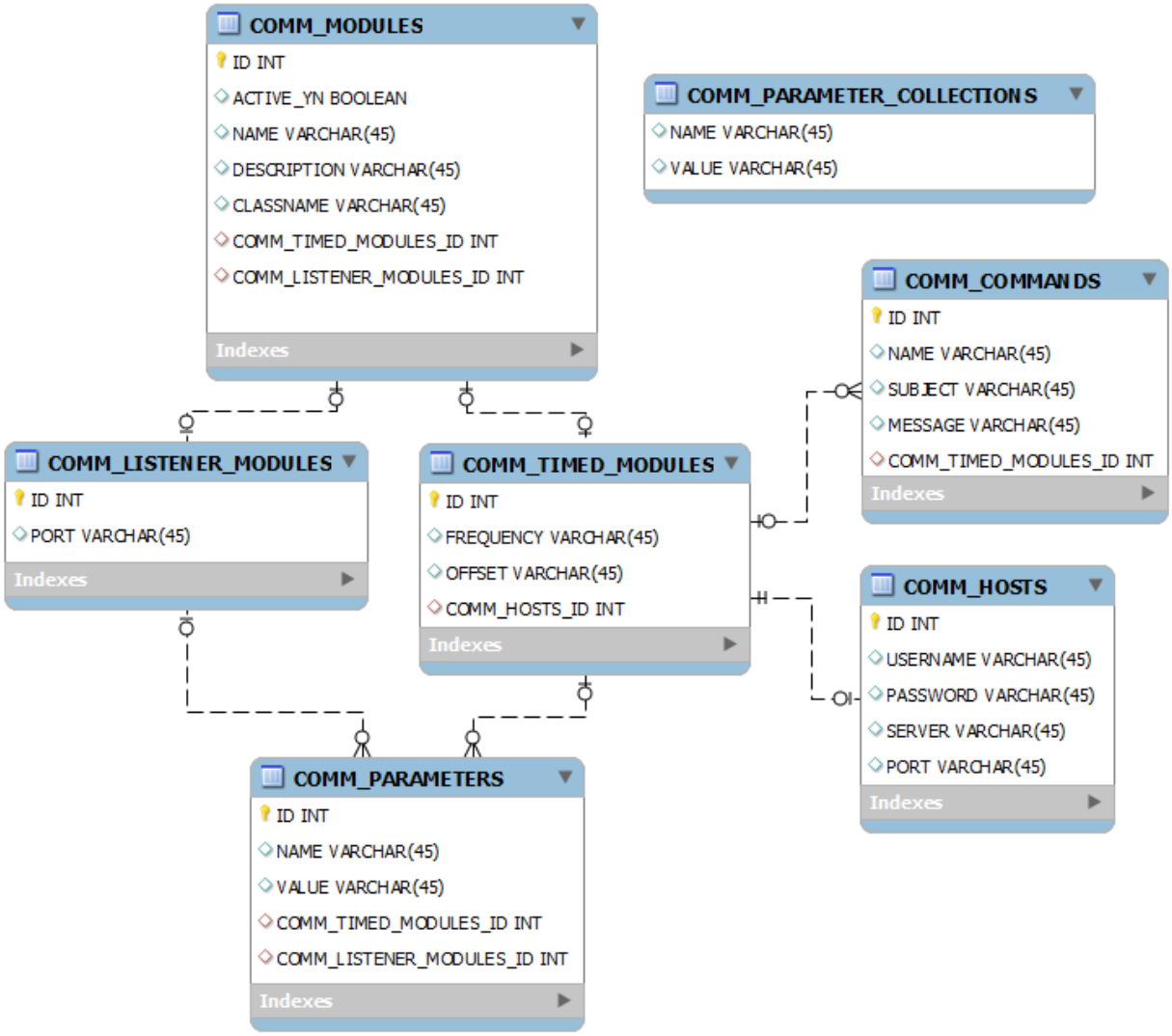
Jeg har også taget hensyn til de forskellige navne konventioner der anvendes i vTrack databasen.



Figur 2 - 2.udkast database model

A.2.3 3. udkast

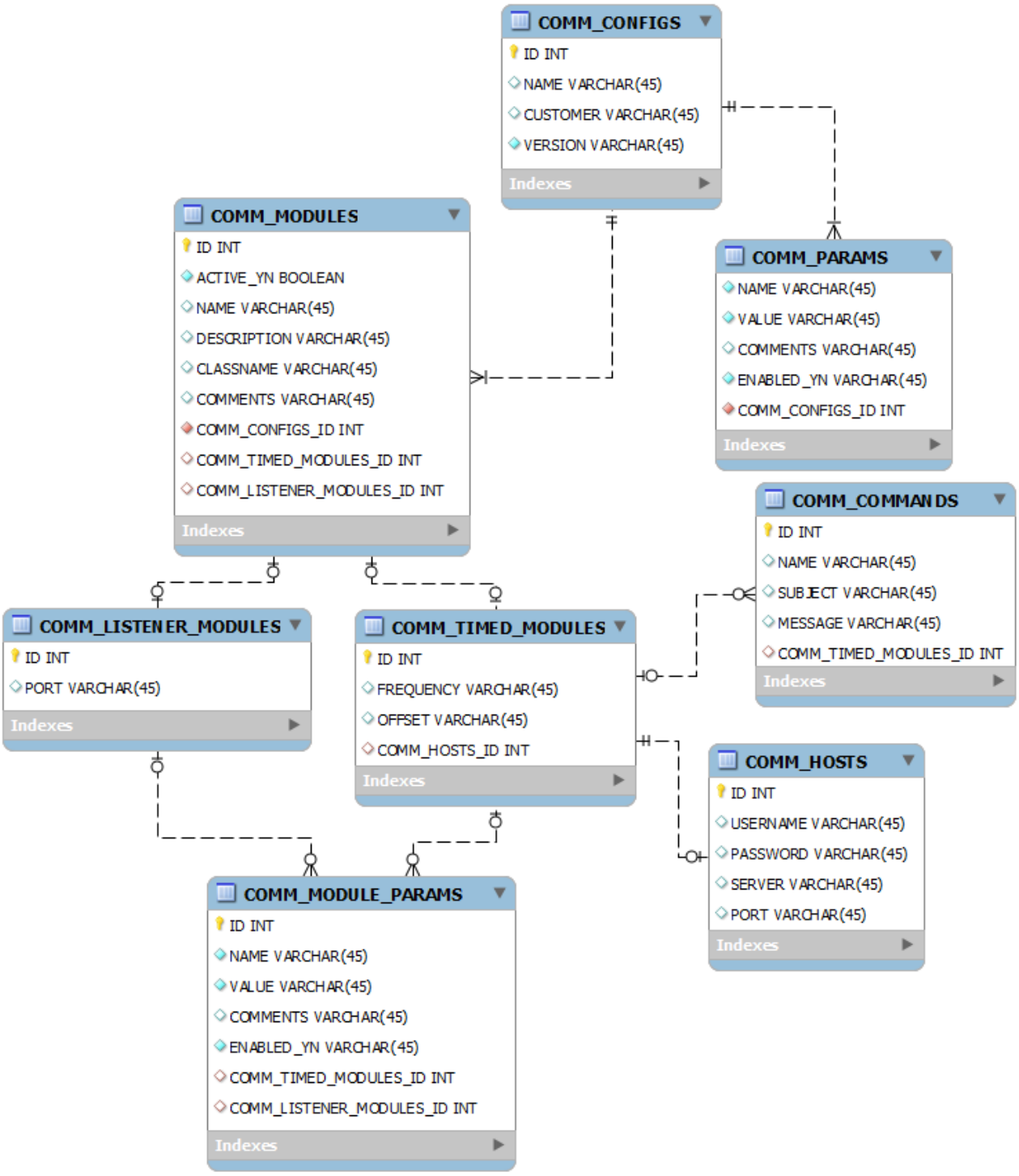
Siden der er nogle tabeller på Figur 2 som ikke bidrager med noget brugbart. Fx id'er som peger på id'er har jeg i denne omgang valgt at fjerne disse fra modellen. Dette resulterede i at jeg kom frem til en mere enkel arkitektur som er afbilledet på Figur 3.



Figur 3 - 3. udkast til database model

A.2.4 4. udkast

Ved denne iteration har jeg tilføjet en ny tabel `COMM_CONFIGS` som forbinder `COMM_MODULES` og `COMM_PARAMETER_COLLECTIONS`, fra Figur 3 således `COMM_PARAMETER_COLLECTIONS` ikke var separeret af de øvrige tabeller. Jeg har også ændret nogle af tabelnavnene ved brug af forkortelser (fx parameter er forkortet til params). Resultatet af disse ændringer ses på Figur 4.

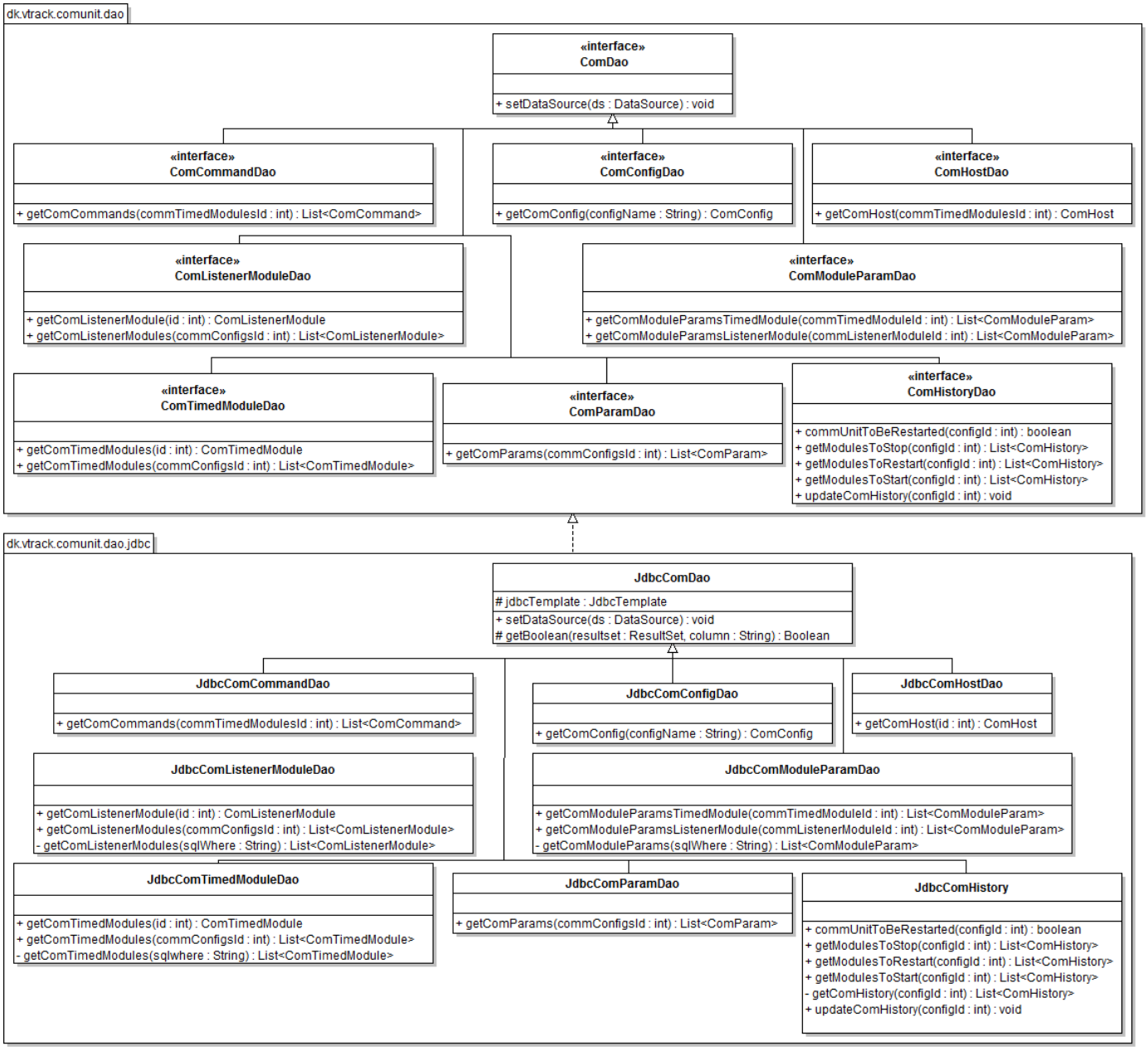


Figur 4 - 4.udkast til database model

Ændringer fra 4. udkast til den endelige model diskuteres i rapporten.

A.3 JDBC og DAO implementering – Kommunikationsenhed

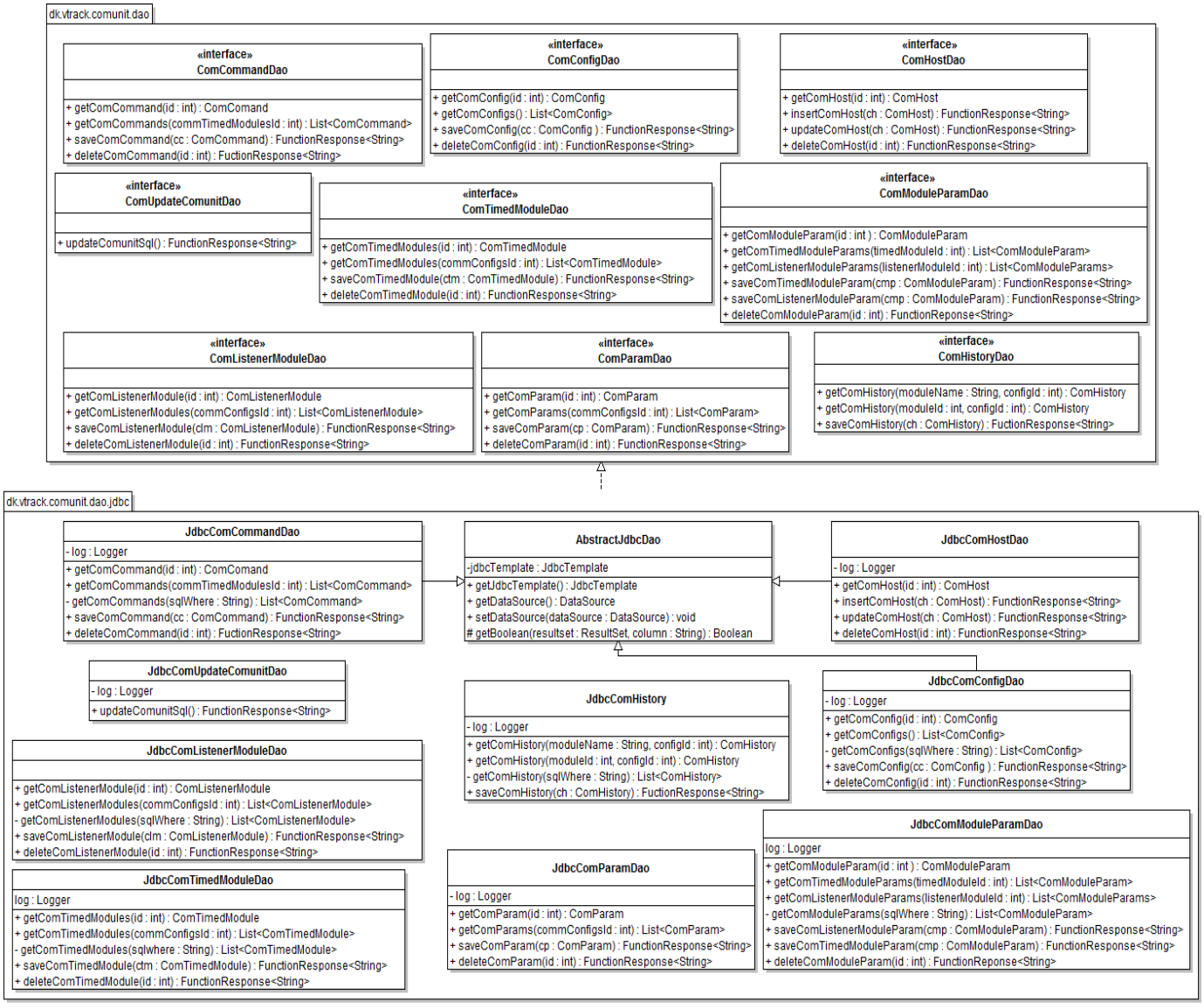
Figuren nedenfor viser den fulde arkitektur over JDBC og DAO implementeringerne i kommunikationsenheden for indlæsning af data fra databasen.
Hver del ligger i sin egen package.
Klasserne vist i den nederste del på figuren implementerer de korresponderende interfaces fra øverste del af figuren.



Figur 5 - JDBC og Dao fuld implementation kommunikationsenhed

A.4 JDBC og DAO implementering – Miljø

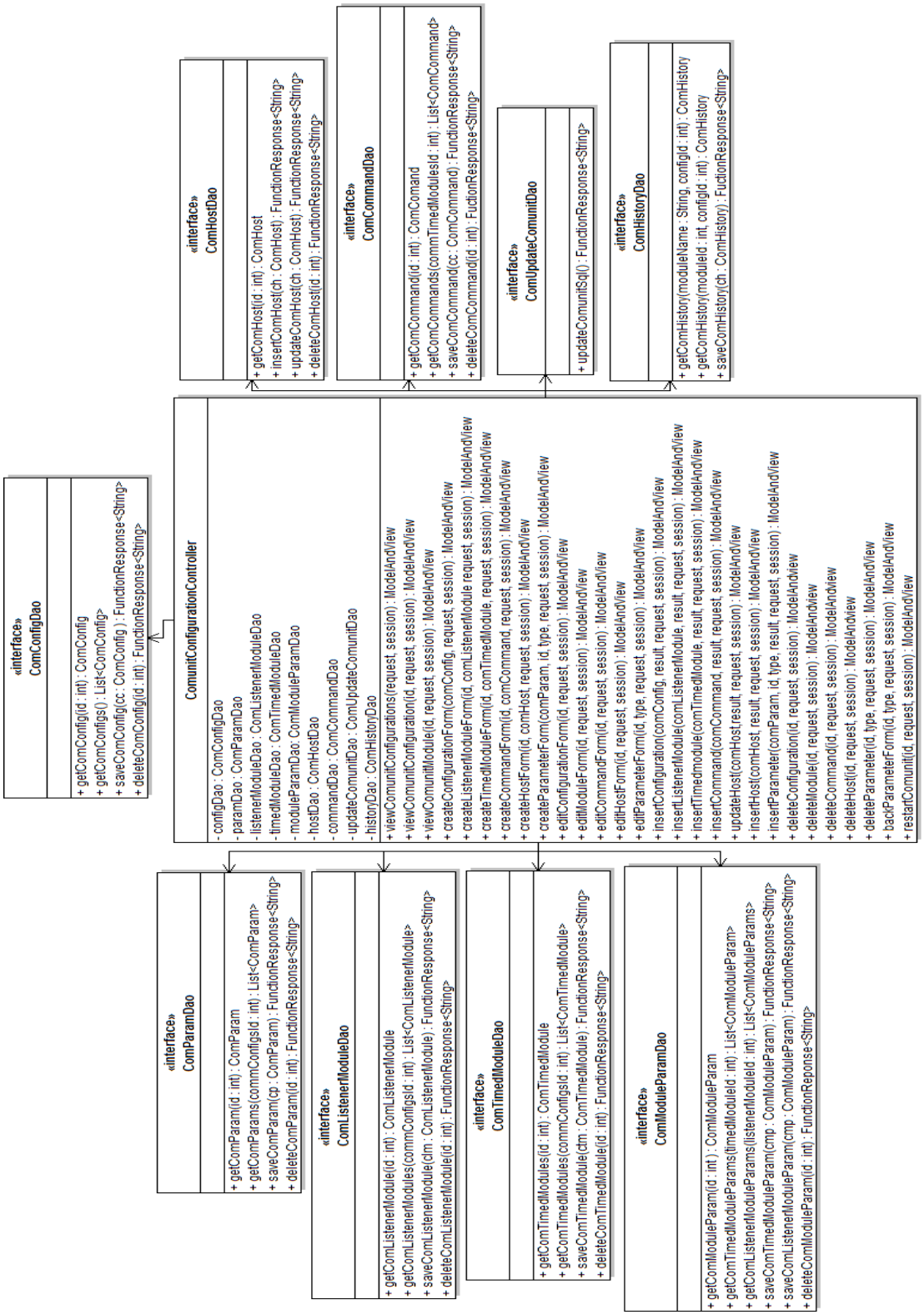
Figuren nedenfor viser den fulde arkitektur over JDBC og AO implementationen som jeg har oprettet for Miljøet (præsentationsenheden) for indlæsning af konfigurationsdata fra database.
Hver del ligger i sin egen package.
Klasserne vist i den nederste del på figuren implementerer de korresponderende interfaces fra øverste del af figuren.



Figur 6 - JDBC og Dao fuld implementation miljø

A.5 Controller – Miljø

Figuren for neden viser klassen for `ComunitConfigurationController`. Dette er MVC controlleren i webapplikationen (Miljøet). Yderligere kan man se hvilke klasse den har associationer til.



Figur 7 - ComunitConfigurationController

Appendix B - Test

B.1 Introduktion

Dette appendix indeholder test resultaterne fra de forskellige tests der er lavet løbet af projektet.

B.2 Database test

Her er test resultaterne for test 1:

Test nummer	Faktisk output
1.1	CREATE TABLE succeeded. ALTER TABLE COMM_CONFIGS succeeded. ALTER TABLE COMM_CONFIGS succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_PARAMS succeeded. ALTER TABLE COMM_PARAMS succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_MODULES succeeded. ALTER TABLE COMM_MODULES succeeded. ALTER TABLE COMM_MODULES succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_LISTENER_MODULES succeeded. ALTER TABLE COMM_LISTENER_MODULES succeeded. COMMENT ON succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_TIMED_MODULES succeeded. ALTER TABLE COMM_TIMED_MODULES succeeded.

	COMMENT ON succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. COMMENT ON succeeded. CREATE INDEX succeeded. CREATE INDEX succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_HOSTS succeeded. ALTER TABLE COMM_HOSTS succeeded. COMMENT ON succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_COMMANDS succeeded. ALTER TABLE COMM_COMMANDS succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_HISTORY succeeded. ALTER TABLE COMM_HISTORY succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded.
1.2	CREATE TABLE succeeded. ALTER TABLE COMM_CONFIGS succeeded. ALTER TABLE COMM_CONFIGS succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_PARAMS succeeded. ALTER TABLE COMM_PARAMS succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_MODULES succeeded. ALTER TABLE COMM_MODULES succeeded. ALTER TABLE COMM_MODULES succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_LISTENER_MODULES succeeded. ALTER TABLE COMM_LISTENER_MODULES succeeded. COMMENT ON succeeded. CREATE TABLE succeeded.

	ALTER TABLE COMM_TIMED_MODULES succeeded. ALTER TABLE COMM_TIMED_MODULES succeeded. COMMENT ON succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. ALTER TABLE COMM_MODULE_PARAMS succeeded. COMMENT ON succeeded. CREATE INDEX succeeded. CREATE INDEX succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_HOSTS succeeded. ALTER TABLE COMM_HOSTS succeeded. COMMENT ON succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_COMMANDS succeeded. ALTER TABLE COMM_COMMANDS succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded. CREATE TABLE succeeded. ALTER TABLE COMM_HISTORY succeeded. ALTER TABLE COMM_HISTORY succeeded. CREATE INDEX succeeded. COMMENT ON succeeded. CREATE SEQUENCE succeeded.

B.3 Database integrationstest

Her er test resultaterne for test 2:

Test nummer	Faktisk output
2.1	1 rows inserted
2.2	1 rows inserted 1 rows inserted
2.3	1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted

	1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted 1 rows inserted
2.4	1 rows inserted
2.5	1 rows inserted 1 rows inserted
2.6	1 rows inserted 1 rows inserted

B.4 Constraint test af database

Her er test resultaterne for test 3:

Test nummer	Faktisk output
3.1	Error starting at line 1 in command: INSERT INTO COMM_CONFIGS(ID, NAME, CUSTOMER, VERSION) VALUES(COMM_CONFIGS_SEQ.NEXTVAL, 'local vTrack.xml', 'Ibrahim2', '2.0.0') Error report: SQL Error: ORA-00001: unique constraint (SYSTEM.UNIQUE_CONFIG_NAMES) violated 00001. 00000 - "unique constraint (%s.%s) violated" *Cause: An UPDATE or INSERT statement attempted to insert a duplicate key. For Trusted Oracle configured in DBMS MAC mode, you may see this message if a duplicate entry exists at a different level. *Action: Either remove the unique restriction or do not insert the key.

3.2	Error starting at line 1 in command: INSERT INTO COMM_MODULES(ID, ACTIVE_YN, NAME, DESCRIPTION, CLASSNAME, COMMENTS, COMM_CONFIGS_ID) VALUES(COMM_MODULES_SEQ.NEXTVAL, 'Y', 'LOGGER', 'LOGGER2', 'dk.vtrack.comunit.database.LogMessageWriter2', NULL, (select id from comm_configs where name = 'local vTrack.xml')) Error report: SQL Error: ORA-00001: unique constraint (SYSTEM.UNIQUE_MODULE_NAMES) violated 00001. 00000 - "unique constraint (%s.%s) violated" *Cause: An UPDATE or INSERT statement attempted to insert a duplicate key. For Trusted Oracle configured in DBMS MAC mode, you may see this message if a duplicate entry exists at a different level. *Action: Either remove the unique restriction or do not insert the key.
3.3	Error starting at line 1 in command: INSERT INTO COMM_MODULE_PARAMS(ID, NAME, VALUE, COMMENTS, ENABLED_YN, COMM_TIMED_MODULES_ID, COMM_LISTENER_MODULES_ID) VALUES(COMM_MODULE_PARAMS_SEQ.NEXTVAL, 'Name', 'Value', 'Test', 'Y', (select id from comm_timed_modules natural join comm_modules where name = 'LOGGER' and comm_configs_id = (select id from comm_configs where name = 'local vTrack.xml')), (select id from comm_listener_modules natural join comm_modules where name = 'COMMAND_SERVER' and comm_configs_id = (select id from comm_configs where name = 'local vTrack.xml'))) Error report: SQL Error: ORA-02290: check constraint (SYSTEM.CHK_MODULE_ID) violated 02290. 00000 - "check constraint (%s.%s) violated" *Cause: The values being inserted do not satisfy the named check *Action: do not insert values that violate the constraint.

B.5 Input test i Miljø

Test af felterne ved oprettelse og opdatering af Parameter

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
9.1	Input validering ved oprettelse/opdatering af parameter	Tomme værdier i alle input felter	Fejl om at navn og value skal angives	Name cannot be empty	Ok
9.2	Input validering ved oprettelse/opdatering af parameter	Name: Value: a	Fejl om at navn skal angives	Name cannot be empty	Ok
9.3	Input validering ved oprettelse/opdatering af parameter	Name: a Value:	Fejl om at value skal angives	Value cannot be empty	Ok
9.4	Limit test ved oprettelse/opdatering af parameter	Name og value på 25 random tegn	Værdierne indsættes i database og der vises en ny række	Data er indsat og en ny række vises	Ok

Test af felterne ved oprettelse og opdatering af Timed module

Denne test tager ikke hensyn til de felter som er ikke obligatoriske (comments), active har gennem denne test altid haft værdien yes.
Ved test på felter der forventer et tal har jeg testet med følgende tal værdier: -500, -500, 1, -500.1, 500,1 og -500,1 (bemærk engelsk og dansk komma).
Frequence og offset skal være et mellem 0 og 999999999 derfor testes der med værdierne 1000000000, 999999999, 999999998 og -1, 0, 1 for dette felt.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
10.1	Input validering ved oprettelse/opdatering af timed module	Tomme værdier i alle input felter	Fejl om at navn skal angives	Name cannot be empty	Ok
10.2	Input validering ved oprettelse/opdatering af timed module	Name: Description: a Classname: a Frequency: 0 Offset: 0	Fejl om at navn skal angives	Name cannot be empty	Ok
10.3	Input validering ved oprettelse/opdatering af timed module	Name: a Description: Classname: a Frequency: 0 Offset: 0	Fejl om at description skal angives	Description cannot be empty	Ok
10.4	Input validering ved oprettelse/opdatering af timed module	Name: a Description: a Classname: Frequency: 0 Offset: 0	Fejl om at classname skal angives	Classname cannot be empty	Ok
10.5	Input validering ved oprettelse/opdatering af timed module	Name: a Description: a Classname: a Frequency: Offset: 0	Fejl om at frequency skal angives	Frequency cannot be empty	Ok
10.6	Input validering ved oprettelse/opdatering af timed module	Name: a Description: a Classname: a Frequency: 0 Offset:	Fejl om at offset skal angives	Offset cannot be empty	Ok
10.7	Test på frequency	Negativt heltal Negativt komma tal Positivt komma tal	Fejl om at frequency ikke kan indsættes	Frequency must be an integer	Ok
10.8	Test på offset	Negativt heltal Negativt komma tal Positivt komma tal	Fejl om at offset ikke kan indsættes	Offset must be an integer	OK
10.9	Limit test på frequence	1000000000 999999999 999999998	Afvis Godkend Godkend	Ved fejl vises: Frequency must between	Ok

		-1 0 1	Afvis godkend godkend	0 and 999999999	
10.10	Limit test på offset	1000000000 999999999 999999998 -1 0 1	Afvis Godkend Godkend Afvis godkend godkend	Ved fejl vises: Offset must between 0 and 999999999	Ok
10.11	Limit test ved oprettelse/opdatering af timed module	Name, description, classname på 25 random tegn	Ingen fejl (bør kunne indsættes i DB)	Data er indsat og en ny række vises	Ok

Test af felterne ved oprettelse og opdatering af Host

Denne test tager ikke hensyn til felter som er ikke obligatoriske (fx server).
Ved test på felter der forventer et tal har jeg testet med følgende tal værdier: -500, -500,1, -500.1, 500,1 og -500,1 (bemærk engelsk og dansk komma).
Port er ikke et obligatorisk felt, men hvis det angivet skal det været et heltal mellem 0 og 65535, derfor testes værdierne: 65536, 65535, 65534 og -1, 0, 1 for dette felt.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
11.1	Input validering ved oprettelse/opdatering af host	Tomme værdier i alle input felter	Fejl om at username skal angives	Username cannot be empty	Ok
11.2	Input validering ved oprettelse/opdatering af host	username: password: a	Fejl om at username skal angives	Username cannot be empty	Ok
11.3	Input validering ved oprettelse/opdatering af host	username: a password:	Fejl om at password skal angives	Password cannot be empty	Ok
11.4	Test på port	Negativt heltal Negativt komma tal Positivt komma tal	Fejl om at port ikke kan indsættes	Port must be an integer	Ok
11.5	Limit test på port	65536 65535 65534 -1 0 1	Afvis Godkend Godkend Afvis godkend godkend	Ved fejl vises: Port must between 0 and 65535	Ok
11.6	Limit test ved oprettelse/opdatering af host	username og password på 25 random tegn	Ingen fejl (bør kunne indsættes i DB)	Data er indsat og en ny række vises	Ok

Test af felterne ved oprettelse og opdatering af Command

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
12.1	Input validering ved oprettelse/opdatering af command	Tomme værdier i alle input felter	Fejl om at navn skal angives	Name cannot be empty	Ok
12.2	Input validering ved oprettelse/opdatering af command	Name: Value: a Subject: a	Fejl om at value skal angives	Name cannot be empty	Ok
12.3	Input validering ved oprettelse/opdatering af command	Name: a Value: Subject: a	Fejl om at subject skal angives	Value cannot be empty	Ok
12.4	Input validering ved oprettelse/opdatering af command	Name: a Value: a Subject:	Fejl om at subject skal angives	Subject cannot be empty	Ok
12.5	Limit test ved oprettelse/opdatering af command	Name, value og subject på 25 random tegn	Ingen fejl (bør kunne indsættes i DB)	Data er indsat og en ny række vises	Ok

Test af felterne ved oprettelse og opdatering af Listener module

Denne test tager ikke hensyn til felter som er ikke obligatoriske (comments), active har gennem denne test altid haft værdien yes.
Ved test på felter der forventer et tal har jeg testet med følgende tal værdier: -500, -500,1, -500.1, 500,1 og -500,1 (bemærk engelsk og dansk komma).
Port skal være et heltal mellem 0 og 65535, derfor testes værdierne 65536, 65535, 65534 og -1, 0, 1 for dette felt.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
13.1	Input validering ved oprettelse/opdatering af listener module	Tomme værdier i alle input felter	Fejl om at navn skal angives	Name cannot be empty	Ok
13.2	Input validering ved oprettelse/opdatering af listener module	Name: Description: a Classname: a Port: 0	Fejl om at navn skal angives	Name cannot be empty	Ok
13.3	Input validering ved oprettelse/opdatering af listener module	Name: a Description: Classname: a Port: 0	Fejl om at description skal angives	Description cannot be empty	Ok
13.4	Input validering ved oprettelse/opdatering af listener module	Name: a Description: a Classname: Port: 0	Fejl om at classname skal angives	Classname cannot be empty	Ok
13.5	Input validering ved oprettelse/opdatering af listener module	Name: a Description: a Classname: a	Fejl om at port skal angives	Port cannot be empty	Ok

		Port:			
13.6	Test på Port	Negativt heltal Negativt komma tal Positivt komma tal	Fejl om at port ikke kan indsættes	Port must be an integer	Ok
13.7	Limit test på port	65536 65535 65534 -1 0 1	Afvis Godkend Godkend Afvis godkend godkend	Ved fejl vises: Port must between 0 and 65535	Ok
13.8	Limit test ved oprettelse/opdatering af listener module	Name, description, classname på 25 random tegn	Ingen fejl (bør kunne indsættes i DB)	Data er indsat og en ny række vises	Ok

B.6 CRUD test i Miljø

I denne test testes der for om de forskellige CRUD funktioner virker i miljøet.
Her er et kortfattet redegørelse over testens forløb:

Der oprettes en konfiguration med to parametre (P1 og P2), to timed modules (TM1 og TM2) og to listener modules (LM1 og LM2).
Til TM1 oprettes en host (TM1H), 2 parametre (TM1P1 og TM1P2) og 2 commands (TM1C1 og TM1C2).
For TM2 oprettes en host (TM2H), 1 parameter (TM2P1) og 1 command (TM2C1).
Til LM1 modulet oprettes 2 parametre (LM1P1 og LM1P2) og for LM2 oprettes en enkel parameter (LM2P1).

Derefter testes der om update og delete funktionerne virker som ønsket på de oprettede elementer.
Til P1 testes update og delete funktionen, det samme gøres for TM1H, TM1P1 og TM1C1.
Dernæst testes update og delete funktionen på selve modulet (TM1), for at se om de resterende værdier er slettet dvs. den tilhørende parameter TM1P2 og command TM1C2.

Til LM1 testes update og delete på LM1P1 og dernæst på selve modulet, for at se om de resterende værdier er slettet dvs. den tilhørende parameter LM1P2.

Update og delete funktionerne testes også på selve konfigurationen, for at se om de resterende værdier er slettet (P1, TM2, TM2H, TM2P1, TM2C1, LM2 og LM2P1).
Alle værdier der anvendes er fiktive og har ingen betydning for testen!

På denne måde testes der for alle CRUD funktioner og man kan bl.a. tjekke for om de tilhørende konfigurationsparametre til en konfiguration slettes, når brugeren ønsker at slette en hel konfiguration.

Test nummer	Test beskrivelse	Input	Forventet output	Faktisk output	Resultat
14.1	Oprettelse af konfiguration	Name: Config Customer: Test Version: 1.0.0	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.2	Oprettelse af P1	Name: P1 Value: P1 Comments: Test Enabled: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok

14.3	Oprettelse af P2	Name: P2 Value: P2 Comments: Test Enabled: No	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.4	Oprettelse af TM1	Name: TM1 Description: TM1 Classname: Dk.vtrack.TM1 Comments: Test TM1 Frequency: 30 Offset: 30 Active: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.5	Oprettelse af TM1H	Username: user1 Password: pass1 Server: s1 Port: 12345	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.6	Oprettelse af TM1P1	Name: Tm1p1 Value: Tm1p1 Comments: Test Enabled: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.7	Oprettelse af TM1P2	Name: Tm1p2 Value: Tm1p2 Comments: Test Enabled: no	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.8	Oprettelse af TM1C1	Name: Tm1c1 Subject: test Message: test	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.9	Oprettelse af TM1C2	Name: Tm1C2 Subject: test Message: test	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.10	Oprettelse af TM2	Name: TM2 Description: TM2 Classname: Dk.vtrack.TM2 Comments: Test TM2 Frequency: 300 Offset: 50 Active: no	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.11	Oprettelse af TM2H	Username: user2 password: pass2 server: s2 port: 54321	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.12	Oprettelse af TM2P1	Name: Tm2p1 Value: Tm2p1 Comments: Test Enabled: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.13	Oprettelse af TM2C1	Name: Tm2c1 Subject: test Message: test	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok

14.14	Oprettelse af LM1	Name: LM1 Description: LM1 Classname: dk.vtrack.LM1 Comments: Test LM1 Port: 4321 Active: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.15	Oprettelse af LM1P1	Name: Lm1P1 Value: Lm1P1 Comments: Test Enabled: Yes	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.16	Oprettelse af LM1P2	Name: Lm1p2 Value Lm1p2 Comments: Test Enabled: No	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.17	Oprettelse af LM2	Name: LM2 Description: LM2 Classname: dk.vtrack.LM2 Comments: Test LM2 Port: 8080 Active: No	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.18	Oprettelse af LM2P1	Name: Lm2p2 Value: Lm2p2 Comments: Test Enabled: no	Værdierne indsættes i database og en ny række vises	Data er indsat og en ny række vises	Ok
14.19	Update af P1	Name: P1U Value: P1U Comments: TestU Enabled: No	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.20	Slet af P1		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok
14.21	Update af TM1H	Username: user1U Password: pass1U Server: s1U Port: 23456	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.22	Slet af TM1H		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok
14.23	Update af TM1P1	Name: Tm1p1U Value: Tm1p1U Comments: TestU Enabled: No	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.24	Slet af TM1P1		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og	Ok

				Miljø	
14.25	Update af TM1C1	Name: Tm1c1U Subject: testU Message: testU	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.26	Slet af TM1C1			Værdien er slettet fra database og Miljø	Ok
14.27	Update af TM1	Name: TM1U Description: TM1U Classname: Dk.vtrack.TM1U Comments: Test TM1U Frequency: 35 Offset: 35 Active: No	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.28	Slet af TM1		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok
14.29	Update af LM1P1	Name: Lm1P1U Value: Lm1P1U Comments: TestU Enabled: No	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.30	Slet af LM1P1		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok
14.31	Slet af LM1		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok
14.32	Update af konfiguration	Name: Config2 Customer: Test Version: 2.0.0	Værdierne opdateres i databasen og de nye data vises i Miljøet	Data er indsat i database og værdierne er opdateret	Ok
14.33	Slet af konfiguration		Værdien er slettet fra database og Miljø	Værdien er slettet fra database og Miljø	Ok

Appendix C – CD rom

C.1 Introduktion

Dette Appendix indeholder en oversigt over de forskellige filer og dokumenter der findes i den vedlagte cd rom, og som er afleveret sammen med denne rapport.

C.2 CD rom filstruktur

I den afleverede cd rom findes følgende filer:

Readme.txt
Analyse af vTrack parametre.xlsx
Database Test
- Constraint test.sql
- Simple local vtrack test.sql
- Simple local vtrack.xml
- Test results constraint test.txt
- Test results creating tabels.txt
- Test results mapping xml to DB.txt

Dokumenter
- Projekt rapport.pdf
- Status rapport
  - Readme.txt
  - Status rapport 1.pdf
  - Status rapport 2.pdf
  - Status rapport 3.pdf
  - Status rapport 4.pdf
  - Status rapport 5.pdf
  - Status rapport 6.pdf
  - Status rapport 7.pdf
  - Status rapport 8.pdf
  - Status rapport 9.pdf

Eclipse plugins
- jaxbbuilder.2.0.1581.zip
- Readme.txt

Kildekode
- vTrack XML Parser.zip

SQL script
- Readme.txt
- Access control webclient.sql
- Access granting comm unit vtrack.sql
- Comm unit data from vtrack xml.sql
- Database create comm unit tables.sql
- Database delete comm unit tables.sql

- Database truncate comm unit tables.sql

vTrack XML Parser

- images
  - error.png
  - icon.png
  - warning.png
- lib
  - ojdbc6.jar
- schema
  - Configuration.xsd
- vTrack XML Parser.jar

vTrack XML Parser test

- Simple local vtrack test.sql
- Simple local.vtrack.xml
- test1.xml
- test2.xml
- vTrackXML.sql

vTrack XML Parser Tutorial

- vtrack.xml
- vtrack_argos.xml
- vtrack_inmarsat.xml
- vtrack_listeners.xml

Xsd schema

- configuration.xsd

Appendix D – Håndbog

D.1 Introduktion

Dette appendix indeholder 2 håndbøger, henholdsvis for Miljøet og for vTrack XML Parseren.
I hver bog findes en installationsguide, beskrivelse af brugergrænsefladen og en tutorial.
Formålet med disse bøger er at de skal give et overblik af programmet og skal sætte læseren i stand til at kunne anvende programmet uden yderligere dokumentation.

Danmarks Tekniske Universitet – Visma Consulting, Hellerup

Comm. Unit konfiguration Håndbog

Ibrahim Nemli
1/28/2013

Forord

Jeg er en IT-Diplom studerende fra DTU som i forbindelse med min afsluttende eksamen har udviklet et program (webapplikation) der er i stand til at kunne ændre i konfigurationsparametrene til vTracks kommunikationsenhed via et grafisk miljø. På denne måde bliver det nemmere for brugerne at kunne vedligeholde og ændre i data, da den nye version af kommunikationsenheden indlæser sin konfiguration fra database.

Denne håndbog er målrettet til at give brugeren anvisninger til hvordan programmet betjenes. Du vil gennem denne håndbog få en grundig gennemgang af programmet, således du vil være i stand til at kunne benytte det nye miljø med det samme.
Yderligere findes der er også en installationsguide til hvordan softwaren installeres på din computer.

Jeg håber at dette produkt vil spare jer tid for den manuelle vedligeholdelse af konfigurationen og I vil have glæde af at benytte programmet. God Fornøjelse!

Ibrahim Nemli
Januar 2013

Indholdsfortegnelse

Figur oversigt

Introduktion

Det grafiske miljø er udviklet med det formål at kunne vedligeholde og ændre konfigurationsparametrene til den nye version af kommunikationsenheden i vTrack på et højere abstraktions niveau end en xml editor, som den gamle konfiguration var baseret på, idet konfigurationen er flyttet over i en database.

Denne håndbog er skrevet som en kombination af tutorials og brugervejledning til dette Miljø. Formålet med denne håndbog er at give brugeren et indblik i hvordan programmet betjenes lige fra installeringen af programmet til at kunne vedligeholde konfigurationen som findes i databasen.
Med dette program kan vi nemmere vedligeholde konfigurationen med håb om at formindske nedetiden for den nye version af kommunikationsenheden.

Miljøet er opsat på en simpel brugervenlig måde hvor brugeren kan med få klik kan få et overblik over den fulde konfiguration og med et par klik udføre de ønskede CRUD (create, read, update, delete) funktioner.

Denne håndbog dækker kun en lille del af det færdige produkt og giver mere en tutorial lignende oplevelse for brugeren, i stedet for en fuldstændig reference manual, hvilket gør, at brugeren får en lettere og mere informationsrig indblik i hvad dette program kan gøre.

Baggrund for læseren

For at give den fulde forståelse for denne håndbog, kan det være en god ide at have en basisviden om de forskellige teknikker og teknologier der anvendes i denne håndbog. I denne håndbog vil emner som Oracle database, SQL, kommunikationsenheden, Tomcat Server og xml parser indgå. Der henvises til Gloser afsnittet i denne håndbog, som giver en kort introduktion til de forskellige begreber.

Derudover er også en god ide at have kendskab til konfigurationsfilen til vTracks kommunikationsenhed. Dog er ingen af disse et 'must' for at kunne betjene programmet.

Installations guide

Dette afsnit er en installationsvejledning, som beskriver hvordan du kører webapplikationen, således du kan anvende Miljøet med det samme.
Denne installationsguide vil starte en webapplikation op gennem Eclipse. Installationsprocessen er relativ kompliceret, så derfor anbefales der at man anvender et Eclipse setup som man kan få af Mikael Grønfelt fra Visma Consulting. Herfra kan man hurtigt bygge og installere de krævede projekter til at kunne starte Miljøet op. Dette setup fylder ca. 4 GB og derfor var det ikke muligt at brænde denne fil i den givne cd-rom.

Software krav:

Her er en liste af softwareprodukter der skal være installeret før Miljøet kan tages i brug. Sørg for at disse produkter er installeret før programmet anvendes.

Først og fremmest skal du have installeret Java Development Kit version 1.6.0 som minimums krav. Uden Java vil du ikke kunne være i stand til at kunne afvikle programmet! (Java kan hentes ned fra følgende link: http://www.oracle.com/technetwork/java/javase/downloads/index.html).

Du skal yderligere have en Oracle database installeret før webapplikationen kan køre. Her anbefales der, at der anvendes Oracle Database 10g Express Edition eller nyere (kan downloades gratis på link: http://www.oracle.com/technetwork/products/express-edition/overview/index.html).

Apache Tomcat Server. For at kunne køre webapplikationen, skal du have installeret en Apache Tomcat server (kan hentes ned fra følgende link: http://tomcat.apache.org/index.html).

Eclipse. Der anbefales at downloade og installere Eclipse 3.6.0. Her kan du bygge og installere webapplikationen på Tomcat serveren. Eclipse kan downloades fra: http://www.eclipse.org/downloads/

Dette projekt er afhængigt af Java Spring frameworket derfor skal du downloade og installere Spring til Eclipse. Se mere i følgende link: http://www.springsource.org/.

Installations guide:

Efter Java og Oracle database er installeret, skal du have oprettet tabellerne til vTrack i databasen. Sørg også for at have installeret tabellerne der er krævet for vTrack (kontakt evt. Klaus Mogensen fra Visma Consulting).

For at kunne køre programmet skal du have bygget og deployet 2 projekter til Tomcat. Brug Eclipse til dette. Du skal have checket præsentationsenheden og webclienten ud fra subversion. Når du har bygget disse 2 projekter kan du starte Tomcat serveren op.
Husk også at have installeret Spring til Eclipse.

Når det hele er kørende kan du gå ind på følgende link: http://localhost/webclient/domain/comunit-configurations for at starte med at anvende Miljøet (Husk at du først skal logge ind).

GUI Oversigt

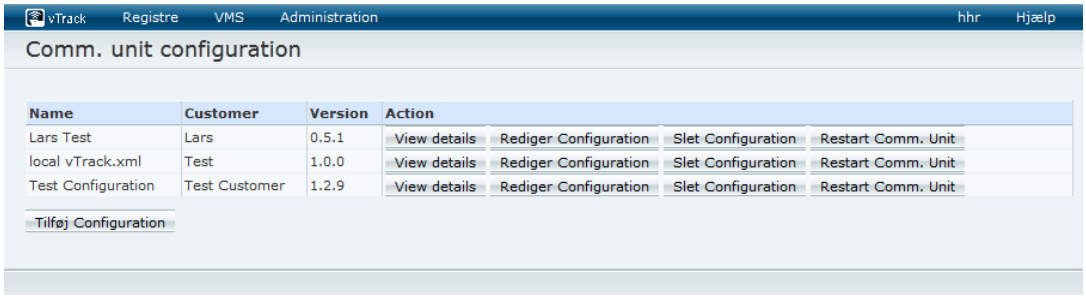
I dette afsnit gennemgås de grafiske komponenter som Miljøet består af. Det er en god ide at stifte kendskab til dette før man udfører tutorialen i næste afsnit.
Der forventes at brugeren der skal anvende programmet, har de nødvendige rettigheder der skal til, før programmet kan betjenes (man skal have rollen ADMIN_VTRACK).

Navigering

Der findes 3 primære sider som giver overblikket af konfigurationen. For hver side går man i mere dybden med konfigurationen og yderligere detaljer vises.

På Figur 1 ses hovedsiden. På denne side vises alle konfigurationer der er gemt i databasen. I dette eksempel findes der 3 konfigurationer.
Hver konfiguration svarer til at en fuld installation/konfiguration til kommunikationsenheden i vTrack.
For hver konfiguration kan man se konfigurationens navn, kunde og version. I den sidste kolonne, kan man ud fra hver konfiguration klikke på den ønskede funktion (View details, Rediger Configuration, Slet Configuration og Restart Comm. Unit).
Nederst på siden kan man tilføje en ny konfiguration, hvis man ønsker dette.

Denne side kan på nuværende tidspunkt ikke tilgås ved brug af den generelle menu i vTrack (se den blå bjælke på figuren). Derfor skal man anvende URL'en: http://localhost/webclient/domain/comunit-configurations for at se denne side.

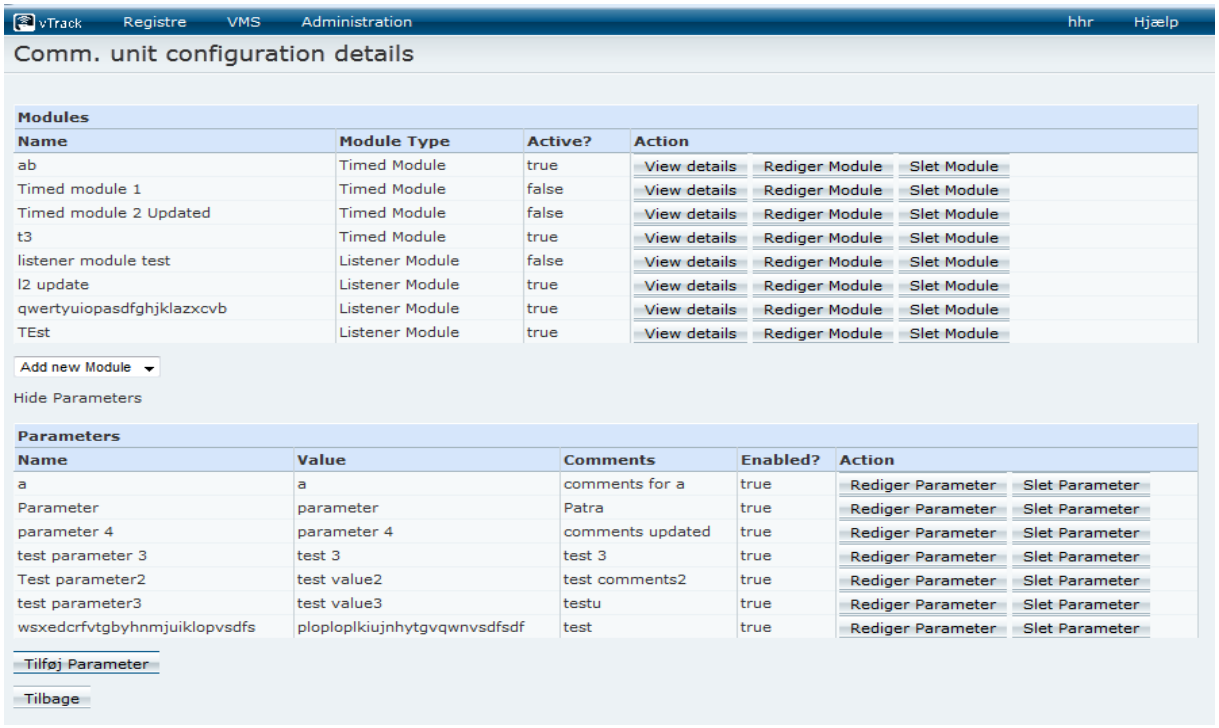


Figur 1 – Hovedside

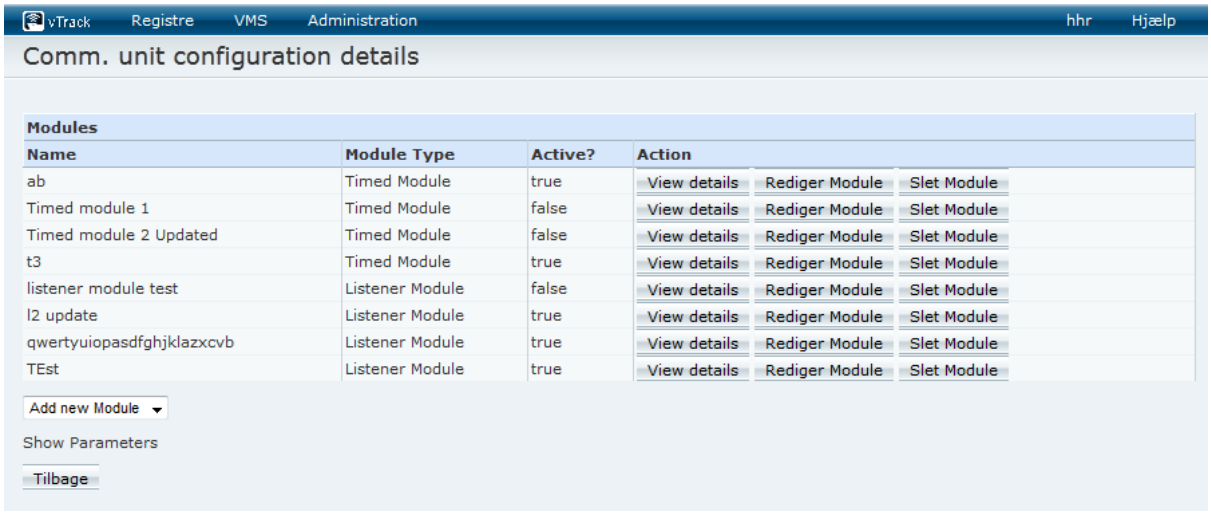
Når man ud fra en konfiguration klikker på "View details" knappen, vil man komme ind på siden vist på Figur 2. Denne side viser parametrene og modulerne for den valgte konfiguration.
For hvert modul kan man se navnet, modul typen og om modulet er aktivt eller ej. Man kan også vælge om man vil se yderligere detaljer om modulet, redigere eller slette modulet ved at klikke på knapperne ude til højre.

Når man kommer ind på denne side er parametrene som standard vist. Klikkes der på linket "Hide Parameters" vil disse skjules. Figur 3 viser et skærmbillede hvor parametrene ikke vises.

Et klik på "Tilbage" knappen nederst på siden vil fører dig tilbage til skærmbilledet på Figur 1.



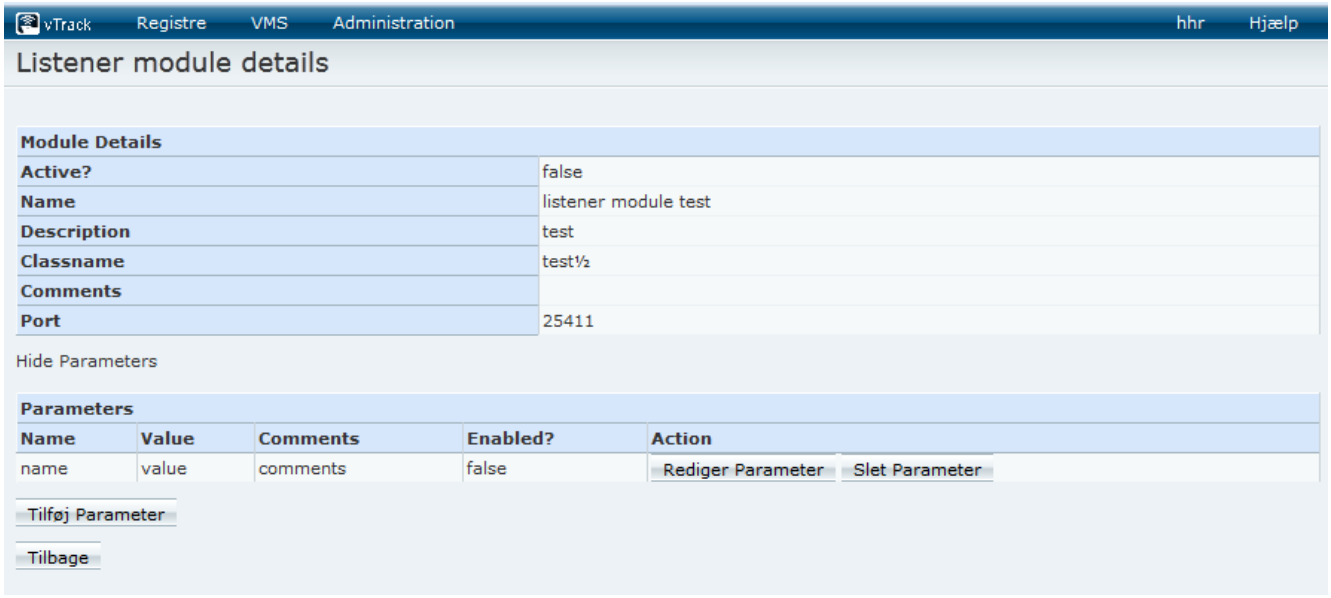
Figur 2 – Konfigurations detaljer



Figur 3 – Parameter og moduler

Man kan igen ved at klikke på linket ”Show Parameters” på Figur 3 få vist parametrene.
De viste parametre fungerer som parameter for alle modulerne (de kaldes for ”globale” parameter).
Ud fra hvert parameter kan man se dens navn, værdi, kommentar og om parameteren er aktiv. Yderligere kan man redigere og slette parameteren, hvis man ønsker dette.
Ønsker du at tilføje en ny parameter til listen, klikkes på ”Tilføj Parameter” knappen.

Som sidste trin kan man se detaljer ved modulerne. Dette gøres ved at klikke på ”View details” knappen ud fra de enkelte moduler på skærmbilledet på Figur 3.

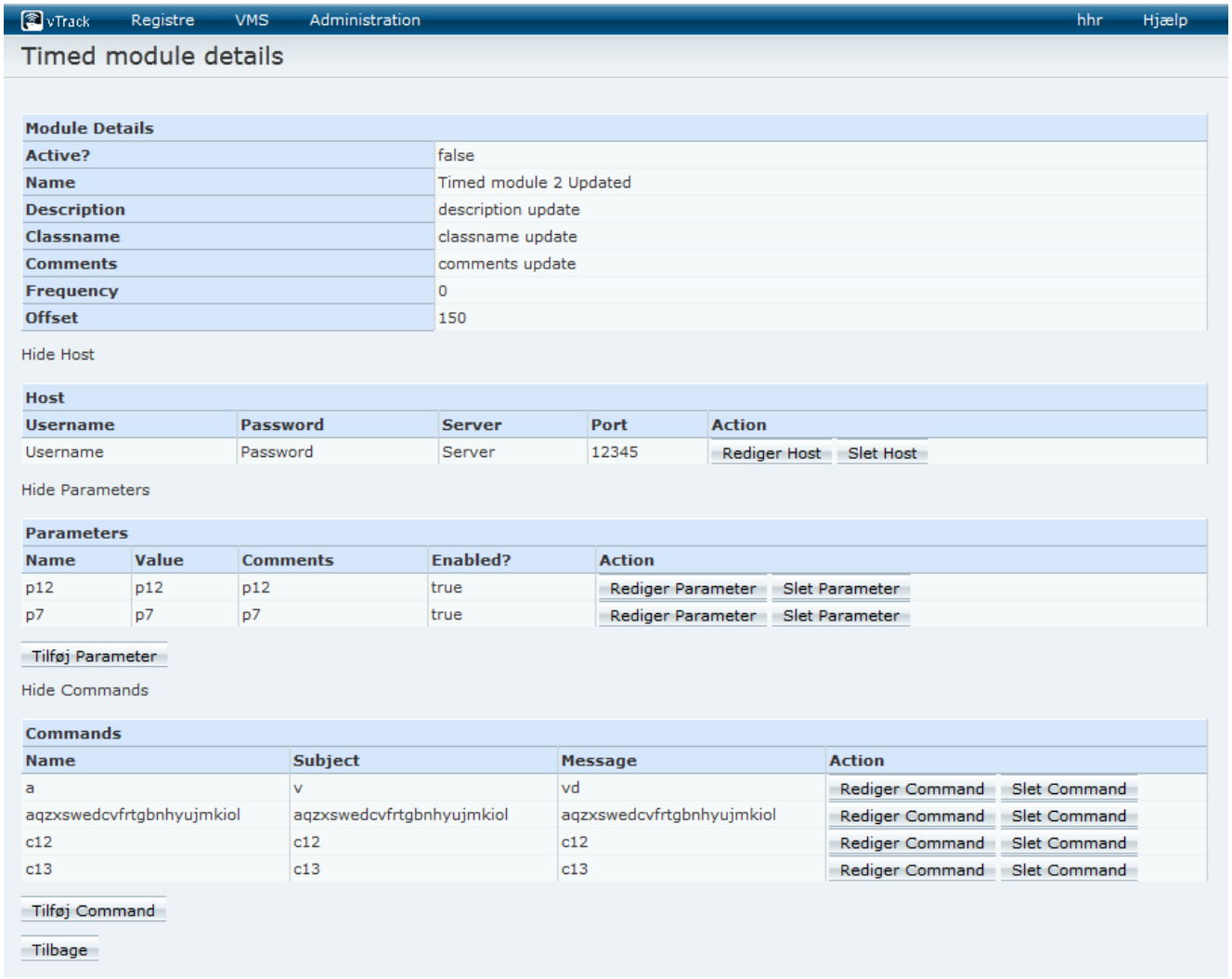


Figur 4 – Listener module

Der findes 2 typer af moduler, Listener modules og Timed modules. Figur 4 overfor viser et skærmbillede hvor detaljerne for et Listener module module vises. Her kan man se alle detaljerne om modulet og dets tilhørende parametre. Som før vil parametrene være viste når siden åbnes, men man kan altid få dem skjult ved at klikke på ”Hide Parameters”.

Den anden type af moduler der findes er Timed modules. Et eksempel på et Timed module ses på Figur 5. Her kan man se alle detaljerne for et Timed module og dets tilhørende værdier (host, parameter og commands).
Bemærk at der højst kan være en host til et Timed module og derfor vil der ikke være en tilføj knap til denne, med mindre hosten ikke findes.

Nederst på skærmbilledet på Figur 4 og Figur 5 findes der en ”Tilbage” knap. Denne knap vil føre dig tilbage til skærmbilledet på Figur 2.



Figur 5 – Timed module

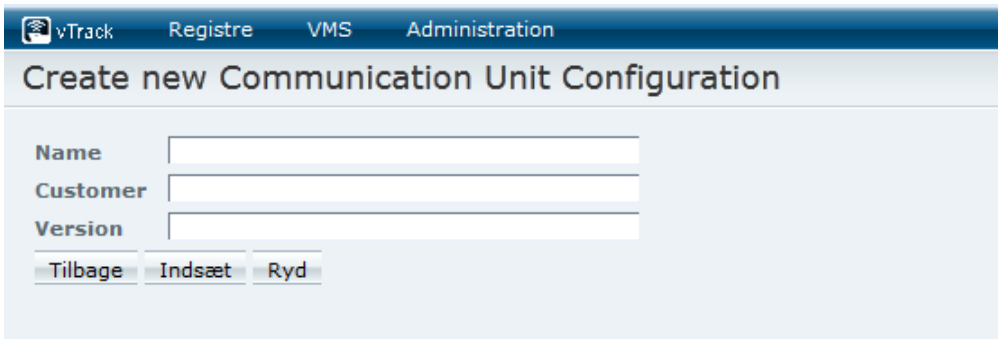
Oprettelse

I dette afsnit vil vi gennemgå hvordan de forskellige elementer til konfigurationen oprettes.

Nederst på siden i Figur 1 findes der en ”Tilføj Configuration” knap som giver mulighed for at tilføje en ny kommunikationsenhed konfiguration. Når der er klikket på denne knap vil du se et skærmbillede som på Figur 6.

Alle 3 felter er obligatoriske og skal udfyldes for at en konfiguration kan oprettes.

Name: Her angives navnet på konfiguration. Det er vigtigt at navnet er unikt og at der ikke findes andre konfigurationer med det samme navn, i dette tilfælde vil oprettelsen fejle.
Customer: Her angives hvem kunden er.
Version: Versionen af konfigurationen. Det anbefales at bruge version 1.0.0 som minimum ved en ny konfiguration.



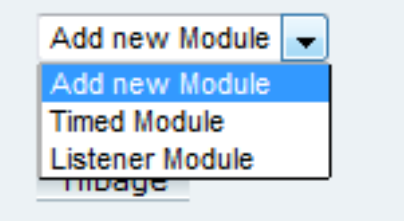
Figur 6 – Oprettelse af ny konfiguration

Når disse 3 felter er udfyldt klikkes der på ”Indsæt” knappen og der vil oprettes en ny (tom) konfiguration. Du vil samtidig blive dirigeret tilbage til skærmbilledet på Figur 1, hvor den nye konfiguration vil være på listen.
Dette vil samtidig blive valideret med, at der vises et ikon som på vist på Figur 7 nederst på siden. Dette ikon vil bliver vist hver gang du udfører en sletning, opdatering eller oprettelse af hvilket som helst element til en konfiguration.
”Ryd” knappen vil slette alt data fra inputfelterne og ”Tilbage” knappen vil stoppe oprettelsesprocessen og vende tilbage til forrige skærmbillede.

Figur 7 – Ok ikon

For at tilføje ydelige data til konfigurationen klikkes der på ”View details” knappen og du kan nu tilføje data til konfigurationen som ønsket.

Man kan oprette et nyt modul til konfigurationen som ses på Figur 2. På midten af siden findes der en dropdown menu (afbilledet på Figur 8). Her kan man vælge om man vil oprette et Timed module eller et Listener module.
Ved et klik på et af de to modul typer vil fører dig hen til siden, hvor værdierne for det nye modul skal indtastes (se Figur 9 og figur 10 på side 11).



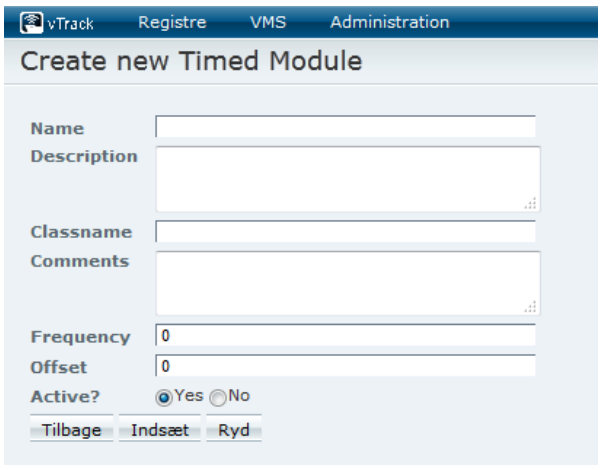
Figur 8 – Oprettelse af et modul

Figur 9 og figur 10 viser siderne for oprettelse af henholdsvis Timed module og Listener module.
For hver side skal alle felter på nær Comments udfyldes, da dette ikke er et obligatorisk felt.

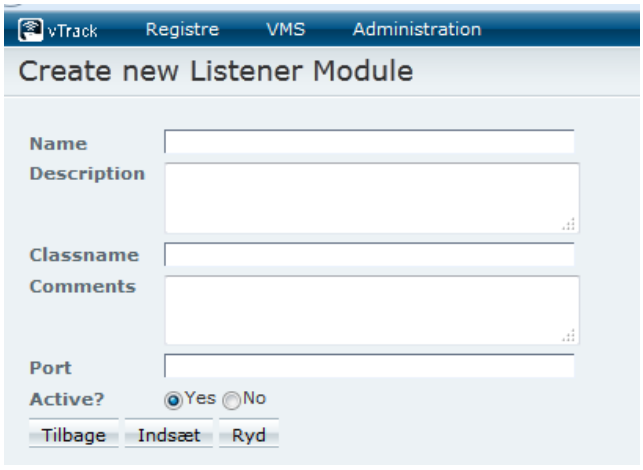
For oprettelse af et Timed module er Frequency og Offset begrænset til at være et helt positivt tal mellem 0 og 999999999. Indtastes en værdi udenfor grænserne vil der vises en fejlmeddelelse.

For Listener module er feltet Port begrænset til at være et helt positivt tal mellem 0 og 65535.

Her er en beskrivelse af de øvrige felter:
Name: Navnet på modulet, hvert modul skal have et unikt navn, da det ellers ikke vil kunne indsættes i databasen.
Description: Her angives en kort beskrivelse af modulet og dets egenskaber.
Classname: Her angives det fulde kvalificerede klassenavn for den Java klasse der skal initialiseres af kommunikationsenheden (fx dk.vtrack.comunit.server.les.ArgosDownloadClient).
Comments: Her kan man skrive en kort kommentar til modulet, hvis man ønsker dette (anbefalet).
Frequency og Offset angiver perioden hvor modulet skal kører (tiden angives i sekunder).
Port: Her angives hvilken port Listener modulet skal lytte på.
Active: Endelige kan man angive om modulet skal være aktivt eller ej. Et deaktiveret modul vil ikke starte op af kommunikationsenheden når den for besked på det (startes ikke op som tråd), men vil findes i databasen som en del af konfigurationen til kommunikationsenheden.



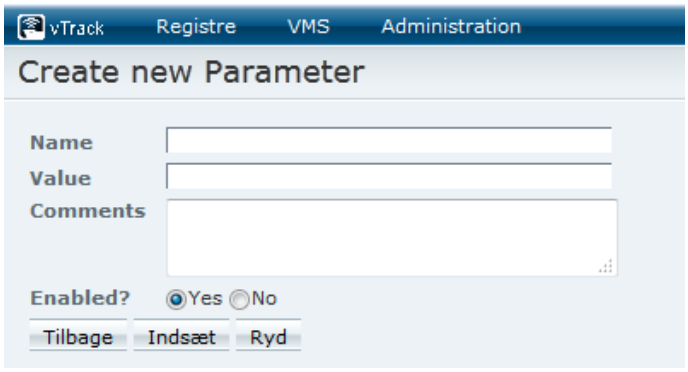
Figur 9 – Oprettelse af Timed module



Figur 10 – Oprettelse af Listener module

Et er de essentielle elementer i en konfiguration er parameter elementet. Man kan oprette en parameter i 3 forskellige steder for en konfiguration. Det ene sted er parameter som er generelle for alle moduler (dem som vi kalder for ”globale” parameter se Figur 3), parameter for et Timed module (se Figur 5) og parameter for et Listener module (se Figur 4).

Siden der vises for oprettelse af parameter for alle disse 3 elementer, er ens. Den eneste forskel er hvad man opretter parameter til.
Figur 11 viser et skærmbillede for oprettelse af en ny parameter.
Her skal navnet og værdien angives. Kommentarerne er valgfrie, men det er altid anbefalet at skrive en kort kommentar til parameteren som skal oprettes. Til sidste vælges om parameteren skal være aktiv eller ej.

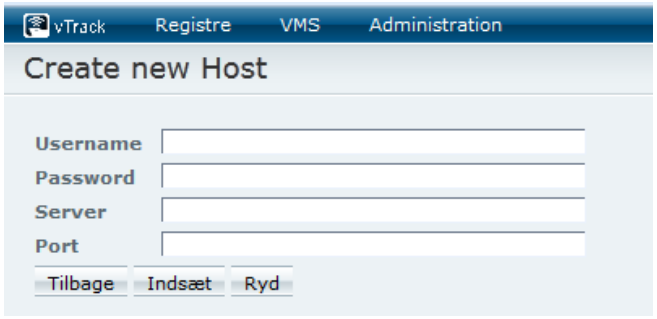


Figur 11 – Oprettelser af parameter

De sidste to ting som gennemgås kan kun oprettes for Timed modules.

Til et Timed module er det muligt at oprette en Host, således modulet kan etablere en forbindelse til en server med en angivet port nummer. Man kan også angive et brugernavn og et password i det tilfælde hvor det er krævet.
Bemærk at der højst kan være én Host til et Timed module. Hvis der en Host til et Timed module vil man ikke have mulighed for at oprette en ny, indtil den nuværende er slettet.

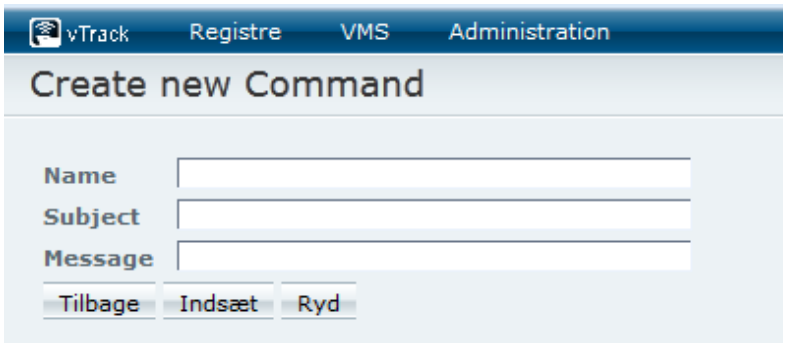
Figur 12 viser et skærmbillede hvor en Host oprettes. Her er Username og Password feltet obligatoriske felter, hvorimod Server og Port er frivillige.
Hvis port angives skal det være et helt positivt tal mellem 0 og 65535.



Figur 12 – Oprettelse af host

Figur 13 viser skærmbilledet hvor Command oprettes. Det er kun muligt at tilføje Command til Timed modules. Alle felter er obligatoriske og skal derfor udfyldes, hvis man ønsker at oprette en ny Command.

Bemærk venligst at det nødvendigvis ikke alle Timed modules der understøtter Command data af kommunikationsenheden!



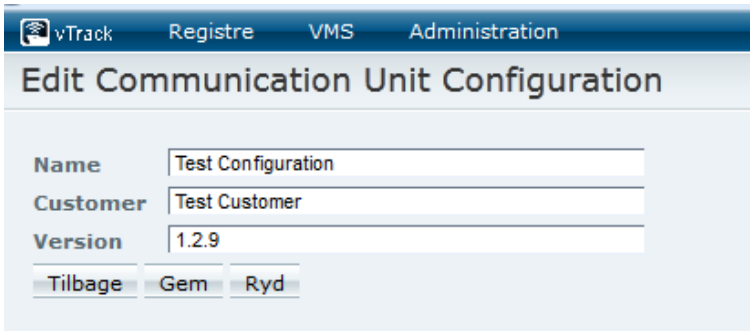
Figur 13 – Oprettelse af command

Opdatering

Det grafiske miljø giver også mulighed for at opdatere de eksisterende værdier i konfigurationen. Ud for de enkelte rækker i de forskellige tabeller som vi har gennemgået indtil nu, kan man klikke på "Rediger" knappen hvorved man vil blive dirigeret om til en side, som giver mulighed for at ændre i konfigurationsdata.

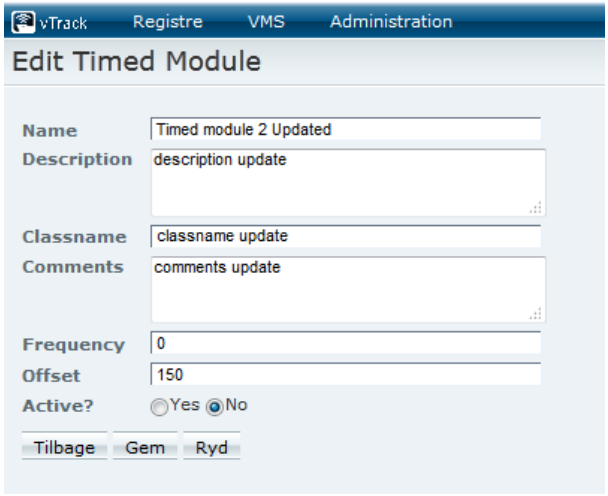
Figur 14 viser skærmbilledet for opdateringen af en konfiguration. Ligesom ved oprettelse af en konfiguration skal man angive de 3 felter som vist på figuren.
Når siden for opdatering loades vil de eksisterende værdier indsættes til de korresponderende felter og et klik på "Ryd" vil indsætte de gamle værdier ind.
Når opdateringerne er indtastet klikkes på "Gem" knappen for at gemme de nye opdateringer. Alle input felter valideres med samme formularer som i afsnittet overfor (Oprettelse).
Et klik på "Tilbage" knappen annuller opdateringsprocessen.

Funktionaliteten for de øvrige opdateringssider er det samme og derfor vil disse ikke blive forklaret dybere.

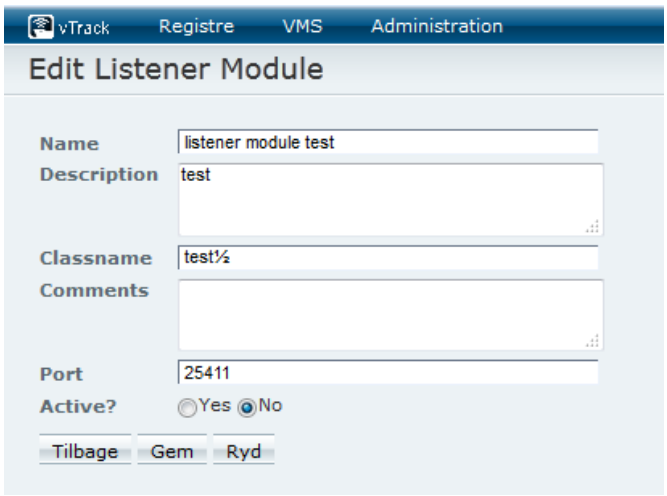


Figur 14 – Opdatering af konfiguration

Skærmbilledet for opdatering af et Timed module vises på Figur 15 og figur 16 viser skærmbilledet for opdatering af et Listener module.

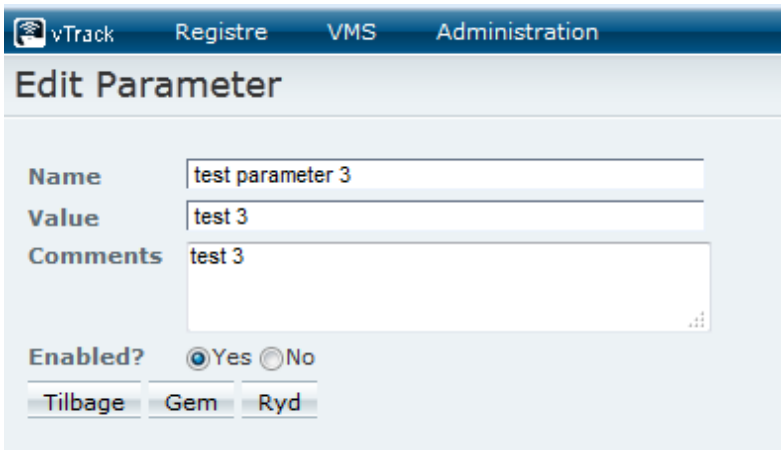


Figur 15 – Opdatering af Timed module



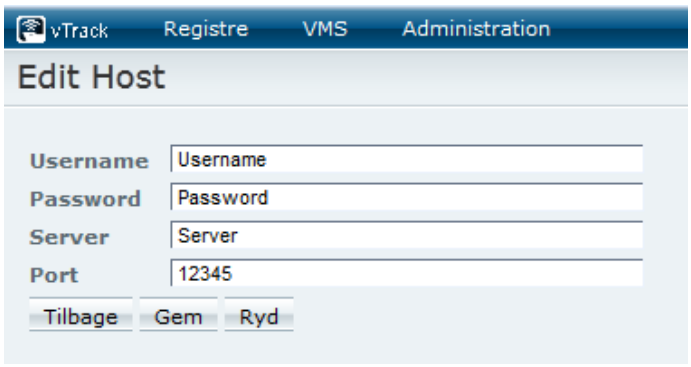
Figur 16 – Opdatering af Listener module

Figur 17 viser skærmbilledet for opdatering af parametre. Det samme vindue åbnes for alle typer af parametre (parametre for de 2 modul typer og for konfigurationen).



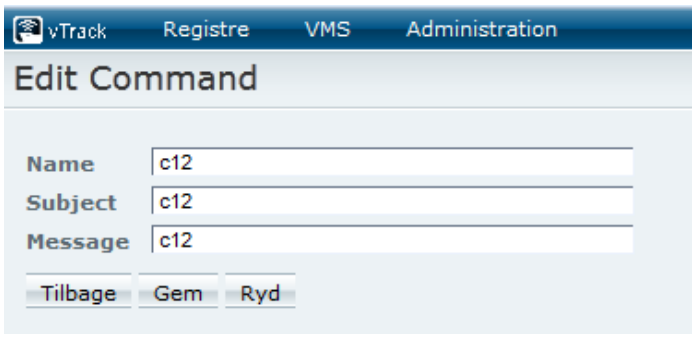
Figur 17 – Opdatering af parameter

Figur 18 – Viser skærmbilledet for opdatering af en Host til et Timed module.



Figur 18 – Opdatering af host

Skærmbilledet på Figur 19 viser opdatering af en Command (kun for Timed module).

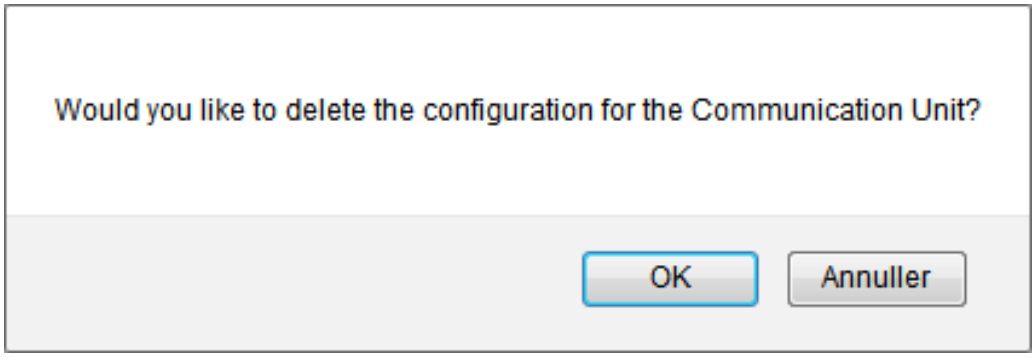


Figur 19 – Opdatering af command

Sletning

Ud fra hver række findes i tabellerne på Figur 1, Figur 3, Figur 4 og Figur 5 findes der en "Slet" knap. Man kan slette den pågældende række fra konfigurationen ved at klikke på denne knap. Uanset hvad du ønsker at slette vil du se en lignende meddelelse som på Figur 20. Her skal du bekræfte at du virkelig ønsker at slette det pågældende element.

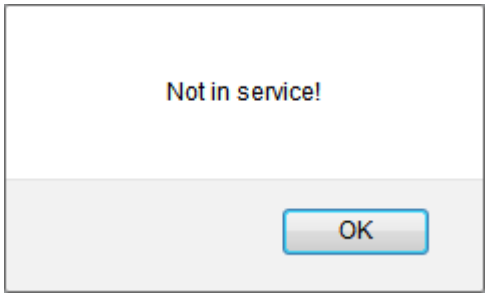
Bemærk ved sletning af et modul vil alle dets korresponderende data blive slettet. Det samme er gældende ved sletning af en konfiguration.



Figur 20 – Sletning af konfiguration

Opdatering af kommunikationsenheden

Et klik på "Restart Comm. Unit" knappen Figur 1 vil sende en påmindelse til kommunikationsenheden om at den skal genindlæse sin konfiguration fra databasen, da nogle af dem er ændret.
Ændringerne af konfigurationsparametrene vil blive opdateret af kommunikationsenheden, uden at man skal genstarte den manuelt.
Husk at have startet kommunikationsenheden før der klikkes på denne knap.
Lige nu virker denne funktionalitet ikke og derfor vises en meddelelse som på Figur 21.



Figur 21 – Opdatering af kommunikationsenheden

Tutorial

Tutorialen i denne håndbog tager udgangspunkt i at man har udført tutorialen for vTrack XML Parser (se tutorial afsnittet i håndbogen for vTrack XML Parser), således der findes mindst en konfiguration som vi kan arbejde med i denne tutorial.

Vi vil i denne tutorial udføre 2 simple opgaver. Vi vil som det første oprette et nyt modul (Timed module) med 2 parametre. Og dernæst vil vi deaktivere et andet modul. Dette er 2 meget almindelige use-cases som vil blive udført, når der kommer nye udvidelser til kommunikationsenheden og når kommunikations-enheden skal vedligeholdes.
Når dette er udført vil vi genstarte kommunikationsenheden med de nye værdier.

Man kunne forestille sig at der var implementeret et nyt modul, som skal kører en gang i døgnet for at slette filer med en bestemt endelse (fx .txt filer) fra en bestemt folder (fx c:/files). Derfor skal der oprettes et nyt modul (Timed module) som vil initialisere dette job, som en tråd, når kommunikationsenheden kører. Denne tutorial forventer ikke at dette er implementeret. Denne tutorial skal kun give et eksempel på hvordan man kan sætte lignende indstillinger op, når der kommer nye udvidelser til kommunikationsenheden. Derfor kan kommunikationsenheden muligvis give fejl når dette modul oprettes i denne tutorial!

Inden du starter på denne tutorial, vær sikker på at webapplikationen er oppe og køre på en server (fx Apache Tomcat). Og gå ind på følgende link: http://localhost/webclient/domain/comunit-configurations

Du vil blive bedt om at logge ind. Indtast venligt dit brugernavn og password for at logge ind.
Når du er logget ind vil du se et skærmbillede som på Figur 1 med et konfigurationsnavn som ”Test Configuration name” customer: ”Test Customer” og version: ”1.0.0”. Ud fra denne række klikkes der på “View details” knappen.

Du vil nu se et skærmbillede lignende som på Figur 2 med de tilhørende parametre og moduler. For at oprette et nyt modul vælges ”Timed module” som afbilledet på Figur 8. Når denne er valgt vil du blive dirigeret om til en side som på Figur 9.
Her indtastes følgende værdier (Alle felter bør angives på engelsk!):

Name: `File delete`
Description: `Delete old files`
Classname: `dk.vtrack.fileIO.FileDelete`
Comments: `This module deletes files from directory – needs 2 parameters.`
Frequency: `86400`
Offset: `0`
Active: `Yes`

Når dette modul er oprettet vil du vende tilbage skærmbilledet på Figur 2.
Vi skal nu have oprettet modulets 2 parametre.
Klik på ”View details” ud fra modulet. Lige efter ”Parameters” tabellen (som vil være tom da der ikke er nogle parametre) findes der en ”Tilføj Parameter” knap. Klik på denne knap og et skærmbillede som på Figur 11 vil komme frem.

Her indtastes værdierne:

Name: `Directory`
Value: `c:/files`
Comments: `Directory where files has to be deleted from.`
Enabled?: `Yes`

Dernæst klikkes på ”Indsæt” knappen og du vil vende tilbage til siden for modulets detaljer. Programmet vil nu have tilføjet parameteren til listen
Vi skal nu have tilføjet det anden parameter. Klik på ”Tilføj Parameter” og indsæt følgende værdier:

Name: `FileExtension`
Value: `.txt`
Comments: `file extension`
Enabled?: `Yes`

Når denne parameter er gemt, skal vi nu have deaktiveret et andet modul, da denne ikke skal kører længere. Klik på knappen ”Tilbage” (nederst på siden) for at vende tilbage til et skærmbillede lignende på Figur 2.

Der findes et modul der hedder: FILE_CLEANUP_LOG. Dette modul skal deaktiveres. Ud fra dette modul klikkes der på ”Redigere Module”. I det nye vindue ændres Active fra Yes til No.
Klik på ”Gem” for at gemme de nye værdier.
Dette vil nu deaktivere modulet i kommunikationsenheden.

Som det sidste skal vi have opdateret kommunikationsenheden. Dette gøres ved at klikke på ”Restart Comm. Unit” knappen (se Figur 1). Denne handling vil sørge for at opdatere kommunikationsenheden med det nye modul og deaktivere det andet modul (denne funktion virker dog ikke lige på nuværende tidspunkt).

Tillykke du har nu oprettet dit første modul med 2 parametre og har stoppet et andet modul som ikke skal kører længere!
Så snart genstart funktionaliteten virker vil du også have mulighed for at genstarte kommunikations-enheden automatisk.

Gloser

Miljø
En webapplikation som giver mulighed for at ændre i konfigurationsparametrene til den nye version af kommunikationsenheden (i vTrack) efter CRUD princippet.

CRUD
Create, Read, Update, Delete. De 4 mest gængse operationer der udføres på persistent data (fx data i database).

vTrack XML Parser
vTrack XML Parser er et program som er udviklet i forbindelse med dette projekt. Det er et program der er i stand til at kunne omsætte de gamle konfigurationsfiler om til SQL kald og persistere disse i database.

Kommunikationsenhed
En software komponent i vTrack som anvendes til at etablere forbindelse med eksterne kilder således der kan modtages og afsendes data (fx til fartøjer og fiskerimyndigheder).

Konfiguration
Kommunikationsenheden aflæser en række værdier fra en fil således at den ved hvilke processer der skal udføres. Denne konfiguration er nu migreret til en database.

Apache Tomcat server
Open source server der er i stand til at kunne afvikle JSP sider. Denne server anvendes til at afvikle de sider der er udviklet til projektets formål.

Oracle database
Databasen der anvendes i forbindelse med projektet. Der er her hvor det data aflæses og anvendes i de forskellige komponenter af vTrack.

Java
Java er et objekt orienteret programmeringssprog som oprindeligt blev udviklet af Sun, men som nu er overtaget af Oracle. Java anvendes som primært programmeringssprog i forbindelse med projektet. vTrack er udviklet i Java.

Eclipse
Et udviklingsmiljø hvor software udvikles. I dette projekt anvendes dette værktøj også til andre formål bl.a. til at bygge vTrack og starte den op gennem Tomcat server.

Danmarks Tekniske Universitet – Visma Consulting, Hellerup

vTrack XML Parser Håndbog

Ibrahim Nemli
1/28/2013

Forord

Jeg er en IT-Diplom studerende fra DTU som i forbindelse med min afsluttende eksamen har udviklet et program der er i stand til at kunne konvertere konfigurationsfilen til vTracks kommunikationsenhed om til SQL kald. Hidtil har kommunikationsenheden anvendt en xml fil som bruges i forbindelse ved opstart. Dette er nu lavet om, så den nye version af kommunikationsenheden indlæser sin konfiguration fra databasen.

Denne håndbog er målrettet til at give brugeren anvisninger til hvordan programmet betjenes. Du vil gennem denne håndbog få oversat din første konfigurationsfil, således vil du være i stand til at kunne bruge den nye version af kommunikationsenheden med det samme.
Yderligere findes der er også en installationsguide til hvordan softwaren installeres på din computer.

Jeg håber at dette produkt vil spare jer tid for den manuelle oversættelse af konfigurationsfilerne og I vil have glæde for at benytte programmet. God Fornøjelse!

Ibrahim Nemli
Januar 2013

Indholdsfortegnelse

Figur oversigt

Introduktion

vTrack XML Parser er et program som er dedikeret til at kunne omsætte den gamle konfiguration (som er baseret på xml filer) til vTracks kommunikationsenhed til SQL kald, da den nye version af kommunikationsenheden læser sin konfiguration fra database.

Denne håndbog er skrevet som en kombination af tutorials og brugervejledning til vTrack XML Parser. Formålet med denne håndbog er at give brugeren et indblik i hvordan programmet betjenes lige fra installeringen af programmet, til at kunne generere SQL kald på baggrund af den gamle konfiguration. Med dette program vil vi kunne opgradere vore kunders kommunikationsenhed til den nyeste version på en hurtigere og mere effektiv måde.

vTrack XML Parser er et simplet program som består af en 4 trins wizard som guider brugeren gradvist til at kunne oversætte de gamle konfigurationsfiler til SQL.

Denne håndbog dækker kun en lille del af det færdige produkt og giver mere en tutorial lignende oplevelse for brugeren, i stedet for en fuldstændig reference manual, hvilket gør, at brugeren får en lettere og mere informationsrig indblik i hvad dette program kan gøre.

Baggrund for læseren

For at give den fulde forståelse for denne håndbog, kan det være god ide at have en basisviden om de forskellige teknikker og teknologier der anvendes i denne håndbog. I denne håndbog vil emner som xml, xsd skema, Oracle database, SQL, og parser indgå. Der henvises til Gloser afsnittet i denne håndbog som giver en kort introduktion til de forskellige begreber.

Derudover er også en god ide at have kendskab til konfigurationsfilen til vTracks kommunikationsenhed før man anvender dette produkt.
Dog er ingen af disse et 'must' for at kunne betjene programmet.

Installations guide

Dette afsnit er en installationsvejledning, som beskriver hvordan du installerer vTrack XML Parser programmet for at få fuld udnyttelse af dets funktioner.

Software krav:

Her er en liste af softwareprodukter der skal installeres før vTrack XML Parser kan tages i brug. Sørg for at disse produkter er installeret før programmet anvendes.

Først og fremmest skal du have installeret Java Runtime Environment version 1.6.0 som minimums krav. Uden Java vil du ikke kunne være i stand til at kunne afvikle programmet! (Java kan hentes ned fra følgende link: http://www.oracle.com/technetwork/java/javase/downloads/index.html).

Med vTrack Xml Parser har man mulighed for at kunne gemme de genererede SQL kald direkte i en Oracle database (Andre typer af databaser understøttes ikke!). I tilfælde du ønsker at kunne bruge denne funktionalitet skal du have installeret en Oracle database. Her anbefales der at der anvendes Oracle Database 10g Express Edition eller nyere (kan downloades gratis på link: http://www.oracle.com/technetwork/products/express-edition/overview/index.html).

Installations guide:

Efter Java og Oracle database er installeret, skal du have oprettet tabeller i databasen, således den genererede data fra vTrack XML Parser kan persisteres i databasen. Der findes en SQL fil i Cd-rommen (SQL script\Database create com unit tables.sql) som du skal have kørt. Kør dette script (Brug evt. Oracle SQL Developer eller lignende). Dette script vil oprette de nødvendige tabeller for programmet.

vTrack XML Parser findes som en kørbar JAR fil (Java ARchive) som ligger på Cd-rommen (vTrack XML Parser\ vTrack XML Parser.jar). Programmet startes op ved at dobbeltklikke på denne fil. Programmet er afhængig af de 3 øvrige mapper (images, lib og schema) som derfor skal findes i samme mappe som programmet og må ikke findes særskilt!

Tillykke du har nu installeret vTrack Xml Parser og er klart til at anvende programmet.

Afvikling af det medfølgende eksempel

Tutorialen i denne håndbog tager udgangspunkt i et eksempel som er findes i mappen vTrack XML Parser Tutorial på Cd-rom. Denne mappe indeholder 4 filer (xml filer), præcist som vTrack konfigurationen.

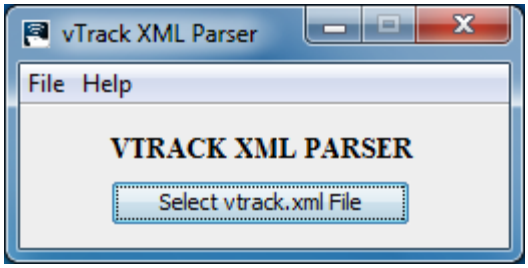
Med hensyn til forretningshemmeligheder er værdier som username, password og server erstattet med værdien ”ask_customer”. Værdier for Port er angivet som ”0000”. Dette har dog ingen betydning for dette program eller for tutorialen, men kan have en betydning når disse værdier skal anvendes i kommunikationsenheden for en givet kunde. Derfor bør værdierne fra disse filer ikke anvendes hos vore kunder!
Disse filer bør kun anvendes til at vise funktionaliteten af dette program til brugeren.

GUI Oversigt

Dette afsnit beskriver de grafiske komponenter som programmet består af og som det anbefales at man stifter kenskab til, før man udfører tutorialen.

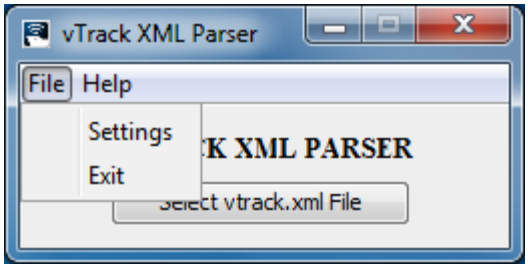
vTrack XML Parser

Dette afsnit handler generelt om de forskellige grafiske komponenter som programmet består af. Senere i afsnittet vil vi komme dybere ind i de øvrige komponenter.



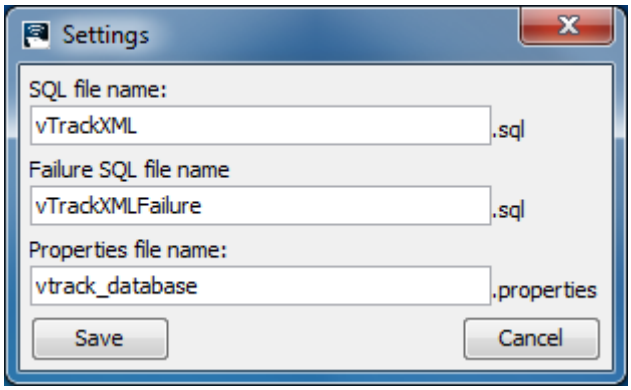
Figur 1 – Hovedmenu

Det første som brugeren vil se når vTrack XML Parser programmet starter op, er billedet på Figur 1. Programmet har en meget simpel brugergrænseflade med én knap og 2 menu knapper (File og Help).



Figur 2 – File Menu

I "File" menuen findes der 2 undermenuer. Man kan vælge mellem at afslutte programmet ved at klikke på "Exit" eller man gå ind i indstillinger ved at klikke på "Settings".



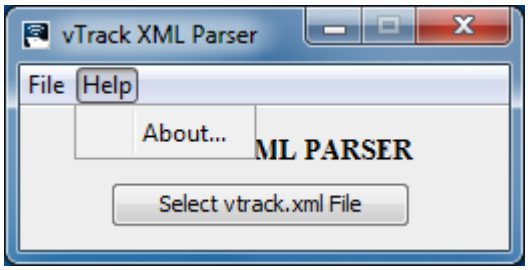
Figur 3 – Settings

I settings menuen vil du se et vindue som på Figur 3. Denne menu giver brugeren mulighed for at give et prædefineret filnavn til de 3 filer der kan genereres af programmet (det er ikke nødvendigvis tilfældet at alle 3 filer genereres ved oversættelse af en konfiguration, dette afhænger af brugerens valg).

De angivne filnavne på figuren er programmeret (hardcoded) og vil altid være det samme når programmet starter op.

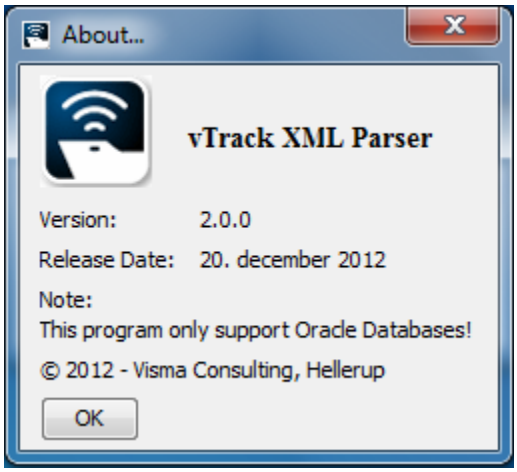
”SQL file name” er filnavnet på filen som indeholder SQL kald der bliver genereret på baggrund af data fra konfigurationsfilerne.
”Failure SQL file name” er navnet på den fil som indeholder SQL kald der fejlede ved indsættelse i databasen.
”Properties file name” er filnavnet på den properties fil der anvendes af den nye version af kommunikationsenheden. Værdierne af denne fil opsættes i wizarden.
Der anbefales ikke at man ændrer ”Properties file name” da dette navn anvendes af kommunikations-enheden (vtrack_database.properties).

Ved ændring af værdierne må ingen af felterne være tomme!



Figur 4 – Help Menu

Under ”Help” menuen finder der en enkelt undermenu og dette er ”About…”. Når man klikker på denne, vil man få vist et vindue som på Figur 5.

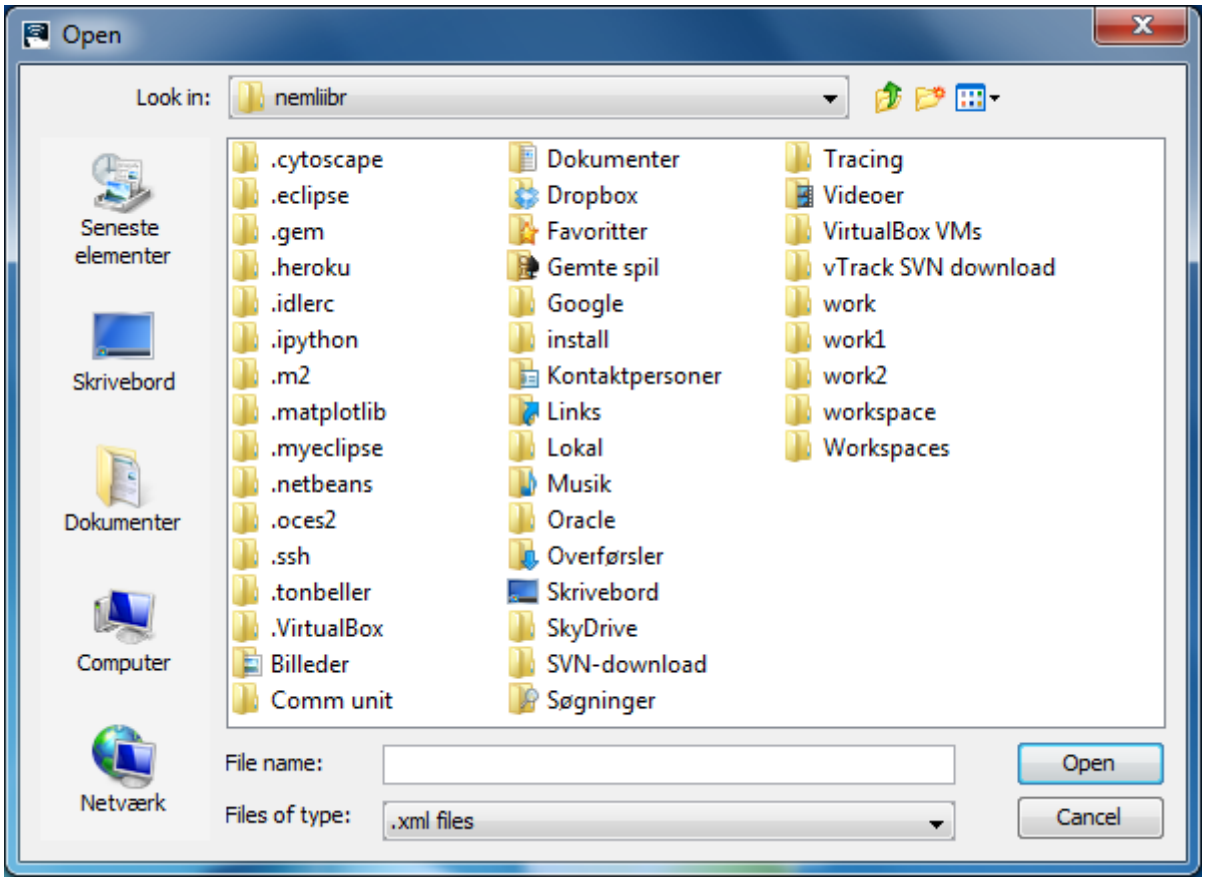


Figur 5 – Om vTrack XML Parser

Figur 5 viser informationer omkring programmet. Her er det muligt at se hvilken version af programmet der anvendes og hvornår denne er udgivet, samt yderligere informationer om programmet.

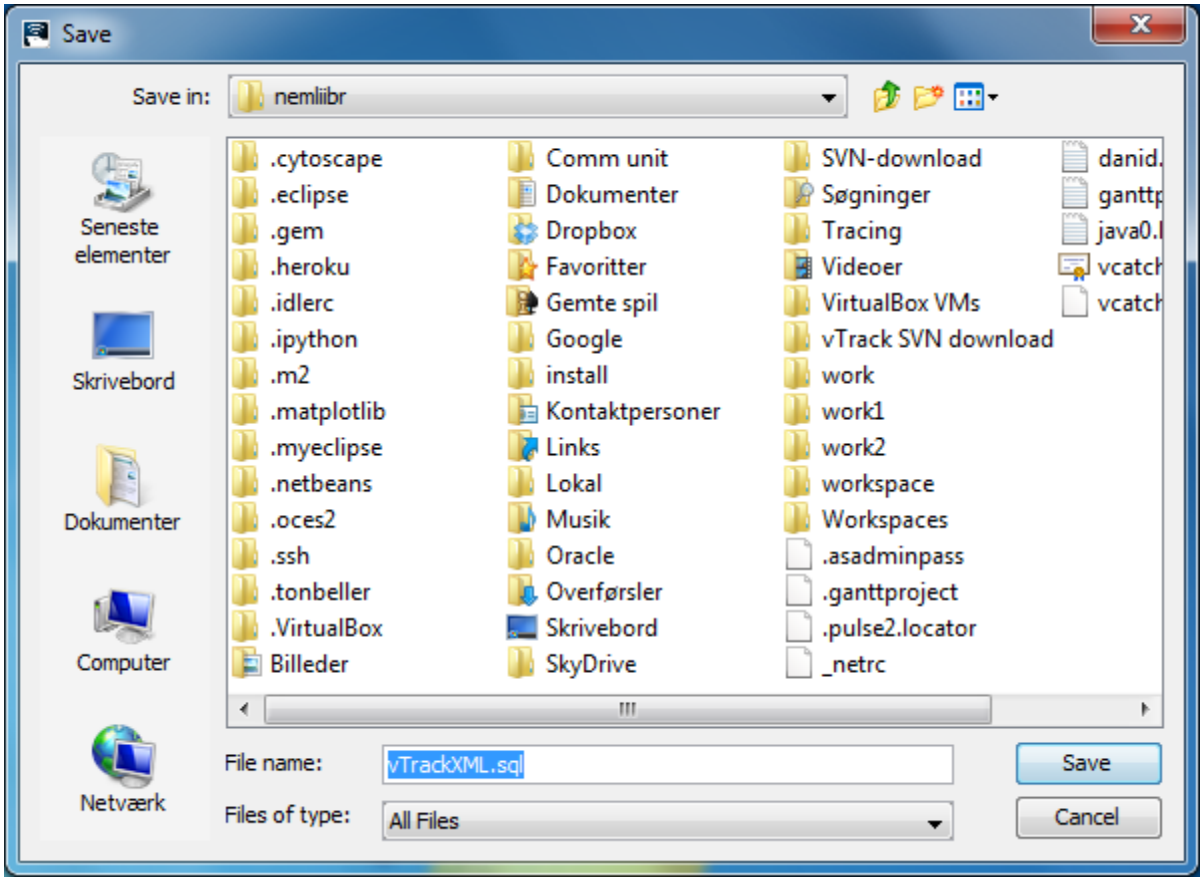
Filhåndtering

For at gøre det nemmere for brugeren at navigere rundt i filsystemet anvender vTrack XML Parser menuer som vist på Figur 6 og Figur 7.



Figur 6 – Menu til fil åbning

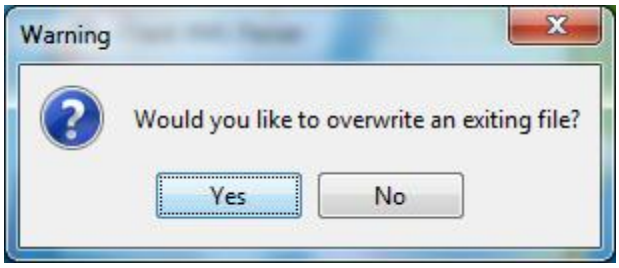
Denne menu anvendes når man skal åbne en fil.
Filtypen som man skal åbne vil altid være filtreret. I figur 6 kan man se at der er filtreret for filer med endelsen ".xml". Dette betyder at programmet forventer at indlæse en fil med endelsen ".xml".



Figur 7 – Menu til at gemme filer

Figur 7 viser menuen der anvendes når programmet skal gemme en fil. Denne menu gør det nemmere for brugeren at gemme den genererede fil det ønskede sted i filsystemet.

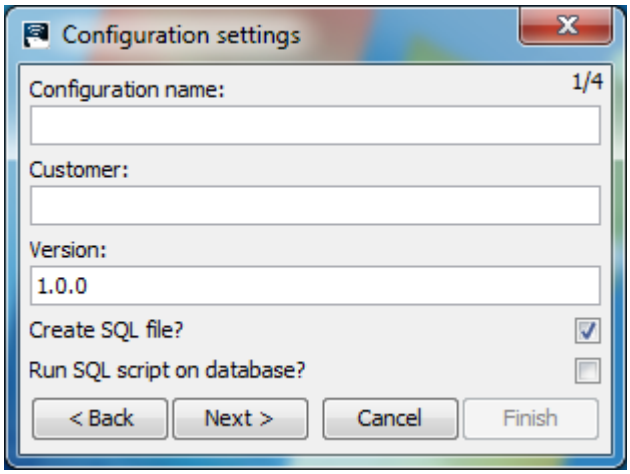
I tilfælde af at den samme fil findes i den folder hvor brugeren ønsker at gemme sin fil, vil en meddelelse som på Figur 8 vises. Her skal brugeren tage hensyn til om den eksisterende fil skal overskrives eller ej. Ønsker du ikke at overskrive den eksisterende fil, vil menuen på Figur 7 vises igen, hvor du har mulighed for enten at angive et nyt filnavn eller vælge en anden placering i filsystemet.
Klikkes der på ”Yes” knappen vil programmet overskrive filen.



Figur 8 – Overskrivning af eksisterende fil

Wizard

Wizarden i vTrack XML Parser er en 4 trins opsætning for oversættelsen, som guider dig igennem til at generere de ønskede filer.
For hvert trin i menuen vil du i øverste højre hjørne se, hvilket trin du er nået til i wizarden. På denne måde kan du altid følge med for hvor langt du er nået. På bunden af hvert vindue findes der 4 knapper som du bruger til at kunne naviger rundt i wizarden.
"< Back" knappen vil føre dig til den forrige menu, "Next >" vil føre dig til næste menu, "Cancel" knappen vil annullerer processen og "Finish" knappen vil afslutte wizarden og generere de ønskede filer. Denne knap er dog kun tilgængeligt (aktiv) i det sidste trin i wizarden.
Det er ikke muligt at afslutte wizarden på noget som helst tidspunkt ved klik på det røde kryds oppe på i højre hjørne. Brug "Cancel" knappen til at afslutte processen.



Figur 9 – Konfigurationsindstillinger

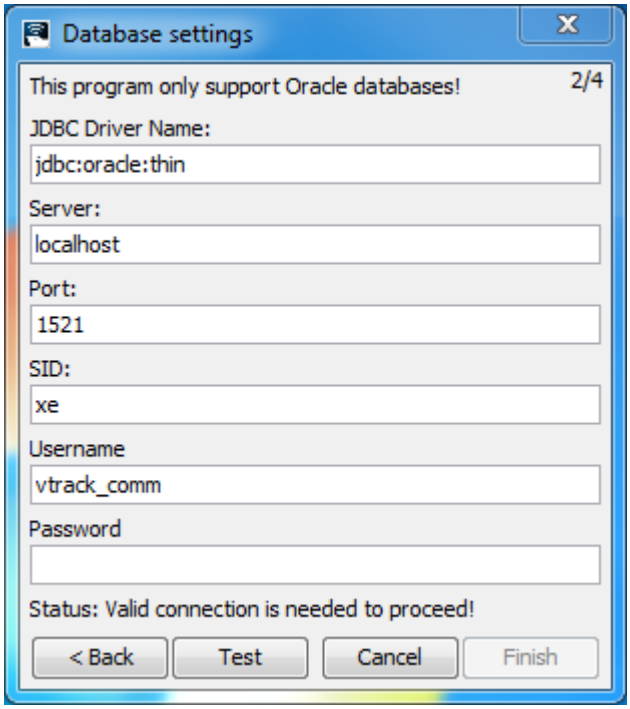
Figur 9 er det første vindue i wizarden. Her skal brugeren angive navnet til konfigurationen, kundens navn og versionen af konfiguration. Disse 3 felter skal være udfyldte før man kan fortsætte til næste menu.

Det er vigtigt at konfigurationsnavnet er unikt i den forstand at man ikke opretter en ny kommunikationsenhed konfiguration med det samme navn i databasen. Hvis der findes en anden konfiguration med det samme navn, kan der muligvis opstå konflikter når kommunikationsenheden startes op. vTrack XML Parser programmet tjekker IKKE for dette, dette er op til brugeren!
En konfiguration med samme navn vil heller ikke kunne indsættes i databasen, pga. denne fejl!

Hvis du ønsker at oprette en SQL fil som indeholder SQL forespørgsel for data i konfigurationsfilerne bør feltet for "Create SQL file?" være tjekket af.

Med vTrack XML Parser har brugeren også mulighed for at køre og derved persistere de genererede SQL kald direkte på en Oracle database. Hvis dette ønskes skal feltet for "Run SQL script on database?" være aftjekket.

Per default er versionen angivet til version 1.0.0 og "Create SQL file?" er aftjekket, når dette vindue vises.

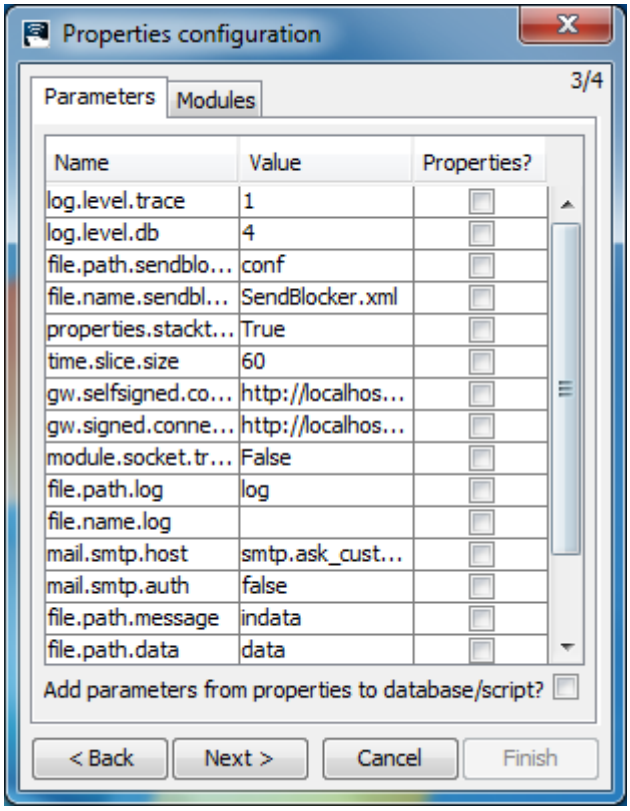


Figur 10 – Databaseindstillinger

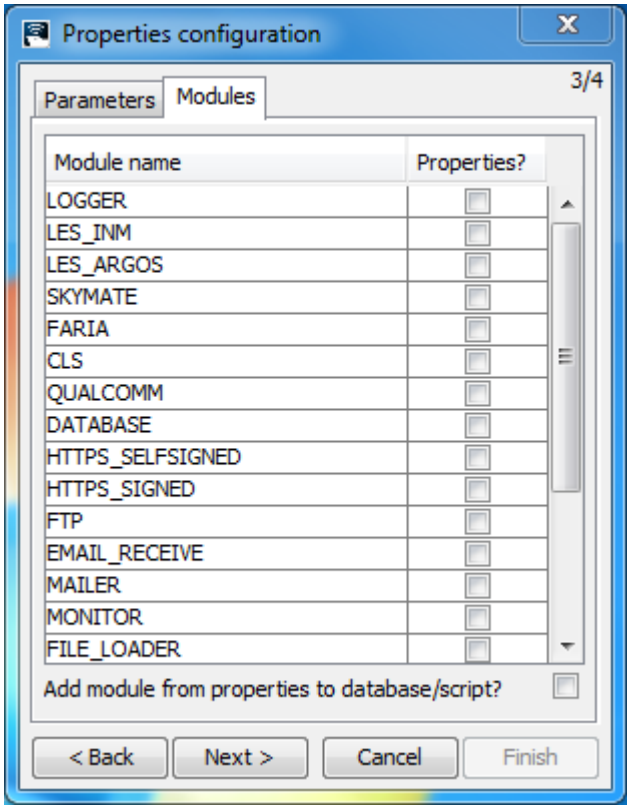
Hvis du i Figur 9 har valgt at du gerne vil indsætte SQL scriptet i databasen vil du se et skærmbillede som på Figur 10 ellers vil du se et skærmbillede som på Figur 11.

I denne menu opsættes forbindelsen til databasen, hvor data skal indsættes. Bemærk at programmet kun understøtter Oracle databaser.
Nogle af felterne vil automatisk blive udfyldt for dig. Dette er afhængigt om ”LOGGER” modulet findes i den indlæste konfigurationsfil.
Dog vil password feltet vil aldrig blive foreslået af sikkerhedsmæssige årsager.

Alle felter skal være udfyldte og der skal være en gyldig database forbindelse før det er muligt at komme videre til næste trin i wizarden. Uden dette vil det ikke være muligt at komme videre til næste menu.



Figur 11 – Properties indstillinger for parameter



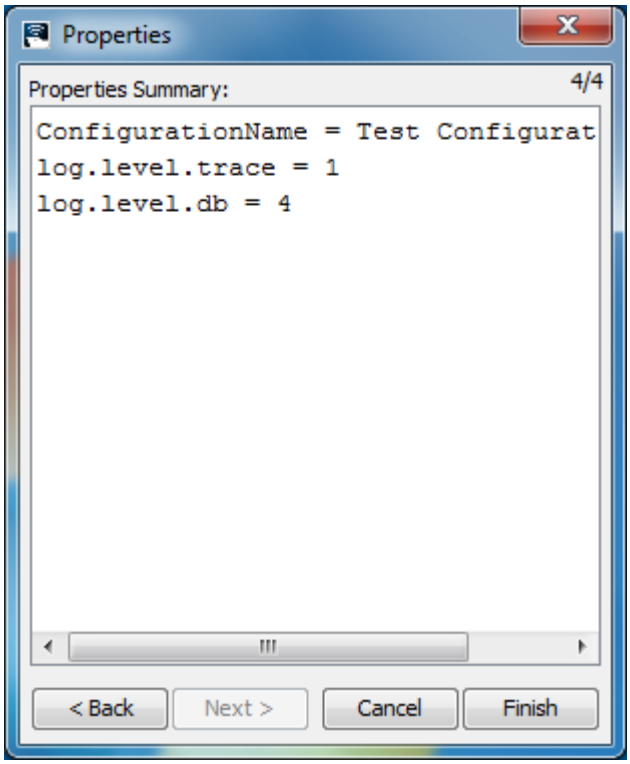
Figur 12 – Properties indstillinger for moduler

Skærmbilledet på Figur 11 og Figur 12 er properties konfigurationen og giver brugeren mulighed for at kunne opsætte properties filen som anvendes af den nye version af kommunikationsenheden. Ved klik på checkboksen ud fra parameteren eller ud fra modulet vil tilføje det til properties filen.

Man bør kun vælge parametre og moduler som er specificeret i den medfølgende rapport (se afsnit 15.3 side 64-65), for at kommunikationsenheden vil starte korrekt op.

Når man vælger et modul, vil alle dets tilhørende værdier tilføjes til properties filen automatisk. Det er ikke muligt at specificere hvilke tilførende værdier der skal med og hvilke der ikke skal med, for et givet modul. Dette kan gøres efter properties filen er genereret.

Som default vil de valgte parametre og moduler ikke tilføjes i den genererede SQL fil (eller indsættes i databasen) da dette vil give redundant data. Programmet giver dog mulighed for dette. For begge menuer er det muligt at angive om man ønsker at tilføje de valgte parametre og moduler i databasen/scriptet som genereres.



Figur 13 – Properties resume

Figur 13 er det sidste trin i wizarden. Dette trin giver mulighed for at se indholdet på properties filen som man har konfigureret i properties konfigurationen, før man genererer de ønskede filer. Det er ikke muligt at redigere i properties filen på dette skærmbillede.

Et klik på ”Finish” knappen vil afslutte wizarden og programmet vil udføre de ønskede funktioner.

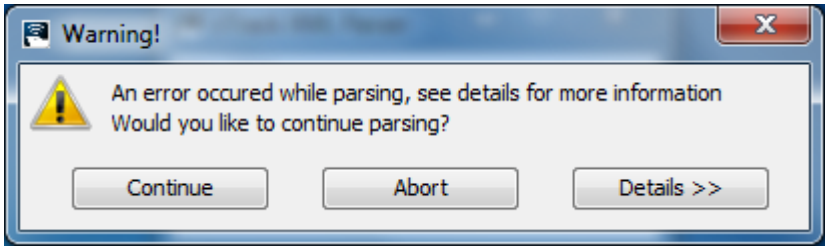
Fejlmeddelelser

Fejlmeddelelser i vTrack XML Parser har deres eget vindue. Dette består af 3 knapper, ”Continue”, ”Abort” og ”Details”.

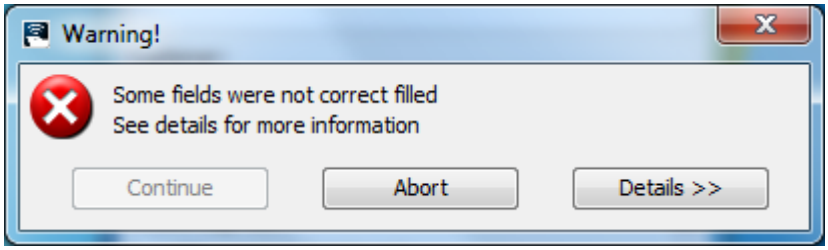
Afhængig af om programmet kan fortsætte eller ej vil ”Continue” knappen være tilgængelig. Dette er afhængig af den specifikke fejlsituation der opstår i programmet.
Uanset hvilken fejl der opstår, er det ikke muligt at klikke på det røde kryds i denne menu (brug ”Abort” knappen til dette formål)!

Figur 14 og Figur 15 viser to eksempler på fejlmeddelelsen hvor der i det første er mulighed for at forsætte processen i programmet og i det andet har man ikke mulighed for at forsætte. Klik på ”Abort” knappen vil afslutte processen. Hvorimod ”Continue” knappen vil fortsætte processen.

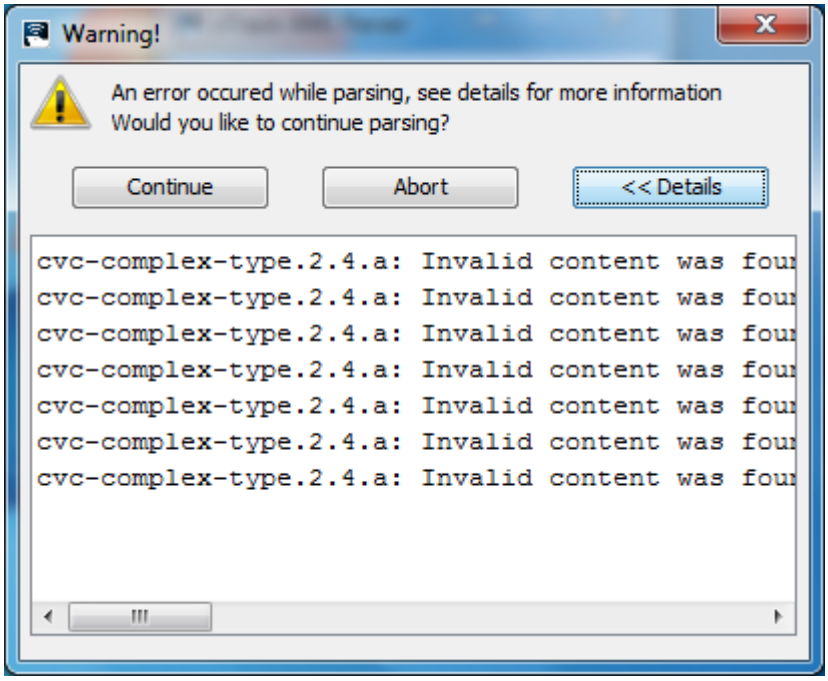
Klik på ”Details” knappen vil vise yderligere detaljer om fejlen. Figur 16 og Figur 17 viser detaljerne for de 2 figurer nedenfor. Endnu et klik på ”Details” knappen vil skjule detaljerne for fejlen.



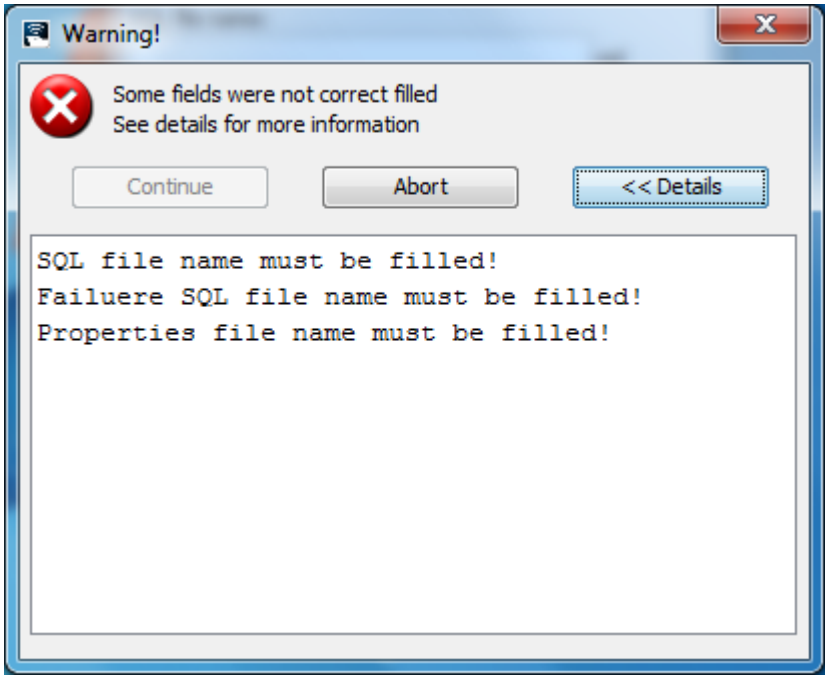
Figur 14 – Fejlbesked 1



Figur 15 – Fejlbesked 2



Figur 16 – Fejlbesked 1 detaljer



Figur 17 – Fejlbesked 2 detaljer

Tutorial

I denne tutorial vil vi oversætte den første konfigurationsfil til SQL forespørgsel og indsætte dette i databasen. Formålet med denne tutorial er at vise så mange funktioner i vTrack XML Parser som muligt.

Når programmet er oppe og køre skal du klikke på ”Select vtrack.xml File” knappen. Dette vil åbne en menu op hvor du har mulighed for at åbne en fil. Programmet har allerede filtreret for mapper og for ”.xml” filer. Vi skal i denne menu have navigeret hen til cd-rommen og herfra skal du åbne ”vtrack.xml” filen som ligger i mappen ”vTrack XML Parser Tutorial”.
Naviger derhen og åben filen.

Når denne fil er valgt vil programmet for det første tjekke om xml filen er valid og derefter vil programmet kontrollere for om xml filen er gyldig. Hvis disse to ting er i orden vil programmet parse filen ellers vil der vises en fejlmeddelelse.
I dette tilfælde har programmet opdaget en fejl. Hvis du klikker på ”Details” knappen vil du se at der 7 ”cvc-complex-type.2.4.a: …” fejl (samme fejlmeddelelse vises på Figur 16). Dette skyldes at xml filen ikke er angivet præcist som beskrevet i xsd skemaet (dette er et skema som anvendes når xml filen tjekkes for om den er gyldig).

Du har nu 2 muligheder:
1: Du kan vælge at stoppe (ved at klikke på ”Abort”), rette fejlen og derved prøve igen eller
2: Som du måske har lagt mærke det, så har du stadig mulighed for at fortsætte med processen ved at klikke på ”Continue” knappen. Vi vælger derfor at fortsætte. Klik på ”Continue” knappen.

Programmet vil nu åbne det første trin i wizarden.
I dette trin skal vi have angivet nogle yderligere informationer til konfigurationen.
Her angives følgende data (da dette er en tutorial er det givne data ikke så nødvendige, men vær sikker på at en anden konfiguration med samme navn ikke eksister i databasen, da dette kan give fejl senere i processen):

Configuration name: `Test Configuration name`
Customer: `Test Customer`
Version: `1.0.0`
Og vi ønsker både at oprette en SQL fil og køre SQL scriptet på databasen.
Når disse felter er udfyldte klikkes på ”Next >” knappen.

Dette vil nu åbne skærmbilledet for indstillinger for etablering af database forbindelsen. Her vil de øverste 5 felter være angivet, da den indlæste konfigurationsfil indeholder et modul der hedder ”LOGGER”. Hvis programmet ikke kan finde dette modul vil disse felter være blanke.
Værdierne er nødvendigvis ikke gyldige for at etablere forbindelse til din database. Sørg for at alle disse felter er korrekte forhold til din database, således der kan etableres en forbindelse. Brug evt. ”Test” knappen for at teste forbindelsen til databasen. Hvis der fejl vil disse vises.
Så snart der er en gyldig forbindelse vil programmet fortsætte til properties konfigurationen.

I properties konfiguration er det muligt at opsætte properties filen til den nye version af kommunikations-enheden. Her bør man vælge parametre og moduler som angivet i rapporten (se afsnit 15.3 side 64-65).
I denne tutorial vælges:

- Log.level.trace
- Log.level.db

fra parametrene og fra modulerne afmærkes modulet:

- LOGGER

For både parametre og moduler ønsker vi ikke at tilføje de afmærkede moduler og parametre i databasen eller i scriptet.
Når disse er valgt klikkes på ”Next >” knappen.

Det sidste trin i wizarden vil nu blive vist. Denne viser indholdet af de properties der vil genereres.
Du kan altid gå tilbage og lave de ønskede ændringer som du har lyst til.
Så snart du er færdig klikkes på ”Finish” knappen og programmet vil nu generere de ønskede filer.

Den første fil der genereres er ”vTrackXML.sql” filen. Denne indeholder alle SQL kald der genereret på baggrund af den indlæste konfigurationsfil.
Gem venligst filen et sted i filsystemet.

Den næste fil der genereres er properties filen som blev sat op i wizarden (denne fil genereres altid og kan ikke fravælges). Gem venligst filen et sted i filsystemet (generelt bør denne fil ligge i conf mappen for kommunikationsenheden). Den genererede properties fil i denne tutorial bør ikke anvendes af kommunikationsenheden!

Som det næste vil programmet nu gemme de genererede SQL kald i databasen. Hvis det lykkedes at indsætte disse kald i databasen vil fortsætte (programmets hovedmenu vil blive vist).
Hvis der sker en fejl vil du se en fejlmeddelelse, hvor der står at der var nogle SQL kald som ikke blev indsat korrekt og at du bedes klikke på ”Continue” knappen for at gemme disse i en separat fil.
Hvis der opstår en fejl i denne tutorial bedes du klikke på ”Continue” og vælge et sted i filsystemet hvor du ønsker at gemme filen.
Denne fil indeholder SQL kald som ikke kunne indsættes af en eller anden grund. For hvert SQL kald er der årsagen for fejlen inkluderet. Disse kald bør rettes og indsættes manuelt.

Indsættelse af data i databasen kan tage et stykke tid afhængig af hvor stor en mængde data der skal indsættes, så vær venligt at vente et øjeblik. Når programmet er færdigt vil programmets hovedmenu vil blive vist.

Tillykke du har nu oversat din første konfigurationsfil og har genereret både en SQL fil og samtidig gemt disse data i databasen.

Gloser

Xml
Extensible Markup Language. Anvendes bl.a. til at gemme og udveksle data mellem forskellige computere og systemer. Xml syntaksen er nem at lære og opbygges som en træstruktur.

Xsd skema
XML skema. Formålet med xsd skemaet er at definer de lovlige tags i et xml dokument. Således kan xsd skemaet anvendes til at validere xml skemaer.

Parser
Parserens hovedopgave er at konverter givet input til et andet format som output. I dette program konverteres en xml fil om til SQL kald.

Kommunikationsenhed
En software komponent i vTrack som anvendes til at etablere forbindelse med eksterne kilder således der kan modtages og afsendes data (fx til fartøjer og fiskerimyndigheder).

Java
Java er et objekt orienteret programmeringssprog som oprindeligt blev udviklet af Sun, men som nu er overtaget af Oracle. Java anvendes som primært programmeringssprog i forbindelse med projektet. vTrack er udviklet i Java.

Oracle database
Databasen der anvendes i forbindelse med projektet. Der er her hvor det data aflæses og anvendes i de forskellige komponenter af vTrack.

SQL
Structured Query Language, et programmerings sprog der anvendes I forbindelse med relationalle databaser og til at håndtere data i databasen.

GUI
Graphical User Interface. En grafisk brugergrænseflade som gør det muligt for brugeren at interagere med systemet.

Appendix E - Timeregnskab

E.1 Introduktion

Jeg har i projektets forløb lavet et timeregnskab, så jeg kan følge hvor meget tid jeg har brugt på de forskellige aktiviteter.
Design og analyse indebærer ting som fx: analyse af kildekode og databasekode, udvikling af modeller og diagrammer.
Felterne implementering og test viser hvor lang tid jeg har brugt på at udvikle kildekode til projektet og hvor lang tid jeg har brugt på at teste den udviklede kode.
Dokumentering er tiden som er brugt på at skrive rapport, herunder statusrapporter, skrive mails mellem vejledere og for at samle projektet (ved afslutningsfasen).
Aktiviteten "Andet" indebærer ting som fx: møder, søgning af kilder på internettet udførelse af tutorials og andre aktiviteter som er relevant for projektets fremgang.

E.2 Timeregnskab

Tabellen nedenfor viser et samlet timeregnskab jeg har brugt på projektet i perioden 21. januar 2013 – 28. januar 2013. De øvrige timeregnskaber kan findes i statusrapporterne på cd-rommen.

Dato	Design	Analyse	Implementering	Test	Dokumentering	Andet	I alt
Januar 2013							
21/01 – 13					8		8
22/01 – 13					8		8
23/01 – 13					8		8
24/01 – 13					8		8
25/01 – 13					8		8
28/01 – 13						4	4
TOTAL	**0**	**0**	**0**	**0**	**40**	**4**	**44**

E.3 Opsummering

Nedenfor findes der en tabel som giver et samlet billede af hvor lang tid jeg har brugt på de forskellige aktiviteter for hele projektet.
Dette regnskab har taget højde for fejl og mangler som er beskrevet i Readme.txt filen som er vedlagt i cd-rommen i samme folder som statusrapporterne.

OPSUMMERING – 3. september 2012 til 28. januar 2013							
	Design	**Analyse**	**Implementering**	**Test**	**Dokumentation**	**Andet**	**I alt**
TOTAL	**56t 15 min**	**68t 45 min**	**150t**	**63t**	**175t**	**84t**	**597t**