HAIBA

Minipas-konvertering

Design, arkitektur og udviklerguide

Indhold

1 Formål 3

2 Arkitektur- og designoverblik 4

3 Særlige forhold 6

3.1 Personhenførbare data 6

4 Fysiske Datamodeller 7

4.1.1 Database 7

4.1.2 Meta-database 7

5 Opsætning af udviklingsmiljø 8

5.1 Kildekode 8

5.2 Byggemiljø 8

5.2.1 Dependencies 8

5.3 Database-setup 8

5.3.1 Database 8

5.4 Test 8

5.5 IDE 9

5.5.1 Eclipse 9

5.6 IntelliJ Idea IDE 10

5.7 Distribution 11

6 Tips og tricks 12

6.1 Tomcat out of memory 12

6.1.1 Beskrivelse 12

6.1.2 Løsning 12

7 Referencer og kilder 13

8 Ændringslog 14

# Formål

Dette dokument giver et overblik over Minipas-konverteringen i HAIBA med fokus på design og arkitektur. Dokumentet er også en guide til udviklere. Guiden gennemgår på overordnet plan de aktiviteter, der er nødvendige for at kunne videreudvikle på servicen.

Dokumentet har som formål at give et indblik i det underliggende design, de udstillede snitflader på det overordnede niveau samt særlige forhold.

Både udviklere, aftagere samt driftsleverandør kan med fordel læse dette dokument.

# Arkitektur- og designoverblik

Minipas-konverteringens formål er at konvertere eksisterende tabeller til et andet tabellayout.

Overordnet set består konverteren af en applikation kørende på en JEE applikationsserver (I dette tilfælde Tomcat).

Konverteren er bygget op via Spring version 3.x (se <http://www.springsource.org/> ) som en standard JEE webapplikation. Kendskab til følgende Spring undermoduler vil være en fordel (MVC, batch og Integration).

Til at bygge systemet bruges Maven 3.x.

Modulerne er strukturerede på følgende måde:

* Preprocessoren har til ansvar at synkronisere rækker fra årgangstabellerne T\_ADM20XX, T\_DIAG20XX etc. (hvor XX er antal år efter år 2000) med tabeller i det eksisterende LPR-format med T\_ADM og T\_KODER. Preprocessoren ligger i pakken: dk.nsi.haiba.minipasconverter.executor, klassen MinipasPreprocessor er det skedulerede job som sørger for at processere data periodisk.
* Modulet dk.nsi.haiba.minipasconverter.model indeholder pojo-objekter, serialiseret fra kildedatabaserne
* Modulet dk.nsi.haiba.minipasconverter.dao indeholder adgang til og fra databaserne. Disse DAO-objekter indeholder også cache fra tabellerne.
* Modulet dk.nsi.haiba.minipasconverter.status indeholder en minimal webside, der kan give informationer om sidste kørsel, igangværende kørsel og tidspunkt for næste kørsel.

Følgende designvalg er gjort:

* De eksterne tabeller T\_ADM20XX, T\_DIAG20XX opdateres hver nat. T\_ADM20XX indeholder kolonnen IDNUMMER som altid refererer til den samme indlæggelse, derfor kan værdierne herfra benyttes til synkronisering. Samme table indeholder også kolonnen SKEMA\_OPDAT med tidspunktet for sidste ændring af den refererede indlæggelse.
* Tabellerne T\_DIAG20XX, T\_SKSUBE20XX samt T\_SKSOPR20XX indeholder data for indlæggelsen. Rækker fra disse tabeller kopieres til destinationstabellen T\_KODER.
* Tabellen T\_MINIPAS\_SYNC husker værdier fra IDNUMMER samt SKEMA\_OPDAT og kan ved næste kørsel se, om der er sket ændringer på indlæggelserne. Er dette tilfældet, genprocesseres rækkerne for denne indlæggelse. Nævnes indlæggelsen ikke, er den fjernet og indlæggelsen fjernes også fra destinationstabellerne samt der indsættes en række I T\_LOG\_SYNC\_HISTORY med denne information.
* Umiddelbart i starten af hver konvertering hentes status for minipas-data fra minipas-status-tabellen (nuværende hedder T\_MINIPAS\_UGL\_STATUS). Er data ikke klar endnu, eller melder status-tabellen fejl (værdien er over 1) forsøges et antal gange (pt. 2 flere forsøg) med et mellemrum på 30 minutter. Herefter stoppes konverteringen. Hvis statustabellen melder at data er klar på de ekstra forsøg, fortsætter konverteringen som normalt.
* Degradering: Hvis databehandlingen afbrydes utilsigtet eks. pga. kodefejl eller fysisk afbrydelse, vil importen blot genstarte ved næste kørsel med en ny komplet synkronisering. Administration af synkronisering og inserts i destinationstabellerne er omfattet af den samme transaktion.
* Performance: for at optimere tidsforbruget under processering er der indført en cache omkring læsning af rækker fra T\_DIAG20XX, T\_SKSUBE20XX samt T\_SKSOPR20XX. Det viste sig at opslag fra disse tabeller var langsommere end batchlæsning. Da rækker fra T\_ADM læses med stigende nøgle, kan det lade sig gøre at hente et antal rækker (også sorteret) med data lig med eller større end denne nøgle, når den tidligere cache ikke længere er aktiv.

Data kopieres som følger (se tabellayout i afsnit 4):

T\_ADM20XX.IDNUMMER -> T\_ADM.V\_RECNUM

T\_ADM20XX.C\_SGH -> T\_ADM.C\_SGH

T\_ADM20XX.C\_AFD -> T\_ADM.C\_AFD

T\_ADM20XX.C\_PATTYPE -> T\_ADM.C\_PATTYPE

T\_ADM20XX.V\_CPR -> T\_ADM.V\_CPR

T\_ADM20XX.D\_INDDTO -> T\_ADM.D\_INDDTO

T\_ADM20XX.D\_UDDTO -> T\_ADM.D\_UDDTO

T\_DIAG20XX.IDNUMMER -> T\_KODER.V\_RECNUM

T\_DIAG20XX.C\_DIAG -> T\_KODER.C\_KODE

T\_DIAG20XX.C\_TILDIAG -> T\_KODER.C\_TILKODE

T\_DIAG20XX.C\_DIAGTYPE -> T\_KODER.C\_KODEART

T\_DIAG20XX.INDBERETNINGSDATO -> T\_KODER.PDTO

T\_DIAG20XX.C\_SGH -> T\_KODER.C\_PSGH

T\_DIAG20XX.C\_AFD -> T\_KODER.C\_PAFD

”dia” -> T\_KODER.V\_TYPE

T\_SKSUBE20XX.IDNUMMER -> T\_KODER.V\_RECNUM

T\_SKSUBE20XX.C\_OPR -> T\_KODER.C\_KODE

T\_SKSUBE20XX.C\_TILOPR -> T\_KODER.C\_TILKODE

T\_SKSUBE20XX.C\_OPRART -> T\_KODER.C\_KODEART

T\_SKSUBE20XX.INDBERETNINGSDATO -> T\_KODER.PDTO

T\_SKSUBE20XX.C\_SGH -> T\_KODER.C\_PSGH

T\_SKSUBE20XX.C\_AFD -> T\_KODER.C\_PAFD

”und” -> T\_KODER.V\_TYPE

T\_SKSOPR20XX: som for T\_SKSUBE20XX, dog

”opr” -> T\_KODER.V\_TYPE

# Særlige forhold

## Personhenførbare data

Data fra konverteringen indeholder personnumre

# Fysiske Datamodeller

Detaljer omkring feltværdier, datatyper og lignende kan ses i SQL skemaerne som ligger på følgende URL: <https://github.com/trifork/HAIBA-minipas-konvertering/tree/master/database>

### Kilde-database

|  |  |
| --- | --- |
| Tabel | Beskrivelse |
| T\_ADM20XX | Indeholder minipas-indlæggelser |
| T\_DIAG20XX | Indeholder minipas-diagnoser |
| T\_SKSOPR20XX | Indeholder minipas-operationer |
| T\_SKSUBE20XX | Indeholder minipas-behandlinger og undersøgelser |
| T\_MINIPAS\_UGL\_STATUS (konfigurerbart tabelnavn) | Indeholder status på sidste eller igangværende data-klargøring fra minipas. |

### Destinations-database

|  |  |
| --- | --- |
| Tabel | Beskrivelse |
| T\_ADM | Indeholder minipas-indlæggelser |
| T\_KODER | Indeholder minipas-diagnoser, operationer, behandlinger og undersøgelser |
| T\_LOG\_SYNC | Indeholder information om seneste processering |
| T\_LOG\_SYNC\_HISTORY | Indeholder information om slettede rækker |

### Meta-database

|  |  |
| --- | --- |
| Tabel | Beskrivelse |
| MinipasImportStatus | Indeholder information om kørsler; start- og sluttid samt udfald. Desuden en kort tekstbesked om evt. fejl. Denne information bruges af statussiden. |
| T\_MINIPAS\_SYNC | Indeholder synkroniseringsmodel mellem kilde- og destinationsdatabase |

# Opsætning af udviklingsmiljø

Opsætningen af udviklingsmiljøet for Medicin-databehandleren forudsætter, at følgende elementer allerede er installeret på udviklerens maskine:

* Java Developer Kit 6.0\_x
* Maven 3.x [MAVEN]
* Git 1.7.x
* MySQL 5.5.x
* Tomcat 7 (Udviklet og testet på version 7.0.34)

Installationsvejledningen indeholder detaljer omkring opsætning af ovenstående komponenter.

## Kildekode

Kildekoden er placeret i et github-repositorie og kan checkes ud på følgende måde:

git clone git@github.com:trifork/HAIBA-Medicinimporter.git

## Byggemiljø

Databehandleren anvender Maven som byggesystem. Strukturen følger de generelle anbefalinger for Maven projekter, og er struktureret

efter Maven layout-konventionen.

For at bygge databehandlersystemet, skal man gøre følgende:

mvn install

Projektet indeholder udover unittests også integrationstests, der kræver en kørende database, for at undlade udførelsen af integrationstests kan parameteren -DskipITs tilføjes mvn install kommandoen.

### Dependencies

For at kunne hente NSI-specifikke afhængigheder (bl.a. nsp-util) i binær form i stedet for at skulle bygge alle afhængigheder selv på det lokale udviklingsmiljø, indeholder pom'en en reference til nexus.trifork.com, som er et artefaktrepository der er placeret hos Trifork. Binære releases af databehandleren findes også i nexus.trifork.com.

Repository’et bør, når det er muligt, udskiftes med et artefaktrepository der er driftet hos NSI. Når et sådant er etableret

## Database-setup

### Database

Databaseskema for prøvesvar mm. ligger i database-folderen.

## Test

Installationen kan verificeres ved at eksekvere databehandlerens test suite.

Testsuiten benytter JUnit og Mockito til test.

Test suiten afvikles ved at udføre følgende kommando i projektroden:

mvn test

Installationen kan yderligere verificeres (Code coverage, kode konventioner o.l.) ved at udføre kommandoen, rapporterne ligger i <projekt rod>/target/site:

mvn verify

## IDE

Databehandleren kan principielt udvikles i enhver Java IDE, der forstår Maven projekters opbygning.

I dette dokument beskrives kort opsætning for to af de pt. mest udbredte Java IDE’er: Eclipse og IntelliJ.

### Eclipse

Eclipse er ikke født med Maven support, og det anbefales derfor, at man installerer m2eclipse inden databehandleren hentes ind i Eclipse, pluginet kan hentes her:

http://www.eclipse.org/m2e/

Herefter importeres projekterne i Eclipse via ”import”:

Alternativt kan man importere projektet ved at udføre følgende kommando:

mvn eclipse:eclipse

Og herefter importere projektet på normal vis i Eclipse.

Kommandoen genererer Eclipse projektfilerne (.project og .classpath) for databehandleren. Denne metode kræver dog, at kommandoen udføres hver gang man ændrer i pom filerne.

## IntelliJ Idea IDE

IntelliJ Idea er født med Maven support, og databehandleren kan derfor direkte importeres. Projektet importeres i IntelliJ ved under ”Create new project” at vælge ”Import project from external model”. Herefter udvælges roden af databehandleren, hvorefter projektet importeres.

Det anbefales i den sammenhæng, at man krydser af i ”Import Maven projects automatically”, hvorefter IntelliJ selv detekterer nye moduler i projektet.

Alternativt kan man importere projektet ved at udføre følgende kommando:

mvn idea:idea

Herefter kan projektet importeres på normal vis i IntelliJ.

*Obs! Denne metode kræver dog, at kommandoen udføres hver gang man ændrer i pom filerne.*

## Distribution

databehandleren kan bygges til distribution eller lokal test ved at udføre:

mvn package

Dette generer en WAR fil, der efterfølgende kan deploye's lokalt eller på et testmiljø.

Til produktion bør Maven release plugin bruges, da det får tag'et bygget og automatisk får opdateret versionsnumre m.v.

Først køres:

mvn release:prepare

Går det godt køres:

mvn release:perform

# Tips og tricks

I det følgende beskrives problemer man som udvikler kan støde på, og forslag til løsning af samme.

## Tomcat out of memory

### Beskrivelse

I Tomcats log, logger den noget i stil med ”out of memory” og nævner “permgenspace”, dette kan ske ved at der hot-deployes ofte, som man typisk gør under udvikling

### Løsning

Forøg Tomcat permgen space ved at sætte følgende miljø variabel:

JAVA\_OPTS="-Xms2048m -Xmx2048m -XX:MaxPermSize=512m"

# Referencer og kilder

|  |  |  |
| --- | --- | --- |
| Reference-id | Indhold / Overskrift | Henvisning |
| [MAVEN] | Welcome to Apache Maven | <http://maven.apache.org/> |

# Ændringslog

Kilden til dette dokument kan findes på:

<https://github.com/trifork/HAIBA-Minipas-konvertering/tree/master/doc>

| Version | Dato | Ændring | Ansvarlig |
| --- | --- | --- | --- |
| 1 | 2014-03-31 | Initielt dokument | Trifork  Aksel Schmidt |
|  |  |  |  |