

SPECIFIKATION AF ELEKTRONISKE PATIENTJOURNALER

Christian Krog
Madsen
c973746

Rasmus Dyhrberg
c973403

Nikolaj Christensen
c973588

18. december 2002

02268 Formelle Aspekter af Software Engineering II
Informatik and Matematisk Modellering
Danmarks Tekniske Universitet

Indhold

1	Introduktion	1
2	Terminologi	2
3	Domænebeskrivelse	3
3.1	Papirbaserede Patientjournaler	4
3.2	RSL specifikation af Papirbaserede Patientjournaler	5
3.3	Førstegenerations Elektroniske Patientjournaler	9
3.4	RSL specifikation af Førstegenerations Elektroniske Patientjournaler	10
4	Kravspecifikation	16
4.1	RSL specifikation af Andengenerations Elektroniske Patientjournaler	18
5	Vurdering af G-EPJ specifikationen	34
5.1	Sundhedsstyrelsens projekt	34
5.2	Systematics projekt	35
5.3	Vurdering	36
6	Udvekslingsformat	37
6.1	Krav til udvekslingsformat	38
6.2	Design af udvekslingsformat	38
6.2.1	Udvidelse af EPJ2	39
6.2.2	Journalregister	47
6.2.3	Brugerkommandoer til V-EPJ	49
6.2.4	Protokol for kommunikation mellem EPJ systemer	52
6.3	V-EPJ grænsesnit	54
6.3.1	XML Schema for udvekslingsformat	62
7	Konklusion	62
A	XML Schema	65

1 Introduktion

Sundhedsområdet er i øjeblikket genstand for en hastig udvikling af næste generation af IT-systemer til registrering og behandling af data. En vigtig del af dette arbejde er udviklingen af den Elektroniske Patientjournal (forkortet EPJ). Når disse systemer bliver fuldt implementeret vil det være muligt at foretage store rationaliseringer i arbejdsprocesserne, i det man i en behandlingssituation vil få umiddelbar adgang til alle tilgængelige data, hvilket samtidigt kan minimere risikoen for fejlbehandlinger.

Der arbejdes i øjeblikket sideløbende med at implementere forskellige EPJ systemer i flere af landets amter. For at realisere de fulde gevinster ved denne elektronisering er det af største vigtighed at de forskellige EPJ systemer kan udveksle data indbyrdes. Derfor har Sundhedsstyrelsen igangsat et arbejde med at definere en fælles ramme og et fælles grænsesnit imellem disse systemer. Dette arbejde er foreløbig udmundet i en specifikation af en grundstruktur (Grundstruktur for Elektroniske Patientjournaler, forkortet G-EPJ [7]), som tjener til at definere en fælles proces og begrebsdannelse for et EPJ system.

Der er også allerede projekter, der arbejder på at implementere EPJ systemer. Software firmaet Systematic har fået til opgave at udvikle et system til Århus amt. Systemet er netop nu i pilotdrift og forventes sat i drift fra starten af marts 2003.

I afsnit 3 tages der udgangspunkt i den eksisterende papirbaserede patientjournal. Der laves en model, der beskriver journalen som en samling af dokumenter. Desuden beskrives de ting der kan gøres med en journal, f.eks. tilføje nye dokumenter, lave notater eller søge efter dokumenter. Dernæst modelleres et førstegenerations EPJ-system, hvor patientjournalens dokumenter principielt bare elektroniseres, dvs. journalerne lagres og tilgås elektronisk, men indholdet er stadig organiseret som ustruktureret fritekst, ligesom i papirjournalen.

Ud fra de to første modeller defineres i afsnit 4 kravene til et andengenerations EPJ-system, hvor data er strukturerede, dvs. større stykker tekst splittes op og tilpasses et fast skema med en række felter. Modellen for systemet er baseret løst på G-EPJ specifikationen.

I afsnit 5 forklarer vi hovedtrækkene i dels Sundhedsstyrelsens G-EPJ model og dels Systematic A/S's DOM model. De to modeller sammenlignes og modellen fra afsnit 4 inddrages.

I andengenerations EPJ systemet haves nu en model, der specificerer elektroniske patientjournaler, på en måde der lægger tæt op af G-EPJ'en. Vi vil dog gerne videreudvikle denne model ved at foretage endnu et par trin i designprocessen.

I afsnit 6 tages udgangspunkt i modellen for andengenerations patientjournaler. Der opstilles en model der beskriver det vi kalder en Virtuel Elektronisk Patientjournal (V-EPJ). V-EPJ gør det muligt at tilgå data der er placeret i et eksternt EPJ system. Dette er nødvendigt i det tilfælde at et centralt nationalt EPJ system ikke er muligt og det derfor er nødvendigt at distribuere systemet ud til de enkelte amter.

Som et sidste trin i design processen udvikles i afsnit 6.3.1 et XML-schema, der skal fungere som et grænsesnit mellem de forskellige EPJ-systemer.

Endelig indeholder afsnit 7 en konklusion.

I formaliseringerne bruges det formelle specifikationsprog RSL [5]. Dette sprog sætter os i stand til præcist at formulere strukturer og processer vha. matematisk logik.

Vi vil gerne takke Pernille Borum, Enhed for Sundhedsinformatik, Sundhedsstyrelsen, for at have afsat tid til at diskutere G-EPJ specifikationen. Vi vil også gerne takke Rikke Drewsen Andersen og Pia Wichmann Madsen, Systematic, for at fortælle om Systematics DOM model og deres erfaringer med udvikling af et andengenerations EPJ system.

2 Terminologi

- **Amtssygehuse.** Et sygehus der er knyttet til et amt.
- **Apotek.** Et udsalg, der indtil for nylig havde monopol på salg af al medicin. Nu gælder dette monopol ikke længere *håndkøbsmedicin*, men kun *receptpligtig medicin*. Der er ikke nogen decideret ejer, men en forpagter, der har rettighed til at drive apoteket.
- **Behandling.** Den process der forløber fra det tidspunkt hvor en patient får stillet en *diagnose* til det tidspunkt hvor *diagnosen* er afhjulpet. En behandling kan enten være en *medicinsk behandling*, *kirurgisk behandling* eller en *psykologisk behandling*.
- **Cpr-nummer.** Et unikt nummer betående af 10 cifre, der indeholder oplysninger om køn og alder. 1 - 2 position angiver personens fødselsdag. 3 - 4 position angiver personens fødselsmåned. 5 - 6 position angiver personens fødselsår, uden århundrede. 7 - 10 position er et løbenummer. Kombinationen af cifrene i positionerne 5, 6 og 7 angiver personens århundrede og 10. position i personnummeret angiver personens køn. Enhver dansker bliver ved fødslen tildelt et cpr-nummer.
- **Den offentlige sektor.** Alle de instanser der drives for skattepenge. Der kan blandt andet nævnes folkeskoler, gymnasier, biblioteker, sygehuse osv.
- **Det offentlige sundhedssystem.** Den del af *den offentlige sektor* der tager sig af sundhed. Omfatter *lægehuse*, *universitetshospitaler*, *amtssygehuse*, *kommunehospitaler*, *praktiserende læger* og *apoteker*.
- **Diagnose.** Før en patient kommer under *behandling* stilles der en diagnose, der fortæller hvad patienten fejler.
- **Dokument.** Ved et dokument forstås i denne rapport et stykke papir der indeholder medicinske oplysninger.
- **Førstegenerations Elektronisk Patientjournal (EPJ1).** En elektronisk *patientjournal*, der i opbygning svarer til en papirbaseret patientjournal.
- **Håndkøbsmedicin.** Medicin der kan købes uden *recept*. Der kan for eksempel nævnes Panodil der ikke er så stærk at der kræves en *recept*.
- **Kirurgisk Behandling.** *Behandling* hvor der foretages operationer.
- **Klinik.** Ved en klinik forstås i denne rapport en institution, der hører under det *offentlige sundhedssystem*.

- **Kliniker.** En kliniker er en person der er ansat på en *klunik* og som har lovlig adgang til *patientjournaler*, fx en læge eller sygeplejerske.
- **Kommunehospital.** Hospital der er knyttet til en kommune.
- **Lægehus.** En institution hvor flere *praktiserende læger* har slået sig sammen og har dannet en fælles *praksis*.
- **Medicinsk behandling.** *Behandling* hvor der indgår brug af medicin.
- **Patientjournal.** En samling af alle medicinske oplysninger vedrørende personen som journalen omhandler.
- **Praksis.** Den institution hvor en patient kommer når han skal undersøges af en *praktiserende læge*.
- **Praktiserende læge.** En læge der har sin egen *praksis*. En *praktiserende læge* har knyttet et antal personer til sin *praksis*. Disse kommer til den pågældende læge når de har brug for sygdomsmæssig vejledning.
- **Psykologisk behandling.** *Behandling* hvor der benyttes psykologi.
- **Recept.** Et stykke papir der udstedes af en læge til en patient. På dette papir skriver lægen under på, at patienten må købe den pågældende medicin på et *apotek*. Dette gøres for at undgå, at stærk medicin falder i forkerte hænder.
- **Receptpligtig medicin.** Medicin hvor der kræves en *recept* for at man kan købe det. Dette gælder som regel kun stærk medicin, der kan være sundhedsfarligt.
- **Resultat.** Det der er opnået når der er udført en *behandling*. Et resultat kan være både positivt og negativt alt efter om den *diagnose* som *behandlingen* skulle afhjælpe rent faktisk blev afhjulpet eller ej.
- **Symptom.** Et symptom er ting der tyder på sygdom. Det kan være feber, smerter, blod i urin osv.
- **Universitetshospital.** Hospital der er knyttet til et universitet.

3 Domænebeskrivelse

I dette afsnit vil domænet, der ligger til grund for denne opgave, blive beskrevet. Normalt vil en domænebeskrivelse blive udarbejdet i tæt samarbejde med personer der har viden indenfor det konkrete fagområde. Det har der desværre kun i begrænset omfang været mulighed for, så det beskrevne domæne er baseret på egne erfaringer samt på sund fornuft. Dette betyder, at der kan forekomme brudstykker i beskrivelsen, der ikke er i overensstemmelse med virkeligheden.

3.1 Papirbaserede Patientjournaler

Når der bliver født et barn i Danmark tildeles barnet en patientjournal (PJ). Denne vil kun være tilgængelig fra den klinik der er tilknyttet fødslen. Bliver personen, som PJ'en tilhører, indlagt på en anden klinik, kan den nye klinik bestille en kopi af PJ'en til eget arkiv. Således kan en person have flere PJ'er.

En PJ er papirbaseret og kan fysisk beskrives som værende en mappe som de kendes fra arkivskabe. På forsiden af mappen findes de vigtigste data omkring personen som journalen tilhører. Her tænkes på cpr-nummer, personens fulde navn, adresse, telefonnummer osv. Ydermere indeholder en PJ et antal dokumenter, der er sorteret efter dato. Disse dokumenter informerer om alle oplysninger angående personen, der er af medicinsk relevans. Dokumenterne er delt op i fire typer, en opdeling der er baseret på det enkelte dokumentets indhold. De fire typer er symptom, diagnose, behandling og resultat. Fælles for alle dokumenter er, at de indeholder oplysninger om hvilken dato dokumentet er udarbejdet samt hvilken kliniker der har udarbejdet det. En kliniker kan være hvilken som helst person der har lovlig adgang til PJ'en, fx læge, sygeplejerske og lægesekretær. Ydermere kan der til hvert dokument tilføjes kommentarer. Ved en kommentar forstås et lille stykke tekst i marginen af et dokument. Ud over disse generelle oplysninger indeholder hver type mere specifikke oplysninger. Således indeholder et symptomdokument en beskrivelse af et symptom. Diagnosedokument indeholder en beskrivelse af diagnosen samt en liste med henvisninger til de symptomer, der ligger til grund for at diagnosen er blevet stillet. Et behandlingsdokument indeholder en beskrivelse af den behandling der er udført, en prioritet der udtrykker hvor meget behandlingen haster samt en liste af henvisninger til de diagnoser som behandlingen har til mål at afhjælpe. Den fjerde og sidste type af dokumenter er resultatdokumenter, der beskriver resultatet af en bestemt behandling, dokumentet refererer derfor til det tilhørende behandlingsdokument.

Alle referencer fra et dokument til et andet sker ved tekstuel henvisning. Det kunne fx være *"behandlingsresultat af 1/7-2002"*. Henvisningen kan så udvides indtil den bliver en unik reference.

Når en person er i kontakt med det offentlige sundhedsvæsen vil der blive tilføjet dokumenter til hans eller hendes PJ. På denne måde holdes der styr på hvad personen er blevet behandlet for igennem årene.

Det er som sagt muligt at oprette en ny PJ og det kan også lade sig gøre at slette en PJ. Dette skal dog først ske når personen som den tilhører er død. Der kan tilføjes et nyt dokument til en PJ, men det er i denne model dog ikke muligt at fjerne et dokument permanent fra en PJ. Istedet kan der, hvis der ikke er tilfredshed med de oplysninger der står i et dokument, tilføjes en kommentar til et dokument.

Søges informationer om en person i en til personen knyttet PJ, kan det gøres på forskellige måder. Der kan søges efter et specifikt dokument ved hjælp af de før omtalte tekstuelle henvisninger. Der kan også søges efter dokumenter ved at skimme den skrevne tekst indtil den ønskede deltekst findes i et dokument. Hvis der for eksempel er interesse for alle dokumenter der omhandler brækkede ben, kræves det at der søges i alle dokumenter, for at der er sikkerhed for at finde alle dokumenter med tekst omhandlende brækkede ben. Endeligt kan der søges i en PJ udfra datoen. Da dokumenterne er sorteret efter dato er det ret lige til at finde alle dokumenter, der er udarbejdet i et bestemt tidsrum. Dette

resulterer i en mængde med nul eller flere dokumenter.

Brugen af PPJ sker ved at der i klinikkens arkiv hentes en ønsket PJ. I denne kan der så søges efter dokumenter eller tilføjes nye dokumenter.

3.2 RSL specifikation af Papirbaserede Patientjournaler

scheme PPJ =

class

type

Klinik' = Person \xrightarrow{m} PJ,

Klinik = { | k : Klinik' • wf_Klinik(k) | },

Klinik' er et map¹ fra personer til patientjournaler.

Klinik er de Klinikker der opfylder et velformethedskriterie.

PJ' :: per : Person dokl : Dokument*,

PJ = { | pj : PJ' • wf_PJ(pj) | },

PJ' (patient journal) består af personoplysninger og en ordnet samling af dokumenter.

PJ er de patientjournaler der opfylder et velformethedskriterie.

Person,
Prioritet,
Dato,
Kliniker,
BehId,

BehId (behandlings-id).

DiagId,

DiagId (diagnose-id).

ResId,

ResId (resultat-id).

SymId,

SymId (symptom-id).

Dokument ::
indh : DokIndhold
did : DokId
dat : Dato
kli : Kliniker
no : Notat,

¹diskret afbildning

Dokument består af dokumentindhold, dokument-id, dato, kliniker og notat.

DokSoegning == mk_dok(Dokument) | tom,

DokSoegning (dokumentsøgning) er det tomme søgeresultat eller et enkelt dokument.

DokIndhold ==
mk_diag(Beskriv, SymId*) |
mk_beh(Beskriv, Prioritet, DiagId*) |
mk_res(Beskriv, BehId) |
mk_sym(Beskriv),

DokIndhold (dokumentindhold) består af et symptom, en diagnose, en behandling, eller et resultat.

DokId = BehId | DiagId | ResId | SymId,

DokId (dokument-id) er en af følgende: symptom-id, diagnose-id, resultat-id, symptom-id.

Beskriv = **Text**,
Notat = **Text**

value

$\geq : \text{Dato} \times \text{Dato} \rightarrow \mathbf{Bool}$,

wf_PJ : PJ' \rightarrow **Bool**

wf_PJ(pj) \equiv

/* Dokumenter er ordnet efter ikke-aftagende dato */

($\forall i, j : \mathbf{Nat} \bullet$

$i < j \wedge \{i, j\} \subseteq \mathbf{inds} \text{ dokl}(pj) \Rightarrow$
 $\text{dat}(\text{dokl}(pj)(i)) \geq \text{dat}(\text{dokl}(pj)(j))) \wedge$

/* Alle henvisninger til dokumenter er gyldige */

($\forall d : \text{Dokument} \bullet$

$d \in \mathbf{elems} \text{ dokl}(pj) \Rightarrow$

let

ids =

case indh(d) **of**

mk_diag(_, idl) \rightarrow **elems** idl,

mk_beh(_, _, idl) \rightarrow **elems** idl,

mk_res(_, id) \rightarrow {id},

mk_sym(_) \rightarrow {}

end

in

($\forall id : \text{DokId} \bullet$

$id \in \mathbf{ids} \Rightarrow$

($\exists d' : \text{Dokument} \bullet$

$$\begin{aligned}
& d' \in \mathbf{elems} \text{ dokl}(pj) \wedge \\
& \text{did}(d') = \text{id}) \\
& \mathbf{end}) \wedge \\
& /* \text{ Alle id'er er unikke */} \\
& (\forall d1, d2 : \text{Dokument} \bullet \\
& \quad d1 \in \mathbf{elems} \text{ dokl}(pj) \wedge \\
& \quad d2 \in \mathbf{elems} \text{ dokl}(pj) \wedge \text{did}(d1) = \text{did}(d2) \Rightarrow \\
& \quad d1 = d2),
\end{aligned}$$

wf_PJ (velformethedskriterie for patientjournal) tager en patientjournal som input og returnerer sand hvis dokumenterne er ordnet efter ikke aftagende dato og alle henvisninger til dokumenter er gyldige.

$$\begin{aligned}
& \text{wf_Klinik} : \text{Klinik}' \rightarrow \mathbf{Bool} \\
& \text{wf_Klinik}(k) \equiv \\
& \quad (\forall p : \text{Person} \bullet p \in \mathbf{dom} k \Rightarrow \text{per}(k(p)) = p),
\end{aligned}$$

wf_Klinik (velformetheds kriterie for klinik) tager en klinik som input og returnerer en boolsk værdi. Denne værdi er sand hvis personen i domænet af klinikken er den samme person som i den patientjournal der bliver henvist til.

$$\begin{aligned}
& \text{kopi} : \text{Klinik} \times \text{Klinik} \times \text{Person} \xrightarrow{\sim} \text{Klinik} \\
& \text{kopi}(kk, kd, p) \equiv kd \cup [p \mapsto \text{kk}(p)] \\
& \mathbf{pre} p \in \mathbf{dom} kk \wedge p \notin \mathbf{dom} kd,
\end{aligned}$$

kopi tager to klinikker og en person som input og kopierer denne persons patientjournal fra den første klinik til den anden.

$$\begin{aligned}
& \text{nyPJ} : \text{Klinik} \times \text{Person} \xrightarrow{\sim} \text{Klinik} \\
& \text{nyPJ}(k, p) \equiv k \cup [p \mapsto (\text{mk_PJ}'(p, \langle \rangle))] \\
& \mathbf{pre} p \notin \mathbf{dom} k,
\end{aligned}$$

nyPJ (ny patientjournal) tager en klinik og en person som input og returnerer den opdaterede klinik med patientjournalen, svarende til personen, inkluderet.

$$\begin{aligned}
& \text{tilfoejDok} : \text{Klinik} \times \text{Person} \times \text{Dokument} \xrightarrow{\sim} \text{Klinik} \\
& \text{tilfoejDok}(k, p, d) \equiv \\
& \quad k \uparrow [p \mapsto (\text{mk_PJ}'(p, \langle d \rangle \wedge \text{dokl}(k(p))))] \\
& \mathbf{pre} p \in \mathbf{dom} k,
\end{aligned}$$

tilfoejDok (tilføj dokument) tager en klinik, en person og et dokument som input og tilføjer dokumentet til starten af dokumentlisten i personens journal i klinikken.

$$\begin{aligned}
& \text{sletPJ} : \text{Person} \times \text{Klinik} \rightarrow \text{Klinik} \\
& \text{sletPJ}(p, k) \equiv k \setminus \{p\},
\end{aligned}$$

sletPJ (slet patientjournal) tager en person og en klinik som input og fjerner personens journal fra klinikken.

soegPJ : Klinik × Person $\xrightarrow{\sim}$ PJ
 soegPJ(k, p) \equiv k(p) **pre** p \in **dom** k,

***soejPJ** (søg efter patientjournal) tager en klinik og en person som input og returnerer patientjournalen for den givne person såfremt den findes i klinikken.*

komDok : Person × DokId × Klinik × Notat \rightarrow Klinik
 komDok(p, i, k, n) \equiv
let dl = dokl(k(p)) **in**
 k †
 [p \mapsto
 mk_PJ'(
 p,
if did(d) = i
then
 mk_Dokument(
 indh(d), did(d), dat(d), kli(d),
 no(d) \wedge n)
else d
end | d **in** dl)]
end,

***komDok** (kommenter dokument) tager en person, et dokument-id, en klinik og et notat som input. Notatet tilføjes til dokumentet svarende til dokument-id for personen på klinikken.*

soegDok : PJ × DokId \rightarrow DokSoegning
 soegDok(pj, di) \equiv
if dokl(pj) = $\langle \rangle$ **then** tom
elseif did(**hd** dokl(pj)) = di **then** mk_dok(**hd** dokl(pj))
else soegDok(mk_PJ'(per(pj), **tl** dokl(pj)), di)
end,

***soegDok** (søg efter dokument) tager en patientjournal og et dokument-id som input og returnerer en dokumentsoegning. Funktionen kører rekursivt igennem og returnerer tom hvis dokument-id ikke findes. Ellers returneres dokumentsoegningen dokumentet svarende til dokument-id.*

soegSpec : PJ × **Text** \rightarrow Dokument-set
 soegSpec(pj, txt) \equiv
 {d |
 d : Dokument •
 d \in dokl(pj) \wedge
case indh(d) **of**
 mk_diag(b, sl) \rightarrow soegibeskriv(b, txt),
 mk_beh(b, p, dl) \rightarrow soegibeskriv(b, txt),

```

        mk_res(b, beh) → soegibeskriv(b, txt),
        mk_sym(b) → soegibeskriv(b, txt)
    end ∨ (no(d) ≠ ⟨⟩ ∧ soegibeskriv(no(d), txt))
},

```

***soegSpec** (søg specifikt) tager en patientjournal og et stykke tekst som input og returnerer mængden af de dokumenter hvor tekststykket indgår i beskrivelsen.*

```

soegibeskriv : Text × Text → Bool
soegibeskriv(bes, txt) ≡
    (∃ a, b, c : Text • a ^ b ^ c = bes ∧ b = txt),

```

***soegibeskriv** (søg i beskrivelse) hjælpefunktion der tager to tekststrengene og undersøger om den anden er indeholdt i den første.*

```

soegDato : PJ × Dato × Dato → Dokument-set
soegDato(pj, d1, d2) ≡
    {d |
        d : Dokument •
            d ∈ elems dokl(pj) ∧ dat(d) ≥ d1 ∧
            d2 ≥ dat(d)}

```

***soegDato** (søg på dato) tager en patientjournal og to datoer og returnerer mængden af dokumenter i patientjournalen, som er oprettet indenfor det tidsrum som de to datoer repræsenterer.*

axiom

```

[Refleksivitet]
    ∀ t : Dato • t ≥ t,
[Anti_symmetri]
    ∀ t1, t2 : Dato • t1 ≥ t2 ∧ t2 ≥ t1 ⇒ t1 = t2,
[Transitivitet]
    ∀ t1, t2, t3 : Dato •
        t1 ≥ t2 ∧ t2 ≥ t3 ⇒ t1 ≥ t3,
[Totalitet]
    ∀ t1, t2 : Dato • t1 ≥ t2 ∨ t2 ≥ t1

```

end

3.3 Førstegenerations Elektroniske Patientjournaler

I afsnit 3.1 blev principperne bag den papirbaserede patientjournal beskrevet. I nogle enkelte amter i Danmark er man gået et skridt videre og har fået ”sat strøm til” patientjournalerne, de såkaldte førstegenerations elektroniske patientjournaler (EPJ1). Hvad dette indebærer vil blive beskrevet i dette afsnit.

Udtrykket "sat strøm til" beskriver meget passende processen ved udviklingen af EPJ1. Den mest markante forskel fra PPJ til EPJ1 er at sidstnævnte er elektroniske. Selve opbygningen af journalerne er stort set uforandret. Forskellen fra PPJ og EPJ1 er hovedsageligt ændringer, der forsøger at udnytte de fordele, der opstår ved at benytte computere. Her tænkes blandt andet på lagring af data, der med lidt omtanke kan simplificere søgninger efter dokumenter.

Der bliver ikke længere knyttet en liste af dokumenter til en PJ, istedet benyttes et map og et unikt id. Herved findes ved bare et enkelt opslag i dette map et bestemt dokument. Herved undgås tidskrævende søgninger gennem alle dokumenter. Soges der derimod efter alle dokumenter der omhandler brækkede ben, er det stadig nødvendigt at lede alle dokumenter igennem. Her kan der kun drages fordel af, at søgning med en computer er langt hurtigere end en manuel søgning.

I modellen der beskriver EPJ1 er det ikke længere muligt at tilføje et notat til et dokument. Istedet findes et antal redigeringsfunktioner til at rette i en til et dokument knyttet beskrivelse.

Selve brugen af EPJ1 sker som sagt ved hjælp af en computer. Via en brugergrænseflade kan en PJ hentes og redigeres.

3.4 RSL specifikation af Førstgenerations Elektroniske Patientjournaler

scheme EPJ1 =

class

type

EPJA' = CPR \overline{m} PJ,

EPJA = { | epj : EPJA' • wf_EPJA(epj) | },

EPJA' (elektronisk patientjournal) er et map fra cpr-numre til patientjournaler.

EPJA er de elektroniske patientjournaler der opfylder et velformethedskriterie.

PJ' :: per : Person dokm : DokId \overline{m} Dokument,

PJ = { | pj : PJ' • wf_PJ(pj) | }

PJ' (patientjournal) består af en person og et dokument map fra dokument-id til dokumenter.

PJ er de patientjournaler der opfylder et velformethedskriterie.

Person = CPR \times Efternavn \times Fornavn \times Andet,

Person består af et cpr-nummer, et efternavn, et fornavn og øvrige personoplysninger.

Kliniker = Person \times KliId,

Kliniker er en person med et kliniker-id.

CPR,

CPR (*cpr-nummer*).

Efternavn,
Fornavn,
Andet,
Prioritet,
Dato,
BehId,

BehId (*behandlings-id*).

DiagId,

DiagId (*diagnose-id*).

ResId,

ResId (*resultat-id*).

SymId,

SymId (*symptom-id*).

KliId,

KliId (*kliniker-id*).

Dokument ::
indh : DokIndhold dat : Dato klid : KliId,

Dokument består af dokument indhold, en dato og et id for den kliniker, som er ansvarlig for dokumentet.

DokIndhold ==
mk_diag(Beskriv, SymId*) |
mk_beh(Beskriv, Prioritet, DiagId*) |
mk_res(Beskriv, BehId) |
mk_sym(Beskriv),

DokIndhold (*dokumentindhold*) er en af følgende: symptom, diagnose, behandling, eller resultat.

DokId = BehId | DiagId | ResId | SymId,

DokId (*dokument-id*) er enten symptom-id, diagnose-id, behandlings-id eller resultat-id.

Beskriv = **Text**

value

$\geq : \text{Dato} \times \text{Dato} \rightarrow \mathbf{Bool}$,

$\text{wf_PJ} : \text{PJ}' \rightarrow \mathbf{Bool}$

$\text{wf_PJ}(\text{pj}) \equiv$

/* Alle henvisninger til dokumenter er gyldige */

$(\forall d : \text{Dokument} \bullet$

$d \in \mathbf{rng} \text{ dokm}(\text{pj}) \Rightarrow$

let

ids =

case $\text{indh}(d)$ **of**

$\text{mk_diag}(_, \text{idl}) \rightarrow \mathbf{elems} \text{ idl}$,

$\text{mk_beh}(_, _, \text{idl}) \rightarrow \mathbf{elems} \text{ idl}$,

$\text{mk_res}(_, \text{id}) \rightarrow \{\text{id}\}$,

$\text{mk_sym}(_) \rightarrow \{\}$

end

in

$\forall \text{id} : \text{DokId} \bullet$

$\text{id} \in \text{ids} \Rightarrow \text{id} \in \mathbf{dom} \text{ dokm}(\text{pj})$

end),

***wf_PJ** (velformethedskriterie for patientjournal) tager en patientjournal som input og returnerer sand hvis alle henvisninger til dokumenter er gyldige.*

$\text{wf_EPJA} : \text{EPJA}' \rightarrow \mathbf{Bool}$

$\text{wf_EPJA}(\text{epj}) \equiv$

$(\forall \text{cpr} : \text{CPR} \bullet$

$\text{cpr} \in \mathbf{dom} \text{ epj} \Rightarrow$

let $(\text{cpr}', \text{en}, \text{fn}, \text{an}) = \text{per}(\text{epj}(\text{cpr}))$ **in**

$\text{cpr} = \text{cpr}'$

end),

***wf_EPJA** (velformethedskriterie for elektronisk patientjournal) tager en elektronisk patientjournal som input og returnerer sand hvis alle de cpr-numre som peger til en patientjournal er identiske med det cpr-nummer, som er anført i patientjournalen.*

$\text{nyPJ} : \text{Person} \times \text{EPJA} \xrightarrow{\sim} \text{EPJA}$

$\text{nyPJ}((\text{c}, \text{e}, \text{f}, \text{a}), \text{epja}) \equiv$

$\text{epja} \uparrow [c \mapsto \text{mk_PJ}'((\text{c}, \text{e}, \text{f}, \text{a}), [])]$

pre $c \notin \mathbf{dom} \text{ epja}$,

***nyPJ** (ny patientjournal) tilføjer en person til en elektronisk patientjournal (EPJ). Funktionen tager en person og en EPJ og returnerer den opdaterede EPJ hvor patientjournalen er tilføjet.*

```

opretDokId : CPR × EPJA  $\xrightarrow{\sim}$  DokId
opretDokId(cpr, e)  $\equiv$ 
  let d : DokId • d  $\notin$  dom dokm(e(cpr)) in d end,

```

opretDokId (opret dokument-id) er en hjælpefunktion der opretter et nyt dokument-id. Tager et cpr-nummer og en EPJ og returnerer et dokument-id som ikke allerede er i EPJ.

```

tilfoejDok : CPR × EPJA × Dokument  $\rightarrow$  EPJA × DokId
tilfoejDok(cpr, epja, dok)  $\equiv$ 
  let did : DokId • opretDokId(cpr, epja) = did in
    (epja †
     [cpr  $\mapsto$ 
      mk_PJ'(
        per(epja(cpr)),
        dokm(epja(cpr)) † [did  $\mapsto$  dok]]), did)
  end,

```

tilfoejDok (tilføj dokument) tilføjer et dokument til en given patients dokumentliste. Tager et cpr-nummer, en EPJ og et dokument som input og returnerer den opdaterede EPJ og det dokument-id svarende til det tilføjede dokument.

```

sletPJ : CPR × EPJA  $\xrightarrow{\sim}$  EPJA
sletPJ(cpr, epja)  $\equiv$  epja \ {cpr}
pre cpr  $\in$  dom epja,

```

sletPJ (slet patientjournal) fjerner en patientjournal fra en EPJ. Tager et cpr-nummer og en EPJ som input og returnerer den opdaterede patientjournal. Det er under forudsætningen af at cpr-nummeret findes i EPJA.

```

soegPJ : CPR × EPJA  $\xrightarrow{\sim}$  PJ
soegPJ(cpr, epja)  $\equiv$  epja(cpr) pre cpr  $\in$  dom epja,

```

soegPJ (søg efter patientjournal) henter en specifik patientjournal fra EPJ ud fra et cpr-nummer. Tager et cpr-nummer og en EPJ som input og returnerer den fundne patientjournal.

```

soegDok : CPR × EPJA × DokId  $\xrightarrow{\sim}$  Dokument
soegDok(cpr, epj, id)  $\equiv$  dokm(epj(cpr))(id)
pre cpr  $\in$  dom epj  $\wedge$  id  $\in$  dom dokm(epj(cpr)),

```

soegDok (søg efter dokument) henter et specifikt dokument i EPJ ud fra et cpr-nummer og et dokument-id. Tager et cpr-nummer, en EPJ og et dokument-id som input og returnerer det fundne dokument.

```

soegibeskriv : Text × Text  $\rightarrow$  Bool
soegibeskriv(bes, txt)  $\equiv$ 
  ( $\exists$  a, b, c : Text • b = txt  $\wedge$  a  $\wedge$  b  $\wedge$  c = bes),

```

soegibeskriv (søg i beskrivelse) hjælpefunktion der tager to tekststreng og undersøger om den anden er indeholdt i den første.

```

soegSpec : CPR × EPJA × Text → Dokument-set
soegSpec(cpr, epj, txt) ≡
  {d |
    d : Dokument •
    d ∈ rng dokm(epj(cpr)) ∧
    (case indh(d) of
      mk_diag(b, _) → soegibeskriv(b, txt),
      mk_beh(b, _, _) → soegibeskriv(b, txt),
      mk_res(b, _) → soegibeskriv(b, txt),
      mk_sym(b) → soegibeskriv(b, txt)
    end)},

```

soegSpec (søg efter specifik streng i dokument) henter de dokumenter hvor en tekststreng indgår i beskrivelsen. Tager et cpr-nummer, en EPJ og en tekststreng som input og returnerer mængden af dokumenter for personen, svarende til cpr-nummeret, hvor tekststrengen indgår i beskrivelsen.

```

soegDato : CPR × EPJA × Dato × Dato → Dokument-set
soegDato(cpr, epj, da1, da2) ≡
  {d |
    d : Dokument •
    d ∈ rng dokm(epj(cpr)) ∧ dat(d) ≥ da1 ∧
    da2 ≥ dat(d)},

```

soegDato (søg på dato) henter samtlige dokumenter for en given person fra en angiven periode. Tager et cpr-nummer, en EPJ og to datoer som input og returnerer samtlige dokumenter for personen, svarende til cpr-nummeret, der daterer til den periode som de to datoer repræsenterer.

```

redDiag :
  CPR × EPJA × DokId × Beskriv × SymId*  $\xrightarrow{\sim}$ 
  EPJA
redDiag(cpr, epja, doki, besk, symidl) ≡
  epja †
  [cpr ↦
    mk_PJ'(
      per(epja(cpr)),
      (dokm(epja(cpr)) †
        [doki ↦
          mk_Dokument(
            mk_diag(besk, symidl),
            dat(dokm(epja(cpr))(doki)),
            klid(dokm(epja(cpr))(doki))])))]
pre cpr ∈ dom epja ∧ doki ∈ dom dokm(epja(cpr)),

```

redDiag (redigér diagnose) overskriver et diagnosedokument i EPJ med et nyt oprettet udfra angivne værdier. Funktionen tager et cpr-nummer, en EPJ, et dokument-id, en beskrivelse og en liste af symptom-id'er som input. Funktionen returnerer EPJ hvor dokumentet, givet ved dokument-id, er overskrevet med et oprettet dokument med den nye beskrivelse og den nye liste af symptom-id'er.

```

redBeh :
  CPR × EPJA × DokId × Beskriv × Prioritet ×
  DiagId*  $\rightsquigarrow$ 
  EPJA
redBeh(cpr, epja, doki, besk, prior, diagidl)  $\equiv$ 
  epja †
  [ cpr  $\mapsto$ 
    mk_PJ'(
      per(epja(cpr)),
      (dokm(epja(cpr)) †
        [ doki  $\mapsto$ 
          mk_Dokument(
            mk_beh(besk, prior, diagidl),
            dat(dokm(epja(cpr))(doki)),
            klid(dokm(epja(cpr))(doki))))))]
  ]
pre cpr  $\in$  dom epja  $\wedge$  doki  $\in$  dom dokm(epja(cpr)),

```

redBeh (redigér behandling) overskriver et behandlingsdokument i EPJ med et nyt oprettet udfra angivne værdier. Funktionen tager et cpr-nummer, en EPJ, et dokument-id, en beskrivelse, en prioritet og en liste af diagnose-id'er som input. Funktionen returnerer EPJ hvor dokumentet, givet ved dokument-id, er overskrevet med et oprettet dokument med den nye beskrivelse og den nye liste af diagnose-id'er.

```

redRes :
  CPR × EPJA × DokId × Beskriv × BehId  $\rightsquigarrow$  EPJA
redRes(cpr, epja, doki, besk, behid)  $\equiv$ 
  epja †
  [ cpr  $\mapsto$ 
    mk_PJ'(
      per(epja(cpr)),
      (dokm(epja(cpr)) †
        [ doki  $\mapsto$ 
          mk_Dokument(
            mk_res(besk, behid),
            dat(dokm(epja(cpr))(doki)),
            klid(dokm(epja(cpr))(doki))))))]
  ]
pre cpr  $\in$  dom epja  $\wedge$  doki  $\in$  dom dokm(epja(cpr)),

```

redRes (redigér resultat) overskriver et resultatdokument i EPJ med et nyt oprettet udfra angivne værdier. Funktionen tager et cpr-nummer, en EPJ, et dokument-id, en beskrivelse

og et behandlings-id som input. Funktionen returnerer EPJ hvor dokumentet, givet ved dokument-id, er overskrevet med et oprettet dokument med den nye beskrivelse og det nye behandlings-id'er.

```

redSym : CPR × EPJA × DokId × Beskriv  $\rightsquigarrow$  EPJA
redSym(cpr, epja, doki, besk)  $\equiv$ 
  epja †
  [ cpr  $\mapsto$ 
    mk_PJ'(
      per(epja(cpr)),
      (dokm(epja(cpr)) †
        [ doki  $\mapsto$ 
          mk_Dokument(
            mk_sym(besk), dat(dokm(epja(cpr))(doki)),
            klid(dokm(epja(cpr))(doki))))))]
  ]
pre cpr  $\in$  dom epja  $\wedge$  doki  $\in$  dom dokm(epja(cpr))

```

redSym (redigér symptom) overskriver et symptomdokument i EPJ med et nyt oprettet ud fra angivne værdier. Funktionen tager et cpr-nummer, en EPJ, et dokument-id og en beskrivelse som input. Funktionen returnerer EPJ hvor dokumentet, givet ved dokument-id, er overskrevet med et oprettet dokument med den nye beskrivelse.

axiom

```

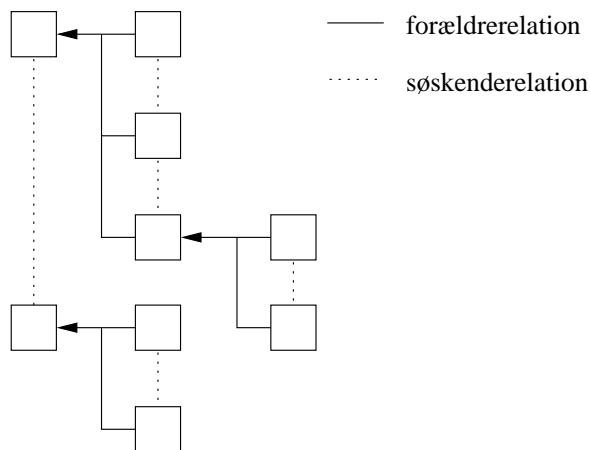
[ Refleksivitet ]
   $\forall t : \text{Dato} \bullet t \geq t,$ 
[ Anti_symmetri ]
   $\forall t1, t2 : \text{Dato} \bullet t1 \geq t2 \wedge t2 \geq t1 \Rightarrow t1 = t2,$ 
[ Transitivitet ]
   $\forall t1, t2, t3 : \text{Dato} \bullet$ 
     $t1 \geq t2 \wedge t2 \geq t3 \Rightarrow t1 \geq t3,$ 
[ Totalitet ]
   $\forall t1, t2 : \text{Dato} \bullet t1 \geq t2 \vee t2 \geq t1$ 
end

```

4 Kravspecifikation

Målet er at udvikle et struktureret elektronisk patientjournal (EPJ) system. I stedet for at være kontaktbaseret som de eksisterende papirsystemer og førstegenerationssystemer, er hensigten at den strukturerede model er forløbsbaseret. Et forløb er en periode med samme sygdom, det vil sige der er muligvis tale om flere hospitalsbesøg og lægebesøg. Det vil sige at en patientjournal skal bestå af personinformation som omtalt i domænebeskrivelsen (afsnit 3) samt et antal forløb sorteret efter et unikt datalogisk id. Et forløb skal indeholde en beskrivelse, som er tiltænkt klinikerne til kommentering. Ydermere bliver forløbet delt op i fire informationselementer svarende til symptomer, diagnoser, behandlinger og resultater.

Diagnoser og behandlinger bliver sorteret i et hierarki der minder om en træstruktur, se fig. 1. Hvert af disse to informationselementer skal således indeholde en liste af noder sorteret efter et unikt datalogisk node id. Noder skal bestå af både data og henvisninger. Symptom- og resultatinformationselementer er blot en samling data.



Figur 1: Hierarkisk struktur, som diagnoser og behandlinger organiseres i.

Data er beskrivelse, status, dato, ansvarlig og et grundlag. Beskrivelsen er en fritekstbeskrivelse af diagnosen. Ansvarlig er den kliniker som er ansvarlig for diagnosen og grundlag er de symptomer, som ligger til grund for den stillede diagnose. For diagnoser er status åben eller lukket og for behandlinger er status enten planlagt, udført, igang eller evalueret. I hver diagnose- og behandlingsnode vil der være henvisninger til en mulig forælder og til søskende i hierarkiet. Dette skal overtage rollen for prioritet, da de forskellige elementer opnår den rette ordning.

For informationselementerne resultat og symptomer er en hierarkisk struktur ikke nødvendig, da resultater er tilknyttet til en specifik behandling og symptomer ikke er sammenhængende. Således skal en resultatnode bestå af beskrivelse, grundlag, som er den tilknyttede behandling, dato og en ansvarlig. Symptomnode består kun af en beskrivelse.

Det skal være muligt at ændre i patientjournalen på flere punkter, ligesom effektiv søgefunktionalitet skal være tilstede. Det skal være muligt at starte et nyt forløb, samt at tilføje informationselementer af hver af de fire slags. Ydermere skal det være muligt at finde en patientjournal for en given person via cpr-nummeret. Det skal være muligt ud fra cpr-nummeret at finde de forløb hvor en given tekststreng indgår i beskrivelsen på de enkelte informationselementer. Det kan være relevant for en kliniker at få en liste af id for diagnoser, behandlinger, resultater eller symptomer i et tidsrum. Forløb skal kunne findes ud fra id og cpr-nummeret på patienten og skal kunne lukkes. Hvis et forløb er lukket betragtes sygdommene som helbredt på tilfredsstillende vis, dvs. når en læge har færdigbehandlet patienten. Samtlige aktive forløb skal kunne findes via et cpr-nummer. Det skal også være muligt at opdatere status på diagnoser og behandlinger.

4.1 RSL specifikation af Andengenerations Elektroniske Patientjournaler

```

scheme EPJ2 =
  class
    type
      EPJA' = CPR  $\overline{m}$  PJ,
      EPJA = { | epj : EPJA' • wf_EPJA(epj) | },

```

EPJA' (elektronisk patientjournal) er et map fra cpr-numre til patientjournaler.
EPJA er de elektroniske patientjournaler der opfylder et velformethedskriterie.

```

PJ' :: per : Person forl : ForId  $\overline{m}$  Forloeb,
PJ = { | pj : PJ' • wf_PJ(pj) | },

```

PJ' (patientjournal) består af en person og et map fra forløbs-id til forløb.
PJ er de patientjournaler der opfylder et velformethedskriterie.

Person = CPR × Efternavn × Fornavn × Andet,

Person består af et cpr-nummer, et efternavn, et fornavn og ekstra information.

```

Forloeb ::
  beskfor : Beskriv
  diaghi : InfElemDiag  $\leftrightarrow$  recon_diag
  behhi : InfElemBeh  $\leftrightarrow$  recon_beh
  res : InfElemRes  $\leftrightarrow$  recon_res
  sym : InfElemSym  $\leftrightarrow$  recon_sym
  forsta : ForStatus  $\leftrightarrow$  recon_forsta,

```

Forloeb (forløb) består af en beskrivelse, et symptominformationselement, et diagnosehierarki, et behandlingshierarki, resultatinformationselement og en status på forløbet.

InfElemDiag = DiagNodeId \overline{m} DiagNode,

InfElemDiag (diagnoseinformationselement) er et map fra diagnosenode-id til diagnose-noder.

InfElemBeh = BehNodeId \overline{m} BehNode,

InfElemBeh (behandlingsinformationselement) er et map fra behandlingsnode-id til behandlingsnoder.

InfElemRes = ResNodeId \overline{m} ResNode,

InfElemRes (resultatinformationselement) er et map fra resultatnode-id til resultatnoder.

InfElemSym = SymNodeId \overline{m} SymNode,

InfElemSym (*symptominformationselement*) er et map fra symptomnode-id til symptomnoder.

```
DiagNode ::  
  besk : Beskriv  
  stat : DiagStatus ↔ recon_sta  
  dat : Dato  
  ansvarlig : KliId  
  grundlag : SymNodeId*  
  fm : OptDiagNodeId  
  soesk : DiagNodeId* ↔ recon_soesk,
```

DiagNode (*diagnosenode*) består af en beskrivelse, en status, en dato, en ansvarlig kliniker, en liste af symptomnode-id som danner grundlag for diagnosen, et muligt diagnosenode-id på forælder i træstrukturen og en liste af diagnosenode-id, som søskende i træstrukturen.

```
BehNode ::  
  besk : Beskriv  
  stat : BehStatus ↔ recon_sta  
  dat : Dato  
  ansvarlig : KliId  
  grundlag : DiagNodeId*  
  fm : OptBehNodeId  
  soesk : BehNodeId* ↔ recon_soesk,
```

BehNode (*behandlingsnode*) består af en beskrivelse, en status, en dato, en ansvarlig kliniker, en liste af diagnosenode-id som danner grundlag for behandlingen, et muligt behandlingsnode-id på forælder i træstrukturen og en liste af behandlingsnode-id, som søskende i træstrukturen.

```
ResNode ::  
  besk : Beskriv  
  grundlag : BehNodeId  
  dat : Dato  
  ansvarlig : KliId,
```

ResNode (*resultatnode*) består af en beskrivelse, et behandlingsnode-id som grundlag for resultatet, en dato og en ansvarlig kliniker.

```
SymNode ::  
  besk : Beskriv  dat : Dato  ansvarlig : KliId,
```

SymNode (*symptomnode*) består af en beskrivelse, en dato og en ansvarlig kliniker.

```
OptDiagNodeId == mk_dId(DiagNodeId) | nil1,
```

OptDiagNodeId (*optionelt diagnosenode-id*) er enten et diagnosenode-id oprettet eller tom.

OptBehNodeId == mk_bId(BehNodeId) | nil2,

***OptBehNodeId** (optionelt behandlingsnode-id) er enten et behandlingsnode-id oprettet eller tom.*

DiagStatus == aaben | lukket,

***DiagStatus** (diagnosestatus) kan antage en af værdierne: åben eller lukket.*

BehStatus == planlagt | udfoert | igang | evalueret,

***BehStatus** (behandlingsstatus) kan antage en af værdierne: planlagt, udført, igang eller evalueret.*

ForStatus == aaben | lukket,

***ForStatus** (forløbsstatus) kan antage en af værdierne: åben eller lukket.*

CPR,
ForId,
Beskriv = **Text**,
DiagNodeId,
BehNodeId,
ResNodeId,
SymNodeId,
Dato,
KliId,
Efternavn,
Fornavn,
Andet

value

$\geq : \text{Dato} \times \text{Dato} \rightarrow \mathbf{Bool}$,

$\text{wf_EPJA} : \text{EPJA}' \rightarrow \mathbf{Bool}$

$\text{wf_EPJA}(\text{epj}) \equiv$

$(\forall \text{cpr} : \text{CPR} \bullet$

$\text{cpr} \in \mathbf{dom} \text{ epj} \Rightarrow$

$\mathbf{let} (\text{cpr}', \text{en}, \text{fn}, \text{an}) = \text{per}(\text{epj}(\text{cpr})) \mathbf{in}$

$\text{cpr} = \text{cpr}'$

$\mathbf{end})$,

***wf_EPJA** (velformeteds-kriterie for elektronisk patientjournal) tager en elektronisk patientjournal som input og returnerer sand hvis alle de cpr-numre som peger til en patientjournal er identiske med det fra personen i patientjournalen.*

```

wf_PJ : PJ' → Bool
wf_PJ(pj) ≡
  /* 1. alle id'er peger rigtigt */
  (∀ forloeb : Forloeb •
    forloeb ∈ rng forl(pj) ⇒
      (∀ dn : DiagNode •
        dn ∈ rng diaghi(forloeb) ⇒
          elems grundlag(dn) ⊆ dom sym(forloeb) ∧
          elems soesk(dn) ⊆ dom diaghi(forloeb) ∧
          case fm(dn) of
            nil1 → true,
            mk_dId(did) →
              did ∈ dom diaghi(forloeb)
          end) ∧
        (∀ bn : BehNode •
          bn ∈ rng behhi(forloeb) ⇒
            elems grundlag(bn) ⊆
              dom diaghi(forloeb) ∧
            elems soesk(bn) ⊆ dom behhi(forloeb) ∧
            case fm(bn) of
              nil2 → true,
              mk_bId(bid) →
                bid ∈ dom behhi(forloeb)
            end) ∧
        (∀ rn : ResNode •
          rn ∈ rng res(forloeb) ⇒
            grundlag(rn) ∈ dom behhi(forloeb))) ∧
      /* 2. diagnosehierarkiet er et trae */
      (∀ dn : DiagNode •
        dn ∈ rng diaghi(forloeb) ⇒
          (∀ did : DiagNodeId •
            did ∈ elems soesk(dn) ⇒
              soesk(dn) =
                soesk(diaghi(forloeb)(did)) ∧
                fm(dn) = fm(diaghi(forloeb)(did)))) ∧
          ~ (∃ nl : DiagNode* •
            len nl > 0 ∧
            (∀ i : Nat •
              i ∈ inds nl \ {1} ⇒
                case fm(nl(i)) of
                  nil1 → false,
                  mk_dId(did) →
                    diaghi(forloeb)(did) = nl(i - 1)
                end) ∧
            ))

```

```

case fm(nl(1)) of
  nil1 → false,
  mk_dId(did) →
    diaghi(forloeb)(did) = nl(len nl)
end) ∧
/* 3. behandlingshierarkiet er et trae */
(∀ bn : BehNode •
  bn ∈ rng behhi(forloeb) ⇒
  (∀ bid : BehNodeId •
    bid ∈ elems soesk(bn) ⇒
    soesk(bn) =
      soesk(behhi(forloeb)(bid)) ∧
      fm(bn) = fm(behhi(forloeb)(bid))) ∧
  ~ (∃ nl : BehNode* •
    len nl > 0 ∧
    (∀ i : Nat •
      i ∈ inds nl \ {1} ⇒
      case fm(nl(i)) of
        nil2 → false,
        mk_bId(bid) →
          behhi(forloeb)(bid) = nl(i - 1)
      end) ∧
      case fm(nl(1)) of
        nil2 → false,
        mk_bId(bid) →
          behhi(forloeb)(bid) = nl(len nl)
      end)),

```

***wf_PJ** (velformethedskriterie for patientjournal) tager en patientjournal som input og returnerer sand hvis alle id i patientjournalen peger rigtigt og begge hierarkier i forløbene i forløbsmappet i patientjournalen har den rette træstruktur.*

```

nyPJ : Person × EPJA → EPJA
nyPJ((cpr, e, f, a), epj) ≡
  epj ∪ [cpr ↦ mk_PJ'((cpr, e, f, a), [])],

```

***nyPJ** (ny patientjournal) tager et cpr-nummer, et efternavn, et fornavn, andet og en EPJ som input og returnerer den opdaterede klinik, hvor der er tilføjet en tom patientjournal for den givne person.*

```

opretForId : EPJA → ForId
opretForId(epj) ≡
  let
    fid : ForId •
      ∀ pj : PJ •
        pj ∈ rng epj ⇒ fid ∉ dom forl(pj)

```

```

in
  fid
end,

```

opretForId (*opret forløbs-id*) opretter et nyt forløbs-id. Tager en EPJ som input og returnerer et forløbs-id hvor om det gælder, at det ikke allerede findes i forløbsmappet i EPJ.

```

nytForloeb : Beskriv × CPR × EPJA  $\xrightarrow{\sim}$  EPJA × ForId
nytForloeb(b, cpr, epj)  $\equiv$ 
  let pj = epj(cpr), fid = opretForId(epj) in
    (epj †
     [ cpr  $\mapsto$ 
       mk_PJ'(
         per(pj),
         forl(pj)  $\cup$ 
         [ fid  $\mapsto$  mk_Forloeb(b, [], [], [], [], aaben) ])
     ], fid)
  end
pre cpr  $\in$  dom epj,

```

nytForloeb (*opret nyt forløb*) tilføjer et nyt forløb til EPJ. Tager en beskrivelse, et cpr-nummer og en EPJ og returnerer den opdaterede EPJ og forløbs-id for det tilføjede forløb.

```

opretDiagNodeId :
  (DiagNodeId  $\xrightarrow{\overline{m}}$  DiagNode)  $\rightarrow$  DiagNodeId
opretDiagNodeId(dm)  $\equiv$ 
  let did : DiagNodeId • did  $\notin$  dom dm in did end,

```

opretDiagNodeId (*opret diagnosenode-id*) hjælpefunktion der tager et diagnosehierarki og danner et nyt diagnosenode-id ud fra dette.

```

indsSoeskDiag :
  DiagNodeId* × OptDiagNodeId × DiagNodeId  $\rightarrow$ 
  DiagNodeId*
indsSoeskDiag(sl, ssoesk, did)  $\equiv$ 
  case ssoesk of
    nil1  $\rightarrow$   $\langle$ did $\rangle$   $\wedge$  sl,
    mk_dId(sid)  $\rightarrow$ 
      if hd sl = sid then  $\langle$ sid, did $\rangle$   $\wedge$  sl
      else
         $\langle$ hd sl $\rangle$   $\wedge$  indsSoeskDiag(tl sl, ssoesk, did)
      end
  end,

```

```

tilfoejDiag :

```

```

Beskriv × DiagStatus × Dato × KliId ×
SymNodeId* × OptDiagNodeId × OptDiagNodeId ×
EPJA × CPR × ForId  $\xrightarrow{\sim}$ 
  EPJA × DiagNodeId
tilfoejDiag(
  b, s, d, a, sidl, foraelder, ssoesk, epj, cpr, fid)  $\equiv$ 
let
  pj = epj(cpr),
  forloeb = forl(pj)(fid),
  diag = diaghi(forloeb),
  did = opretDiagNodeId(diag),
  nydiag =
    diag †
    [p  $\mapsto$ 
      recon_soesk(
        indsSoeskDiag(soesk(diag(p)), ssoesk, did),
        diag(p)) |
      p : DiagNodeId •
      p  $\in$  dom diag  $\wedge$  fm(diag(p)) = foraelder]  $\cup$ 
    [did  $\mapsto$ 
      mk_DiagNode(
        b, s, d, a, sidl, foraelder,
        let
          p : DiagNodeId •
          p  $\in$  dom diag  $\wedge$ 
          fm(diag(p)) = foraelder
        in
          indsSoeskDiag(soesk(diag(p)), ssoesk, did)
        end)]
in
  (epj †
  [cpr  $\mapsto$ 
    mk_PJ'(
      per(pj),
      forl(pj) †
      [fid  $\mapsto$  recon_diag(nydiag, forloeb)]), did)
end
pre
  cpr  $\in$  epj  $\wedge$  fid  $\in$  dom forl(epj(cpr))  $\wedge$ 
case foraelder of
  nil1  $\rightarrow$  true,
  mk_dId(pid)  $\rightarrow$ 
    pid  $\in$  dom diaghi(forl(epj(cpr))(fid))
end  $\wedge$ 

```

```

case ssoesk of
  nil1  $\rightarrow$  true,
  mk_dId(sid)  $\rightarrow$ 
    fm(diaghi(forl(epj(cpr))(fid))(sid)) = foraelder
end,

```

***tilfoejDiag** (tilføj diagnose) tilføjer en diagnose til et diagnosehierarki. Tager al diagnose-nodedata; beskrivelse, diagnosestatus, dato, kliniker-id, symptomnode-id liste, optionelt diagnosenode-id samt en EPJ, et cpr-nummer og et forløbs-id som input. Funktionen returnerer den opdaterede EPJ og diagnosnode-id på den tilføjede diagnose.*

```

opretBehNodeId : (BehNodeId  $\xrightarrow{m}$  BehNode)  $\rightarrow$  BehNodeId
opretBehNodeId(bm)  $\equiv$ 
  let bid : BehNodeId • bid  $\notin$  dom bm in bid end,

```

***opretBehNodeId** (opret behandlingsnode-id) hjælpefunktion der tager et behandlingshierarki og danner et nyt behandlingsnode-id ud fra dette.*

```

indsSoeskBeh :
  BehNodeId*  $\times$  OptBehNodeId  $\times$  BehNodeId  $\rightarrow$ 
  BehNodeId*
indsSoeskBeh(sl, ssoesk, bid)  $\equiv$ 
  case ssoesk of
    nil2  $\rightarrow$   $\langle$ bid $\rangle \wedge$  sl,
    mk_bId(sid)  $\rightarrow$ 
      if hd sl = sid then  $\langle$ sid, bid $\rangle \wedge$  sl
      else
         $\langle$ hd sl $\rangle \wedge$  indsSoeskBeh(tl sl, ssoesk, bid)
      end
  end,

```

```

tilfoejBeh :
  Beskriv  $\times$  BehStatus  $\times$  Dato  $\times$  KliId  $\times$ 
  DiagNodeId*  $\times$  OptBehNodeId  $\times$  OptBehNodeId  $\times$ 
  EPJA  $\times$  CPR  $\times$  ForId  $\xrightarrow{\sim}$ 
  EPJA  $\times$  BehNodeId
tilfoejBeh(
  b, s, d, a, didl, foraelder, ssoesk, epj, cpr, fid)  $\equiv$ 
let
  pj = epj(cpr),
  forloeb = forl(pj)(fid),
  beh = behhi(forloeb),
  bid = opretBehNodeId(beh),
  nybeh =
    beh  $\dagger$ 
    [p  $\mapsto$ 

```

```

    recon_soesk(
      indsSoeskBeh(soesk(beh(p)), ssoesk, bid),
      beh(p)) |
    p : BehNodeId •
    p ∈ dom beh ∧ fm(beh(p)) = foraelder ] ∪
  [bid ↦
    mk_BehNode(
      b, s, d, a, didl, foraelder,
      let
        p : BehNodeId •
        p ∈ dom beh ∧ fm(beh(p)) = foraelder
      in
        indsSoeskBeh(soesk(beh(p)), ssoesk, bid)
      end)]
in
  (epj †
  [cpr ↦
    mk_PJ'(
      per(pj),
      forl(pj) † [fid ↦ recon_beh(nybeh, forloeb)]
    )], bid)
end
pre
  cpr ∈ epj ∧ fid ∈ dom forl(epj(cpr)) ∧
  case foraelder of
    nil2 → true,
    mk_bId(pid) →
      pid ∈ dom behhi(forl(epj(cpr))(fid))
  end ∧
  case ssoesk of
    nil2 → true,
    mk_bId(sid) →
      fm(behhi(forl(epj(cpr))(fid))(sid)) = foraelder
  end,

```

***tilfoejBeh** (tilføj behandling) tilføjer en behandling til et behandlingshierarki. Tager al behandlingsnodedata; beskrivelse, behandlingsstatus, dato, kliniker-id, behandlingsnode-id liste, optionelt behandlingsnode-id samt en EPJ, et cpr-nummer og et forløbs-id som input. Funktionen returnerer den opdaterede EPJ og behandlingsnode-id på den tilføjede behandling.*

```

opretSymNodeId : (SymNodeId  $\overline{m}$  SymNode) → SymNodeId
opretSymNodeId(sm) ≡
  let sid : SymNodeId • sid ∉ dom sm in sid end,

```

opretSymNodeId (opret symptomnode-id) hjælpefunktion der tager et symptominformationselement og danner et nyt symptomnode-id ud fra dette.

```

tilfoejSym :
  Beskriv × Dato × KliId × EPJA × CPR × ForId  $\xrightarrow{\sim}$ 
  EPJA × SymNodeId
tilfoejSym(b, d, a, epj, cpr, fid)  $\equiv$ 
  let
    pj = epj(cpr),
    forloeb = forl(pj)(fid),
    sym = sym(forloeb),
    sid = opretSymNodeId(sym),
    nysym = sym † [sid  $\mapsto$  mk_SymNode(b, d, a)]
  in
    (epj †
     [cpr  $\mapsto$ 
      mk_PJ'(
        per(pj),
        forl(pj) † [fid  $\mapsto$  recon_sym(nysym, forloeb)]
      )], sid)
  end
pre cpr  $\in$  dom epj  $\wedge$  fid  $\in$  dom forl(epj(cpr)),

```

tilfoejSym (tilføj symptom) tilføjer et symptom til symptominformationselementet. Tager en beskrivelse, en dato, et kliniker-id, en EPJ, et cpr-nummer og et forløbs-id som input. Funktionen returnerer den opdaterede EPJ og symptomnode-id på den nye symptomnode.

```

opretResNodeId :
  (ResNodeId  $\xrightarrow{m}$  ResNode)  $\xrightarrow{\sim}$  ResNodeId
opretResNodeId(rm)  $\equiv$ 
  let rid : ResNodeId • rid  $\notin$  dom rm in rid end,

```

opretResNodeId (opret resultatnode-id) tager et resultatinformationselement og danner et nyt resultatnode-id ud fra dette.

```

tilfoejRes :
  Beskriv × BehNodeId × Dato × KliId × EPJA ×
  CPR × ForId  $\rightarrow$ 
  EPJA × ResNodeId
tilfoejRes(b, bid, d, a, epj, cpr, fid)  $\equiv$ 
  let
    pj = epj(cpr),
    forloeb = forl(pj)(fid),
    res = res(forloeb),
    rid = opretResNodeId(res),
    nyres = res † [rid  $\mapsto$  mk_ResNode(b, bid, d, a)]

```

```

in
  (epj †
   [cpr ↦
    mk_PJ'(
      per(pj),
      forl(pj) † [fid ↦ recon_res(nyres, forloeb)]
    )], rid)
end
pre cpr ∈ dom epj ∧ fid ∈ dom forl(epj(cpr)),

```

***tilfoejRes** (tilføj resultat) tilføjer et resultat til resultatinformationselementet. Tager en beskrivelse, et behandlingsnode-id, en dato, et kliniker-id, en EPJ, et cpr-nummer og et forløbs-id som input. Funktionen returnerer den opdaterede EPJ og resultatnode-id på den nye resultatnode.*

```

soegPJ : CPR × EPJA  $\xrightarrow{\sim}$  PJ
soegPJ(cpr, epj) ≡ epj(cpr) pre cpr ∈ dom epj,

```

***soegPJ** (søg efter patientjournal) henter en specifik patientjournal fra EPJ udfra et cpr-nummer. Tager et cpr-nummer og en EPJ som input og returnerer den fundne patientjournal.*

```

soegibeskriv : Text × Text → Bool
soegibeskriv(bes, txt) ≡
  (∃ a, b, c : Text • b = txt ∧ a ^ b ^ c = bes),

```

***soegibeskriv** (søg i beskrivelse) hjælpefunktion der tager to tekststrengene og undersøger, om den anden er indeholdt i den første.*

```

soegBeskriv :
  Text × CPR × EPJA  $\xrightarrow{\sim}$ 
  (ForId  $\vec{m}$ 
   (DiagNodeId-set × BehNodeId-set ×
    SymNodeId-set × ResNodeId-set))
soegBeskriv(txt, cpr, epj) ≡
  let pj = epj(cpr) in
  [fid ↦
   ({did |
    did : DiagNodeId •
     let diag = diaghi(forl(pj))(fid) in
     did ∈ diag ∧
     soegibeskriv(besk(diag(did)), txt)
    } end),
   {bid |
    bid : BehNodeId •
     let beh = behhi(forl(pj))(fid) in

```

```

        bid ∈ beh ∧
        soegibeskriv(besk(beh(bid)), txt)
    end},
    {sid |
    sid : SymNodeId •
        let sym = sym(forl(pj)(fid)) in
        sid ∈ sym ∧
        soegibeskriv(besk(sym(sid)), txt)
    end},
    {rid |
    rid : ResNodeId •
        let res = res(forl(pj)(fid)) in
        rid ∈ res ∧
        soegibeskriv(besk(res(rid)), txt)
    end} | fid : ForId • fid ∈ dom forl(pj)]
end
pre cpr ∈ dom epj,

```

***soegBeskriv** (søg efter beskrivelse) henter de dele af de forløb hvor en tekststreng indgår i beskrivelsen af informationselementerne. Tager en tekststreng, et cpr-nummer og en EPJ som input og returnerer et map fra forløbs-id til de dele af forløbene hvor tekststrengen indgår.*

```

diagSoegDato :
    Dato × Dato × EPJA × CPR  $\rightsquigarrow$  DiagNodeId-set
diagSoegDato(dst, dsl, epj, cpr)  $\equiv$ 
    let pj = epj(cpr) in
    {did |
    did : DiagNodeId •
        (∃ fid : ForId •
            fid ∈ dom forl(pj) ∧
            let diag = diaghi(forl(pj)(fid)) in
            did ∈ dom diag ∧
            dat(diag(did)) ≥ dst ∧
            dsl ≥ dat(diag(did))
        end)}
    end
pre cpr ∈ dom epj,

```

***diagSoegDato** (søg diagnoser efter dato) finder de diagnosenode-id'er for en given person i EPJ som er oprettet indenfor et bestemt tidsrum. Tager to datoer, en EPJ og et cpr-nummer som input og returnerer en mængde af diagnosenode-id'er, som er i det tidsrum som de to datoer repræsenterer.*

```

behSoegDato :
    Dato × Dato × EPJA × CPR  $\rightsquigarrow$  BehNodeId-set

```

```

behSoegDato(dst, dsl, epj, cpr) ≡
  let pj = epj(cpr) in
  {bid |
    bid : BehNodeId •
    (∃ fid : ForId •
      fid ∈ dom forl(pj) ∧
      let beh = behhi(forl(pj)(fid)) in
      bid ∈ dom beh ∧
      dat(beh(bid)) ≥ dst ∧
      dsl ≥ dat(beh(bid))
    end)}
  end
pre cpr ∈ dom epj,

```

behSoegDato (søg behandlinger efter dato) finder de behandlingsnode-id'er for en given person i EPJ som er oprettet indenfor et bestemt tidsrum. Tager to datoer, en EPJ og et cpr-nummer som input og returnerer en mængde af behandlingsnode-id'er, som er i det tidsrum som de to datoer repræsenterer.

```

symSoegDato :
  Dato × Dato × EPJA × CPR  $\xrightarrow{\sim}$  SymNodeId-set
symSoegDato(dst, dsl, epj, cpr) ≡
  let pj = epj(cpr) in
  {sid |
    sid : SymNodeId •
    (∃ fid : ForId •
      fid ∈ dom forl(pj) ∧
      let sym = sym(forl(pj)(fid)) in
      sid ∈ dom sym ∧
      dat(sym(sid)) ≥ dst ∧
      dsl ≥ dat(sym(sid))
    end)}
  end
pre cpr ∈ dom epj,

```

symSoegDato (søg symptomer efter dato) finder de symptomnode-id'er for en given person i EPJ som er oprettet indenfor et bestemt tidsrum. Tager to datoer, en EPJ og et cpr-nummer som input og returnerer en mængde af symptomnode-id'er, som er i det tidsrum som de to datoer udgør.

```

resSoegDato :
  Dato × Dato × EPJA × CPR  $\xrightarrow{\sim}$  ResNodeId-set
resSoegDato(dst, dsl, epj, cpr) ≡
  let pj = epj(cpr) in

```

```

    {rid |
      rid : ResNodeId •
      (∃ fid : ForId •
        fid ∈ dom forl(pj) ∧
        let res = res(forl(pj)(fid)) in
          rid ∈ dom res ∧
          dat(res(rid)) ≥ dst ∧
          dsl ≥ dat(res(rid))
        end)}
  end
pre cpr ∈ dom epj,

```

***resSoegDato** (søg resultater efter dato) finder de resultatnode-id'er for en given person i EPJ som er oprettet indenfor et bestemt tidsrum. Tager to datoer, en EPJ og et cpr-nummer som input og returnerer en mængde af resultatnode-id'er, som er i det tidsrum som de to datoer udgør.*

```

findAktiveForloeb : CPR × EPJA  $\xrightarrow{\sim}$  ForId-set
findAktiveForloeb(cpr, epj) ≡
  let pj = epj(cpr) in
    {fid |
      fid : ForId •
      fid ∈ dom forl(pj) ∧
      forsta(forl(pj)(fid)) = aaben}
  end
pre cpr ∈ dom epj,

```

***findAktiveForloeb** (find aktive forløb) finder de aktive forløb i EPJ for et givent cpr-nummer. Tager et cpr-nummer og en EPJ som input og returnerer en mængde forløbs-id'er på de aktive forløb svarende til cpr-nummer i EPJ.*

```

soegForloeb : ForId × CPR × EPJA  $\xrightarrow{\sim}$  Forloeb
soegForloeb(fid, cpr, epj) ≡ forl(epj(cpr))(fid)
pre cpr ∈ dom epj ∧ fid ∈ dom forl(epj(cpr)),

```

***soegForloeb** (søg efter forløb) finder et forløb, ud fra forløbs-id, for en given EPJ og cpr-nummer.*

```

lukForloeb : ForId × CPR × EPJA  $\xrightarrow{\sim}$  EPJA
lukForloeb(fid, cpr, epj) ≡
  let pj = epj(cpr) in
    epj †
    [cpr ↦
      mk_PJ'(
        per(pj),
        forl(pj) †

```

```

    [fid ↦ recon_forsta(lukket, forl(pj)(fid))] ]
  end
pre cpr ∈ dom epj,

```

lukForloeb (*luk forløb*) lukker et forløb for et givet cpr-nummer og EPJ. Tager forløbs-id, cpr-nummer og EPJ som input og returnerer den opdaterede EPJ hvor status på forløbet givet ved forløbs-id er ændret til lukket.

```

naesteDiagStat : DiagStatus → DiagStatus
naesteDiagStat(s) ≡
  case s of
    aaben → lukket,
    lukket → lukket
  end,

```

naesteDiagStat (*næste diagnosestatus*) hjælpefunktionen der opdaterer status på en diagnose.

```

opdatDiagStat :
  DiagNodeId × ForId × CPR × EPJA  $\rightsquigarrow$  EPJA
opdatDiagStat(did, fid, cpr, epj) ≡
  let
    pj = epj(cpr),
    forloeb = forl(pj)(fid),
    diag = diaghi(forloeb),
    nydiag =
      diag †
      [did ↦
        recon_sta(
          naesteDiagStat(stat(diag(did))), diag(did))]
  in
    epj †
    [cpr ↦
      mk_PJ'(
        per(pj),
        forl(pj) †
        [fid ↦ recon_diag(nydiag, forloeb)]] ]
  end
pre
  cpr ∈ dom epj ∧ fid ∈ dom forl(epj(cpr)) ∧
  did ∈ dom diaghi(forl(epj(cpr))(fid)),

```

opdatDiagStat (*opdater diagnosestatus*) opdaterer status på en diagnose for en person i EPJ.

```

naesteBehStat : BehStatus → BehStatus

```

```

naesteBehStat(s) ≡
  case s of
    planlagt → igang,
    igang → udfoert,
    udfoert → evalueret,
    evalueret → evalueret
  end,

```

naesteBehStat (næste behandlingsstatus) hjælpefunktion der returnerer næste status for en behandling.

```

opdatBehStat :
  BehNodeId × ForId × CPR × EPJA  $\xrightarrow{\sim}$  EPJA
opdatBehStat(bid, fid, cpr, epj) ≡
  let
    pj = epj(cpr),
    forloeb = forl(pj)(fid),
    beh = behhi(forloeb),
    nybeh =
      beh †
      [ bid ↦
        recon_sta(
          naesteBehStat(stat(beh(bid))), beh(bid)) ]
  in
    epj †
    [ cpr ↦
      mk_PJ'(
        per(pj),
        forl(pj) † [ fid ↦ recon_beh(nybeh, forloeb) ]
      )
    ]
  end
pre
  cpr ∈ dom epj ∧ fid ∈ dom forl(epj(cpr)) ∧
  bid ∈ dom behhi(forl(epj(cpr))(fid))

```

opdatBehStat (opdater behandlingsstatus) opdaterer status på en behandling for en person i EPJ.

```

axiom
  [Refleksivitet]
    ∀ t : Dato • t ≥ t,
  [Anti-symmetri]
    ∀ t1, t2 : Dato • t1 ≥ t2 ∧ t2 ≥ t1 ⇒ t1 = t2,
  [Transitivitet]
    ∀ t1, t2, t3 : Dato •
      t1 ≥ t2 ∧ t2 ≥ t3 ⇒ t1 ≥ t3,

```

```

[Totalitet]
  ∀ t1, t2 : Dato • t1 ≥ t2 ∨ t2 ≥ t1
end

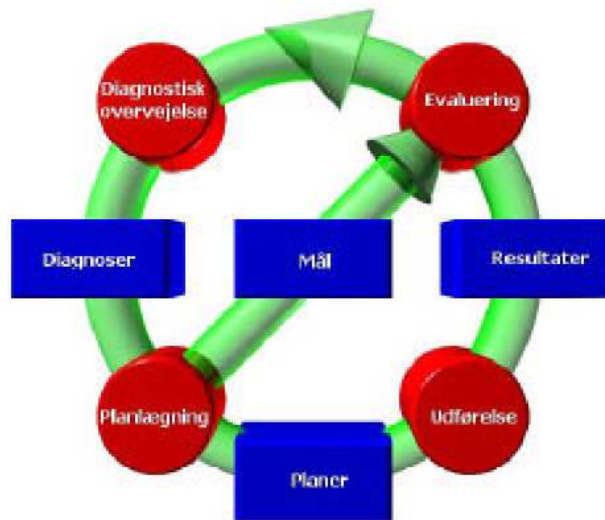
```

5 Vurdering af G-EPJ specifikationen

Dette afsnit indledes med en kort beskrivelse af de projekter som henholdsvis Sundhedsstyrelsen og Systematic arbejder på. Derefter følger en vurdering af projekterne. Her vil fordele og ulemper blive diskuteret.

5.1 Sundhedsstyrelsens projekt

Sundhedsstyrelsen arbejde går som tidligere nævnt ud på at udvikle en model for et fælles struktureret grundlag for kommunikation mellem EPJ-systemer. Dette projekt går under navnet ”Grundstrukturen for Elektroniske Patientjournaler” [7] (G-EPJ). Grundstrukturen består dels af en begrebsmodel, som definerer en standard for processer og navngivning, og dels en Reference Informations Model (RIM), som er en datalogisk model, der implementerer begrebsmodellen. Denne kan illustreres ved figur 2.



Figur 2: Grafisk repræsentation af begrebsmodellen. Figuren er gengivet fra [7].

Klinikere indenfor alle faggrupper anerkender, at begrebsmodellen er en god model for deres arbejdsgange. For klinikerne kræver det dog lidt tilvænning, at modellen er forløbsorienteret frem for den traditionelle kontaktorienterede papirjournal. I papirjournalen er oplysningerne sorteret efter hvornår de er indsamlet, dvs. typisk hver gang patienten har været i kontakt med en kliniker. I den forløbsorienterede model er oplysningerne sorteret i længerevarende behandlingsforløb, som kan bestå af mange kontakter. På den måde

knyttes oplysninger, som er relateret til de samme symptomer, sammen. Modellen går ned i hver en lille detalje hvilket gør den ret statisk og kompleks.

Arbejdet med modellen er som sagt stadig igang, dette betyder at der stadig er hjørner, der ikke er færdigudviklede. G-EPJ specifikationen er nu udgivet i version 1. Planen er, at specifikationen skal afprøves i et pilotprojekt med mindst 100 patienter. Som et til-læg til grundmodellen har Sundhedsstyrelsen udviklet det såkaldte medicinmodul, som er en specifikation for kommunikation af medicinoplysninger i elektroniske patientjournaler. Medicinmodulet har været afprøvet i et pilotprojekt og er nu klar til at blive brugt i praksis.

Fra politisk side er det vedtaget, at der på alle offentlige sygehuse skal være EPJ systemer i brug i 2005. Som Sundhedsministeren også selv erkender, er dette et ambitiøst mål [6]. Det er langt fra sikkert at denne deadline kan overholdes.

5.2 Systematics projekt

I Århus amt kører et projekt der har til formål at få udviklet et andengenerations EPJ system. Århus har altså valgt ikke at vente på, at Sundhedsstyrelsen får færdiggjort den grundlæggende model. Til at udvikle kernen af dette system har man valgt firmaet Systematic.

Systematic har i denne forbindelse udviklet et system, der tager udgangspunkt i den såkaldte domæne objekt model (DOM) som Århus amt har fået udviklet. DOM'en er en modelbeskrivelse af data, der ønskes behandlet i en elektronisk patientjournal. Modellen tager udgangspunkt i hændelser og standardforløb. En hændelse kan for eksempel være en temperaturmåling eller en behandling. Samlinger af hændelser eller sammenhænge mellem information er eksempler på standardforløb. Systemet skal ses som en bro mellem de allerede eksisterende såkaldte fødesystemer og et antal nyudviklede brugermoduler. Et fødesystem er for eksempel it-systemet på et apotek, eller i en blodbank eller det it-system som de praktiserende læger benytter sig af og brugermodulerne danner brugergrænseflade for fødesystemerne. Den opgave systemet derfor skal udføre er at modtage data fra fødesystemerne, lagre dem og sende dem videre til andre fødesystemer eller brugermoduler efter behov.

En af de store fordele ved det system som Systematic udvikler er dets store fleksibilitet og dynamik, der skyldes DOM'ens struktur. Denne dynamik gør, at systemet kan tilpasses varierende ønsker og behov samt følge den udvikling, der sker i sundhedsvæsenet.

Systemet bygges op ud fra et såkaldt integrationssystem (EPJI). Oven på EPJI bygges de nødvendige brugermoduler. Desuden kobles EPJI sammen med forskellige fødesystemer via den såkaldte informationsbus. I EPJI findes også moduler, der sørger for bla. sikkerheden i systemet. Figur 3 illustrerer denne opbygning.

Systemet der er under udvikling i Århus er det første andengenerations EPJ system i landet. Dette kan være en af grundene til, at der ikke er blevet lagt vægt på at sikre, at det kan kommunikere sammen med andre lignende systemer – en mulighed der ellers er af stor relevans i fremtiden hvor flere amter vil få indført EPJ. Der er dog udarbejdet et dokument, der beskriver hvorledes modellen kan repræsentere data fra G-EPJ, således at man kan tilpasse systemet når der kommer en endelig standard på området.

Hovedparten af Systematics leverance blev godkendt af Århus Amt i september 2002

arbejde sammen. Samtidig skabes der problemer hvad angår inkonsistens, problemer som hverken Sundhedsstyrelsen eller Systematic har en løsning for på nuværende tidspunkt.

Udviklingen af G-EPJ har ikke kun været en programmeringsmæssig udfordring. Politik er en stor del af denne udvikling - det er ikke altid at denne politiske indflydelse har en positiv indvirkning på udviklingshastigheden, det modsatte er nok oftere tilfældet.

En anden problemstilling ved EPJ er det etiske spørgsmål. Ved indførelse af EPJ vil alle klinikere ved udgangspunktet have adgang til alle patientjournaler. Ifølge dansk lovgivning skal en patient have mulighed for at nægte personer adgang til sin journal. Systematic kan i deres model meget specifikt styre hvem der har adgang til hvilke journaler. Dette giver dog også problemer. En tænkt situation er, hvis en patient kun har givet et meget begrænset antal klinikere adgang til patientens journal. Kommer den pågældende patient ud for en ulykke og den læge, der skal behandle, ikke har adgang til journalen, hvad så? Hos Sundhedsstyrelsen tænkes dette problem løst ved ikke direkte at nægte klinikere adgang. Istedet vil der blive ført en logfil. Hermed skabes der overblik over hvem der har redigeret eller læst i hvilke journaler, og på denne måde kan en kliniker blive udspurgt om, hvorfor vedkommende har haft forbindelse til en specifik journal. Det, at alle klinikere har adgang til alle journaler, gør også, at der konsekvent skal ses kritisk på de oplysninger, der står i en journal – ikke alt skal tages for gode varer.

Ved udviklingen af medicinmodulet har Systematic støttet sig til Sundhedsstyrelsens tilsvarende. Sundhedsstyrelsen har dog siden videreudviklet deres medicinmodul for at gøre det konsistent med G-EPJ 1.0. Derfor vil Systematic's modul formodentlig kræve noget tilretning for at sikre fuld kompatibilitet med Sundhedsstyrelsens model. Ifølge EPJ-Observatoriets årsrapport for 2002 [2] er forskellene imellem de to modeller imidlertid små.

I G-EPJ har klinikere mulighed for at opstille et mål for en behandling. Når resultatet af behandlingen foreligger, bliver det automatisk sammenlignet med målet. Vi stiller os tvivlende overfor om det altid er muligt at opstille målbare mål for en behandling. Sundhedsstyrelsens svar på dette er, at målet ikke er obligatorisk, så hvis man ikke kan definere et målbart mål, kan man undlade at opstille et. Som udgangspunkt virker det fornuftigt at opstille et mål for en behandling, så det kan dokumenteres, at klinikerne har gjort sig klart, hvad vedkommende ønsker at opnå med behandlingen. Der burde derfor i G-EPJ være mulighed for at opstille et mål, som enten kan være målbart og vil blive vurderet automatisk, eller kan være ikke målbart og vil kræve en klinikers subjektive vurdering.

6 Udvekslingsformat

Som beskrevet i det foregående anser vi det for urealistisk, at der vil kunne opnås enighed om en central national database for EPJ data. Derfor tager vi i dette afsnit udgangspunkt i en model, hvor der eksisterer et antal forskellige EPJ systemer med hver deres database.

Først opstilles en række krav til et format for udveksling af data mellem EPJ systemer. Dernæst beskrives et design, hvori formatets syntaks defineres og gives en operationel semantik. Endelig konverteres formatet til XML.

6.1 Krav til udvekslingsformat

Overordnet set skal udvekslingsformatet sikre, at enhver klinisk information, der er lagret i ét EPJ system kan tilgås fra ethvert andet EPJ system. Tilgangen til eksternt lagrede data skal være transparent, dvs. lokale og eksterne data skal integreres i en fælles brugergrænseflade. Således skal det principielt ikke være muligt for brugeren at se hvilke data, der kommer fra det lokale EPJ system, og hvilke, der hentes eksternt.

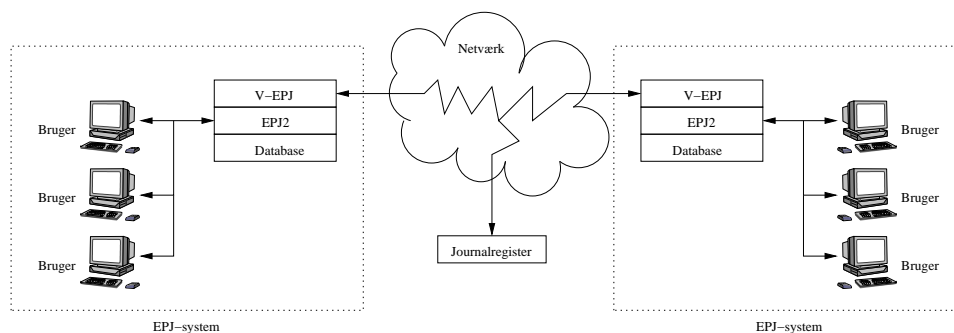
Der forudsættes at der findes en pålidelig måde at sende elektroniske beskeder fra ethvert EPJ system til ethvert andet EPJ system.

Det må ikke være muligt at skrive data i et eksternt EPJ system. Den eneste undtagelse fra dette krav er, at det skal være muligt at ændre henvisninger. Hvis en bruger vil tilføje data til en journal, som findes i et eksternt system, skal disse data gemmes i det lokale system. Desuden skal der tilføjes en henvisning i det eksterne system, som peger på de netop lagrede data i det lokale system. Baggrunden for dette krav er, at selvom hvert EPJ system eksponerer sine data igennem et fælles grænsesnit, kan de godt internt indeholde yderligere data som ikke er tilgængelige gennem grænsesnippet. For at sikre at disse data også udfyldes korrekt, er det nødvendigt at det gøres fra det lokale system. Dermed åbnes der mulighed for, at nogle amter kan specificere et EPJ system, som har flere funktionaliteter end det fælles grænsesnit.

6.2 Design af udvekslingsformat

Det design for udveksling af data mellem EPJ systemer, som præsenteres i det følgende, benævnes Virtuel Elektronisk Patientjournal (forkortet V-EPJ). V-EPJ fungerer som et abstraktionslag ovenpå EPJ systemerne, således at det gennem V-EPJ er muligt at udtrække og sammensætte data fra flere EPJ systemer.

V-EPJ består af fire komponenter: et journalregister, et udvekslingsformat og nogle EPJ systemer med tilhørende V-EPJ grænsesnit. Hver af disse komponenter specificeres i de følgende underafsnit. V-EPJ systemet er skitseret i figur 4.



Figur 4: Skematisk fremstilling af V-EPJ systemet.

Dele af V-EPJ grænsesnippet er ikke fuldt specificeret. Det drejer sig om de steder hvor en forespørgsel til eksterne EPJ'er er nødvendig. For at dette skal kunne lade sig gøre kræves der en omorganisering af måden hvorpå data i et forløb er lagret.

Problemet kan afhjælpes på to måder. Enten skal der, når der foretages en forespørgsel på et forløb, returneres en kopi af forløbet, således at forløbet efterfølgende er lagret i begge EPJ-systemer. Herved gøres det muligt at tilgå de nødvendige data. Hvis denne løsningsmetode vælges, kræver det, at der hver gang der kopieres et forløb for en patient fra et EPJ-system til et andet, foregår en opdatering af det tilsvarende forløb i modtagersystemet, således at det endelige forløb indeholder alle data fra både det lokale og det eksterne. Denne opdateringsproces kan ved store forløb bliver særdeles ineffektiv, og denne løsningsmodel er derfor ikke at foretrække.

En anden måde at løse problemet på er, at udskille dele af de data der er lagret i et forløb og placere dem på et for eksterne EPJ-systemer, lettere tilgængeligt niveau i EPJ-systemet. Forløbet vil istedet blot henvise til de data der før var en del af forløbet selv. Dette vil ikke kræve store opdateringsprocedurer og denne løsningsmodel er derfor at foretrække.

6.2.1 Udvidelse af EPJ2

For at V-EPJ kan bygges ovenpå EPJ2 specifikationen, som den er præsenteret i afsnit 4, er det nødvendigt at udvide alle id'er til at kunne henvise til data som er lagret enten lokalt eller eksternt. Denne ændring er gennemført for en del af EPJ2 specifikationen herunder. For kommentarer henvises til EPJ2 specifikationen.

scheme VEPJ =

class

type

```

VEPJ' = CPR  $\xrightarrow{m}$  PJ,
VEPJ = { | epj : VEPJ' • wf_VEPJ(epj) | },
PJ' :: per : Person forl : ForId  $\xrightarrow{m}$  Forloeb,
PJ = { | pj : PJ' • wf_PJ(pj) | },
Person = CPR × Efternavn × Fornavn × Andet,
Forloeb ::
  beskfor : Beskriv
  diaghi : InfElemDiag  $\leftrightarrow$  recon_diag
  behhi : InfElemBeh  $\leftrightarrow$  recon_beh
  res : InfElemRes  $\leftrightarrow$  recon_res
  sym : InfElemSym  $\leftrightarrow$  recon_sym
  forsta : ForStatus  $\leftrightarrow$  recon_forsta,
InfElemDiag = DiagNodeId  $\xrightarrow{m}$  DiagNode,
InfElemBeh = BehNodeId  $\xrightarrow{m}$  BehNode,
InfElemRes = ResNodeId  $\xrightarrow{m}$  ResNode,
InfElemSym = SymNodeId  $\xrightarrow{m}$  SymNode,
DiagNode ::
  besk : Beskriv
  stat : DiagStatus  $\leftrightarrow$  recon_sta
  dat : Dato
  ansvarlig : KliId

```

```

grundlag : ESymNodeId*
fm : OptEDiagNodeId
soesk : EDiagNodeId* ↔ recon_soesk,
BehNode ::
  besk : Beskriv
  stat : BehStatus ↔ recon_sta
  dat : Dato
  ansvarlig : KliId
  grundlag : EDiagNodeId*
  fm : OptEBehNodeId
  soesk : EBehNodeId* ↔ recon_soesk,
ResNode ::
  besk : Beskriv
  grundlag : EBehNodeId
  dat : Dato
  ansvarlig : KliId,
SymNode ::
  besk : Beskriv  dat : Dato  ansvarlig : KliId,
  OptEDiagNodeId == mk_dId(EDiagNodeId) | nil1,
  OptEBehNodeId == mk_bId(EBehNodeId) | nil2,
  DiagStatus == aaben | lukket,
  BehStatus == planlagt | udfoert | igang | evalueret,
  ForStatus == aaben | lukket,
  CPR,
  ForId,
  Beskriv = Text,
  DiagNodeId,
  BehNodeId,
  ResNodeId,
  SymNodeId,
  OptDiagNodeId = DiagNodeId,
  OptBehNodeId = BehNodeId,
  EForId :: lokal : ForId  ekstern : EPJId,
  EDiagNodeId :: lokal : DiagNodeId  ekstern : EPJId,
  EBehNodeId :: lokal : BehNodeId  ekstern : EPJId,
  EResNodeId :: lokal : ResNodeId  ekstern : EPJId,
  ESymNodeId :: lokal : SymNodeId  ekstern : EPJId,
  EId =
    EForId | ESymNodeId | EDiagNodeId | EBehNodeId |
    EResNodeId,
  EPJId,
  Dato,
  KliId,
  Efternavn,

```

Fornavn,
Andet

value

$\geq : \text{Dato} \times \text{Dato} \rightarrow \mathbf{Bool}$,
LokEPJId : EPJId,

wf_VEPJ : VEPJ' \rightarrow **Bool**

wf_VEPJ(epj) \equiv

(\forall cpr : CPR •

cpr \in **dom** epj \Rightarrow

let (cpr', en, fn, an) = per(epj(cpr)) **in**

cpr = cpr'

end),

/* The wellformed conditions is not specified
for the VEPJ. */

wf_PJ : PJ' \rightarrow **Bool**,

nyPJ : Person \times VEPJ \rightarrow VEPJ

nyPJ((cpr, e, f, a), epj) \equiv

epj \cup [cpr \mapsto mk_PJ'((cpr, e, f, a), [])],

opretESymNodeId :

(SymNodeId \xrightarrow{m} SymNode) \rightarrow ESymNodeId

opretESymNodeId(sm) \equiv

let sid : SymNodeId • sid \notin **dom** sm **in**

mk_ESymNodeId(sid, LokEPJId)

end,

tilfoejSym :

Beskriv \times Dato \times KliId \times VEPJ \times CPR \times ForId $\xrightarrow{\sim}$

VEPJ \times ESymNodeId

tilfoejSym(b, d, a, epj, cpr, fid) \equiv

let

pj = epj(cpr),

forloeb = forl(pj)(fid),

sym = sym(forloeb),

sid = opretESymNodeId(sym),

nysym = sym \dagger [lokal(sid) \mapsto mk_SymNode(b, d, a)]

in

(epj \dagger

[cpr \mapsto

mk_PJ'(

```

        per(pj),
        forl(pj) † [fid ↦ recon_sym(nysym, forloeb)
    ]], sid)
    end
pre cpr ∈ dom epj ∧ fid ∈ dom forl(epj(cpr)),

opretEForId : VEPJ → EForId
opretEForId(epj) ≡
    let
        fid : ForId •
        ∀ pj : PJ •
            pj ∈ rng epj ⇒ fid ∉ dom forl(pj)
    in
        mk_EForId(fid, LokEPJId)
    end,

nytForloeb :
    Beskriv × CPR × VEPJ  $\xrightarrow{\sim}$  VEPJ × EForId
nytForloeb(b, cpr, epj) ≡
    let pj = epj(cpr), fid = opretEForId(epj) in
        (epj †
        [cpr ↦
            mk_PJ'(
                per(pj),
                forl(pj) ∪
                [lokal(fid) ↦
                    mk_Forloeb(b, [], [], [], [], aaben)]), fid)
    end
pre cpr ∈ dom epj,

opretEResNodeId :
    (ResNodeId  $\xrightarrow{\vec{m}}$  ResNode)  $\xrightarrow{\sim}$  EResNodeId
opretEResNodeId(rm) ≡
    let rid : ResNodeId • rid ∉ dom rm in
        mk_EResNodeId(rid, LokEPJId)
    end,

tilfoejRes :
    Beskriv × EBehNodeId × Dato × KliId × VEPJ ×
    CPR × ForId →
        VEPJ × EResNodeId
tilfoejRes(b, bid, d, a, epj, cpr, fid) ≡
    let
        pj = epj(cpr),

```

```

    forloeb = forl(pj)(fid),
    res = res(forloeb),
    rid = opretEResNodeId(res),
    nyres =
      res † [lokal(rid) ↦ mk_ResNode(b, bid, d, a)]
  in
    (epj †
     [cpr ↦
      mk_PJ(
        per(pj),
        forl(pj) † [fid ↦ recon_res(nyres, forloeb)]
      )], rid)
  end
pre cpr ∈ dom epj ∧ fid ∈ dom forl(epj(cpr)),

```

```

soegPJ : CPR × VEPJ  $\rightsquigarrow$  PJ
soegPJ(cpr, epj)  $\equiv$  epj(cpr) pre cpr ∈ dom epj,

```

```

soegibeskriv : Text × Text → Bool
soegibeskriv(bes, txt)  $\equiv$ 
  (∃ a, b, c : Text • b = txt ∧ a ^ b ^ c = bes),

```

```

diagSoegDato :
  Dato × Dato × VEPJ × CPR  $\rightsquigarrow$  EDiagNodeId-set
diagSoegDato(dst, dsl, epj, cpr)  $\equiv$ 
  let pj = epj(cpr) in
    {did |
      did : EDiagNodeId •
        (∃ fid : ForId •
          fid ∈ dom forl(pj) ∧
          let diag = diaghi(forl(pj)(fid)) in
            lokal(did) ∈ dom diag ∧
            dat(diag(lokal(did))) ≥ dst ∧
            dsl ≥ dat(diag(lokal(did))) ∧
            ekstern(did) = LokEPJId
          end)}}
  end
pre cpr ∈ dom epj,

```

```

behSoegDato :
  Dato × Dato × VEPJ × CPR  $\rightsquigarrow$  EBehNodeId-set
behSoegDato(dst, dsl, epj, cpr)  $\equiv$ 
  let pj = epj(cpr) in
    {bid |

```

```

    bid : EBehNodeId •
    (∃ fid : ForId •
      fid ∈ dom forl(pj) ∧
      let beh = behhi(forl(pj)(fid)) in
      lokal(bid) ∈ dom beh ∧
      dat(beh(lokal(bid))) ≥ dst ∧
      dsl ≥ dat(beh(lokal(bid))) ∧
      ekstern(bid) = LokEPJId
    end})
  end
pre cpr ∈ dom epj,

symSoegDato :
  Dato × Dato × VEPJ × CPR  $\xrightarrow{\sim}$  ESymNodeId-set
symSoegDato(dst, dsl, epj, cpr) ≡
  let pj = epj(cpr) in
  {sid |
    sid : ESymNodeId •
    (∃ fid : ForId •
      fid ∈ dom forl(pj) ∧
      let sym = sym(forl(pj)(fid)) in
      lokal(sid) ∈ dom sym ∧
      dat(sym(lokal(sid))) ≥ dst ∧
      dsl ≥ dat(sym(lokal(sid))) ∧
      ekstern(sid) = LokEPJId
    end})
  end
pre cpr ∈ dom epj,

resSoegDato :
  Dato × Dato × VEPJ × CPR  $\xrightarrow{\sim}$  EResNodeId-set
resSoegDato(dst, dsl, epj, cpr) ≡
  let pj = epj(cpr) in
  {rid |
    rid : EResNodeId •
    (∃ fid : ForId •
      fid ∈ dom forl(pj) ∧
      let res = res(forl(pj)(fid)) in
      lokal(rid) ∈ dom res ∧
      dat(res(lokal(rid))) ≥ dst ∧
      dsl ≥ dat(res(lokal(rid))) ∧
      ekstern(rid) = LokEPJId
    end})
  end

```

pre cpr ∈ **dom** epj,

findAktiveForloeb : CPR × VEPJ $\xrightarrow{\sim}$ EForId-set

findAktiveForloeb(cpr, epj) \equiv

let pj = epj(cpr) **in**

{fid |

fid : EForId •

lokal(fid) ∈ **dom** forl(pj) ∧

forsta(forl(pj)(lokal(fid))) = aaben ∧

ekstern(fid) = LokEPJId}

end

pre cpr ∈ **dom** epj,

soegForloeb : EForId × CPR × VEPJ $\xrightarrow{\sim}$ Forloeb

soegForloeb(fid, cpr, epj) \equiv

forl(epj(cpr))(lokal(fid))

pre

cpr ∈ **dom** epj ∧

lokal(fid) ∈ **dom** forl(epj(cpr)) ∧

ekstern(fid) = LokEPJId,

lukForloeb : EForId × CPR × VEPJ $\xrightarrow{\sim}$ VEPJ

lukForloeb(fid, cpr, epj) \equiv

let pj = epj(cpr) **in**

epj †

[cpr ↦

mk_PJ'(

per(pj),

forl(pj) †

[lokal(fid) ↦

recon_forsta(lukket, forl(pj)(lokal(fid)))]]

end

pre cpr ∈ **dom** epj ∧ ekstern(fid) = LokEPJId,

naesteDiagStat : DiagStatus → DiagStatus

naesteDiagStat(s) \equiv

case s **of**

aaben → lukket,

lukket → lukket

end,

opdatDiagStat :

EForId × EForId × CPR × VEPJ $\xrightarrow{\sim}$ VEPJ

opdatDiagStat(did, fid, cpr, epj) \equiv

```

let
  pj = epj(cpr),
  forloeb = forl(pj)(lokal(fid)),
  diag = diaghi(forloeb),
  nydiag =
    diag †
    [ lokal(did) ↦
      recon_sta(
        naesteDiagStat(stat(diag(lokal(did)))),
        diag(lokal(did))) ]
in
  epj †
  [ cpr ↦
    mk_PJ'(
      per(pj),
      forl(pj) †
      [ lokal(fid) ↦ recon_diag(nydiag, forloeb) ] ]
end
pre
  cpr ∈ dom epj ∧
  lokal(fid) ∈ dom forl(epj(cpr)) ∧
  lokal(did) ∈
    dom diaghi(forl(epj(cpr))(lokal(fid))) ∧
  ekstern(did) = LokEPJId ∧ ekstern(fid) = LokEPJId,

```

naesteBehStat : BehStatus → BehStatus

naesteBehStat(s) ≡

```

case s of
  planlagt → igang,
  igang → udfoert,
  udfoert → evalueret,
  evalueret → evalueret
end,

```

opdatBehStat :

EBehNodeId × EForId × CPR × VEPJ $\xrightarrow{\sim}$ VEPJ

opdatBehStat(bid, fid, cpr, epj) ≡

```

let
  pj = epj(cpr),
  forloeb = forl(pj)(lokal(fid)),
  beh = behhi(forloeb),
  nybeh =
    beh †
    [ lokal(bid) ↦

```

```

        recon_sta(
            naesteBehStat(stat(beh(lokal(bid)))),
            beh(lokal(bid))) ]
    in
    epj †
    [ cpr ↦
      mk_PJ'(
        per(pj),
        forl(pj) †
        [ lokal(fid) ↦ recon_beh(nybeh, forloeb) ] ] ]
    end
pre
cpr ∈ dom epj ∧
lokal(fid) ∈ dom forl(epj(cpr)) ∧
lokal(bid) ∈
    dom behhi(forl(epj(cpr))(lokal(fid))) ∧
ekstern(bid) = LokEPJId ∧ ekstern(fid) = LokEPJId

axiom
[ Refleksivitet ]
  ∀ t : Dato • t ≥ t,
[ Anti_symmetri ]
  ∀ t1, t2 : Dato • t1 ≥ t2 ∧ t2 ≥ t1 ⇒ t1 = t2,
[ Transitivitet ]
  ∀ t1, t2, t3 : Dato •
    t1 ≥ t2 ∧ t2 ≥ t3 ⇒ t1 ≥ t3,
[ Totalitet ]
  ∀ t1, t2 : Dato • t1 ≥ t2 ∨ t2 ≥ t1
end

```

6.2.2 Journalregister

I en situation med distribuerede databaser opstår behovet for at finde ud af hvor data for en bestemt patient er lagret. Til dette formål specificeres et såkaldt journalregister. Der findes kun ét centralt journalregister.

Journalregisteret har tre funktioner:

Opslag For en given person fås en mængde af henvisninger til EPJ systemer, hvori der er lagret data om personen.

Tilføjelse Givet en person og en henvisning til et EPJ system, tilføjes denne henvisning til registeret.

Sletning Givet en person og en henvisning til et EPJ system, slettes denne henvisning fra den givne person.

Protokollen for kommunikation mellem registeret og EPJ systemerne er specificeret nedenfor.

```

scheme REGISTER =
  extend VEPJ with
  class
    type
      Register = CPR  $\overline{m}$  EPJId-set,

```

***Register** består af et cpr-nummer og en mængde af EPJId.*

```

  RegKommando ==
    mk_Opslag(EPJId, CPR) |
    mk_Tilfoej(EPJId, CPR) |
    mk_Slet(EPJId, CPR),

```

***RegKommando**(Registerkommando) kan være et opslag, en tilføjelse eller en sletning. EPJId angiver hvor kommandoen kommer fra.*

```

  RegResultat ==
    mk_Udfoert(EPJId) |
    mk_OK(EPJId, EPJId-set) |
    mk_Fejl(EPJId)

```

***RegResultat**(Registerresultat) kan være en udførelse, en ok-meddelelse eller en fejlmeddelelse. EPJId angiver hvor resultat skal sendes hen.*

channel ind : RegKommando, ud : RegResultat

***channel.** En indkanal til registerkommandoer og en udkanal til registerresultater.*

```

value
  RegisterProces : Register  $\rightarrow$  in ind out ud Unit
  RegisterProces(reg)  $\equiv$ 

```

***RegisterProces.** Proces der lytter på indkanalen og sender ud på udkanalen.*

```

  let kom = ind? in
    case kom of
      mk_Opslag(eid, p)  $\rightarrow$ 
        if p  $\in$  dom reg
          then
            ud!mk_OK(eid, reg(p)) ; RegisterProces(reg)
          else ud!mk_Fejl(eid) ; RegisterProces(reg)
        end,

```

mk_Opslag()(foretag opslag) hvis opslaget kan udføres returneres en ok meddelelse med EPJid og en mængde af EPJid. Derefter lyttes videre i processen.

```
mk_Tilfoej(eid, p) →
  ud!mk_Udfoert(eid) ;
  if p ∈ dom reg
  then
    RegisterProces(
      reg † [p ↦ reg(p) ∪ {eid}])
  else RegisterProces(reg ∪ [p ↦ {eid}])
  end,
```

mk_Tilfoej()(foretag tilføjelse) der returneres en udført meddelelse. Derefter lyttes videre i processen på den opdaterede patientjournal.

```
mk_Slet(eid, p) →
  if p ∈ dom reg then ud!mk_Udfoert(eid)
  else ud!mk_Fejl(eid)
  end ;
  RegisterProces(reg \ {p})
```

mk_Slet()(foretag sletning) Hvis sletningen kan udføres returneres en udført meddelelse med EPJid. Derefter lyttes videre i processen på det opdaterede register uden det slettede element.

```
      end
    end
  end
```

6.2.3 Brugerkommandoer til V-EPJ

Af hensyn til modelleringen af grænsesnittet mellem V-EPJ systemerne i de følgende afsnit, defineres nu en syntaks for brugerkommandoer internt i et EPJ system. Disse kommandoer svarer til funktionerne i EPJ2 grænsesnittet. Meningen er at disse kommandoer bliver genereret af brugergrænsefladen og sendt til EPJ systemet.

Det er ikke nødvendigt for et EPJ system præcis at implementere disse kommandoer, men det skal tilbyde lignende funktionalitet. To forskellige EPJ systemer kan altså sagtens have forskellige brugerkommandoer, hvilket åbner mulighed for at leverandører kan vælge at tilbyde ekstra funktionalitet internt i EPJ systemet. Den ekstra funktionalitet vil derimod ikke kunne udnyttes fra et andet EPJ system, da alle EPJ systemer skal præsentere det samme grænsesnit (som defineres i det følgende afsnit) udadtil.

```
scheme UserEPJ =
  extend REGISTER with
  class
```

type

Bruger_Kommando ==

***Bruger_Kommando** er en sammensat type bestående af følgende muligheder: opret ny patientjournal, opret nyt forløb, tilføj diagnose, tilføj behandling, tilføj symptom, tilføj resultat, luk forløb, søg efter patientjournal, søg i beskrivelse, søg efter beskrivelse, søg efter dato, find forløb og find specifikt forløb.*

mk_nypj(Person) |

***mk_nypj()** (ny patientjournal) kommando til oprettelse af en ny patientjournal. Består af en person.*

mk_nytforloeb(Beskriv × CPR) |

***mk_nytforloeb()** (nyt forløb) er en kommando til oprettelse af **nyt forløb**. Består af en beskrivelse og et cpr-nummer.*

mk_tilfoej_diag(
Beskriv × DiagStatus × Dato × KliId ×
ESymNodeId* × OptEDiagNodeId ×
OptEDiagNodeId × CPR × EForId) |

***mk_tilfoej_diag()** (tilføjelse af diagnose) er en kommando til at tilføje en diagnose. Består af en beskrivelse, en diagnosestatus, en dato, et klinikerid på den ansvarlige, en liste af symptomid, som ligger til grund for diagnosen, to optionelle diagnosenodeid, et cpr-nummer og et forløbsid.*

mk_tilfoej_beh(
Beskriv × BehStatus × Dato × KliId ×
EDiagNodeId* × OptEBehNodeId ×
OptEBehNodeId × CPR × EForId) |

***mk_tilfoej_beh()** (tilføjelse af behandling) er en kommando til at tilføje en behandling. Består af en beskrivelse, en behandlingsstatus, en dato, et klinikerid på den ansvarlige, en liste af diagnosenodeid, som ligger til grund for behandlingen, to optionelle behandlingsnodeid, et cpr-nummer og et forløbsid.*

mk_tilfoej_sym(
Beskriv × Dato × KliId × CPR × EForId) |

***mk_tilfoej_sym()** (tilføjelse af symptom) er en kommando til at tilføje et symptom. Består af en beskrivelse, en dato, et klinikerid på den ansvarlige, et cpr-nummer og et forløbsid.*

mk_tilfoej_res(
Beskriv × EBehNodeId × Dato × KliId × CPR ×
EForId) |

***mk_tilfoej_res()** er en kommando til at tilføje et resultat. Består af en beskrivelse, et tilknyttet behandlingsnodeid en dato, et klinikerid på den ansvarlige, et cpr-nummer og et forløbsid.*

`mk_luk_forloeb(EForId × CPR) |`

***mk_luk_forloeb()** (luk forløb) er en kommando til at ændre status på et forløb til lukket. Består af et forløbsid og et cpr-nummer.*

`mk_soeg_pj(CPR) |`

***mk_soeg_pj()** (søg efter patientjournal) er en kommando til at søge efter en patientjournal. Består af et cpr-nummer.*

`mk_soeg_i_beskriv(Beskriv × Text) |`

***mk_soeg_i_beskriv()** (søg i beskrivelse) er en kommando til at søge efter en tekststreng i en beskrivelse. Består af en beskrivelse og et stykke tekst.*

`mk_soeg_beskriv(Text × CPR) |`

***mk_soeg_beskriv()** (søg efter beskrivelse) er en kommando til at søge efter en beskrivelse. Består af et stykke tekst og et cpr-nummer.*

`mk_soeg_dato(Dok × Dato × Dato × CPR) |`

***mk_soeg_dato()** (søg på tidsrum) er en kommando til at finde dokumenter for en person i et givent tidsrum. Består af et dokument, samt to datoer som repræsenterer tidsrummet og et cpr-nummer.*

`mk_find_forloeb(CPR) |`

***mk_find_forloeb()** (find forløb) er en kommando til at finde forløb for en given person. Består af et cpr-nummer.*

`mk_soeg_forloeb(EForId × CPR),`

***mk_soeg_forloeb()** (find specifikt forløb) er en kommando til at finde et forløb ud fra dets id. Består af et forløbsid og et cpr-nummer.*

Dok ==
Diag(dId : EDiagNodeId) |
Beh(bId : EBehNodeId) |
Sym(sId : ESymNodeId) |
Res(rId : ESymNodeId),

***Dok** (dokument) er diagnose, behandling, symptom eller et resultat.*

Bruger_Resultat =
Bruger_Maengde | Bruger_Vaerdi | Bruger_System,

Bruger_Resultat består af en mængde, en værdi eller en systembesked.

```
Bruger_Maengde ==  
  mk_ForloebS(EForId-set) |  
  mk_SymS(ESymNodeId-set) |  
  mk_DiagS(EDiagNodeId-set) |  
  mk_BehS(EBehNodeId-set) |  
  mk_ResS(EResNodeId-set),
```

Bruger_maengde er en forløbsidmængde, symptomidmængde, diagnoseidmængde, behandlingsidmængde eller en resultatidmængde.

```
Bruger_Vaerdi ==  
  mk_PersonV(Person) |  
  mk_ForloebV(Beskriv, ForStatus) |  
  mk_SymV(SymNode) |  
  mk_DiagV(DiagNode) |  
  mk_BehV(BehNode) |  
  mk_ResV(ResNode),
```

Bruger_vaerdi er værdien af en person, et forløb, bestående af beskrivelse og status på forløbet, en symptomnode, en diagnosenode, en behandlingsnode eller en resultatnode.

```
Bruger_System == mk_OK | mk_Fejl
```

Bruger_System(brugersystembesked) kan enten være ok eller fejl.

end

6.2.4 Protokol for kommunikation mellem EPJ systemer

Udvekslingsformatets syntaks er gjort meget simpel. Dermed reduceres kompleksiteten i V-EPJ grænsesnittet. Formatet gives en operationel semantik i afsnit 6.3.

Formatet består af to forskellige typer af beskeder:

- en kommando, som kan være en forespørgsel eller en opdatering;
- et resultat, som er et svar på en kommando. Svaret kan enten være en samling data, en bekræftelse på at kommandoen blev udført, eller en fejl.

En forespørgsel kan være simpel eller komplet. En simpel forespørgsel spørger efter alle informationer af en bestemt type (person-information eller forløb). En standard forespørgsel spørger efter specifik information (symptomer, diagnoser, behandlinger eller resultater) for en given patient og et givent forløb.

En opdateringskommando fortæller, at der i det EPJ system, som sender beskeden, er blevet tilføjet information til et forløb, som er lagret i det modtagende EPJ system.

```

scheme InterEPJ =
  extend UserEPJ with
  class
    type
      Besked ==
        mk_Kom(EPJId, EPJId, Kommando) |
        mk_Res(EPJId, EPJId, Resultat),

```

***Besked** er enten en kommando eller et resultat. I begge indgår to EPJId og henholdsvis kommando-typen og resultat-typen. Det første EPJId identificerer modtageren, og det anden identificerer afsenderen.*

```

      Kommando ==
        mk_SimpelForespg(CPR, Type) |
        mk_Forespg(CPR, Type, ForId, EId) |
        mk_Opdat(CPR, Type, ForId, EId, EId),

```

***Kommando** er enten en simpel forespørgsel, en standard forespørgsel eller en opdatering. En simpel forespørgsel består af et cpr-nummer og en type. En standard forespørgsel består af et cpr-nummer, en type, et forløbs-id og et id.*

```

      Type == Person | Forloeb | Sym | Diag | Beh | Res,

```

***Type** kan enten være person, forløb, symptom, diagnose, behandling eller et resultat. Dette resultat skal ikke forveksles med resultat for en besked.*

```

      Resultat = Liste | Vaerdi | System,

```

***Resultat**(resultat besked) består af en mængde, en værdi eller en systembesked.*

```

      Maengde ==
        mk_ForloebS(ForId-set) |
        mk_SymS(SymNodeId-set) |
        mk_DiagS(DiagNodeId-set) |
        mk_BehS(BehNodeId-set) |
        mk_ResS(ResNodeId-set),

```

***Maengde**(mængde) er en forløbsidmængde, symptomidmængde, diagnoseidmængde, behandlingsidmængde eller en resultatidmængde.*

```

      Vaerdi ==
        mk_PersonV(Person) |
        mk_ForloebV(Beskriv, ForStatus) |
        mk_SymV(SymNode) |
        mk_DiagV(DiagNode) |
        mk_BehV(BehNode) |
        mk_ResV(ResNode),

```

Værdi(*værdi*) er værdien af en person, et forløb, bestående af beskrivelse og status på forløbet, en symptomnode, en diagnosenode, en behandlingsnode eller en resultatnode.

System == mk_OK | mk_Fejl

System(*systembesked*) kan enten være ok eller fejl.

end

6.3 V-EPJ grænsesnit

I dette afsnit gives udvekslingsformatet en operationel semantik i form af en specifikation, som definerer hvordan formatets beskeder skal behandles. For nogle af beskederne er specifikationen dog ikke fuldt færdigudviklet.

I specifikationen nedenfor defineres en proces P , som konstant venter på en besked fra enten en lokal bruger eller et eksternt EPJ system. For hver type af besked definerer processen de trin som behandlingen af beskeden indeholder.

```
scheme Interface =
  extend InterEPJ with
  class
    channel
      int_ind : Bruger_Kommando,
      int_ud : Bruger_Resultat,
      eks_ind : Besked,
      eks_ud : Besked,
      reg_ind : RegResultat,
      reg_ud : RegKommando
```

6 kanaler. En til at lytte efter brugerkommandoer (*int_ind*) og en til at sende resultater til brugeren (*int_ud*). En indkanal (*eks_ind*) og en udkanal (*eks_ud*) til besked mellem de forskellige EPJ'er. Ydermere findes der er en indkanal (*reg_ind*) til at modtage resultater fra registeret og en udkanal (*reg_ud*) til at sende kommandoer til registeret.

```
value
  P :
    VEPJ × EPJId →
      in int_ind, eks_ind, reg_ind
      out int_ud, eks_ud, reg_ud Unit
```

P (*process*) er en process der lytter på de 3 indkanaler og sender på de tre udkanaler.

```
P(epj, eid) ≡
  let kom = int_ind? in
```

Der lyttes på indkanalen for brugerkommandoer.

```
case kom of
  mk_nypj((cpr, en, fn, an)) →
    int_ud!mk_OK ;
    reg_ud!mk_Tilfoej(eid, cpr) ;
    P(nyPJ((cpr, en, fn, an), epj), eid),
```

mk_nypj() (ny patientjournal) I det tilfælde at kommandoen er ny patientjournal, sendes på den interne ud kanal **systembeskeden ok**, og på register udkanalen sendes, at patientjournal skal tilføjes registeret. Ydermere lyttes der videre på EPJ med den nye patientjournal inkluderet via funktionen nyPJ fra VEPJ.

```
mk_nytforloeb(bes, cpr) →
  int_ud!mk_OK ;
  let (epj', fid) = nytForloeb(bes, cpr, epj) in
    P(epj', eid)
  end,
```

mk_nytforloeb (nyt forløb) I det tilfælde at kommandoen er nyt forløb, sendes på den interne ud kanal **systembeskeden ok**. Derefter lyttes videre med på EPJ med det nye forløb inkluderet via funktionen nytForloeb fra VEPJ.

```
mk_tilfoej_diag(
  bes, dias, dat, kid, symnl, odial, odig2,
  cpr, fid) →
  int_ud!mk_OK ; P(epj, eid),
```

Funktionen er ikke blevet færdig implementeret og sender derfor kun systembeskeden ok på den interne ud kanal og lytter videre.

```
mk_tilfoej_beh(
  bes, behs, dat, kid, dianl, obeh1, obeh2,
  cpr, fid) →
  int_ud!mk_OK ; P(epj, eid),
```

Funktionen er ikke blevet færdig implementeret og sender derfor kun systembeskeden ok på den interne ud kanal og lytter videre.

```
mk_tilfoej_sym(bes, dat, kid, cpr, fid) →
  if ekstern(fid) = eid
  then
    int_ud!mk_OK ;
  let
    (epj2, esnid) =
      tilfoejSym(
```

```

        bes, dat, kid, epj, cpr, lokal(fid))
    in
        P(epj2, eid)
    end
else int_ud!mk_OK ; P(epj, eid)
end,

```

mk_tilfoej_Sym() (tilføj symptom). Funktionen er kun implementeret for det tilfælde, at det eksterne id er identisk med id fra den lokale EPJ. For kommandoen tilføj symptom, sendes på den interne udkanal systembeskeden ok. Ydermere lyttes der videre på EPJ med det nye symptom inkluderet via funktionen tilfoejSym fra VEPJ.

```

mk_tilfoej_res(bes, beh1, dat, kid, cpr, fid) →
    if ekstern(fid) = eid
    then
        int_ud!mk_OK ;
    let
        (epj2, ernid) =
            tilfoejRes(
                bes, beh1, dat, kid, epj, cpr,
                lokal(fid))
    in
        P(epj2, eid)
    end
else int_ud!mk_OK ; P(epj, eid)
end,

```

mk_tilfoej_res() (tilføj resultat). Funktionen er kun implementeret for det tilfælde, at det eksterne id er identisk med id fra den lokale EPJ. For kommandoen tilføj resultat, sendes på den interne udkanal systembeskeden ok. Ydermere lyttes der videre på EPJ med det nye symptom inkluderet via funktionen tilfoejRes fra VEPJ.

```

mk_luk_forloeb(fid, cpr) →
    int_ud!mk_OK ; P(epj, eid),

```

Funktionen er ikke blevet færdig implementeret og sender derfor kun **systembeskeden ok** på den interne udkanal og lytter videre.

```

mk_soeg_pj(cpr) → int_ud!mk_OK ; P(epj, eid),

```

Funktionen er ikke blevet færdig implementeret og sender derfor kun **systembeskeden ok** på den interne udkanal og lytter videre.

```

mk_soeg_i_beskriv(bes, txt) →
    int_ud!mk_OK ; P(epj, eid),

```

Funktionen er ikke blevet færdig implementeret og sender derfor kun **systembeskeden ok** på den interne udkanal og lytter videre.

```
mk_soeg_beskriv(txt, cpr) →
  int_ud!mk_OK ; P(epj, eid),
```

Funktionen er ikke blevet færdig implementeret og sender derfor kun systembeskeden ok på den interne udkanal og lytter videre.

```
mk_find_forloeb(cpr) →
  case reg_ud!mk_Opslag(eid, cpr) ; reg_ind? of
    mk_Fejl(_) → int_ud!mk_Fejl,
    mk_OK(_, es) →
      int_ud!
      mk_ForloebS(hentForloeb(eid, es, cpr))
  end ;
  P(epj, eid),
```

***mk_find_forloeb()** (find forløb). Kommandoen find forløb spørger på register udkanalen efter de steder hvor der eksisterer forløb på personen angivet ved cpr-nummeret. Derefter lyttes efter svar fra registeret. Hvis svaret er en fejl sendes den på den interne udkanalen. Ellers findes en EPJid mængde i registeret og der sendes en mængde af forløbs id på den interne udkanal, via hjælpefunktionen hentForloeb.*

```
mk_soeg_forloeb(fid, cpr) →
  int_ud!mk_OK ; P(epj, eid)
```

Funktionen er ikke blevet færdig implementeret og sender derfor kun systembeskeden ok på den interne udkanal og lytter videre.

```
end
end
[]
case eks_ind? of
```

Der lyttes på indkanalen for besked mellem EPJ'er.

```
mk_Kom(dest, kilde, mk_SimpelForespg(cpr, tp)) →
```

***mk_SimpelForespg()** (Simpel forespørgsel) Tilfældet hvor beskeden er en simpel forespørgsel. Der er angivet hvor den kommer fra og hvor resultatet skal hen.*

```
if cpr ∈ dom epj
then
  case tp of
    Person →
      eks_ud!
      mk_Res(
        kilde, eid, mk_PersonV(per(epj(cpr)))) ;
      P(epj, eid),
```

I det tilfælde at typen på den simple forespørgsel er person, sendes på den eksterne udkanal resultatet af at slå personens data op via cpr-nummeret i EPJ og der lyttes videre.

```

Forloeb →
  eks_ud!
  mk_Res(
    kilde, eid,
    mk_ForloebS(dom forl(epj(cpr)))) ;
P(epj, eid),

```

I det tilfælde at typen på den simple forespørgsel er et forløb, sendes på den eksterne udkanal resultatet af at slå forløbsdata op via cpr-nummeret i EPJ og der lyttes videre.

```

— →
  eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
P(epj, eid)

```

For enhver anden type i en simpel forespørgsel sendes resultatet af systembeskeden fejl på den eksterne udkanal.

```

  end
  else
    eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
    P(epj, eid)
  end,
mk_Kom(dest, kilde, mk_Forespg(cpr, tp, fid, id)) →

```

mk_Forespg() (standard forespørgsel) I dette tilfælde er beskeden en standard forespørgsel. Der er angivet hvor den kommer fra og hvor resultatet skal hen.

```

if
  cpr ∈ dom epj ∧
  fid ∈ dom forl(epj(cpr))
then
  case tp of
    Sym →
      case id of
        mk_ESymNodeId(lokal, ekst) →
          if ekst = eid
          then
            eks_ud!
            mk_Res(
              kilde, eid,
              mk_SymV(
                sym(
                  findLokaltForloeb(
                    epj, cpr, fid))(lokal))) ;

```

```

        P(epj, eid)
    else
        eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
        P(epj, eid)
    end,
- →
    eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
    P(epj, eid)
end,

```

Der spørges efter symptomer. De lokale forløb findes via hjælpefunktionen findLokaltForloeb, symptomerne hentes og sendes ud på den eksterne udkanal.

```

Diag →
    case id of
        mk_EDiagNodeId(lokal, ekst) →
            if ekst = eid
                then
                    eks_ud!
                    mk_Res(
                        kilde, eid,
                        mk_DiagV(
                            diaghi(
                                findLokaltForloeb(
                                    epj, cpr, fid))(lokal))) ;
                    P(epj, eid)
                else
                    eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
                    P(epj, eid)
                end,
- →
            eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
            P(epj, eid)
    end,

```

Der spørges efter diagnosehierarkiet. De lokale forløb findes via hjælpefunktionen findLokaltForloeb, diagnosehierarkiet hentes og sendes ud på den eksterne udkanal.

```

Beh →
    case id of
        mk_EBehNodeId(lokal, ekst) →
            if ekst = eid
                then
                    eks_ud!
                    mk_Res(
                        kilde, eid,

```

```

        mk_BehV(
            behhi(
                findLokaltForloeb(
                    epj, cpr, fid))(lokal))) ;
    P(epj, eid)
else
    eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
    P(epj, eid)
end,
- →
    eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
    P(epj, eid)
end,

```

Der spørges efter behandlingshierarkiet. De lokale forløb findes via hjælpefunktionen findLokaltForloeb, behandlingshierarkiet hentes og sendes ud på den eksterne udkanal.

```

Res →
case id of
    mk_EResNodeId(lokal, ekst) →
        if ekst = eid
            then
                eks_ud!mk_Res(kilde, eid, mk_OK) ;
                P(epj, eid)
            else
                eks_ud!
                mk_Res(
                    kilde, eid,
                    mk_ResV(
                        res(
                            findLokaltForloeb(
                                epj, cpr, fid))(lokal))) ;
                P(epj, eid)
            end,

```

Der spørges efter resultater. De lokale forløb findes via hjælpefunktionen findLokaltForloeb, resultaterne hentes og sendes ud på den eksterne udkanal.

```

- →
    eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
    P(epj, eid)
end
end
else
    eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
    P(epj, eid)

```

```

    end,
    mk_Kom(
        dest, kilde, mk_Opdat(cpr, tp, fid, id, id') →

```

mk_opdat() (opdatering) Beskeden er en opdatering. Der er angivet hvor den kommer fra og hvor resultatet skal hen.

```

    if
        cpr ∈ dom epj ∧
        fid ∈ dom forl(epj(cpr))
    then
        case tp of
            Sym →
                eks_ud!mk_Res(kilde, eid, mk_OK) ;
                P(epj, eid),
            Diag →
                eks_ud!mk_Res(kilde, eid, mk_OK) ;
                P(epj, eid),
            Beh →
                eks_ud!mk_Res(kilde, eid, mk_OK) ;
                P(epj, eid),
            Res →
                eks_ud!mk_Res(kilde, eid, mk_OK) ;
                P(epj, eid)
        end
    else
        eks_ud!mk_Res(kilde, eid, mk_Fejl) ;
        P(epj, eid)
    end,
    mk_Res(dest, kilde, _) →
        eks_ud!mk_Res(kilde, eid, mk_Fejl)
end,

```

Ingen af funktionerne til opdatering er færdig implementeret.

```

hentForloeb :
    EPJId × EPJId-set × CPR →
    in eks_ind out eks_ud EForId-set
hentForloeb(eid, es, cpr) ≡
    if card es = 0 then {}
    else
        let a : EPJId • a ∈ es in
            eks_ud!
            mk_Kom(a, eid, mk_SimpelForespg(cpr, Forloeb)) ;
            case eks_ind? of

```

```

mk_Res(⟦, ⟦, mk_ForloebS(fids)) →
  {mk_EForId(fid, a) |
   fid : ForId • fid ∈ fids} ∪
  hentForloeb(eid, es \ {a}, cpr),
  _ → {}
end
end
end,

```

hentForloeb() (*hent forløb*). Hjælpefunktion der spørger til forskellige EPJ'er om forløb på patienten givet ved cpr-nummeret og returnerer en mængde af disse forløb.

```

findLokaltForloeb : VEPJ × CPR × ForId ≃ Forloeb
findLokaltForloeb(epj, cpr, fid) ≡
  forl(epj(cpr))(fid)
pre cpr ∈ dom epj ∧ fid ∈ dom forl(epj(cpr))
end

```

findLokaltForloeb (*find lokalt forløb*). Hjælpefunktion der finder et forløb i den EPJ, givet et forløbsid og et cpr-nummer.

6.3.1 XML Schema for udvekslingsformat

Udvekslingsformatets beskeder, der er defineret i modulerne InterEPJ og VEPJ ovenfor, kan umiddelbart formuleres i XML, som er en anerkendt standard for udveksling af data mellem it-systemer [3]. XML standarden definerer en meget generel syntaks for formatering af beskeder i ASCII tekst. I anvendelser af XML udvides denne syntaks til på passende vis at beskrive de data, som skal udveksles. Med XML Schema specifikationen [4] har man defineret en form for typebegreb på XML beskeder.

Ved hjælp af et XML Schema defineres strukturen af en velformet XML besked i udvekslingsformatet. Strukturen i schema'et følger strukturen i RSL specifikationen meget tæt, og navngivningen er såvidt muligt identisk. Til definition af formatet for et cpr-nummer henvises til XML Schema'et CPR_CivilRegistrationNumber.xsd, som stammer fra det danske offentlige XML standardiserings projekts NøgleData samling [8].

Det komplette schema for udvekslingsformatet er listet i appendix A.

7 Konklusion

I dette dokument er den papirbaserede patientjournal og det førstegenerations elektroniske patientjournal system blevet beskrevet. Derefter er der opstillet krav til et andengenerations EPJ system. Anden generation af EPJ systemer adskiller sig fra første generation ved, at de er forløbsorienterede i stedet for kontaktorienterede og ved, at informationen i journalen er langt mere struktureret. De nye EPJ systemer repræsenterer også et brud med tidligere praksis for patientjournaler, idet det nu er meningen, at såvel læger som sygeplejersker skal opdatere journalen.

De præsenterede modeller abstraherer fra virkeligheden på en række punkter. I kravspecifikationen er det forsøgt at medtage de vigtigste egenskaber ved G-EPJ specifikationen, herunder idéen om den forløbsorienterede journal og den hierarkiske strukturering af diagnoser og behandlinger.

Blandt de åbenbare fordele ved en elektronisk patientjournal er muligheden for at gøre journalen tilgængelig fra alle steder i sundhedsvæsenet. Således vil det ikke længere være nødvendigt at faxe eller sende en kopi af journalen med posten, når f.eks. en patient overflyttes fra et hospital til et andet, eller når den praktiserende læge skal orienteres i forbindelse med, at en af vedkommendes patienter har været behandlet på et sygehus. Det er oplagt, at dette vil kunne medføre besparelser.

Sundhedsstyrelsen anbefaler en model, hvor der etableres én central national database for EPJ data, som de forskellige leverandørers systemer kan tilgå. Denne anbefaling beror på de problemer, der vil opstå, hvis hvert amt laver deres egen database, som skal kunne udveksle information og holdes synkroniseret med alle de andre amters databaser.

Politiske hensyn gør imidlertid, at en central national EPJ database synes udelukket, givet de investeringer nogle amter allerede har foretaget. Alternativet er, at amterne hver for sig opstiller kravene til deres EPJ systemer, og da er det nødvendigt, at det sikres, at systemerne kan kommunikere indbyrdes. Sker det ikke, vil det ikke være muligt at høste det fulde udbytte af elektroniseringen.

Sundhedsstyrelsens mål er, at de EPJ systemer, som udvikles til danske amter, med tiden skal kunne udveksle informationer. Første trin er at blive enige om en fælles begrebsdannelse for kliniske processer og klinisk dokumentation. Det er netop hvad Sundhedsstyrelsen forsøger med specifikationen af G-EPJ. Næste trin er så at definere et udvekslingsformat, således at brugere fra ét EPJ system kan tilgå og ændre information, som er lagret i et andet EPJ system. Dette trin er Sundhedsstyrelsen begyndt på, men der er stadig lang vej.

Det første rigtige andengenerations EPJ system er i øjeblikket under udvikling af Systematic til Århus Amt. Dette system vil give fælles adgang til patientjournalerne fra alle amtets sygehuse. Systemet er baseret på en anden model end G-EPJ, men det vurderes, at systemet vil kunne tilbyde et grænsesnit baseret på G-EPJ modellen. Systematic har dog ikke overvejet hvordan udveksling med andre amters EPJ systemer skal foregå, ligesom der ikke er nogen mulighed for at udveksle data med de systemer, som de praktiserende læger bruger.

Systematic er også leverandør på centrale dele af Ringkjøbing Amts EPJ system. En del af amtets begrundelse for valget af Systematic er at hovedparten af de patienter, som amtets sygehus ikke selv kan tilbyde behandling, overføres til sygehuse i Århus Amt, og der derfor vil være fordele ved at lægge sig tæt op ad Århus Amts system. Det er på den baggrund bemærkelsesværdigt, at det ikke er overvejet hvordan udveksling af information skal foregå.

I anden del af denne rapport er der taget udgangspunkt i problemet med oprettelsen af en central national EPJ database. Det syntes udelukket at skabe enighed blandt alle amter om en sådan database. Der er derfor blevet udviklet et system, der gør det muligt at kommunikere mellem forskellige EPJ-systemer. Denne kommunikation vil for brugeren være transparent. Brugeren vil altså ikke lægge mærke til om hvorvidt de data han får

op på sin skærm lægger i det lokale system eller er hentet i et eksternt system. Dette udvekslingsformat kaldes V-EPJ (virtuel elektronisk patientjournal). For at kommunikation mellem EPJ systemer skal muliggøres har det været nødvendigt at udvikle et journalregister. Dette register holder styr på hvor der ligger oplysninger om hvilke patienter. Således kan et EPJ-system slippe for at lede alle eksterne systemer igennem, men blot søge i dem der ifølge registret indeholder oplysninger om en given patient.

Desuden er der udviklet en syntaks for kommunikation mellem EPJ systemer og mellem en bruger og et EPJ system. Endeligt er der blevet udviklet et grænsesnit, der giver udvekslingsformatet en operationel semantik. Dette er dog kun udført på et udsnit af de tilgængelige kommandoer. En fuldstændig implementering kræver en omorganisering af de lagrede data.

Endelig er der som det sidste trin i designprocessen blevet opstillet et XML schema, der udgør et konkret grænsesnit mellem EPJ-systemer.

Litteratur

- [1] Systematic EPJ arkitektur.
<http://www.systematic.dk/DK/Sundhed/Elektronisk+patientjournal/Arkitektur/Arkitektur.htm>.
36
- [2] S.K. Andersen, C. Nøhr, S. Vingtoft, K. Bernstein, and M. Bruun-Rasmussen. Statusrapport 2002. Technical report, EPJ Observatoriet, 2002. Rapporten kan downloades fra adressen <http://www.epj-observatoriet.dk/statusra.htm>. 37
- [3] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.0 (second edition), October 2000.
<http://www.w3.org/TR/REC-xml>. 62
- [4] W3 Consortium. XML Schema.
<http://www.w3c.org/XML/Schema>. 62
- [5] The RAISE Language Group. *The RAISE Specification Language*. BCS Practitioner Series. Prentice Hall, 1982 – check. 1
- [6] Lars Løkke Rasmussen. De politiske forventninger til EPJ-udviklingen. Tale ved EPJ Observatoriets årskonference 29.-30. oktober 2002. 35
- [7] Sundhedsstyrelsen. Grundstruktur for elektronisk patientjournal.
http://www.sst.dk/faglige_omr/informatik/epj/grundstruk/ver1.0/g-epj_ver_1.01.pdf,
Januar 2001. Version 1.01. 1, 34
- [8] Videnskabsministeriet. Offentlig Information Online (OIO).
<http://www.oio.dk/XML>. 62

A XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema targetNamespace="http://www.sst.dk/epj/"
  xmlns="http://www.sst.dk/epj/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cpr="http://rep.oio.dk/cpr.dk/xml/schemas/core/2002/06/28/">
<xs:import namespace="http://rep.oio.dk/cpr.dk/xml/schemas/core/2002/06/28/"
  schemaLocation="CPR_CivilRegistrationNumber.xsd" />
<xs:element name="Besked" type="Besked"/>

<xs:complexType name="Besked">
  <xs:choice minOccurs="1" maxOccurs="1">
    <xs:element name="Kommando" type="Kommando" />
    <xs:element name="Resultat" type="Resultat" />
  </xs:choice>
</xs:complexType>
<xs:complexType name="Kommando">
  <xs:choice minOccurs="1" maxOccurs="1">
```

```

        <xs:element name="SimpelForespg" type="SimpelForespg" />
        <xs:element name="Forespg" type="Forespg" />
        <xs:element name="Opdat" type="Opdat" />
    </xs:choice>
</xs:complexType>
<xs:complexType name="SimpelForespg">
    <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="CPR" type="cpr:CivilRegistrationNumberType" />
        <xs:element name="Type" type="Type" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Forespg">
    <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="CPR" type="cpr:CivilRegistrationNumberType" />
        <xs:element name="Type" type="Type" />
        <xs:element name="ForId" type="ForId" />
        <xs:element name="EId" type="EId" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Opdat">
    <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="CPR" type="cpr:CivilRegistrationNumberType" />
        <xs:element name="Type" type="Type" />
        <xs:element name="ForId" type="ForId" />
        <xs:element name="EId" type="EId" />
        <xs:element name="EId" type="EId" />
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="Type">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Person" />
        <xs:enumeration value="Forloeb" />
        <xs:enumeration value="Sym" />
        <xs:enumeration value="Diag" />
        <xs:enumeration value="Beh" />
        <xs:enumeration value="Res" />
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="Resultat">
    <xs:choice minOccurs="1" maxOccurs="1">
        <xs:element name="Maengde" type="Maengde" />
        <xs:element name="Vaerdi" type="Vaerdi" />
        <xs:element name="System" type="System" />
    </xs:choice>
</xs:complexType>
<xs:complexType name="Maengde">
    <xs:choice minOccurs="1" maxOccurs="1">
        <xs:element name="ForloebS" type="ForloebS" />
        <xs:element name="SymS" type="SymS" />
        <xs:element name="DiagS" type="DiagS" />
        <xs:element name="BehS" type="BehS" />
        <xs:element name="ResS" type="ResS" />
    </xs:choice>
</xs:complexType>
<xs:complexType name="Vaerdi">

```

```

<xs:choice minOccurs="1" maxOccurs="1">
  <xs:element name="PersonV" type="Person" />
  <xs:element name="ForloebV" type="ForloebV" />
  <xs:element name="SymV" type="SymNode" />
  <xs:element name="DiagV" type="DiagNode" />
  <xs:element name="BehV" type="BehNode" />
  <xs:element name="ResV" type="ResNode" />
</xs:choice>
</xs:complexType>
<xs:simpleType name="System">
  <xs:restriction base="xs:string">
    <xs:enumeration value="OK" />
    <xs:enumeration value="Fejl" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="EId">
  <xs:choice>
    <xs:element name="EForId" type="EForId" />
    <xs:element name="ESymNodeId" type="ESymNodeId" />
    <xs:element name="EDiagNodeId" type="EDiagNodeId" />
    <xs:element name="EBehNodeId" type="EBehNodeId" />
    <xs:element name="EResNodeId" type="EResNodeId" />
  </xs:choice>
</xs:complexType>
<xs:complexType name="Person">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="CPR" type="cpr:CivilRegistrationNumberType" />
    <xs:element name="Efternavn" type="xs:string" />
    <xs:element name="Fornavn" type="xs:string" />
    <xs:element name="Andet" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Forloeb">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="beskfor" type="xs:string" />
    <xs:element name="diaghi" type="InfElemDiag" />
    <xs:element name="behhi" type="InfElemBeh" />
    <xs:element name="res" type="InfElemRes" />
    <xs:element name="sym" type="InfElemSym" />
    <xs:element name="forsta" type="ForStatus" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ForloebS">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EForId" type="EForId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SymS">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="ESymNodeId" type="ESymNodeId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DiagS">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EDiagNodeId" type="EDiagNodeId" />
  </xs:sequence>
</xs:complexType>

```

```

</xs:sequence>
</xs:complexType>
<xs:complexType name="BehS">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EBehNodeId" type="EBehNodeId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ResS">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EResNodeId" type="EResNodeId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ForloebV">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="Beskriv" type="xs:string" />
    <xs:element name="ForStatus" type="ForStatus" />
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="ForStatus">
  <xs:restriction base="xs:string">
    <xs:enumeration value="aaben" />
    <xs:enumeration value="lukket" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="InfElemSym">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="SymPar" type="SymPar" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="InfElemDiag">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="DiagPar" type="DiagPar" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="InfElemBeh">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="BehPar" type="BehPar" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="InfElemRes">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="ResPar" type="ResPar" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SymPar">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="SymNodeId" type="SymNodeId" />
    <xs:element name="SymNode" type="SymNode" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DiagPar">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="DiagNodeId" type="DiagNodeId" />
    <xs:element name="DiagNode" type="DiagNode" />
  </xs:sequence>

```

```

</xs:complexType>
<xs:complexType name="BehPar">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="BehNodeId" type="BehNodeId" />
    <xs:element name="BehNode" type="BehNode" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ResPar">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="ResNodeId" type="ResNodeId" />
    <xs:element name="ResNode" type="ResNode" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SymNode">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="Beskriv" type="xs:string" />
    <xs:element name="Dato" type="Dato" />
    <xs:element name="ansvarlig" type="KliId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DiagNode">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="Beskriv" type="xs:string" />
    <xs:element name="Dato" type="Dato" />
    <xs:element name="ansvarlig" type="KliId" />
    <xs:element name="DiagStatus" type="DiagStatus" />
    <xs:element name="Grundlag" type="DiagGrundlag" />
    <xs:element name="fm" type="EDiagNodeId" minOccurs="0" maxOccurs="1" />
    <xs:element name="soesk" type="DiagSoesk" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="BehNode">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="Beskriv" type="xs:string" />
    <xs:element name="Dato" type="Dato" />
    <xs:element name="ansvarlig" type="KliId" />
    <xs:element name="BehStatus" type="BehStatus" />
    <xs:element name="Grundlag" type="BehGrundlag" />
    <xs:element name="fm" type="EBehNodeId" minOccurs="0" maxOccurs="1" />
    <xs:element name="soesk" type="BehSoesk" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ResNode">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="Beskriv" type="xs:string" />
    <xs:element name="Dato" type="Dato" />
    <xs:element name="ansvarlig" type="KliId" />
    <xs:element name="Grundlag" type="ResGrundlag" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DiagSoesk">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EDiagNodeId" type="EDiagNodeId" />
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="BehSoesk">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EBehNodeId" type="EBehNodeId" />
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="DiagStatus">
  <xs:restriction base="xs:string">
    <xs:enumeration value="aaben" />
    <xs:enumeration value="lukket" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="BehStatus">
  <xs:restriction base="xs:string">
    <xs:enumeration value="planlagt" />
    <xs:enumeration value="udfoert" />
    <xs:enumeration value="igang" />
    <xs:enumeration value="evalueret" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="DiagGrundlag">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="ESymNodeId" type="ESymNodeId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="BehGrundlag">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EDiagNodeId" type="EDiagNodeId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ResGrundlag">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="EBehNodeId" type="EBehNodeId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EForId">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="lokal" type="ForId" />
    <xs:element name="ekstern" type="EPJId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ESymNodeId">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="lokal" type="SymNodeId" />
    <xs:element name="ekstern" type="EPJId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EDiagNodeId">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="lokal" type="DiagNodeId" />
    <xs:element name="ekstern" type="EPJId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EBehNodeId">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="lokal" type="BehNodeId" />
  </xs:sequence>
</xs:complexType>

```

```

    <xs:element name="ekstern" type="EPJId" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EResNodeId">
  <xs:sequence minOccurs="1" maxOccurs="1">
    <xs:element name="lokal" type="ResNodeId" />
    <xs:element name="ekstern" type="EPJId" />
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="ForId">
  <xs:restriction base="xs:integer" />
</xs:simpleType>
<xs:simpleType name="SymNodeId">
  <xs:restriction base="xs:integer" />
</xs:simpleType>
<xs:simpleType name="DiagNodeId">
  <xs:restriction base="xs:integer" />
</xs:simpleType>
<xs:simpleType name="BehNodeId">
  <xs:restriction base="xs:integer" />
</xs:simpleType>
<xs:simpleType name="ResNodeId">
  <xs:restriction base="xs:integer" />
</xs:simpleType>
<xs:simpleType name="KliId">
  <xs:restriction base="xs:integer" />
</xs:simpleType>
<xs:simpleType name="EPJId">
  <xs:restriction base="xs:integer" />
</xs:simpleType>
<xs:simpleType name="Dato">
  <xs:restriction base="xs:date" />
</xs:simpleType>
</xs:schema>

```