A A R H U S U N I V E R S I T E T
Datalogisk Institut

Speciale

Teknisk Informationsteknologi - Distribuerede Realtidssystemer

December 2005

"Modellering og analyse af en kommunikationsprotokol til vindmøller"

Studerende, Simon Tjell (20034258)

Vejleder, Jens Bæk Jørgensen

1. Resumé/abstract

Dansk: *"Modellering og analyse af en kommunikationsprotokol til vindmøller"*

Dette speciale omhandler modellering og analyse af et kommunikationssystem, som anvendes internt i en ny generation af vindmøller fra Vestas. Kommunikationen foregår imellem en gruppe af indlejrede computere, som i en distribueret sammenhæng har ansvaret for styring og overvågning af energiproduktionen i en vindmølle. Vestas har haft problemer med stabiliteten af kommunikationssystemet, som på samme tid, er underlagt krav til realtidsegenskaber og ressourceforbrug. Der arbejdes med at identificere problemet ud fra en grundig analyse af systemet. Identifikationen af problemet efterfølges af et forslag på en mulig løsning. Efterfølgende modelleres løsningsforslaget og systemet i helhed vha. Farvede Petri Net. Modellen anvendes som grundlag for en efterfølgende analyse og evaluering af løsningsforslaget. Specialet afsluttes med en evaluering af den modelbaserede analyse og en perspektivering af den anvendte metode i forhold til alternative metoder.

Engelsk: *"Modeling and analysis of a communication protocol for windmills"*

This thesis describes the process of modeling and analysis of a communication system, which is applied internally in a new generation of windmill from Vestas. The communication takes place within a group of embedded computers, which are connected in a distributed system. The embedded computers have the responsibility of controlling and monitoring the production of energy in a windmill. At Vestas, stability problems have been experienced of the communication system, which, is subject to requirements for both the real time properties and the resource consumption at the same time. The work on the thesis is initiated by identifying the problem based on a thorough analysis of the system. The identification of the problem is followed by a proposal to a solution to the problem. The proposed solution is modeled along with the system in general. The model is developed using Coloured Petri Nets. The model is used for analyzing and evaluating the proposed solution. The thesis is concluded by a discussion about the method being used for modeling and conducting the analysis.

2. Indholdsfortegnelse

3. Indledning

Oliekrisen i begyndelsen af 1970erne medførte en øget efterspørgsel på alternative energiformer. Den øgede efterspørgsel dannede grundlag for investeringer i bl.a. vindmølleindustrien, som i de efterfølgende årtier har oplevet en kraftig vækst [VMI1]. Samtidig har ønsket om en mere miljørigtig energiproduktion bidraget til gode politiske og økonomiske kår for væksten. Det stadigt voksende marked for vindenergi har resulteret i en løbende udvikling af vindmøllerne og den anvendte teknologi. Der udvikles stadigt større og mere effektive vindmøller [VMI2]. Rundt om i verden etableres store vindmølleparker, og samtidig bliver den enkelte vindmølle en mere og mere kompleks maskine. Overvågningen og styringen af energiproduktionen forfines for at sikre bedst mulig effektivitet. En moderne vindmølle reguleres vha. avancerede computersystemer, som reagerer ud fra indsamlede målinger fra flere hundrede sensorer. Som et resultat af den løbende videreudvikling af vindmøllerne, bliver også disse computersystemer stadigt mere komplekse.
Det danske firma Vestas Wind Systems [VWS] er en af de største producenter af vindmøller på verdensmarkedet. Hos Vestas oplever man også en stigende grad af kompleksitet indenfor de systemer der anvendes til styring og overvågning af vindmøllerne. Det stiller store krav til udviklingen af den software der varetager styringen.

3.1. Software i vindmøller

Hos Vestas udvikler man avancerede distribuerede computersystemer til styring og overvågning af firmaets vindmøller. Disse systemer bliver gradvist mere komplekse efterhånden som vindmøllerne vokser i størrelse og kapacitet. Forøgelsen af kompleksitet sker med det formål at forbedre mulighederne for optimering af energiproduktionsprocessen. Stigningen i kompleksitet viser sig i form af et stigende udvalg af målbare fysiske egenskaber vedrørende vindmøllens aktuelle tilstand, som inddrages i den løbende styring.
Reguleringssystemet i en moderne vindmølle hos Vestas består af tre eller flere noder (indlejrede computere), som via et netværk samarbejder om at løse opgaven med at bearbejde de indsamlede måledata og håndtere den løbende styring af vindmøllen. Noderne indgår i et tætforbundet distribueret system, hvor fælles adgang til distribuerede variable muliggør samarbejdet om løsningen af opgaven. Indsamlede måledata tilgås som delte parametre i de enkelte noder, og det samlede system indeholder over 2000 af denne slags variable. Endvidere er den fysiske indsamling af data fordelt ud over flere forskellige noder. Hver enkelt node har på den måde direkte lokal adgang til forskellige

delmængder af den samlede gruppe variable, mens andre variable hentes via netværket fra de andre noder i systemet. Nogle variable aflæses samtidigt af flere noder, mens andre fungerer som bindeled imellem reguleringsalgoritmer i systemet. Da de indsamlede måledata repræsenterer systemets billede af den omgivende fysiske verden, optræder der ofte stramme krav til realtidsegenskaberne for dels reguleringen i sig selv, og dels den kommunikation der danner grundlag for den nødvendige udveksling af variable imellem noderne. Kravene er nødvendige for at sikre, at reguleringsalgoritmerne til enhver tid opererer på en aktuelt og konsistent billede af omgivelserne. Det er således nødvendigt at sikre en tilpas høj opdateringsfrekvens af variablerne i systemet for at kunne garantere korrekt funktion af de forskellige reguleringsalgoritmer.

3.2. Design af kommunikationssystemet

Den beskrevne arkitektur for det distribuerede system stiller store krav til rettidighed og generel konsistens i forbindelse med udveksling af variable. Der er flere forskellige egenskaber ved systemet, som vanskeliggør designprocessen i forbindelse med udvikling af et sikkert medium for udveksling af variable imellem noderne. Vindmøllens mekanik og generator giver et elektrisk set meget støjfyldt miljø for noderne, hvilket potentielt resulterer i større problemer med indstråling og medførende forvanskning eller tab af data. Endvidere indgår systemets noder i et meget dynamisk og komplekst samspil, som gør det svært at forudsige spidsbelastningssituationer og opstille kriterier for forekomsten af disse.

I Vestas' nyeste generation af vindmøller anvendes et applikationsframework ved navn Distributed and Active Objects (DAO). DAO er grundlaget for udvikling af de applikationer, som afvikles på systemets noder. Frameworket er udviklet hos Vestas med fokus på en række domænespecifikke krav til realtidsegenskaber for vindmøllesystemer.

I det typiske scenarium vil flere noder dele en delmængde af variablerne i systemet. Som en direkte følge af arkitekturen i DAO-frameworket, er kommunikation imellem noderne en relativt tidskrævende opgave i forhold til forbrug af processeringsressourcer i den enkelt node. Dette skyldes flere aspekter i arkitekturen; bl.a. kompleksiteten af den anvendte protokolstak, fejlsikringsalgoritmer i protokolstakken og den løbende komponentskedulering.

For at hindre overbelastning og medførende ustabilitet som følge af overforbrug af processeringskraft, er det nødvendigt at anvende multicast-kommunikation med det formål at begrænse antallet af udvekslede beskeder imellem noderne mest muligt. Ved multicast-kommunikation forstås generelt en kommunikationsform, hvor en besked kan sendes til en udvalgt gruppe af modtagere på én gang [MULT]. Målet med denne form for kommunikation er at begrænse den processeringstid, en given node anvender til at ved-

ligeholde mængden af variable. Motivationen for brug af multicast-kommunikation er baseret på den forudsætning, at en stor del af systemets variable vil blive tilgået af flere end to noder. I modsat fald, er multicast-kommunikation ikke en god løsning. Kombinationen af skrappe realtidskrav, multicast-kommunikation imellem flere samtidigt kommunikerende noder og ønsket om begrænsning af mængden af udvekslede beskeder imellem noderne stiller store krav til kommunikationssystemet. Det eksisterende system til udveksling af parametre imellem noderne via DAO-frameworket har vist sig utilstrækkeligt på flere områder:

- Manglende forudsigelighed i forhold til realtids- og konsistensegenskaber
- Begrænsede muligheder for at estimere ressourcebelastning under designet af et system
- Ikke tilstrækkelig fejltolerance i forhold til bl.a. mistede beskeder
- Manglende mulighed for identifikation af overbelastningssituationer
- Grovkornet dimensionering af systemet – ud fra udetaljerede worst-case scenarier

Vestas har et ønske om at få udviklet et nyt design af det kommunikationssystem, der danner grundlag for DAO-frameworket og samtidig opnå en mere præcis indsigt i dette systems virkemåde. Det sidste ønske bliver stadigt mere aktuelt når kompleksiteten af det samlede system stiger. Der ønskes endvidere en stærkere grad af mulighed for vurdering af egenskaberne for et givent kommunikationssystem. Den gængse fremgangsmåde baserer sig på tidlige antagelser, som efterprøves vha. prototypeimplementationer. Et af problemerne ved denne fremgangsmåde er, at det ofte ikke giver noget praktisk grundlag for analyse af kommunikationssystemets dynamiske egenskaber. Denne problemstilling er arketypisk for systemer af denne type, hvorfor arbejdet med metoder til analyse og modellering, kan forventes at kunne anvendes i forbindelse med andre systemer med lignende arkitektur.

3.3. Specialets formål

Målet med dette speciale er at levere et forslag til en løsning af den beskrevne problemstilling. Egnetheden af den foreslåede løsning evalueres herefter med udgangspunkt i analyser på baggrund af modeller. Disse analyser skal gøre det muligt at vurdere løsningsforslagets egenskaber i forhold til realtidskravene og de generelle begrænsninger i forbindelse med systemets ressourceforbrug. Analyserne baseres på resultater af automatiserede og systematiske eksperimenter udført vha. de beskrevne modeller.
Vestas bidrager med oplysninger om og erfaringer med det eksisterende system samt oplæg til krav til egenskaberne for et kommunikationssystem der kan anvendes sammen

med DAO-frameworket. Disse krav vil være de overordnede forudsætninger for arbejdet med udvælgelse af et egnet design for kommunikationssystemet. Opnåelse og dokumentation af erfaring i forbindelse med anvendelse af modeller til analyse af realtidskritiske systemer er centralt i arbejdet med specialet. Endvidere repræsenterer specialet en praktisk tilgang til processen i forbindelse med anvendelse af modeller til analyse af en kommunikationsprotokol, som skal anvendes i et eksisterende implementeret system.
Den omtalte modellering udføres vha. en teknik kaldet Coloured Petri Nets (CPN) [CPN1], som er en udvidelse af en ældre teknik kaldet Petri Nets [PETRI]. CPN er en metode til modellering og eksekvering af modeller, som er anvendelig i en række forskellige sammenhænge, heriblandt protokolanalyse og -verifikation. Aarhus Universitet huser en forskningsgruppe indenfor området [CPNGR]. Denne gruppe har bl.a. arbejdet med udvikling af det generiske modelleringsværktøj CPN Tools [CPN2]. Værktøjet anvendes til modellering, simulering og verificering af systemer vha. CPN. Dette værktøj vil blive anvendt til modelleringen og analyse af protokoller undervejs i specialet. Værktøjet muliggør analytisk evaluering af modeller på baggrund af en eksekvering af disse.
Efter at have vist hvordan modellen danner grundlag analyse af løsningsforslaget indledes en mere generel diskussion omkring den anvendte metode. Diskussionen fokuseres omkring en række generelle kriterier for kvalitetsvurdering af modelleringsmetoder. Diskussionen afrundes med en sammenligning mellem den anvendte metode og to alternative modelbaserede metoder.

3.4. Specialets struktur

- Specialet er struktureret på følgende måde: Efter dette afsnits indledning, beskrives DAO med fokus på kommunikationen imellem noder. Herefter gives en beskrivelse af principperne bag signaleringsprotokoller, som er en særlig type multicast-protokoller. I næste afsnit argumenteres der for, at netop en signaleringsprotokol kunne være relevant til løsning af den aktuelle problemstilling. Herefter udvælges en passende type signaleringsprotokol og denne protokol danner grundlag for den efterfølgende CPN-modellering. Forud for selve beskrivelsen af modellen, gives en introduktion til principperne indenfor CPN. CPN-modellen af løsningsforslaget bruges til at indsamle måledata, som anvendes i næste afsnits analyse af løsningsforslagets egenskaber. Arbejdet med den modelbaserede analyse af løsningsforslaget afsluttes med en diskussion omkring den anvendte metode med udgangspunkt i en række generelle egenskaber. I samme afsnit sammenlignes den anvendte metode med to alternative metoder til modelbaseret analyse. Specialet afsluttes med et konkluderende afsnit, hvor resultaterne opsummeres.

3.5. Læserens forudsætninger

Det forudsættes at læseren af dette speciale har en generel faglig forståelse for kommunikationsprotokoller og distribuerede systemer. Der anvendes et mindre antal eksempler på kildekode. Derfor er det en fordel, at have en overordnet forståelse af de anvendte programmeringssprog (C++ og Standard/CPN ML). De steder, hvor det vurderes nødvendigt, vil der blive henvist til uddybende litteratur. Der forudsættes ikke forudgående kendskab til vindmølledomænet eller CPN.

4. Vindmøllesoftware hos Vestas

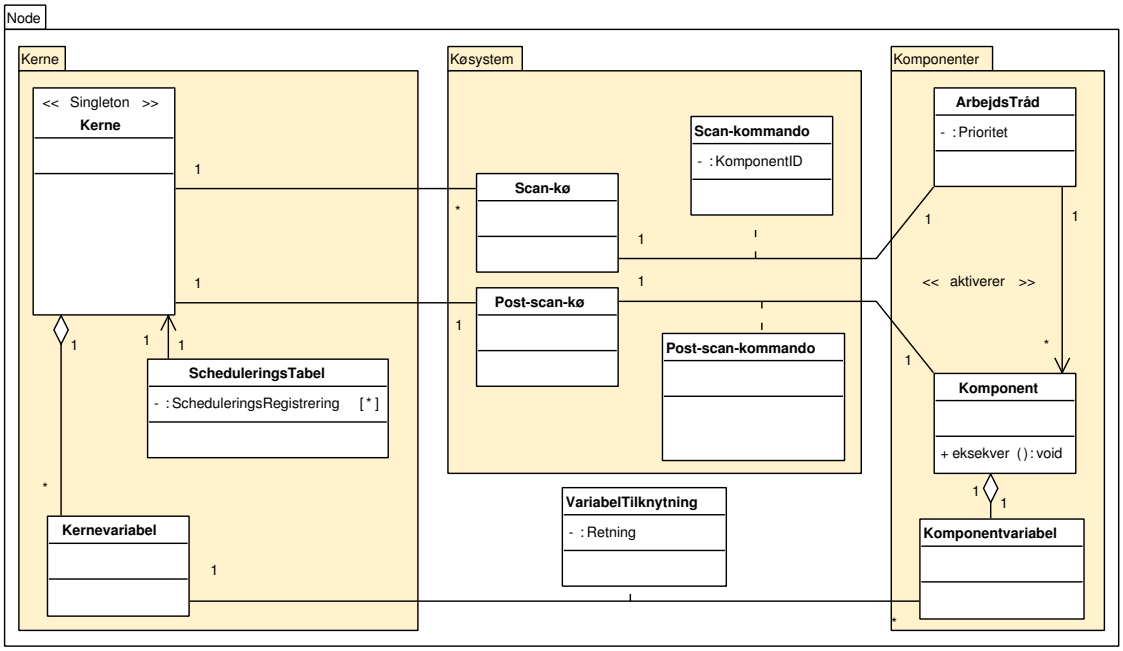
Der er holdt en række møder med Jesper Schmidt fra Vestas, som er ansvarlig for design og udvikling af DAO-frameworket. Formålet med møderne var i første omgang at opnå forståelse for designet bag DAO og dernæst at få beskrevet de problemstillinger i forhold til multicast-kommunikationen, som vil blive behandlet efterfølgende. Dette afsnit beskriver DAO med fokus på kommunikationen imellem noderne. Afsnit 4.1 beskriver DAOs opbygning og virkemåde og afsnit 4.2.14 beskriver problemstillingen.
Vestas har haft dette afsnit til gennemlæsning i flere forskellige versioner og har igennem løbende tilbagemeldinger bidraget til at gøre beskrivelsen af DAO så præcis som muligt. Den endelige version er godkendt af Jesper Schmidt. Som supplement til denne beskrivelse indeholder afsnit 13.2 en opsummering af en lang række af de konkrete oplysninger om DAO, som er indsamlet i forbindelse med samarbejdet med Vestas.

4.1. Distributed and Active Objects (DAO)

Vestas har udviklet et komplet applikationsframework (DAO), som danner fundament for udviklingen af de applikationer, som varetager overvågning og styring af vindmøllerne. DAO-frameworket er designet, så det passer til den distribuerede natur, som er kendetegnende for størstedelen af de applikationer der udvikles til afvikling internt i vindmøllerne. I en typisk vindmølleapplikation indgår en række distribuerede reguleringsalgoritmer, som genererer output til aktuatorer på baggrund af input fra sensorer. Disse algoritmer opererer på flere forskellige logiske niveauer og nogle af dem deles om inputvariable mens andre fungerer som led i en kæde af reguleringsalgoritmer, hvor delte variable udgør bindeled i kæden. DAO-frameworket håndterer bl.a. al udveksling og opdatering af de variable som anvendes til input og output af de softwarekomponenter, som implementerer reguleringsalgoritmerne. Desuden muliggør frameworket en dynamisk opbygning af arkitekturen i det distribuerede system, således at reguleringssløjfer kan til- og frakobles i et kørende system. Denne egenskab gør det samtidig muligt at tilkoble monitoreringsudstyr til systemet i en fungerende vindmølle i marken i forbindelse med optimering og fejlsøgning.
Som beskrevet tidligere, indeholder en vindmølle typisk tre eller flere indlejrede systemer, som arbejder sammen via et netværk. Hver af disse noder eksekverer en instans af DAO. Det centrale i en DAO-instans er DAO-kernen, som har ansvar for schedulering og aktivering af de komponenter der eksisterer på den enkelte node. En komponent er en software-implementation af en delmængde af den samlede applikations funktionalitet. Komponenter eksekveres periodisk af DAO-kernen.

4.2. Bestanddele i DAO

På Figur 1 illustreres den overordnede arkitektur DAO-frameworket og de komponenter der afvikles på en fysisk node. Som det ses, består systemet på en enkelt node grundlæggende af én DAO-kerne og en gruppe komponenter. Desuden findes et køsystem som anvendes til kommunikation imellem DAO-kernen og nodens komponenter. De enkelte elementer beskrives i de følgende afsnit.



Figur 1 - DAO-frameworkets arkitektur

4.2.1. Komponenter

En komponent i DAO er et objekt, som har en eksekveringsmetode, der kan aktiveres periodisk. Al funktionalitet i en styringsapplikation er implementeret i komponenter, som kan være distribueret over flere fysiske noder. En komponent er tilknyttet en række af systemets globale variable – i form af output- og/eller input-tilknytning. En komponent kan påvirke systemets tilstand igennem de variable, som komponenten måtte have tilknyttet som output. Samtidig har en komponent adgang til at læse systemets globale tilstand via de variable, som komponenten har tilknyttet som input-variable. En komponent afvikles fuldstændig afkapslet fra systemets øvrige komponenter, og grænsefladen imellem komponenter er baseret på deling af variable.

4.2.2. Kerne

DAO-kernen er den centrale del af DAO-frameworket på hver de fysisk adskilte noder. Kernen har en række vigtige funktioner, som bla. omfatter kommunikation med DAO-kerner på andre node, periodisk aktivering af nodens komponenter og opdatering af nodens lokalt lagrede variable. Grundlæggende er DAO-kernen ansvarlig for at afvikle komponenter og håndtere de variable der anvendes af komponenterne. Kernen gennemløber periodisk en fast cyklus af opgaver, som beskrives i detaljer i afsnit 4.2.11.

4.2.3. Scheduleringstabel

DAO-kernen anvender en scheduleringstabel, som beskriver hvilke komponenter der skal aktiveres periodisk og aktiveringens periodetid. Denne tabel opdateres dynamisk, når komponenter bliver initialiseret på noden.

4.2.4. Kerne- og komponentvariable

I DAO-frameworket foregår kommunikationen igennem en stor samling variable. Som nævnt kan komponenter anvende en variabel til output, input eller begge dele. To komponenter kan således udveksle data igennem en parameter. Der eksisterer typisk flere tusinde variable i et distribueret system i en vindmølle, og de variable indeholder meget forskellige typer data; måledata, setpunkter for reguleringsalgoritmer, konfigurationsdata osv. Der skelnes imellem to forskellige typer variable:

- Kernevariable – denne type variabel indeholder den aktuelle værdi for variablen. Hver variabel af denne type eksisterer i én instans på hver af de noder, hvor der findes komponenter som har tilknytning til variablen.

- Komponentvariable – dette er en lokal kopi af indholdet af den en kernevariabel. De lokale kopier eksisterer internt i komponenter i form af én instans for hver variabel, som komponenten har tilknytning til.

Når DAO-kernen gør klar til at aktivere en komponent, kopieres indholdet af alle komponentens tilknyttede variable fra samlingen af kernevariable til komponentens interne samling af komponent-variable. Denne operation kaldes *latching* og beskrives nærmere i afsnit 4.2.7. Udover at kunne fungere som bindeled imellem to kommunikerende komponenter, repræsenterer systemets variable også komponenternes grænseflade til de fysiske omgivelser. Det sker i kraft af at en variabel kan indeholde aflæsninger fra en

fysisk sensor eller output-værdier til en fysisk aktuator. Forbindelsen imellem en variabel og enten en sensor til input eller en aktuator til output sker altid igennem én enkelt komponent. Det vil sige, at én komponent har ansvaret for enten at aflæse sensoren eller skrive til aktuatoren. Alle andre komponenter kan tilgå de fysisk forbundne variable på samme måde som alle andre variable i systemet.
Der findes desuden en variant af variable som kaldes parametre. Disse anvendes til at konfigurere systemet, og kan tilgås fra komponenterne på samme måde som variable - dog kun i form af input-tilknytninger. Parametrene fungerer således som konstanter, set fra komponenterne.

4.2.5. Arbejdstråde

Den aktive eksekvering af komponenter i DAO foregår i en pulje af tråde. Trådpuljen indeholder et antal aktive klasser (arbejdstråde), som eksekveres uafhængigt af hinanden i hver sin tråd. Arbejdstrådene i trådpuljen har indbyrdes forskellige prioriteter som alle er højere (dvs. lavere prioriteret) end prioriteten for den tråd der afvikler DAO-kernen. Arbejdstrådene er sammen med DAO-kernen de eneste aktive klasser i DAO-frameworket. De har ansvar for at eksekvere komponenter på baggrund af DAO-kernens schedulering af disse.

4.2.6. Køsystem og kommandoer

Arbejdstrådene kommunikerer med DAO-kernen igennem et system af FIFO-køer. Formålet med køsystemet er, at afkoble aktiveringen af komponententer fra scheduleringen i DAO-kernen, som afvikles i sin egen tråd. Kommunikationen består af to typer beskeder:

- **Scan-kommandoer:** Denne type beskeder placeres i scan-køerne af DAO-kernen og aftages herfra af arbejdstrådene. En scan-kommando giver arbejdstråden besked om at eksekvere en specifik komponent via dennes eksekveringsmetode. Arbejdstråden venter herefter på at komponentens eksekveringsmetoder afsluttes, før der hentes en ny scan-kommando fra den tilknyttede scan-kø. Hver enkelt arbejdstråd er tilknyttet én scan-kø hvorfra den modtager kommandoer fra DAO-kernen. Scan-kommandoen indeholder en ID for den komponent der skal eksekveres. DAO-kernens valg af scan-kø til placering scan-kommandoer hænger sammen med DAO-frameworkets prioriterings- og scheduleringspolitik, som beskrives i detaljer i afsnit 4.2.8.

- **Post-scan-kommandoer: N**år en komponent afslutter sin eksekveringsmetode, placeres en post-scan-kommando i post-scan-køen. I modsætning til scan-køerne findes der kun én post-scan-kø, som er fælles for alle komponenter. Post-scan-kommandoerne indeholder information om afsluttede komponenter. Som beskrevet tidligere, opererer komponenter under afviklingen på komponentvariable, som er kopier af det globale indhold af variablerne. Post-scan-kommandoen fra en bestemt komponent, identificerer komponenten og signalerer, at komponenten er afsluttet. Når en komponent er afsluttet, kan indholdet af komponentens komponentvariable kopieres tilbage til de tilsvarende kernevariable. På den måde opdateres evt. ændrede værdier. En post-scan-kommando indeholder således oplysninger om, at der er sket ændringer i komponentvariable, som kan overføres på de globale kernevariable.

- **Attach/detach-kommandoer:** Attach- og detach-kommandoer placeres ligeledes i den delte post-scan-kø, og anvendes når komponenter tilknytter eller frakobler sig adgang til systemets variable. Attach-kommandoer indeholder en udpegning af variablen, en ID for komponenten og en angivelse af tilknytningens retning: Input (I) eller input/output (IO).

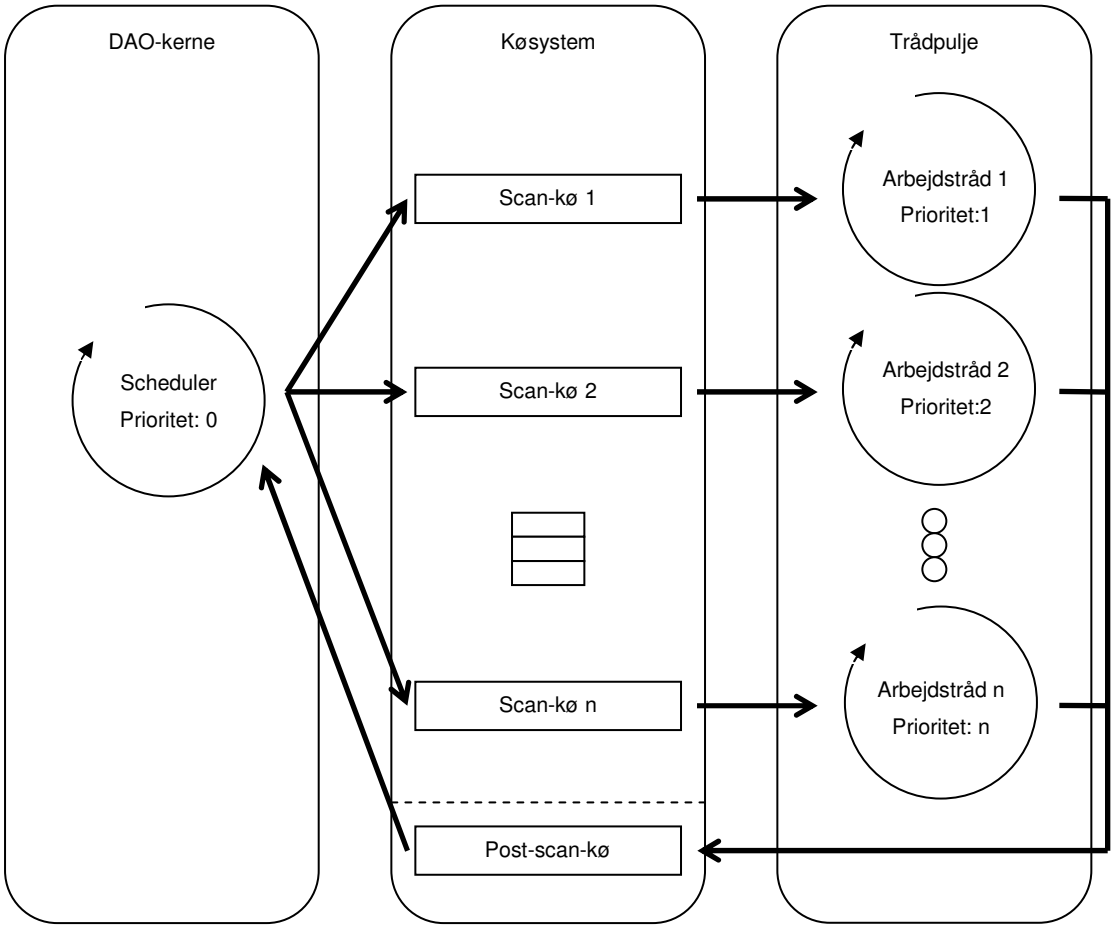
4.2.7. Latching

For at sikre at en komponent arbejder på et konsistent billede af de variable den har adgang til, foretages en såkaldt *latching*-operation inden komponenten aktiveres. Operationen dækker over en kopiering af alle de variable, som komponenten har tilknyttet som input-variable. Indholdet af variablene kopieres fra hver enkelt variabels repræsentant i gruppen af kernevariable til dennes modsvarende repræsentant i gruppen af komponent-variable hos den aktuelle komponent. Denne operation lader en komponent arbejde på et fastfrosset billede af de tilknyttede variable. Det er i denne sammenhæng vigtigt at bemærke at der, som tidligere beskrevet, altid kun er én komponent som har tilknyttet en given variabel til output - mens en flere komponenter kan have tilknyttet den samme variabel til input.

4.2.8. Aktivering af komponenter

Som beskrevet håndteres aktiveringen af komponenter af DAO-kernen. Hver enkelt komponent på en node er tildelt en statisk prioritet. Denne prioritet er valgt på baggrund

af komponentens eksekveringsperiode og afviklingstid. Prioriteten for en komponent er bestemmende for i hvilken scan-kø, DAO-kernen placerer den scan-kommando som får en arbejdstråd i trådpuljen til at aktivere komponenten. Da hver arbejdstråd kun aftager scan-kommandoer fra en enkelt scan-kø og ingen arbejdstråde modtager scan-kommandoer fra den samme scan-kø, fungerer DAO-kernens udvælgelse af scan-kø som en mekanisme til at sikre at den vedtagne scheduleringspolitik overholdes. Den anvendte scheduleringspolitik kaldes *Rate Monotonic Scheduling* [RMS] og beskrives i afsnit 4.2.9. Da komponenternes prioritet i forhold til hinanden er fastlagt statisk og fast ved kompileringstid, vil scan-kommandoer til aktivering af en given komponent altid blive placeret i en bestem scan-kø – og vil altid blive aftaget af en bestemt arbejdstråd.
Figur 2 illustrerer DAO-kernens anvendelse af en gruppe scan-køer til udvælgelse af arbejdstråde.



Figur 2 - Beskedtrafikken på en DAO-node

De fremhævede pile på Figur 2 viser hvordan scan- og post-scan-kommandoer cirkulerer.

4.2.9. Princippet bag Rate Monotonic Scheduling

Rate-monotonic scheduling (RMA) beskrives i [RMS] og er en scheduleringsalgoritme der baserer sig på statisk tildeling af prioriteter til et sæt af periodiske tasks. Begrebet *statisk* betyder i denne sammenhæng, at prioriteterne tildeles offline - dvs. før programmet kompileres – i modsætning til en dynamisk algoritme, hvor prioriteterne ændres løbende mens programmet afvikles. I forbindelse med DAO-frameworket udgøres de tasks der scheduleres af komponenterne på en enkelt node. Som beskrevet tidligere aktiveres komponenternes eksekveringmetoder periodisk og med forskellig frekvens. Aktiveringen udføres af arbejdstrådene i trådpuljen og komponenter kan prioriteres forskelligt ved at sende scan-beskeder igennem forskellige scan-køer i køsystemet. Det grundlæggende princip i RMA bygger på at tildele prioriteter ud fra periodetid for de tasks der skal scheduleres. Ordet *monotonic* dækker over, at tildelingen sker ud fra en monoton funktion af periodetiden. Tildelingen sker således at tasks med højest eksekveringsfrekvens (og dermed lavest periodetid) prioriteres højest. Anvendelsen af RMA forudsætter at tasks er i stand til at afbryde hinanden. Operativsystemet skal altså understøtte preemptive scheduling således at en igangværende task kan afbrydes af en højere prioriteret task.
I [RMS] bevises det, at RMA er en optimal algoritme for scheduling af periodiske tasks. Der føres bevis for, at ingen anden statisk scheduleringsalgoritme kan schedulere et task-sæt, som ikke kan scheduleres vha. RMA. Desuden defineres et mål for den maksimalt mulige udnyttelse af den tilrådeværende processeringstid for et givet sæt af periodiske tasks. Denne udnyttelsesgrad falder, desto større sættet bliver. Det skyldes grundlæggende, at jo flere tasks der skacheduleres, jo sværere bliver det at periodetiderne til at passe sammen og dermed udnytte processeringstiden.

4.2.10. Et konkret eksempel på en komponent

Kildekoden i Figur 3 viser implementationen af en simpel komponent i DAO. Komponenten er et tænkt eksempel på implementation af et speedometer, som udregner hastighed ud fra acceleration. Det ses hvordan komponenten er tilknyttet to variable *acceleration* og *velocity* som anvendes til henholdsvis input og output. De to variable eksisterer i form af komponent-variable (*a* og *v*) intern i klassen og disse kobles vha. *attach*-metoden til systemets kerne-variable. De kerne-variable, som de to komponent-variable knyttes til, udpeges vha. af deres tekstuelle navne: *"acceleration"* og *"velocity"*.

```
class speedometer : public component<speedometer>{
  private:
    si::time dt;
    input<si::acceleration> a;
    output<si::velocity> v;

  public:
    void scan(){
      v += a * dt;
    }

    speedometer(ini&){
      dt = 10 * si::msec();
      rate(dt);
      attach(a, ”acceleration”);
      attach(v, ”velocity”);
    }
};
```

Figur 3 - Eksempel på kildekode for DAO-komponent

I constructor-metoden ses det, hvordan komponenten registrerer sig hos DAO-kernen vha. *rate*-metoden. I forbindelse med registreringen angiver *dt*-parameteren, hvor ofte komponenten skal aktiveres (*dt* angiver eksekveringsperioden). Efter registreringen vil DAO-kernen for hver *dt* millisekunder aktivere komponenten vha. af køsystemet og trådpuljen. Når en arbejdstråd modtager en scan-kommando fra DAO-kernen via scan-køen, vil arbejdstråden aktivere komponenten ved at kalde *scan*-metoden på det objekt der instantierer komponentens klasse.

4.2.11. Eksekveringscyklus i DAO-kernen

DAO-kernen gennemløber kontinuerligt en cyklus af handlinger, hvoraf de vigtigste er beskrevet i de foregående afsnit. Her beskrives den samlede sekvens af handlinger, sammen med en beskrivelse relationerne til resten af systemet:

1) Håndtering af indkommende kommunikation

DAO-kernen modtager indkommende beskeder fra systemets øvrige DAO-kerner (dvs. DAO-kerner placeret på andre noder) via netværket som forbinder noderne. Disse beskeder indeholder to typer information:
- Information om ændringer i variabel-værdier. Beskeden optræder når variable i systemet har ændret værdi, og viden om denne ændring har relevans for komponenter på den modtagende node. Dette er tilfældet hvis en komponent på den modtagende node har tilknyttet variablen som input.
- Attach/detach-kommandoer. Denne type beskeder optræder som indkommende kommunikation, når den variabel kommandoen henviser til eksisterer på den modtagende node - men komponenten eksisterer på en anden node.

2) Håndtering af post-scan-kommandoer
DAO-kernen aftager post-scan-kommandoer fra post-scan-køen og opdaterer om nødvendigt indholdet af de lokale kerne-variable ud fra værdierne af komponentvariable.

3) Håndtering af attach/detach-kommandoer
DAO-kernen håndterer de beskeder om tilknytning til variable, som er indkommet fra både den indgående kommunikation og den lokale post-scan-kø.

4) Latching
DAO_kernen kopierer indholdet af de kerne-variable der benyttes som input i de lokale komponenter over i de tilhørende komponent-variable.

5) Generering af scan-kommandoer
Ud fra registreringer i scheduleringstabellen, generer scan-kommandoer som fører til aktivering af de komponenter der skal aktiveres i denne cyklus. Scheduleringstabellen indeholder de nødvendige informationer, så DAO-kernen ved hvornår hver enkelt komponent skal aktiveres næste gang.

6) Håndtering af event-kommunikation
Udover variabel-systemet har DAO-kernerne mulighed for at sende og modtage event-provokerede beskeder imellem noderne. Beskederne anvendes bl.a. i forbindelse med opsætning af systemet.

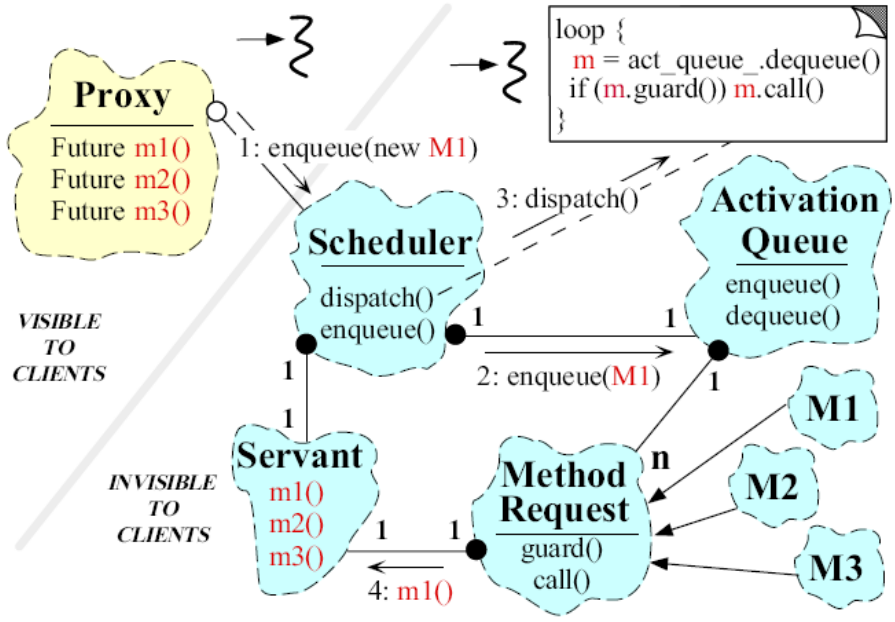
7) Håndtering af udgående kommunikation
DAO-kernen videresender information om opdateringer af lokale kerne-variable, som har relevans for andre noder i systemet. Informationen har relevans for andre noder, hvis disse noder indeholder komponenter, som har tilknyttet sig de ændrede variable som

input. Den udgående kommunikation omfatter også videresending af attach-kommandoer, som lokale komponenter har placeret i post-scan-køen for at tilslutte sig en variabel, som ikke eksisterer lokalt på samme node som komponenten. Hvis den givne variabel ikke eksisterer lokalt på noden, vil DAO-kernen videresende attach-kommandoen til de andre DAO-kerner.

8) Gennemløb ny cyklus fra punkt 1

4.2.12. Sammenligning med kendt design pattern

Det princip der er anvendt i forbindelse med designet af DAO, minder på mange måder om et designmønster ved navn Active Object (AO), som beskrives i [AO]. Figur 4 viser de klasser, der indgår i mønstret.



Figur 4 - Klasser i Active Object-mønstret. Kilde: [AO]

AO bygger på et princip om at adskille invokering og eksekvering af metoder. En klient-klasse har adgang til en række objekter igennem et objekt af typen Proxy, som repræsenterer interfacet til de aktive objekter. Når et klient-objekt invokerer en metode hos et af de aktive objekter via et kald til en metode i et Proxy-objekt, placerer Proxy-objektet et objekt af typen MethodRequest i en kø - ActivationQueue. MethodRequest-objekterne opsamles fra ActivationQueue af et objekt af typen Scheduler. Scheduler-objektet har

herefter ansvaret for at aktivere metoderne i det egentlige objekt - altså det aktive objekt, som er indkapslet. Disse metoder ligger placeret i klassen Servant. I forbindelse med klient-objektets kald af metoden i Proxy, oprettes der et objekt af typen Future, hvortil Servant-objektets metode skriver resultatet af eksekveringen af metoden. Future-objektet oprettes umiddelbart invokeringen - i modsætning til efter eksekveringen, som er det normale tidspunkt for returnering af resultat. Klient-objektet kan herefter vente på resultatet i Future-objektet asynkront (vha. polling) eller synkront (vha. blokerende kald).
Med udgangspunkt i figuren og den forudgående beskrivelse af DAO følger her en kort gennemgang af de mest grundlæggende forskelle imellem mønstret og designet i DAO:

Egenskab	**AO**	**DAO**
Implementation	I AO implementeres applikationen i en række klient-klasser, som anvender aktive objekter.	I DAO implementeres selve applikationen i de komponenter som i grove træk modsvarer de aktive objekter i AO.
Identifikation af komponenter	I AO anvendes MethodRequest-objekter til at udpege den metode der skal eksekveres i det aktive objekt.	I DAO sker udpegningen på komponentniveau vha. en komponent-id.
Metodetyper	I AO kan alle typer objekter omskrives til at indgå som aktive objekter i mønstret.	I DAO har komponenterne et fast interface, og kun en metode der kan aktiveres; *scan*.
Opsamling af aktiveringsbeskeder	I AO anvendes et objekt af typen Scheduler til at opsamle aktiverings-beskeder.	I DAO sker denne opsamling i trådpuljen.
Formidling af resultat	I AO placeres resultatet af en eksekvering i et Future-objekt.	I DAO påvirker komponenterne systemets tilstand via komponentvariable, som efter eksekveringen af scan-metoden overføres til kernevariable.
Prioritering	I AO ligestilles alle invokeringer af metoder.	I DAO håndteres prioritering af komponenterne vha. en scheduleringspolitik; Rate Monotonic Scheduling.

Tabel 1 - Sammenligning af designmønstret AO og designet i DAO

AO anvendes blandt andet i JAWS, som er et framework til brug i forbindelse med udvikling af webservere [JAWS].
I [AO] foreslås en udvidelse af mønsteret i form af distribution af systemets elementer. Denne model svarer dog ikke til den måde DAO er designet på. I [AO] bygger idéen om distribuerede aktive objekter på en stub-skeleton-arkitektur, hvor klient-objekter og aktive objekter kan eksistere på forskellige fysiske noder. I dette tilfælde er det Proxy-klassen som udvides og opdeles i en stub-klasse, som befinder sig på samme node som klient-objekterne og en skeleton-klasse, som befinder sig på samme side som de aktive objekter - dvs. resten af mønstrets klasser. Stub-klassen har ansvar for marshalling af MethodRequest-objekter, således at de kan sendes over netværk til skeleton-klassen, som ved modtagelsen af objekter vil stå for demarshalling og placering af objekterne i ActivationQueue.

4.2.13. Kommunikationslaget i DAO

Når en DAO-kerne på én node kommunikerer med en DAO-kerne på en anden node, foregår kommunikationen via et LAN-netværk. DAO-kernerne tilgår dette netværk for at modtage og afsende beskeder via en protokolstak. Der indgår ikke routere i forbindelsen imellem noderne og alle noder har den samme direkte adgang til kommunikationsmediet via protokolstakken. DAO-kernens grænseflade til protokolstakken udgøres af transportlagsprotokollen User Datagram Protocol (UDP) [UDP]. UDP er en meget simpel protokol til udveksling af beskeder - såkaldte datagrammer. Protokollen er i modsætning til TCP ikke forbindelsesorienteret og giver ingen garantier for pålidelighed. Det vil sige at tab af pakker ikke detekteres som en indbygget egenskab i protokollen. UDP indeholder dog en checksum, som medsendes datagrammet og forhindrer at forvanskede data når frem til en modtager.

Felt	**Størrelse**	**Anvendelse**
Modtagerport	16 bit	Den port hos modtageren hvortil datagrammet sendes
Afsenderport	16 bit	Den lokale port hvorfra datagrammet afsendes
Datalængde	16 bit	Længden i bytes af indholdet i datafeltet
Checksum	16 bit	16 bit 1's komplement af data og header
Data	0-65536 bytes	

Tabel 2 - UDP pakkeformat

Fordelen i forhold til TCP er at den simplere UDP-protokol resulterer i mindre headere og ingen kontrolbeskeder, hvilket giver en mere effektiv udnyttelse af netværksbåndbredden når pakker ikke går tabt. Af samme grund anvendes UDP ofte i realtidskritiske applikationer som fx. streaming af video og lyd.
I UDP-datagrammets datafelt anvendes en DAO-specifik indpakning af de data der overføres.
Denne indpakning giver mulighed for at afsende to typer af beskeder:

1. Attach/detach-beskeder - denne type besked bruges til at oprette eller nedlægge sammenknytninger imellem komponenter og variable. En besked indeholder adressen på den node, hvor komponenten eksisterer og navnet på en variabel, som komponenten skal enten til- eller afkobles. Der findes i alt 4 forskellige attach/detach-beskeder: I-attach, IO-attach, I-detach og IO-detach.
2. Beskeder med information om ændringer af variabelværdier - denne type beskeder indeholder oplysninger om ændringer i værdierne af variable. Et UDP-datagram kan indeholde information om ændringer af flere variable samtidigt. Beskederne udpeger navngivne variable og leverer den aktuelle værdi for disse variable. Denne information (variabelnavn og aktuel værdi) fylder 5-11 bytes pr. ændret variabel (se størrelsesangivelserne i Tabel 3).

Tabel 3 viser sammensætningen af den indpakning af data, som DAO foretager før data placeres i UDP-datagrammet. For hver ændret variabel skabes én forekomst af denne struktur i UDP-datagrammets datafelt.

Felt	**Størrelse**	**Anvendelse**
ID-type	1 byte	Angiver datatypen og derigennem længden for ID-feltet
ID	1-4 bytes	Identificerer den ændrede variabel
Værdi-type	1 byte	Angiver datatypen og derigennem længden for værdi-feltet
Værdi	1-4 bytes	Indeholder den aktuelle værdi af den ændrede variabel
Terminator	1 byte	Indikerer afslutning på beskeden for denne variabel

Tabel 3 - DAO-specifik indpakning af besked om ændring af variabel-værdi

Der findes i alt 5 forskellige typer beskeder: 4 slags attach/detach-beskeder og én slags besked med variabelændringer. De forskellige typer beskeder afsendes har hver deres unikke type-ID. På den måde er den modtagende DAO-kerne i stand til at skelne imellem de to typer beskeder.

Den DAO-specifikke indpakning kan ses som et ekstra protokollag ovenpå UDP-protokollen. Alle UDP-datagrammer sendes via protokolstakkens underliggende IP-protokol. UDP-datagrammerne multicastes via de underliggende lag i protokolstakken til alle DAO-kernerne i systemet. Når multicast anvendes, afsendes kun én besked fra afsenderen. Denne besked modtages af alle aktive noder i systemet, som er tilmeldt gruppen af modtagere.
Afsendelse og modtagelse af beskeder udføres af en dedikeret DAO-komponent, som aktiveres periodisk med en cyklus på 50 ms (kan konfigureres). Denne komponent svarer til systemets øvrige komponenter, med den undtagelse at kommunikationskomponenten har adgang til nodens protokolstak og derigennem kan den kommunikere med kommunikationskomponenterne på de øvrige noder. Målinger hos Vestas har vist at afsendelse og modtagelse af UDP-datagrammer kan vare op til 850 µs (mikrosekunder)- dvs. kommunikationskomponenten kan bruge op til 850 µs CPU-tid på at håndtere en en ud- eller indgående besked. Hvis der er ændringer i de lokale variable hos en node, vil dette resultere i udsendelse af 1 datagram - eftersom flere variable samles i 1 datagram. Hvis der er ændringer i variable hos andre noder, vil dette resultere i modtagelse af 1 datagram fra hver af de noder, hvor der forekommer ændrede variable. Der kan således laves en simpel beregning for forbruget af CPU-tid i kommunikationskomponenten:

$$T_{kommunikation} = T_{send} + n_{AntalNoderMedÆndringer} * T_{\mathrm{mod}\,tag}$$

Formel 1 - Tidsforbrug på håndtering af beskeder

Hvor $T_{send} = T_{modtag} = 850$ µs og $0 \leq n_{AntalNoderMedÆnringer} \leq$ antal noder i systemet.
Hvis den lokale node ikke har ændrede variable udgår T_{send} i beregningen ($T_{send} = 0$).
Kommunikationskomponentens procentvise forbrug af de samlede CPU-ressourcer kan herefter beregnes således:

$$U_{kommunikation} = \frac{T_{kommunikation}}{P_{aktivering}}$$

Formel 2 - Udnyttelse af CPU-ressourcer

Hvor $P_{aktivering}$ er kommunikationskomponentens aktiveringscyklus på 50 ms.

4.2.14. Kommunikationsproblemstillingen i DAO

Problemet i den nuværende udgave af DAO er, at det ikke bliver detekteres, hvis en pakke mistes imellem afsender og modtager. Som beskrevet anvendes de udvekslede pakker til at distribuere information om systemets tilstand i form af delte variable imellem systemets noder. Da mange af disse variable repræsenterer måledata, som anvendes af reguleringsalgoritmer er den manglende forudsigelighed i forhold til kommunikationssystemet et generelt problem i det nuværende system. Der eksisterer altså en uforudsigelighed i forhold til hvorvidt afsendte måledata når frem tilpas regelmæssigt. Hos Vestas opererer man med en fælles maksimal opdateringstid for alle variable. Dette krav specificerer, at hvis en variabel ændres, skal alle noder have kendskab til denne ændring før der er gået 50 ms. Kravet sikrer at systemets mange forskellige reguleringsalgoritmer alle er i stand til at operere tilfredsstillende. Som systemet er i dag, er det for det meste muligt at opfylde dette krav. Undtagelsen opstår, når et UDP-datagram med oplysninger om ændringer i variable går tabt under kommunikationen. Et UDP-datagram kan gå tabt af flere forskelle årsager:

- Indholdet er blevet forvansket undervejs og checksum-beregningen passer derfor ikke hos modtageren.
- Modtagerens inputbuffer er fuld og der er derfor ikke plads til det nye indkommende datagram. UDP indeholder i modsætning til TCP ikke congestion control.

I begge tilfælde går datagrammet tabt - og UDP leverer ikke information om dette til applikationslaget. Da DAO-frameworket heller ikke detekterer eller håndterer fejl i form af mistede pakker, vil en mistet pakke resultere i at kravet til den maksimale opdateringstid ikke kan opfyldes, da informationen er gået tabt og ikke vil blive gentransmitteret. Målinger hos Vestas har vist at 1 UDP-datagram for hver 100.000 går tabt i gennemsnit. Datagrammer går med ligeligt fordelt sandsynlighed tabt hos enten afsender eller modtager af et datagram. Effekten af de to situationer er forskellig. Hvis et datagram går tabt i hos afsenderen, vil ingen af de øvrige noder modtage beskeden. Hvis datagrammet til gengæld går tabt hos en modtagende node, er det stadig muligt for de øvrige modtagende noder, at modtage datagrammet korrekt. Som beskrevet i afsnit 4.2.13 kan UDP-datagrammerne indeholde information om flere ændrede variable, så når et datagram går tabt kan det have negativ betydning for en række komponenter fordelt på forskellige noder. Desuden kan UDP-datagrammet indeholde attach/detach-beskeder, som også kan gå tabt.

5. Signaleringsprotokoller

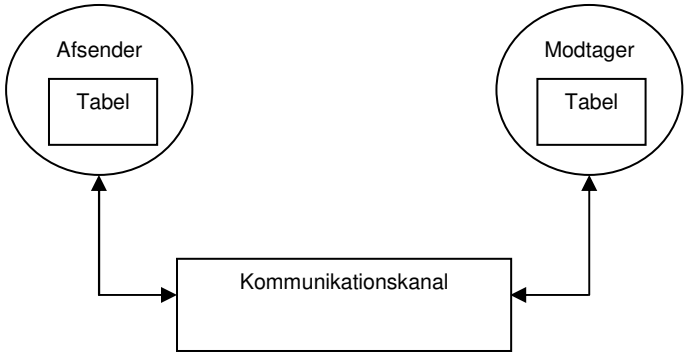
Det foregående afsnit beskrev den kommunikation der foregår imellem DAO-noderne for at udveksle opdaterede værdier af de delte variable. Et af de mest centrale krav til et system som DAO er at alle noder har det samme billede af variablenes værdier samt at oplysninger om ændrede værdier formidles imellem systemets noder med mindst mulig forsinkelse og bedst mulig udnyttelse af netværksbåndbredden. Denne slags krav minder i høj grad om kravene til en bestemt type kommunikationsprotokoller; signaleringsprotokoller. Denne type protokoller har en række forskellige anvendelser, men fælles for dem er, at formålet er at vedligeholde et fælles og konsistent billede af tilstand, som deles af de kommunikerende parter. Dette afsnit vil beskrive princippet bag signaleringsprotokoller ved at give en gennemgang af en gruppe grundlæggende typer af signaleringsprotokoller.
Begrebet *signaleringsprotokol* dækker over en bred vifte af protokoltyper, som anvendes i mange forskellige sammenhænge; eksempelvis dynamisk reservation af netværksbåndbredde med RSVP som signaleringsprotokol [RSVP], opdatering af fillister i fildelingstjenester [SS2], etablering af virtuelle kredsløb i netværk [ATM] og håndtering af multimedia-sessioner med SIP som signaleringsprotokol [SIP].

5.1. Systemmodel

Som udgangspunkt for beskrivelserne anvendes en simpel systemmodel. Systemmodellen definerer de parter og hændelser, der indgår i beskrivelsen af signaleringsprotokoller i de efterfølgende afsnit. Hændelserne bliver beskrevet på et abstrakt niveau, eftersom det afhænger af specifikationen af hver enkelt signaleringsprotokol hvordan hændelserne bliver udført.

5.1.1. Kommunikerende parter, tilstande og tabeller

Modellen indeholder to noder: En afsender og en modtager. Hver af disse noder har en tabel bestående af en gruppe {nøgle, værdi}-sæt [SS1]. Hvert sæt beskriver en tilstand. Værdien i sættet kan ændres hos afsenderen. Når der sker en ændring hos afsenderen, er det dennes opgave at få *opdateret* den værdi med samme nøgle som eksisterer hos modtageren. Det er ligeledes afsenderens opgave at *installere* og *slette* {nøgle, værdi}-sæt hos modtageren så tabellen hos modtageren har det samme indhold som tabellen hos afsenderen. Signaleringsprotokollerne specificerer disse mekanismer i forbindelse med kommunikationen imellem afsenderen og modtageren.



Figur 5 - Systemmodel

5.1.2. Hændelser

Her beskrives det sæt af hændelser, som kan forekomme mens systemet kører. Hændelserne er beskrevet uden hensyntagen til de bagvedliggende protokol- og implementationsspecifikke detaljer. Flere detaljer omkring hver af hændelserne, kan findes i beskrivelsen af forskellige typer af signaleringsprotokoller i afsnit 5.2.

- **Installation af tilstande hos modtageren**
  Hvis en applikation på afsender-noden ønsker at dele en tilstand med modtagernoden, placerer applikationen denne tilstand i tabellen. Når en ny tilstand bliver oprettet i afsenderens tabel, er det afsenderens ansvar at sørge for, at denne tilstand tilføjes til modtagerens tabel.

- **Vedligeholdelse af tilstande hos modtageren**
  Efter oprettelsen af en tilstand hos modtageren, har afsenderen ansvaret for at holde denne tilstand løbende opdateret, så dens værdi hos modtageren passer med værdien hos afsenderen. Denne vedligeholdelse foretages ligeså længe tilstanden i form af et {nøgle, værdi}-sæt eksisterer i afsenderens tabel.

- **Sletning af tilstande hos modtageren**
  Tilstande eksisterer i afsenderens tabel indtil de bliver slettet af en applikation hos afsenderen. Når en tilstand bliver slettet i afsenderens tabel, er det afsenderens ansvar at den tilsvarende tilstand bliver slettet fra tabellen hos modtageren. En sletning kan udføres på tre forskellige måder:

1. Aktiv sletning

Afsenderen kan have en dedikeret beskedtype til fjernelse af tilstande hos modtageren. I denne type besked identificeres den tilstand, som skal fjernes ved hjælp af dens unikke nøgle.

2. Sletning ved timeout

I nogle signaleringsprotokoller anvendes periodisk opdatering af tilstandene hos modtageren - dvs. afsenderen udsender løbende beskeder med indholdet af tilstandene i dens tabel. I denne sammenhæng kan alle tilstande i modtagerens tabel have tilknyttet en nedtællingstimer, som bliver genopfrisket hver gang værdien af tilstanden bliver opdateret. Hvis opdateringen ikke sker indenfor den fastsatte tid, vil tilstanden blive slettet. Det betyder, at tilstande som ikke længere findes i afsenderens tabel automatisk vil blive slettet fra modtagerens tabel, når timeren udløber. Den automatiske sletning vil også ske, hvis afsender bryder ned og ikke længere sender opdateringsbeskeder.

3. Falsk sletning

Som beskrevet ovenfor, anvender nogle signaleringsprotokoller bevidst nedtællingstimere til sletning af tilstande i modtagerens tabel. Da nogle protokoller samtidig anvender upålidelige mekanismer til afsendelse af beskeder, kan det ske at en opdateringsbesked går tabt. Det kan resultere i, at nedtællingstimeren løber ud selvom tilstanden stadig eksisterer hos afsenderen, hvilket forårsager at tilstanden bliver slettet fra modtagerens tabel selvom den stadig eksisterer i afsenderens tabel. Dette kaldes en falsk sletning.

5.1.3. Kommunikationskanal

Afsender og modtager kommunikerer via en ikke-pålidelig kommunikationskanal. Afsender og modtager befinder sig på samme lokalnetværk, og udvekslede pakker skal derfor ikke bevæge sig igennem routere osv. De to noder kan sende beskeder til hinanden igennem denne kanal. En besked der afsendes fra afsenderen vil efter en vis forsinkelse blive afleveret hos modtageren, medmindre den går tabt undervejs. Forsinkelsen stammer primært fra processeringstid hos afsenderen og modtageren. En besked kan gå tabt, hvis den forvanskes undervejs og derfor afvises hos modtageren. Beskeder kan også gå tabt, hvis modtagerens inputbuffer er fuld, og derfor ikke har plads til flere indkommende beskeder. Tab af beskeder vil ikke blive meddelt til hverken afsender eller modtager.

5.2. Typer af signaleringsprotokoller

Signaleringsprotokoller kan inddeles i to grundlæggende kategorier: Soft og Hard State-protokoller [INT]. Begrebet Soft State blev første gang introduceret i [DARPA] som et alternativ til de allerede eksisterende signaleringsprotokoller, som herefter fik navnet Hard State-protokoller. Soft State-protokoller er baseret på upålidelig kommunikation mens Hard State-protokoller er baseret på pålidelig kommunikation. Begge protokoltyper baserer sig på kommunikation via en underliggende upålidelig protokol som fx. UDP.
De to typer af protokoller anvender forskellige metoder til at opnå højest mulig konsistens: Pålidelige opdateringsbeskeder for Hard State-protokollernes vedkommende og periodiske opdateringsbeskeder for Soft State-protokollernes vedkommende. Pålideligheden i Hard State-protokollerne bygger på afsendelse af bekræftelser på modtagne beskeder hos modtageren.
Underafsnittene 5.2.1 og 5.2.2 giver generiske beskrivelser af de to forskellige typer af signaleringsprotokoller. Beskrivelserne tager udgangspunkt i de modeller for Soft og Hard State-protokoller, som anvendes i [SS4].

5.2.1. Soft State

Når en afsender ønsker at installere en tilstand i tabellen hos en modtager, udsendes en installationsbesked, som indeholder tilstandens aktuelle værdi. Samtidig indeholder installationsbeskeden en angivelse af tilstandens maksimale opdateringsperiode. Dvs. den tid der maksimalt må gå imellem opdatering af tilstandens værdi - også selvom denne ikke er ændret siden sidste opdatering. Denne værdi kaldes $T_{Timeout}$. Alle installerede tilstande hos en modtager vil løbende blive vedligeholdt med opdateringsbeskeder, som modtages fra afsenderens. Opdateringsbeskederne sendes løbende og ikke nødvendigvis med samme frekvens. Hver enkelt tilstand i afsenderens tabel har tilknyttet en nedtællings-timer, hvis udløbstid er bestemt af værdien $T_{Opdater}$, for hvilken det gælder at $T_{Opdater} < T_{Timeout}$. Nedtællings-timeren initialiseres til værdien $T_{Opdater}$ hver gang afsenderen har udsendt en opdateringsbesked for den relaterede tilstand og hver gang nedtællingstimeren udløber afsendes en opdateringsbesked for den relaterede tilstand. Alle tilstande i tabellen hos modtageren har en individuel nedtællings-timer, hvis udløbstid er bestemt af den værdi, $T_{Timeout}$, som fulgte med installationsbeskeden fra afsenderen, da tilstanden blev oprettet i tabellen. Når modtageren får en opdatering af en tilstand i form af en opdateringsbesked, initialiseres tilstandens timeout-timer. Den samme nulstilling sker når den første installationsbesked for en tilstand modtages. Hvis en tilstands timer når at

løbe ud før en opdateringsbesked modtages, vil denne tilstand blive slettet fra tabellen. Dette er den eneste situation, hvor en tilstand vil blive slettet fra modtagerens tabel. En sletning af en tilstand hos modtageren kan således have tre årsager i Soft State-protokoller:

- Tilstanden er slettet hos afsenderen, som derfor ikke længere udsender opdateringsbeskeder
- Opdateringsbeskeder er gået tabt imellem afsender og modtager
- Afsender er gået i stå og udsender ikke opdateringsbeskeder rettidigt

De to sidste tilfælde kaldes *falsk sletning*, da tilstanden slettes hos modtageren, selvom den stadig eksisterer i afsenderens tabel. Den sidste årsag er dog i mange tilfælde hensigtsmæssig, da en ikke-fungerende afsender heller ikke kan forventes at have gyldige værdier for tilstandene i sin egen tabel. En vigtig egenskab for Soft State-protokoller er altså at der ikke findes nogen beskedtype til aktivt at slette en tilstand fra modtagerens tabel. Dette sker alene som resultat af en udløbet opdateringstimer i modtagerens tabel. Hvis en falsk-sletning skyldes tab af opdateringsbeskeder, er der en sandsynlighed for at efterfølgende opdateringsbeskeder fra afsender vil nå frem til modtageren. Hvis modtageren på denne måde modtager opdateringsbeskeder for en tilstand, som ikke findes i dens tabel (fordi den er blevet berørt af en falsk sletning), vil denne opdateringsbesked blive betragtet som en installationsbesked, og tilstanden vil igen blive tilføjet til modtagerens tabel. Samtidig med risikoen for falske sletninger, findes der også en positiv konsekvens af anvendelsen af nedtællingstimere hos modtageren i Soft State protokoller; tilstande uden afsender vil automatisk blive fjernet fra modtagerens tabel, når timeren udløber. Dette gør, at der ikke er risiko for en ophobning af afsenderløse tilstande hos modtageren.
Al kommunikation imellem afsender og modtager foregår via en tabsbehæftet kommunikationskanal, eftersom modtager aldrig sender kvitteringer for modtagne pakker og afsenderen derfor ikke har mulighed for at detektere tab af pakker og evt. gensende.
I Soft State-protokoller opstår der en forsinkelse fra en tilstand har ændret værdi i afsenderens tabel og til denne ændring er afspejlet i modtagerens tabel. Forsinkelsen er en tid, $T_{Forsinkelse}$ som er karakteriseret ved: $0 < T_{Forsinkelse} < T_{Timeout}$ forudsat at tilstanden allerede eksisterer i modtagerens tabel og ikke bliver udsat for en falsk sletning før opdateringsbeskeden når frem. Denne forsinkelse skyldes at afsendelsen af opdateringsbeskeder hos afsenderen ikke er aktivt synkroniseret med ændringen af tilstandenes værdier i dennes tabel. En lignende forsinkelse gør sig gældende i forbindelse med sletning af en værdi i afsenderens tabel, men med den forskel at $T_{Timeout}$ her vil være afgørende for hvornår tilstanden slettes i modtagerens tabel.

Soft State-protokoller anvendes bla. i RSVP [RSVP] til vedligeholdelse af oplysninger om dynamisk konfigurerede ruter igennem et netværk.

5.2.2. Hard State

I Hard State-protokoller etableres der i modsætning til i Soft State-protokoller en pålidelig kommunikationskanal imellem afsenderen og modtageren. Herigennem kan tilstande opdateres pålideligt hos modtageren. I modsætning til Soft State anvendes der ikke periodisk udsendelse af opdateringsbeskeder. Den pålidelige kommunikationskanal gør, at det kun er nødvendigt at udsende opdateringsbeskeder for tilstande, som har værdi siden sidste udsendelse af opdateringsbesked. Grundlaget for den pålidelige kommunikation er anvendelsen af eksplicit pålidelige beskeder til alle opgaver: Installation, opdatering og sletning af tilstande. Pålideligheden implementeres ved hjælp af retransmissionstimere hos afsenderen. Hver gang en besked afsendes, vil en retransmissionstimer blive initialiseret med værdien $T_{Retransmission}$. Når modtageren modtager en besked fra afsender, er det modtagerens besked at sende en bekræftelse tilbage afsenderen. Hvis afsenderens retransmissionstimer når at løbe ud før der er modtaget en bekræftelsesbesked fra modtageren, vil afsenderen gensende opdateringsbeskeden for den berørte tilstand. Hvis tilstanden har ændret værdi undervejs, vil indholdet af retransmissionsbeskeden indeholde den nyeste værdi. Der findes ingen retransmissionstimere hos modtageren, da denne del af protokollen håndteres af afsenderen alene.
Hard State-protokoller anvender en særlig besked til aktivt at fjerne tilstande fra modtagerens tabel. Dette er nødvendigt, da der ikke anvendes nedtællingstimere og periodiske opdateringsbeskeder som i Soft State og tilstandene derfor ikke bliver fjernet fra tabellen automatisk. På samme måde som i Soft State anvender afsenderen en installationsbesked med tilstandens aktuelle værdi, til at installere en ny tilstand i modtagerens tabel. Efter tilstanden er installeret, vil afsenderen kun sende nye beskeder med relation til denne tilstand, når dens værdi ændres eller tilstanden slettes i afsenderens tabel.
I forbindelse med Hard State-protokoller kan der opstå et problem i forhold til sletning af ikke-aktive tilstande i modtagerens tabel - dvs. tilstande som er installeret af en afsender som pga. af nedbrud er stoppet med at vedligeholde deres indhold. I Soft State-protokoller, vil sletningen ske automatisk, hvis opdateringsbeskederne udebliver. Denne automatik findes ikke i Hard State-protokoller, da der ikke anvendes periodiske opdateringsbeskeder. I [SS2] foreslås en løsning på dette problem; en separat hjerteslagsprotokol, som anvendes til at detektere nedbrud af en afsender. Princippet er enkelt og handler om at lade afsender udsende en periodisk besked om at den stadig opererer korrekt. Denne besked er generel for afsenderen og knytter sig ikke til nogen bestemt

tilstand. Hvis modtageren opdager, at den periodiske besked udebliver, kan alle de tilstande, som afsenderen måtte have installeret i modtagerens tabel slettes.
Princippet fra Hard State-protokoller anvendes bla. i TCP [TCP] til opdatering af de sekvensnumre, som identificerer pakker i en fragmenteret datastrøm. I denne sammenhæng fungerer en Hard State-protokol som en integreret del af den samlede protokol.

Tabel 4 indeholder en opsummering af de beskrevne egenskaber for Hard og Soft State-protokoller:

	Hard State	**Soft State**
Vedligeholdelse af tilstande	Én skrivning pr. ændring i tilstanden	Løbende opdatering af tilstanden
Sletning af tilstande	Eksplicit	Udløb af timer
Oprydning af afsenderløse tilstande	Manuelt	Automatisk

Tabel 4 - Kilde: [SS3]

5.2.3. Hybridprotokoller

Med Soft og Hard State-protokollerne som yderpunkter, beskriver [SS4] en række mellemliggende hybrider, som her vil blive bekrevet kort:

- **Soft State med eksplicit sletning**
  Inspireret af Hard State-protokollerne har denne protokoltype en ekstra beskedtype, som anvendes til at fjerne tilstande fra modtagerens tabel. Denne besked sendes uden garanti. Denne hybrid-protokol er sammen med den rene Soft State-protokol kendetegnet ved, at modtageren aldrig afsender beskeder. Alle beskeder afsendes fra afsenderen.

- **Soft State med pålidelig installation**
  Denne protokoltype anvender eksplicit pålidelige installationsbeskeder på samme måde som Hard State, men fungerer ellers som den rene Soft State-protokol. Se afsnit 5.2.2 for en beskrivelse af hvordan eksplicit pålidelige beskeder håndteres.

- **Soft State med pålidelig installation/fjernelse**

I denne protokoltype anvendes eksplicit pålidelige beskeder i forbindelse med både installation og fjernelse af tilstande hos modtageren.

6. Løsningsforslag

Dette afsnit vil beskæftige sig med udvælgelse af en egnet protokoltype til anvendelse i forbindelse med DAO. Udvælgelsen sker med udgangspunkt i det foregående kapitels beskrivelse af signaleringsprotokoller og sammenholdes med de konkrete krav til et kommunikationssystem i DAO.

Som beskrevet tidligere, er kommunikationen imellem noderne i DAO karakteriseret ved sine krav til rettidig udveksling af opdateringsbeskeder for de delte variable. I [SPEC] defineres et realtidssystem som et system, hvor tiden indgår som afgørende faktor for om beskeder bliver udvekslet korrekt. En besked kan nå uforvansket fra en afsender til en modtager, men hvis beskeden kommer for sent frem til modtageren vil den blive betragtet som ugyldig. Der kan altså stilles krav til maksimalt tilladt forsinkelse i forbindelse med udveksling af beskeder. Dette er også tilfældet i DAO, hvor systemets reguleringsalgoritmer har behov for at kende til ændringer i værdierne af de delte variable indenfor en vis tidsramme for at kunne fungere efter hensigten. I et system som DAO findes der ingen delt opfattelse af den globale tid, så en modtager er altså ikke i stand til at vurdere alderen af en modtaget besked - og kan derfor ikke forkaste beskeder, som måtte være blevet forsinket for meget undervejs fra afsender til modtager.

Der vil altid eksistere en vis forsinkelse i forbindelse med udveksling af opdateringsbeskeder imellem afsender og modtager. Forsinkelsen stammer hovedsageligt fra den tid det for afsenderen og modtageren at håndtere henholdsvis afsendelse og modtagelse af beskeden igennem hver deres protokolstak. Selve transmissionsforsinkelsen på netværket udgør en mindre del af den samlede forsinkelse. Med udgangspunkt i systemmodellen fra afsnit 5.1, vil en yderligere forsinkelse imellem afsendelse og modtagelse opstå i forbindelse med retransmission af en besked fra afsenderen. I protokoller uden retransmission vil en tabt besked ikke nå frem til modtageren, mens den i protokoller med retransmission vil nå frem med denne yderligere forsinkelse. Ud fra den overordnede definition af et realtidssystem, kan denne yderligere forsinkelse i sig selv være skyld til at beskeden er ugyldig når den når frem efter retransmission fordi den er blevet forældet. I en sådan kommunikation vil retransmission være formålsløst - og samtidig betyde overflødig brug af processeringsressourcer til kommunikation i forbindelse med retransmissionen.

Retransmission af beskeder fra afsenderen forekommer i alle protokoltyper undtagen ren Soft State og Soft State med eksplicit sletning (se afsnit 5.2). I de to øvrige Soft State-protokoltyper (Soft State med pålidelig installation og Soft State med pålidelig installation/fjernelse) anvendes retransmission af beskeder til at sikre pålidelig installation og fjernelse af tilstande. Da retransmissionerne for hybridprotokollerne kun vedrører installations- og sletningsbeskederne, har den forsinkelse der kan opstå i forbindelse med re-

transmission ikke nogen direkte indvirkning på korrektheden af DAO-komponenternes reguleringsalgoritmer på samme måde som i forbindelse med Hard State, hvor det er selve opdateringsbeskederne med nye værdier som kan blive retransmitteret (og dermed forældede). Problemet i denne sammenhæng ligger i, at båndbredde- og processeringsressourcer går til spilde, hvis retransmitterede beskeder når for sent frem hos modtageren. I de Soft State-protokoller, hvor der ikke anvendes aktiv retransmission (ren Soft State og Soft State med eksplicit sletning) kan der stadig forekomme uacceptable forsinkelser. Disse forsinkelser vil opstå, hvis opdateringsbeskeder går tabt og deres størrelse vil være bestemt af frekvensen for udsendelse af opdateringsbeskeder, fordi denne bestemmer hvornår en ny besked bliver sendt ud - og eventuelt når frem til modtageren.
Prisen for periodisk opdatering af ikke-ændrede tilstande i Soft State består i at der anvendes processeringsressourcer på udsendelse af redundant data. Med udgangspunkt i antallet af afsendte beskeder, resulterer valget imellem en protokol med retransmission eller en protokol uden retransmission altså i beslutning om, hvor der skal anvendes ekstra beskeder for at sikre konsistens. Begge protokoltyper resulterer i et overhead af beskeder i forhold til en helt naiv protokol, hvor besked om ændring i tilstande kun afsendes én gang via en upålidelig kommunikationskanal. I protokoller med retransmission består overheadet af kvitteringsbeskeder, som afsendes fra modtageren. Disse beskeder er overflødige, når beskederne fra afsenderen når sikkert frem. I protokoller uden retransmission består overheadet af opdateringsbeskeder, som udsendes selvom en tilstand ikke er ændret. Disse beskeder er overflødige så længe den foregående opdateringsbesked ikke er gået tabt - dvs. når tilstanden er konsistent imellem afsender og modtager.
I sammenhæng med DAO, indgår yderligere en dimension i valget; nemlig behovet for deterministisk begrænsning af ressourceforbrug. Denne problemstilling beskrives i afsnit 6.1.

6.1. Maksimalt ressourceforbrug

Et andet problem i forbindelse med kommunikation imellem noderne, er det ressourceforbrug, som udvekslingen af beskeder resulterer i. I DAO er det vigtigt at sikre, at en tilstrækkelig mængde af processeringsressourcerne står til rådighed for DAO-komponenterne på hver enkelt node. Reguleringsalgoritmerne er implementeret i DAO-komponenter, som aktiveres periodisk. For at denne aktivering kan udføres med tilpas høj frekvens er det vigtigt at hver enkelt komponent ikke bruger for stor en del af den samlede processeringstid. Som beskrevet i afsnit 4.2.13, håndteres kommunikationen i DAO i en dedikeret kommunikationskomponent, som opererer på lige fod med DAO-

nodens øvrige komponenter. Der kan defineres et maksimum for tilladt forbrug af processeringstid for kommunikationskomponenten. Dette maksimum sikrer at de øvrige komponenter på noden kan aktiveres korrekt.
Både afsendelse og modtagelse af beskeder tager tid og kræver dermed processeringsressourcer i kommunikationskomponenten. Hvis det skal være muligt at sikre, at kommunikationskomponenten holder sig under det maksimalt tilladte forbrug af processeringstid er det nødvendigt at vide hvor mange beskeder der vil blive afsendt og modtaget af hver enkelt kommunikationskomponent. Kommunikationskomponentens ressourceforbrug hænger direkte sammen med antallet af afsendte og modtagne beskeder over tid. Der kan derfor fastsættes en maksimal grænse for frekvensen af afsendelse/modtagelse af beskeder på en node ud fra den maksimale grænse for kommunikationskomponentens forbrug af processeringstid. I en protokol hvor der anvendes pålidelige beskeder (fx Hard State), vil afsenderen modtage en kvittering fra hver enkelt modtager. Den tid der bruges på modtagelse af kvitteringer for afsendte beskeder hos afsenderen indgår i det samlede billede af kommunikationskomponentens forbrug af processeringsressourcer. I protokoltyper uden pålidelig beskedudveksling, vil afsenderen ikke skulle håndtere indkommende beskeder.
Kravet om forudsigelighed i forhold til forbrug af processeringsressourcer kan ikke umiddelbart opfyldes af Hard State-protokollerne, da det her ikke er muligt at give en deterministisk øvre grænse for antallet af beskeder pr. tidsenhed pga. muligheden for retransmission af beskeder. Det samme gør sig gældende for de to hybridprotokoller; Soft State med pålidelig sletning og Soft State med pålidelig installation/sletning. Den rene Soft State-protokol og Soft State med eksplicit sletning er i stand til at overholde kravet til mulighed for deterministisk begrænsning af antallet af afsendte beskeder. Det skyldes, at disse to protokoltyper anvender en fast opdateringsfrekvens for hver tilstand. På samme måde anvendes en fast frekvens for udsendelse af installationsbeskeder som reaktion på tilføjelse af tilstande i afsenderens tabel. Opdateringsfrekvenserne varierer fra tilstand til tilstand, men ligger fast på kompileringstidspunktet og kan anvendes som udgangspunkt i en analyse af ressourceforbruget i systemet. Frekvensen for udsendelse af installationsbeskeder for nye tilstande er fælles for alle tilstande. Denne frekvens bestemmer hvor ofte der kigges efter nye tilstande i afsenderens tabel. Hvis der opdages en ny tilstand udsendes en installationsbesked. De faste opdateringsfrekvenser gør det muligt at garantere et loft for, hvor mange indkommende og udgående beskeder en node maksimalt vil skulle behandle.

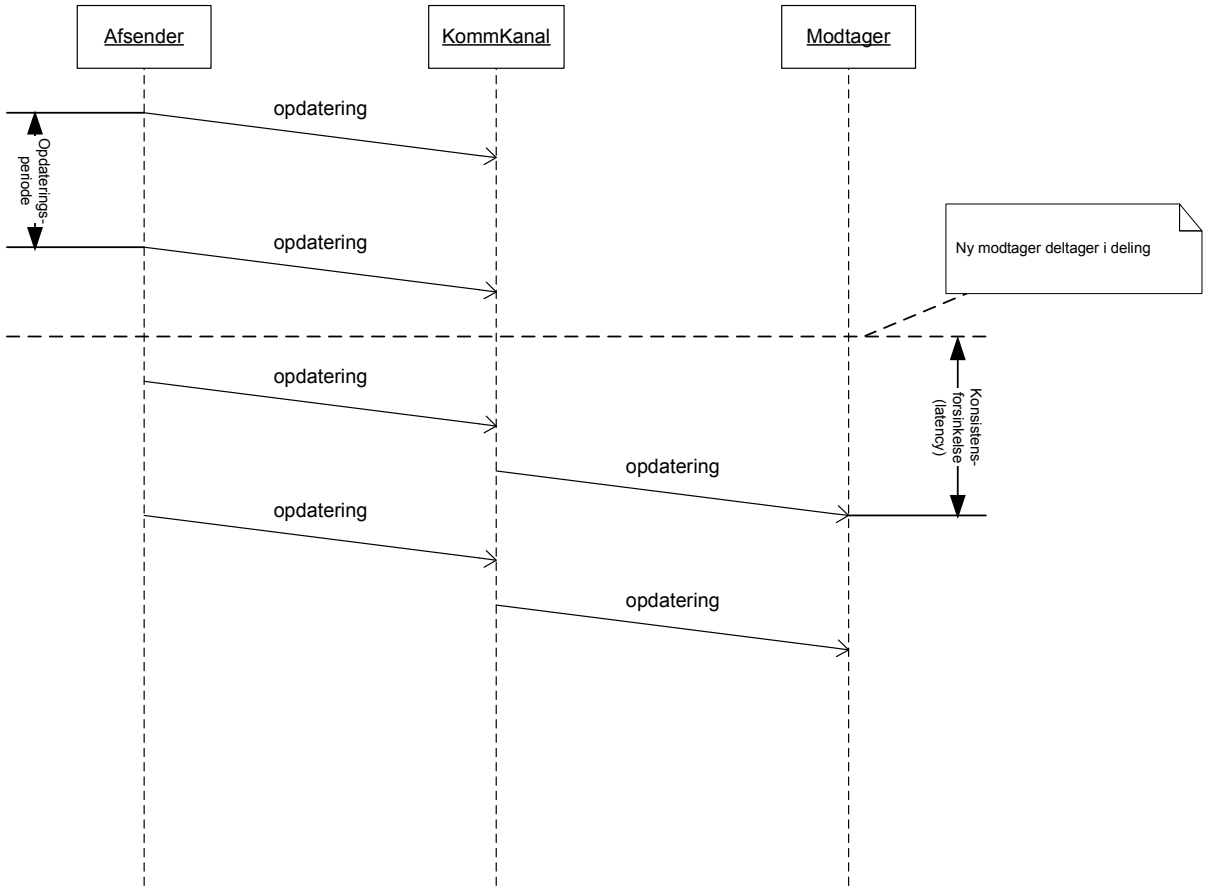
6.2. Robusthed i forhold til ændringer

I et typisk system er alle noderne ikke altid aktive. Mens en gruppe af noder kommunikerer og deler tilstande, kan en ny node blive aktiv og have behov for at deltage i delingen af tilstande. Det vil afhænge af protokoltypen, hvordan den nye node indlemmes i gruppen af aktive noder. Her beskrives forskellen med udgangspunkt i protokolbeskrivelserne fra afsnit 5.2:

- **Hard State**
  I Hard State sker indlemmelsen trinvist efterhånden som der sker ændringer i de delte tilstande. Der vil således ikke ske nogen aktiv installation af alle afsenderens tilstande hos en nytilkommen modtager. Det betyder at den forsinkelse der vil opstå før modtageren har et konsistent billede af alle afsenderens tilstande, vil afhænge af hvor ofte der sker ændringer i værdierne af tilstandene i afsenderens tabel. Hyppigheden af ændringer er direkte afgørende for hyppigheden af udsendelse af opdateringsbeskeder, og vil derfor være afgørende for hvornår den nytilkomne node bliver opmærksom på de tilstande, som eksisterer i afsenderens tabel.

- **Soft State**
  I Soft State sker der heller ikke nogen aktiv installation af tilstande i modtagerens tabel, men forsinkelsen før afsender og modtager er konsistente er mindre end for Hard State. Takket være de periodiske opdateringsbeskeder i Soft State, afhænger forsinkelsen kun af de forskellige tilstandes opdateringsfrekvenser. Alle tilstande vil blive installeret i modtagerens tabel, efterhånden som der udsendes og modtages opdateringsbeskeder om dem.

Forskellen på hvordan tilstande installeres i tabellen hos en ny node betyder, at der i denne sammenhæng drages nytte af de periodiske opdateringsbeskeder i Soft State-protokollen. De bidrager til at begrænse den forsinkelse (latency), som opstår før den nye node har et konsistent billede af de delte variable [SS1]. Figur 6 illustrerer forsinkelsen for en enkelt tilstand i forbindelse med Soft State. En afsender sender periodiske opdateringsbeskeder til kommunikationskanalen (KommKanal). Disse beskeder når ikke videre, før en modtager begynder at deltage i delingen af tilstande (markeret med den vandrette streg). Herefter vil modtageren begynde at modtage de periodiske opdateringsbeskeder fra afsenderen via kommunikationskanalen. For en Hard State-protokol vil opdateringsbeskederne fra afsender til kommunikationskanal først optræde, når der sker en ændring af tilstandens værdi hos en afsender. Derfor afhænger konsistensforsinkelsen i denne sammenhæng af hvornår en sådan ændring forekommer.



Figur 6 - Konsistensforsinkelse for Soft State efter deltagelse af ny modtager

Der er også forskel på de to protokoltyper i forbindelse med, at en node forlader en gruppe. I Hard State anvendes eksplicitte beskeder til sletning af installerede tilstande hos modtageren. Hvis afsenderen fejler efter en eller flere tilstande er installeret, vil disse tilstande forblive i modtagerens tabel, selvom de er afsenderløse. I Soft State foregår fjernelsen af tilstande bevidst ud fra detektering af udløbne opdateringstimere hos modtageren. Det forhindrer ophobning af afsenderløse tilstande i modtagerens tabel.
I forbindelse med at en afsender forlader gruppen af kommunikerende noder uden fejl, er Hard State potentielt hurtigere til at få etableret konsistens. Det skyldes anvendelsen af de eksplicitte sletningsbeskeder. Disse vil blive afsendt før afsenderen forlader gruppen, og pålidelighedskravene gør, at en afsender ikke vil forlade gruppen før sletning af dens tilstande i modtagernes tabeller er bekræftet med kvitteringer. Forsinkelsen i Hard State i forbindelse med at en afsender forlader gruppen af noder afhænger altså af den tid det tager, at udveksle sletningsbeskeder på en pålidelig måde. I Soft State forholder det sig anderledes. Her vil forsinkelsen afhænge af de nedtællingstimere, som er tilknyt-

tet tilstandene i modtagerens tabel, da de er afgørende for hvornår modtageren sletter de givne tilstande - og dermed opnår konsistens efter afsenderen har forladt gruppen.
I DAO eksisterer samme situation, hvor noder har behov for at deltage i eller forlade en eksisterende deling af variable. I det eksisterende kommunikationssystem anvendes attach/detach-beskederne, når en ny node ønsker at deltage i eller forlade delingen af variable. Disse beskeder sendes ligesom opdateringsbeskederne uden nogen form for pålidelighed. Der er samtidig ikke er nogen periodisk udsendelse af opdateringsbeskeder, og opdateringsbeskeder i forbindelse med ændringer udsendes ikke medmindre der er registreret en tilkobling til den berørte variabel. Det betyder, at tabet af en attach-besked ikke vil blive korrigeret efterfølgende.

6.3. Soft State med multicast-kommunikation

I DAO anvendes flere end én modtager og én afsender. Når en besked afsendes vil det ske fra én afsender til en gruppe af modtagere. Dette sker ved at afsende et UDP-datagram til en multicast-adresse, som definerer gruppen af modtagere. I Soft State-protokollen anvendes der aldrig kvittering fra modtageren – eller modtagerne i DAOs tilfælde. I afsnit 5.2.1 beskrives Soft State-protokollen på baggrund af en systemmodel med kun én afsender og én modtager. Da der ikke skal holdes styr på modtagerkvitteringer i Soft State, kan protokollen uden omdesign udvides til at kommunikere via multicast-addresser [SS1]. Det betyder at installations- og opdateringsbeskeder vil blive afsendt vha. UDP-multicast i DAO i modsætning til vha. generisk unicast-kommunikation i systemmodellen for protokol-beskrivelserne. I forbindelse med multicast-kommunikation kan installations- og opdateringsbeskeder blive leveret til ingen, nogen af eller alle modtagerne. Hvis én eller flere installationsbeskeder går tabt, vil en efterfølgende opdateringsbesked, som når frem forårsage at den aktuelle tilstand bliver installeret og opdateret hos modtageren. På samme måde vil tabet af en opdateringsbesked undervejs til én eller flere af modtagerne blive opvejet af en efterfølgende opdateringsbesked. Den enkle udvidelse til anvendelse af multicast-kommunikation er endnu et argument for valget af Soft State-protokollen. I Hard State anvendes kvitteringer fra modtageren for at sikre en pålidelig overførsel af beskeder. Princippet med kvitteringer gør det mere problematisk at introducere flere samtidige afsendere og modtagere. I modsætning til Soft State kan alle beskeder ikke sendes med multicast i Hard State - dvs. det er ikke tilstrækkeligt blot at sende alle beskeder fra afsenderen som multicast-beskeder i stedet for som unicast-beskeder. Ét problem er mængden af kvitteringer, som sendes fra gruppen af modtagere til afsenderen. I Hard State sender en modtager en kvitteringer når en besked er modtaget korrekt. Det bliver til én kvittering for hver modtager, når beskeden er sendt som multicast til en gruppe af modtagere. Det betyder, at håndteringen af indkommende

kvitteringer hos afsender kan blive en stor ressourcebelastning. Situationen kaldes ACK-implosion [RM] og er klassisk i sammenhæng med pålidelig multicast-kommunikation.

6.4. Valg af protokoltype

Gennem undersøgelsen af Soft og Hard State-protokollerne er det blevet klargjort, at Soft State egner sig godt til et system som DAO. Det skyldes flere forskellige egenskaber, hvor Soft State sammenholdt med kravene til et kommunikationssystem til DAO adskiller sig positivt fra Hard State. Det vigtigste argument for at anvende Soft State er muligheden for deterministisk forudsigelighed af antallet af afsendte beskeder (se afsnit 6.1). Manglen på denne egenskab gør Hard State uegnet til brug i den realtidskritiske sammenhæng, hvori DAO-noderne kommunikerer. Soft State har også en stor fordel i sammenhæng med DAO, når man kigger på den robusthed der tilbydes i forbindelse med dynamiske ændringer (join/leave) i gruppen af kommunikerende noder (se afsnit 6.2). Et sidste og vigtigt argument for at vælge Soft State er protokollens egenskaber i forhold til multicast-kommunikation (se afsnit 6.3), som er simple og ressourcebegrænsende set i forhold til Hard State i sammenhæng med multicast-kommunikation.
Det vælges at anvende Soft State som protokol i det videre arbejde med at finde et egnet design for kommunikationskomponenten i DAO. De efterfølgende afsnit vil tage udgangspunkt i dette valg og modellerne vil basere sig på Soft State protokollen som den beskrives i afsnit 1. I DAO-anvendes attach/detach-beskeder. Disse beskeder gør i afsenderen i stand til at vide hvilke modtagere er tilsluttet variable i afsenderens tabel. På den måde kan afsenderen afgøre, om det er nødvendigt at udsende opdateringsbeskeder. Formålet er at mindske antallet af beskeder. Dette princip bibeholdes i det videre arbejde med løsningsforslaget.

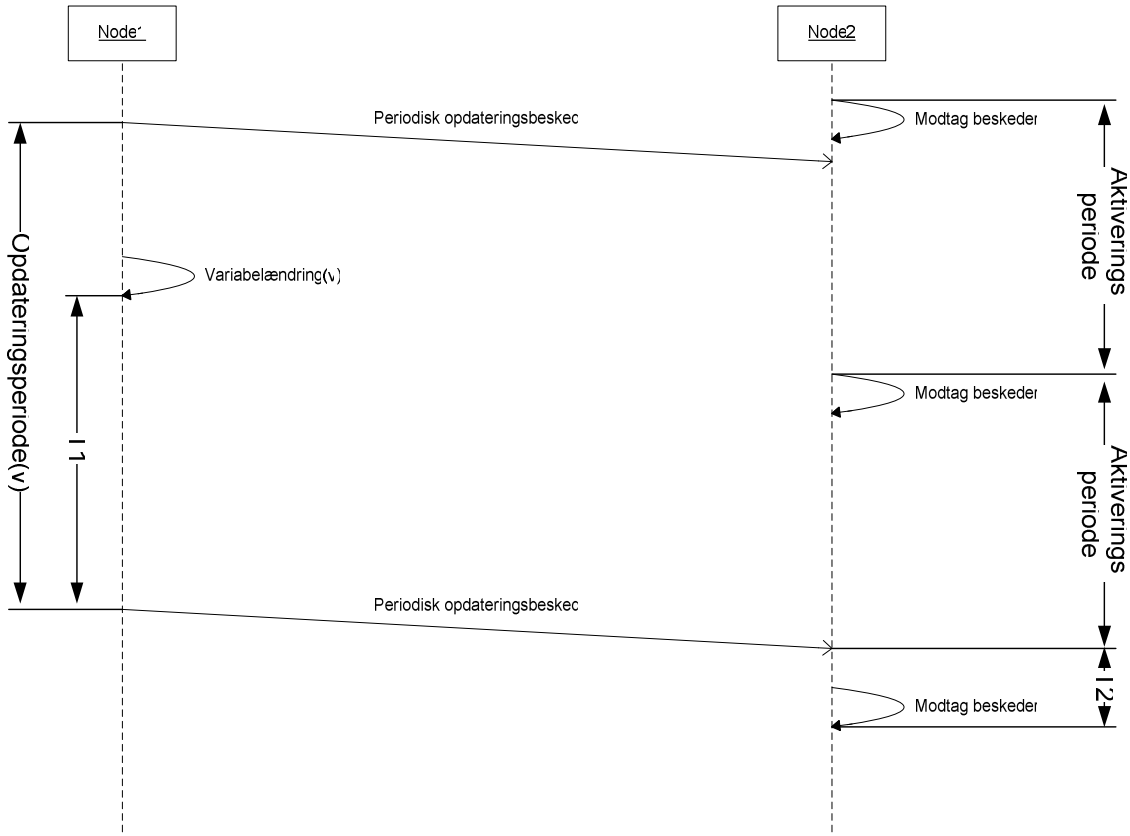
6.4.1. Valg af opdateringsperioder

Dette underafsnit beskriver den metode, som vil blive anvendt til at bestemme opdateringsperioder i forbindelse med kombinationen af løsningsforslaget og DAO. Hver kernevariabel får tilknyttet et individuelt sæt bestående af to tællere; en opdateringstæller og en timeout-tæller. De to tællere anvendes til at implementere Soft State-protokollen i forbindelse med hhv. periodiske opdateringsbeskeder og sletning. Startværdien for en kernevariabels opdateringstæller er afgørende for den frekvens, hvormed der udsendes opdateringsbeskeder for kernevariablen. Kernevariablene i et DAO-system kan grupperes efter hvor hyppigt deres værdier ændres af nodernes komponenter. Ud fra denne gruppering kan er det muligt at fastlægge frekvensen for udsendelse af opdateringsbeskeder

og dermed startværdier for opdateringstællerne. Kendskabet til ændringsfrekvenser er baseret på viden om de målinger, som repræsenteres af de forskellige variable.
Vestas har givet følgende oplysninger for frekvensen af ændringer i værdierne af et samlet systems kernevariable:

- 5 % af alle kernevariable ændres med en frekvens på 100 Hz
- 10 % ændres med 50 Hz
- 15 % ændres med 10 Hz
- 20 % ændres med 1 Hz
- 50 % ændres med >1 Hz

På baggrund af en kernevariabels unikke nøgle, kendes kernevariablens gennemsnitlige ændringsfrekvens. Ændringsfrekvensen for en given gruppe af kernevariable kendes som en statisk parameter i modellen. I modellen initialiseres noderne tilfældigt med en samling kernevariable, som fordeler sig efter ovenstående sandsynligheder. Det er vigtigt at skelne imellem forsinkelse og opdateringsfrekvens i denne sammenhæng. I afsnit 4.2.14 beskrives kravet om, at ændringer i kernevariable skal viderekommunikeres til alle noder indenfor 50 ms. Dette krav sætter en nedre grænse for frekvensen for udsendelse af opdateringsbeskeder. Det skyldes, at selvom et signal i form af en kernevariabel ændres sjældent, er det stadig vigtigt at få viderekommunikeret ændringen hurtigt - dvs. indenfor 50 ms. Et eksempel kan være en kernevariabel, hvis værdi repræsenterer tilstanden for en manuelt betjent afbryder med to tilstande; tændt eller slukket. Selvom knappen kun betjenes meget sjældent, er det vigtigt at systemet reagerer hurtigt når det sker.
Den nedre grænse for opdateringsfrekvensen, som fastsættes som en afledning af den højeste ændringsfrekvens resulterer i en nedre grænse for kommunikationskomponentens aktiveringsfrekvens. Den højeste opdateringsfrekvens fastsætter hvor ofte kommunikationskomponenten skal aktiveres, for at kunne afsende opdateringsbeskeder tilpas hyppigt. Jo hyppigere værdien af kernevariablen ændres, jo hyppigere må opdateringsbeskeder udsendes for at tilfredsstille kravet om den maksimale forsinkelse.



Figur 7 - Opdateringsperiode hos DAO-node

Figur 7 illustrerer hvordan forsinkelsen i forbindelse med en variabelændring opstår. Hos "Node1" ændrer variabel ”v” værdi ("Variabelændring(v)"). Fra det tidspunkt, hvor ændringen sker, går tiden T_1 før variablens opdateringstæller løber ud. Når tælleren udløber, bliver en opdateringsbesked med variablens ny værdi udsendt ("Periodisk opdateringsbesked"). Noderne er ikke synkroniseret indbyrdes, og derfor vil afsendelsen af beskeden fra "Node1" ske uafhængigt af aktiveringen af kommunikationskomponenten hos "Node2". Beskeden kommer frem til "Node2" via kommunikationskanalen og derefter går tiden T_2 før beskeden bliver modtaget af kommunikationskomponenten. Denne forsinkelse afhænger af hvornår kommunikationskomponenten på "Node2" næste gang bliver aktiveret - og beskeden modtages og behandles. Samtidig afhænger forsinkelsen af den forsinkelse som opstår i forbindelse med modtagelse af en besked samt af hvor mange beskeder der bliver modtaget indenfor aktiveringsperioden. Beskedforsinkelsen er en kendt konstant og antallet af modtagne beskeder har en øvre grænse, som afhænger af antallet af noder. Maksimalværdierne for de to forsinkelser (T_1 og T_2) beskriver den situation, hvor timingen imellem de to noder passer dårligst i forhold til hinanden:

$$\max(T_1) = T_{Opdateringsperiode} + T_{Besked}$$

$$\max(T_2) = T_{Aktiveringsperiode} + (n-1) * T_{Besked}$$

Formel 3

hvor
$T_{Opdateringsperiode}$ = Periodetiden for udsendelse af opdateringsbeskeder for variablen
T_{Besked} = Tiden det tager at sende eller modtage en besked
$T_{Aktiveringsperiode}$ = Periodetiden for aktivering af kommunikationskomponenten
n = Antal noder

Den samlede maksimale forsinkelse fra en variabel ændres til ændringen er videreformidlet er derfor:

$$T_{Max} = \max(T_1) + \max(T_2) = T_{Opdateringsperiode} + T_{Aktiveringsperiode} + n * T_{Besked}$$

Formel 4

Kriteriet i forhold til maksimal opdateringsforsinkelse er opfyldt når udtrykket resulterer i $T_{Max} \leq 50$ ms. For at sikre, at opdateringsbeskeder udsendes tilpas hyppigt for alle kernevariable er den lavest mulige aktiveringsfrekvens lig med den højest forekommende ændringsfrekvens. Dermed kan den øvre grænse for $T_{Aktiveringsperiode}$ fastfryses i udtrykket. Da T_{Besked} er en kendt konstant og den maksimalt tilladte værdi for T_{Max} er kendt, er det nu muligt at bestemte den højest tilladte værdi for $T_{Opdateringsperiode}$.

$$T_{Opdateringsperiode.Max} = T_{Max} - T_{Aktiveringsperiode} - n * T_{Besked}$$

Formel 5

Hvis opdateringsperioden for en given kernevariabel holdes under denne maksimumværdi, vil det være muligt at overholde den maksimalt tilladte forsinkelse i forbindelse med opdatering af variabelværdien - forudsat at opdateringsbeskeden ikke går tabt. Motivationen for at finde de rette opdateringsperioder for systemets kernevariable ligger i et ønske om at begrænse antallet af afsendte beskeder, for at begrænse kommunikationskomponentens ressourceforbrug. Desto lavere opdateringsperiode for en kernevariabel, desto flere opdateringsbeskeder vil der blive udsendt. I modellen anvendes en funktion ("find_opdat_p") til at finde opdateringsperioden for en given kernevariabel. Som udgangspunkt anvendes kernevariablens ændringsperiode som opdateringsperiode. Hvis

ændringsperiode overskrider den maksimalt tilladte opdateringsperiode, anvendes $T_{Opdateringsperiode}$ i stedet som opdateringsperiode.

6.5. Forskelle mellem eksisterende og foreslået løsning

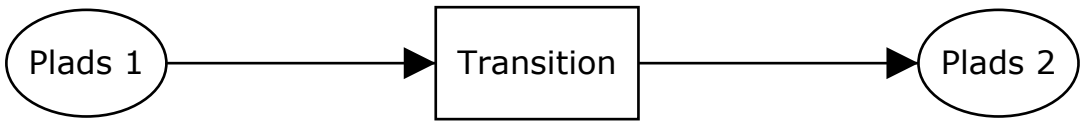
Den mest centrale forskel imellem løsningsforslaget, som præsenteres i dette afsnit og det eksisterende kommunikationssystem i DAO er, at der udsendes periodiske opdateringsbeskeder for både ændrede og uændrede variable. Ulempen ved dette er, at der vil blive udsendt flere beskeder end det er teoretisk nødvendigt. Fordelen er, at det gøres muligt at forudsige ressourceforbruget og at konsekvensen af tabte beskeder minimeres. Et tænkt eksempel på det sidstnævnte kunne være følgende:
En variabel repræsenterer tilstanden for en kontakt, som kan have to forskellige tilstande; tændt eller slukket. Kontakten er fysisk forbundet til en node, hvorpå komponenten A overvåger tilstanden. Samtidig eksisterer komponenten B på en anden node. Komponenten B kan tænde og slukke for en lampe. Når kontakten er tændt, skal lampen være tændt og når kontakten er slukket skal lampen være slukket. For at implementere denne funktionalitet er det nødvendigt for B, at kende den aktuelle tilstand for kontakten. Samtidig er det vigtigt, at skift af tilstanden kendes hurtigst muligt. I det eksisterende kommunikationssystem i DAO, vil der blive udsendt én besked, hvis kontaktens tilstand går fra tændt til slukket eller omvendt. Beskeden udsendes altså som reaktion på en ændring. Denne besked udsendes, når DAO-kernen registrerer at værdien for den variabel, som repræsenterer kontaktens tilstand er blevet ændret. Selve ændringen af variabelværdien udføres af komponenten A, som adgang til at aflæse kontaktens fysiske tilstand og samtidig har IO-tilknytning til variablen. På den anden node har komponenten B oprettet I-tilknytning til variablen. Beskeden med den ændrede værdi for variablen udsendes fra den node, hvor komponenten A eksisterer og modtages af noden, hvor komponenten B eksisterer. Når beskeden bliver modtaget, opdateres den lokale kopi af variablen med den nye værdi. Når komponenten B herefter aktiveres, vil variablen have denne rette værdi i forhold til kontaktens fysiske tilstand. I DAO opstår der et problem, hvis beskeden med den ændrede variabelværdi går tabt i forbindelse med transmissionen. Hvis det sker, vil den variabelkopi, som benyttes af komponenten B ikke blive opdateret med den nye værdi og lampen vil ikke modsvare kontaktens tilstand. Dette vil være tilfældet indtil kontaktens næste gang tændes eller slukkes, hvilket vil forårsage udsendelse af endnu en opdateringsbesked (som også kan gå tabt). I modsætning til i DAO, sker udsendelse af opdateringsbeskeder altså som en reaktion på at en fastsat tidsperiode er udløbet. Dette giver redundans (samme information udsendes flere gange), men mindsker den potentielle periode med inkonsistens, som vil opstå i forbindelse med tab af opdateringsbeskeder.

7. Coloured Petri Nets

I de foregående afsnit er DAO blevet beskrevet, efterfulgt af en beskrivelse af signaleringsprotokoller og et valg af et konkret løsningsforslag. Næste skridt er modellering af det valgte løsningsforslag. Modelleringen foregår ved hjælp af Coloured Petri Nets (CPN) [CPN1]. Som optakt til selve modelleringen af løsningsforslaget giver dette afsnit en introduktion til CPN. Dette afsnit giver en uformel beskrivelse af CPN. Formålet er at skabe et tilstrækkeligt grundlag for forståelse af de modeller, som beskrives i afsnit 8. I forbindelse med den efterfølgende beskrivelse af den konkrete modellering af løsningsforslaget, vil en række yderligere CPN-egenskaber blive beskrevet. En formel beskrivelse af CPN kan findes i [CPN1].

CPN er et modelleringssprog, som er særligt egnet til modellering af distribuerede systemer. Modelleringssproget kan repræsenteres grafisk eller i form af matematiske udtryk. Den grafiske fremstilling af modeller muliggør en intuitiv specifikation af systemer vha. modeller mens den matematiske repræsentation danner grundlag for forskellige typer formel analyse af modellerne. Beskrivelsen i dette afsnit fokuserer på den grafiske repræsentation af modeller.

CPN-modeller er bygget op af tre strukturelle grundelementer: *Pladser*, *transitioner* og *pile*. Udover disse elementer, anvendes *inskriptioner* til at specificere detaljerne for en model. Alle tre elementer kan have inskriptioner, som typisk specificeres i et funktionelt sprog. Pilene forbinder pladser og transitioner. Pladserne indeholder et antal *brikker* af en bestemt type. Mængden af brikker på en plads repræsenterer pladsens tilstand. En models samlede tilstand udgøres af mængden af brikker på modellens pladser. Modellens tilstand ændres, når brikker flyttes imellem pladser. Flytning af brikker imellem pladser sker ved at transitioner *udføres* (occurrence). En transition har via pile tilknyttet to grupper af pladser: Input- og output-pladserne. Intuitivt set, kan en transition flytte brikker fra inputpladserne til outputpladserne. Undervejs kan brikkerne evalueres og modificeres. Figur 8 viser to pladser, som via pile og én transition er forbundet. Plads 1 er inputplads for transitionen og Plads 2 er outputplads.

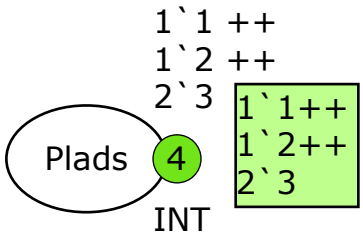


Figur 8 - Pladser forbindes med pile og transitioner

Brikker kan altså flyttes imellem modellens pladser via pilene og de mellemliggende transitioner.

7.1. Pladser og brikker

Figur 9 viser en plads med de mulige inskriptioner. Nederst til højre for pladsen ses inskriptionen "INT", som specificerer typen af brikker, som pladsen kan indeholde (i dette tilfælde heltalsværdier - dvs. af typen "INT"). Indholdet af brikker på en plads kaldes pladsens *mærkning* (marking). Enhver plads har en aktuel mærkning, som svarer til pladsens aktuelle tilstand. På Figur 9 ses den aktuelle mærkning i firkanten til højre for pladsen. Pladsers mærkning specificeres vha. en multimængde. En multimængde adskiller sig fra et almindeligt sæt, ved at multimængden har en multiplicitetsangivelse for hvert element, som angiver antallet af forekomster af elementer med den givne værdi i multimængden. Den lille cirkel med firtallet det aktuelle antal brikker på pladsen - dvs. størrelsen af pladsens aktuelle mærkning. Øverst til højre for pladsen ses en anden mærkningsspecifikation; pladsens startmærkning. Startmærkningen specificerer starttilstanden for pladsen - altså pladsens indhold af brikker for en nulstillet model før der er udført nogle transitioner.

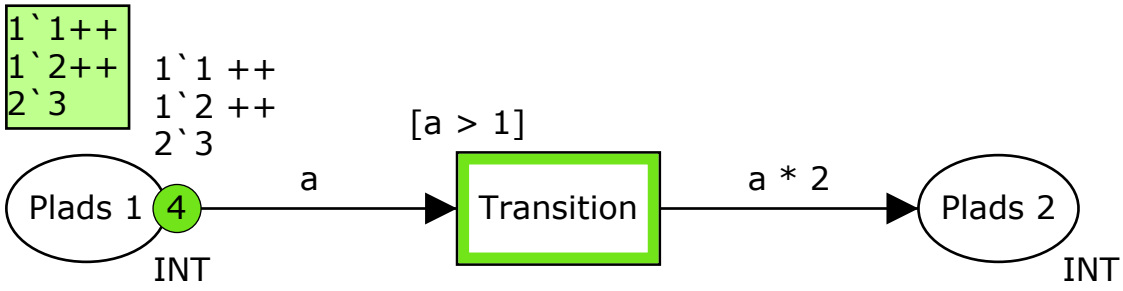


Figur 9 - En plads

I CPN er alle brikker instanser af en bestemt type og har en værdi. De mulige briktyper deklareres i inskriptionssproget og kan være simple eller komplekse typer. Der findes en lille samling simple typer i inskriptionssproget. Disse kan både anvendes direkte i modellen og indgå i deklarering af komplekse typer. De indbyggede typer omfatter simple typer såsom heltal og strenge og komplekse typer såsom lister. De indbyggede typer har tilknyttet en række metoder, som kan bruges til evaluering og manipulering af brikkernes værdier.

7.2. Transitioner og pile

Figur 10 viser to pladser og en mellemliggende transition. Plads 1 har en mængde brikker og en af disse brikker vil blive *konsumeret*, hver gang transitionen bliver udført. Samtidig vil transitionen *producere* brikker på Plads 2, hver gang den bliver udført. På pilen fra Plads 1 ses inskriptionen "a". Denne inskription er et udtryk, som er skrevet i inskriptionssproget. I dette tilfælde identificerer udtrykket en deklareret variabel. Når transitionen konsumerer en brik fra Plads 1 vil brikkens værdi blive bundet til variablen "a", som er af samme type som brikkerne på pladsen. For at dette kan lade sig gøre, skal der være en brik pladsen, som kan bindes til variablen (enabling). Desuden skal transitionens *udførselskriterium* (guard) være opfyldt. Dette kriterium er specificeret øverst til venstre for transitionen. Kriteriet er specificeret i inskriptionssproget. Udtrykket specificerer i dette tilfælde, at transitionen kan udføres for brikker med værdier over 1. Transitionen kan altså udføres, når der findes brikker på inputpladsen og når mindst én af disse brikker opfylder udførselskriteriet. Når transitionen udføres, indgår variablen i pilen, som fører til outputpladsen Plads 2. Det ses hvordan inskriptionen specificerer, at brikkernes værdi ganges med 2, før de placeres på Plads 2. Når der er flere mulige brikker på Plads 1, som kan bindes til variablen for at udføre transitionen vil valget af brik til en given udførsel af transitionen ske nondeterministisk (med mindre modellen simuleres interaktivt, hvor bindingen vælges).



Figur 10 - Mærkninger

En transition kan have flere end én inputplads. I det tilfælde kan transitionen udføres, hvis der kan konsumeres brikker fra alle inputpladserne. Transitioner kan også eksistere uden inputpladser. På samme måde kan transitioner have flere eller ingen outputpladser. Når transitioner udføres, ændres modellens samlede tilstand. Den samlede tilstand ville ændre sig på denne måde, så længe det er muligt at udføre transitioner. I eksemplet på Figur 10 vil dette være muligt indtil kun brikken med værdien 1 er tilbage på Plads 1.

Når denne tilstand er nået, vil tilstanden ikke kunne ændres fordi transitionen ikke kan udføres.

7.3. Inskriptionssproget CPN ML

Inskriptionerne i en CPN-model kan bl.a. specificere startmærkning for pladser, modificering af brikker i forbindelse med udførsel af transitioner, udførselskriterier for transitioner, udvælgelseskriterier for brikker fra inputpladser. Alle inskriptioner i CPN-modeller specificeres som udtryk vha. det funktionelle sprog CPN ML (CPN Meta Language). CPN ML er baseret på Standard ML [SML1] og hører til klassen af funktionelle programmeringssprog [FUNK1]. Et funktionelt programmeringssprog bygger på princippet fra matematiske funktioner. Dette skal ses i modsætning til procedurelle sprog som C++, Java osv., hvor programmerne bygges op omkring variable og sekvenser af kommandoer. I et procedurelt sprog kan det returnerede resultat fra en metode afhænge af tilstande udenfor metoden (fx globale eller statiske variable). Dette er ikke muligt i funktionelle programmeringssprog, hvor det returnerede resultat for en funktion kaldet med et sæt inputparametre altid vil være det samme, hver gang funktionen kaldes med dette sæt inputparametre. På samme måde vil en funktion i et funktionelt sprog aldrig påvirke ydre tilstande direkte [CPN1]. Det gør sproget egnet til brug i forbindelse med CPN, hvor der er behov for at kunne behandle modellerne (inkl. inskriptionerne) formelt. CPN ML har en række indbyggede funktioner, som bidrager til at opretholde et højt abstraktionsniveau i CPN-modellerne. Disse funktioner omfatter bl.a. mønstergenkendelse, rekursion, polymorfi og modularitet. Eksempler på anvendelse af CPN ML kan ses i afsnit 8.

7.4. Eksekvering af CPN-modeller

En af de mest grundlæggende fordele ved CPN-modeller er, at de er eksekverbare. Det betyder at de kan anvendes til at simulere det modellerede system og derved hjælpe til at indsamle viden om systemet. Simuleringen kan ske i et dedikeret modelleringsværktøj (fx CPN Tools [CPN2]). Her kan modellerne specificeres og eksekveres. Simulering kan udføres på to forskellige måder: Interaktivt og automatisk. I den interaktive simulering er det op til brugeren at afgøre rækkefølgen for udførsel af transitioner og valg af brikker i forbindelse med binding til variable. Valget sker indenfor de normale rammer for udførsel af transitioner - dvs. udførselskriterier osv. skal være opfyldt. Som alternativ til den interaktive simulering findes automatisk simulering. I dette tilfælde vil simuleringsværktøjet træffe alle valg nondeterministisk ud fra en indbygget tilfældighedsgenerator. I begge tilfælde gør CPN Tools det muligt at følge simuleringen grafisk. Modellens aktuelle

tilstand opdateres løbende og gør det muligt at overvåge pladsernes indhold af brikker. Afhængigt af sammenhæng, kan viden om det modellerede system hentes fra løbende indsamling af data under eksekveringen og/eller modellens sluttilstand efter opfyldelse af et slutkriterium. Et slutkriterium kunne være et maksimalt antal transitionsudførsler eller når det ikke er muligt at udføre flere transitioner. Udover den simuleringsbaserede analyse er det desuden muligt at analysere med udgangspunkt i modellens tilstandsrum. I denne form for analyse kan det undersøges om bestemte tilstande kan nås eller ikke nås, om modellen kan nå en fastlåst tilstand osv.

8. CPN-modellering af løsningsforslaget

Dette afsnit beskriver den CPN-model, som er udviklet til brug i den efterfølgende analyse af løsningsforslaget. Modellen er udviklet med udgangspunkt i de foregående afsnits beskrivelser af DAO, Soft State-protokoller og det valgte løsningsforslag. Beskrivelsen af modellen vil samtidig danne grundlag for en mere detaljeret gennemgang af de anvendte teknikker indenfor CPN. Flere af disse teknikker bruges i forskellige sammenhænge. De vil blive beskrevet første sted de anvendes.

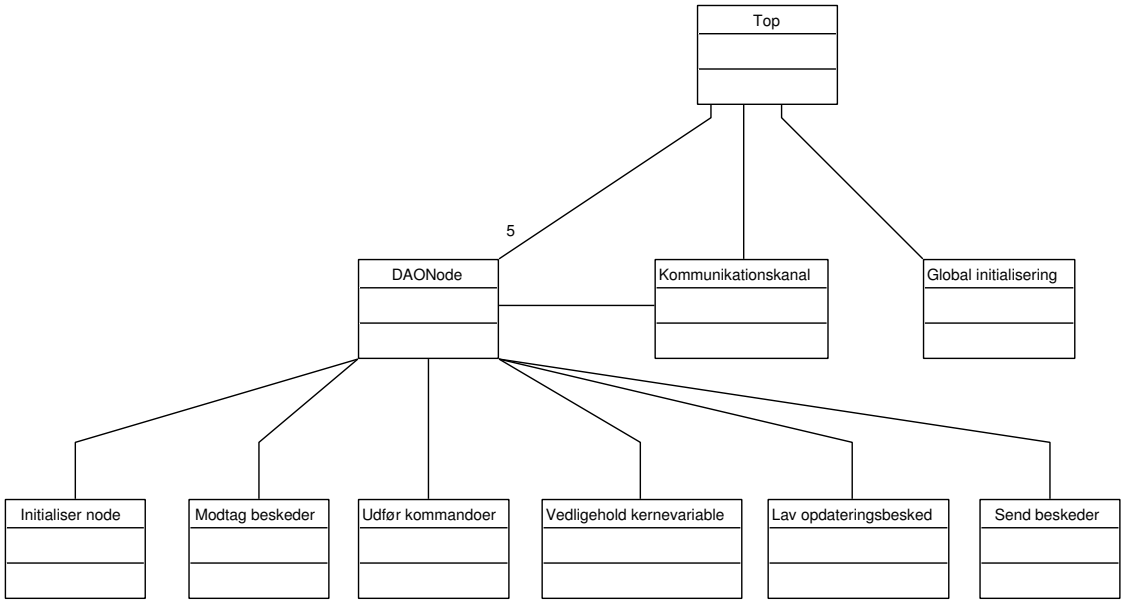
8.1. Fokus for model

Fokus i modellen er lagt på at modellere den løbende opdatering og deling af variabelværdier i et kørende system bestående af en gruppe DAO-noder. Der modelleres et statisk system i forhold til tilknytningerne mellem komponenter og noder. Alle tilknytninger initialiseres under opstarten. Hver node har de tilknytningstabeller, hvori tilknytningerne er registreret, men deres indhold vil ikke ændre sig efter initialiseringen. Set i forhold til den modellerede signaleringsprotokol og de medtagne DAO-principper betyder det, at kun udveksling af opdateringsbeskeder er modelleret. Modellen fremstiller en tilstand af systemet, hvor en række Attach-beskeder allerede er udvekslet imellem noderne. Valget af fokusering er baseret på et ønske om at undersøge mulighederne for udvikling af et konsistensoptimeret og realtidsorienteret kommunikationssystem til DAO. Selve etableringen af tilknytninger er mindre kritisk i forhold til realtidsegenskaberne og kan derfor i højere grad understøttes af traditionelle teknikker til pålidelig beskedudveksling.

8.2. Modellens hierarkiske struktur

En CPN-model kan sammensættes af en gruppe sammenhængende sider, som hver indeholder delmængder af den samlede models elementer. I dette tilfælde er modellen opdelt i en hierarkisk struktur bestående af én side på det øverste niveau og en række undersider. Den samlede hierarkiske struktur består af i alt tre niveauer. Det betyder, at der på det mellemste niveau findes sider, som på samme tid er undersider (i forhold til modellens topside på øverste niveau) og topsider (i forhold til undersider på modellens nederste niveau). På Figur 11 ses en grafisk repræsentation af modellens hierarkiske struktur i form af et entitetsrelationsdiagram [SPEC]. Et entitetsrelationsdiagram kan indeholde angivelse af kardinaliteter, som specificerer et forholdsmæssigt antal entiteter i forbindelse med en relation imellem to entiteter. På figuren er kardinaliteter på 1 udeladt. Figurens eneste kardinalitetsangivelse specificerer, at der for hver instans af siden

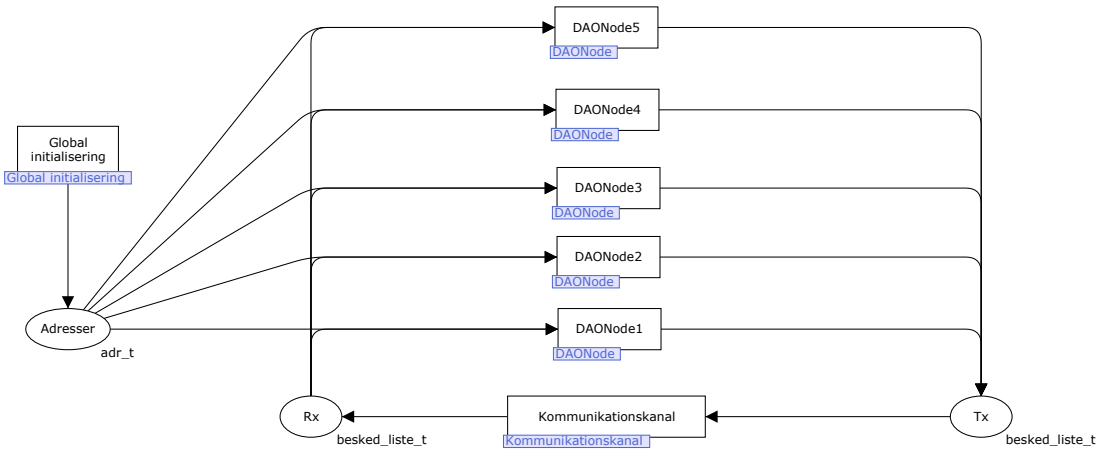
"Top" eksisterer netop fem instanser af undersiden "DAONode". Da kardinaliteter på 1 er udeladt, er det underforstået, at der eksempelvis for hver instans af siden "DAONode" findes netop én instans af undersiden "Modtag beskeder". Figuren viser desuden forbindelser imellem modellens sider.



Figur 11 - Modellens hierarkiske struktur

8.3. Modellens øverste niveau

På Figur 12 ses det øverste niveau af den hierarkiske struktur, som modellen af systemet udgør. Her beskrives det øverste niveau i grove træk. De efterfølgende afsnit indeholder mere detaljerede beskrivelser af undersiderne.



Figur 12 - Side: Top

Modellen beskriver op til 5 noder, som er forbundet via en kommunikationskanal. Antallet af aktive noder afhænger af antallet af initialiserede adresser. Alle noder deler adgang til pladsen "Adresser", hvorfra en node skal hente en unik adressebrik for at fungere som aktiv node. Antallet af aktive noder reguleres af antallet af unikke adressebrikker på pladsen "Adresser". Disse brikker placeres under initialisering af modellen af undersiden "Global Initialisering". Denne underside foretager desuden al nødvendig initialisering af modellen før opstart. Når initialiseringen er afsluttet, placeres det konfigurerede antal adresser på pladsen "Adresser". Nodernes grænseflade til kommunikationskanalen består af to FIFO-buffere - én til udgående beskeder og én til indkommende beskeder. På pladsen "Tx" kan noderne placere beskedbrikker, som vil blive modtaget af de øvrige noder. Efter at have passeret kommunikationskanalen, vil de sendte beskedbrikker blive placeret på pladsen "Rx", hvorfra de kan aftages af systemets aktive noder.

8.3.1. Substitutionstransitioner

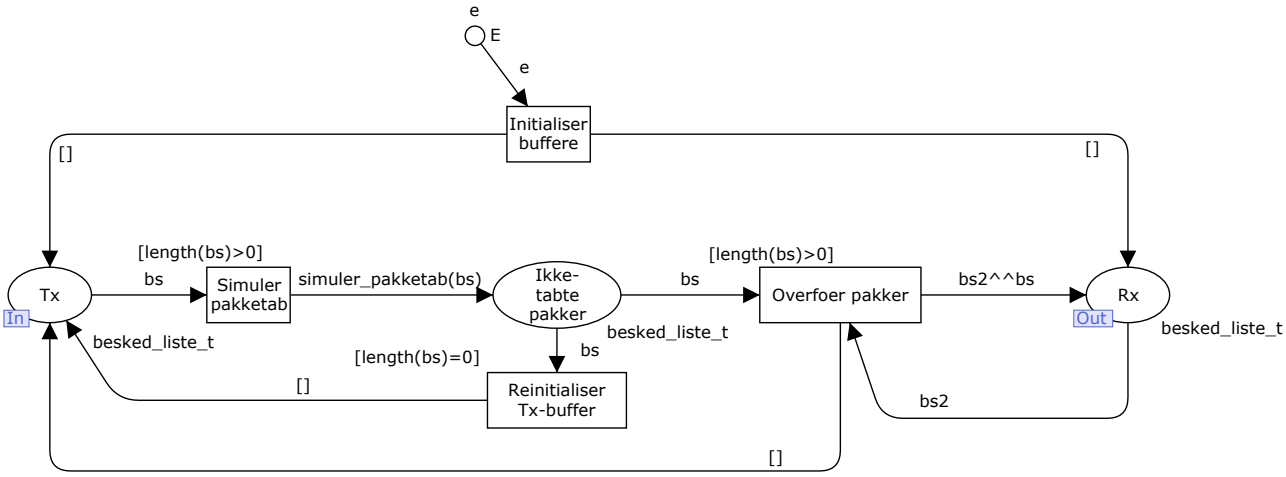
På figuren ses et mærkat på alle transitionerne: "Global initialisering", "DAONode" og "Kommunikationskanal". Disse mærkater fortæller, at transitionerne er substitutionstransitioner [CPN1]. En substitutionstransition anvendes i forbindelse med hierarkisk strukturering af en model. Mærkaterne på transitionen henviser til undersider i modellen. Undersiderne indeholder en modellering af funktionaliteten for hver af substitutionstransitionerne på et højereliggende niveau i en hierarkisk model. På det niveau, som illustreres på Figur 12, ses de forskellige transitioners forbindelse til en lille gruppe af pladser: "Adresser", "Rx" og "Tx". Disse pladser vil fungere som grænseflader for undersidernes mere detaljerede modeller. Eksempler herpå vil blive beskrevet i de efterfølgende beskrivelser af modellens undersider. Det er desuden vigtigt at bemærke, at selvom der

findes fem forskellige DAO-noder i modellen ("DAONode1...5"), henviser de alle til den samme underside; "DAONode". Resultatet heraf vil være, at der bliver genereret en instans af den samme underside for hver gang mærkatet er anvendt på det højereliggende niveau. I dette tilfælde vil der altså blive skabt i alt fem kopier af undersiden "DAONode". Undersiderne er instanser af den samme modelspecifikation. Til gengæld har de fem instanser hver deres uafhængige tilstand. Fordelingen af brikker på undersidernes pladser og dermed undersidernes tilstand vil være individuel for hver enkelt underside. Det gør, at teknikken kan benyttes til at modellere de flere samtidige DAO-noder - og samtidig undgå at skulle vedligeholde fem forskellige modeller af en DAO-node. Adressebrikkerne med unikke værdier konsumeres af undersiderne fra pladsen "Adresser", og benyttes til at differentiere de forskellige instanser af undersiderne.
Pladserne "Tx" og "Rx" er som nævnt substitutionstransitionernes grænseflade til den hierarkiske struktur. Disse pladser er repræsenteret både på det niveau, som vises på Figur 12 og i den underliggende beskrivelse af substitutionstransitionerne. De to repræsentationer kaldes hhv. substitutionstransitionens *stik* (*sockets*) og *porte*. I det konkrete tilfælde findes altså eksempelvis ét "Rx"-stik og fem "Rx"-porte med forbindelse til stikket - én på hver instans af undersiden "DAONode". Stikkene og portene vil altid indeholde samme mængde brikker på tværs af modellens hierarkiske struktur. Derimod er pladser på undersiden - dvs. detaljebeskrivelsen af substitutionstransitionen - ikke sammenkædede på tværs af undersiderne og vil derfor kunne indeholde forskellige mængder brikker. Det betyder at undersidernes interne tilstande er uafhængige af hinanden, men afhængige af hinanden i forhold til de delte pladser (stikkene og portene).

8.4. Underside: Kommunikationskanal

Kommunikationskanalen forbinder alle systemets DAO-noder. Figur 13 viser hvordan kommunikationskanalen er modelleret. Den vigtigste funktion for kommunikationskanalen er at overføre beskeder fra pladsen "Tx" hvortil alle noderne afleverer de pakker der skal sendes videre til pladsen "Rx" hvorfra alle noder modtager beskeder. De to pladser eksisterer i modellens øverste niveau og fungerer således som bindeled imellem alle noderne og kommunikationskanalen, hvis underside findes i én instans. Her ses det første simple eksempel på beskrivelse af detaljerne for en substitutionstransition: Pladserne "Tx" og "Rx" er forsynet med hhv. "In"- og "Out"-mærkater. Disse mærkater specificerer, at pladserne er kædet sammen med pladser på en side i et højereliggende niveau af modellen. Samtidig er selve undersiden ("Kommunikationskanal") kædet sammen med en enkelt transition af samme navn på den højereliggende side. I dette tilfælde findes de pladser og transitionen, som undersiden er tilknyttet, på modellens topside (se Figur 12). Denne form for sammenkædning betyder, at pladserne på undersiden og topsiden

er ens i alle aspekter: Antallet og retningen af pile til og fra pladsen, opstartstilstanden, typen af tilladte brikker osv. Mærkatet på undersidens pladser angiver en retning for sammenkædningen. For pladsen "Tx" betyder mærkatet "In", at topsiden producerer brikker til pladsen - dvs. topsiden har kun pile fra transitioner, som peger imod pladsens repræsentation på topsiden. Det modsatte er tilfældet for pladsen "Rx", hvorfra topsiden kun vil konsumere brikker. Udover disse to typer mærkater findes også typen "I/O", som anvendes når topsiden både kan konsumere brikker fra og producere brikker til den berørte plads. Sammenkædningen af pladser foretages i modelleringsværktøjet. Efter sammenkædning af to pladser, vil de to pladsers tilstande (dvs. mængden af brikker) være den samme. Der kan konsumeres og produceres brikker til og fra pladserne på både topsiden og undersiden og mængden af brikker vil ændres på samme måde som de gør for en normal plads i modellen. I forbindelse med kommunikationskanalen omfatter sammenkædningen kun to pladser; én på undersiden og én på topsiden.



Figur 13 - Side: Kommunikatioskanal

8.4.1. Simulering af pakketab

Modellen af kommunikationskanalen har til formål at simulere den fysiske forbindelse hvorigennem en gruppe af DAO-noder kommunikerer. I forbindelse med udveksling af beskeder over denne forbindelse, kan beskeder med en vis sandsynlighed gå tabt på vejen fra afsender til modtager. For at kunne analysere på en virkelighedstro model af systemet, bliver tabet af beskeder (eller pakker) simuleret i modellen for kommunikationskanalen. Simuleringen af pakketab sker i forbindelse med transitionen "Simuler pakketab", ved at udføre funktionen "simuler_pakketab" på alle beskederne fra pladsen "Tx". Beskederne findes på pladsen i form af en liste af beskeder. Funktionen tager den-

ne liste som input og generer en ny liste som output. Det genererede output repræsenterer mængden af overlevende beskeder. Tab af en pakke kan ske på to forskellige måde med hver deres sandsynlighed:

- En pakke går tabt i forbindelse med afsendelse eller transmission: Denne situation kan eksempelvis opstå, hvis beskedens data blive forvansket undervejs i transmissionen. I denne situation vil beskeden gå tabt for alle systemets modtagere. Situationen simuleres ved at fjerne en hel besked fra listen af beskeder fra pladsen "Tx" før resten af listen overføres til pladsen "Rx".
- Én eller flere af modtagerne (evt. alle) modtager ikke en afsendt pakke: Hvis en beskeds indhold forvanskes efter modtagelsen, vil beskeden gå tabt på grund af et fejlende CRC-check [UDP]. Denne situation opstår hos individuelle noder og uafhængigt af situationen hos systemets øvrige noder. Derfor vil et pakketab ikke typisk ikke berøre alle systemets noder. Situationen simuleres ved at fjerne én eller flere af modtageradresserne i én eller flere af beskederne i listen fra pladsen "Tx" før den modificerede liste placeres på pladsen "Rx". Metoden til adressering af modtagere i modellen beskrives i detaljer i afsnit 8.8.

Sandsynligheden for hver af de to situationer fordeler sig ligeligt. 1 ud af 100.000 pakker går tabt (se afsnit 4.2.13). I halvdelen af tilfældene skyldes det den ene fejlsituation og i den anden halvdel af tilfældene skyldes det den anden fejlsituation. Funktionen "simuler_pakketab" har ansvaret for at simulere begge situationer med den korrekte fordeling og samlede sandsynlighed.

8.4.2. FIFO-mekanisme

For at simulere kommunikationen imellem noderne korrekt er modellen af kommunikationskanalen bygget op omkring en FIFO-semantisk udveksling af beskeder. Denne mekanisme får kommunikationskanalen til at fungere som en FIFO-buffer, hvor beskederne vil blive modtaget af noder fra pladsen "Rx" i samme rækkefølge som de blev placeret på pladsen "Tx" [CPN3]. Noderne i et virkeligt system sidder placeret i et tætforbundet netværk uden mellemliggende routere osv. Det betyder, at beskeder vil blive modtaget i den rækkefølge de er afsendt og denne egenskab er afspejlet i modellen. Formålet med FIFO-mekanismen er at modellere denne egenskab. Sikringen af kommunikationskanalens FIFO-egenskaber er opnået ved at anvende lister af beskeder som type for brikkerne på pladserne "Tx" og "Rx". Når en node afsender en besked, vil det resultere i, at en besked-brik tilføjes i enden af den ordnede liste af beskeder, som allerede findes på

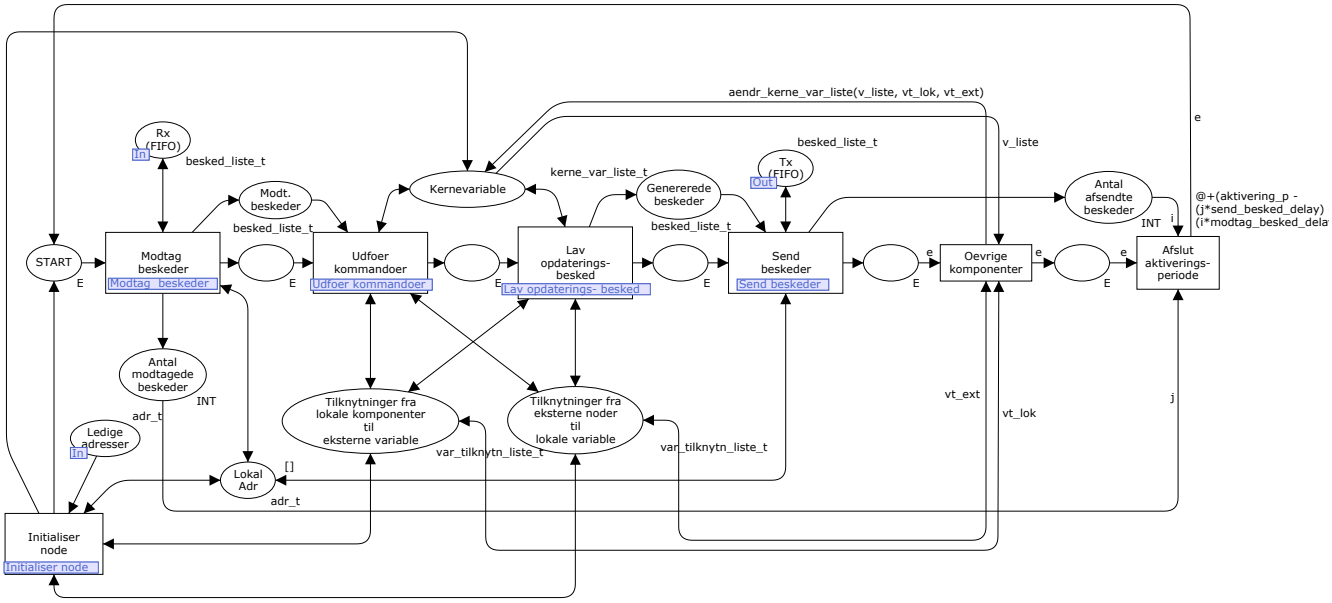
"Tx"-pladsen. På samme måde overfører kommunikationskanalen en hel ordnet liste af beskeder, som tilføjes i enden af en allerede eksisterende ordnet liste af beskeder på "Rx"-pladsen. Denne operation udføres som det ses på figuren vha. pilen fra transitionen "Overfør pakker" til pladsen "Rx". CPN ML-operatoren "^^" anvendes til sammenkædning af de to lister. Resultatet af sammenkædning er en ny liste. Rækkefølgen af de to lister i den resulterende liste afhænger af rækkefølgen af listerne i udtrykket. ”x^^y” vil give en liste med alle elementer fra listen ”x” i den oprindelige rækkefølge efterfulgt alle elementer fra listen ”y” i den oprindelige rækkefølge. FIFO-mekanismen er yderligere understøttet i modelleringen af DAO-nodernes modtagelse og afsendelse af beskeder, som beskrives i hhv. afsnit 8.8 og afsnit 8.11.

8.5. Underside: DAONode

Figur 14 viser den underside, som specificerer den grundlæggende opførsel for hver af de op til fem DAO-noder, som optræder på topsiden (afsnit 8.3). Indholdet af denne underside giver en detaljeret model for substitutionstransitionerne med mærkatet "DAONode" på topsiden. Som beskrevet tidligere, findes undersiden i et antal instanser, som passer med det konfigurerede antal noder i systemet (bestemt af parameteren "antal_noder"). Som det ses på Figur 11 udgør "DAONode"-undersiden et centralt bindeled i den hierarkiske struktur imellem topsiden og en række undersider. Modellen specificerer den sekvens af handlinger, som udføres af kommunikationskomponenten hos en DAO-node. Samtidig modelleres effekten af de øvrige komponenters manipulation af nodens variable. Hver af disse handlinger er repræsenteret i modellen på Figur 14 i form af en substitutionstransition. Hver af substitutionstransitionerne refererer til en underside, hvor detaljerne bag handlingerne specificeres. På "DAONode"-undersiden specificeres altså primært den overordnede sekvens for gennemførsel af kommunikationskomponentens opgaver. Afviklingen af opgaver er baseret på gennemløb af en tidslinie. Efter afslutning af nodens initialisering placerer transitionen "Initiliaser node" en brik på pladsen "START". Dette muliggør den første afvikling af tidslinien, som starter med pladsen ”START” og derudover består af de fem unavngivne pladser og de transitioner, som de sammenkæder. For enden af tidslinien findes transitionen "Afslut aktiveringsperiode". Denne transition anvender tidsmekanismen i CPN, til at tilpasse tidspunktet for den næste aktivering af kommunikationskomponenten (se afsnit 8.5.2). Detaljerne i forbindelse med de forskellige opgaver vil blive beskrevet i de efterfølgende afsnit, hvor undersiderne beskrives.

Undersiden findes i et konfigurerbart antal instanser. For at kunne skelne de forskellige instanser, bliver hver instans af undersiden initialiseret med en unik adresse, som placeres på pladsen "Lokal adr.". Brikken på denne plads vil indeholde en værdi af typen

"adr_t" og vil blive anvendt i forbindelse med afsendelse og modtagelse af beskeder til og fra systemets øvrige noder. For hver instans af undersiden vil brikken have en unik værdi.



Figur 14 - Side: DAO-node

8.5.1. Modellering af øvrige komponenter

Udover en enkelt kommunikationskomponent, indeholder hver DAO-node i det virkelige system en gruppe DAO-komponenter, som aktiveres periodisk på samme måde som kommunikationskomponenten. Disse komponenter implementerer systemets forskellige reguleringsalgoritmer. I den forbindelse tilgås de delte variable ved skrivning og læsning. I forbindelse med modellering af kommunikationskomponenten er det nødvendigt at simulere effekten af de øvrige komponenters skrivninger til de delte variable. Det gøres i transitionen "Øvrige komponenter", hvor funktionen "aendr_kerne_var_liste" udføres på indholdet af nodens variabeltabel før de modificerede indhold lægges tilbage på pladsen "Kernevariable". Funktionen ændrer værdien for udvalgte variable ud fra den antagne ændringsfrekvens for hver enkelt kernevariabel (se afsnit 13.2). To kriterier skal være opfyldt før værdien af en given variabel på en given node vil blive af funktionen: For variablen skal der skal findes en IO-tilknytning fra en lokal komponent og der må ikke findes nogen IO-tilknytninger fra eksterne komponenter. De to tilknytningstabeller inddrages for at efterprøve opfyldelsen af disse to kriterier. Pladsen "Tilknytninger fra lokale komponenter til eksterne variable" indeholder en tabel, som bruges til at verificere det

første kriterium og pladsen "Tilknytninger fra eksterne noder til lokale variable" indeholder en tabel, som bruges til at verificere det andet kriterium.

8.5.2. Periodisk aktivering

Kommunikationskomponenten aktiveres periodisk og dette modelleres på "DAONode"-undersiden. Den periodiske aktivering modelleres ved, at det med et fast interval gøres muligt at gennemløbe tidslinien. Én brik på denne plads gør det muligt at gennemløbe tidslinien én gang. Et gennemløb af tidslinien slutter med transitionen "Afslut aktiveringsperiode". Denne transition placerer den næste brik på pladsen "START". Formålet med den afsluttende transition er at simulere den resterende tid af en aktiveringsperiode. Mængden af resterende tid for en aktiveringsperiode kan beregnes ud fra hvor mange beskeder, kommunikationsmodulet har håndteret i løbet af den indeværende aktiveringsperiode. Periodetiden for kommunikationskomponenten er sammen med processeringstiden pr. besked konfigureret vha. tre konstanter, som indgår i modellen: "aktivering_p", "modtag_besked_forsinkelse" og "send_besked_forsinkelse". Når en aktiveringsperiode afsluttes, udregnes den resterende periodetid vha. disse konstanter sammen med oplysninger om antallet af afsendte og modtagne beskeder indenfor perioden. Resultatet af regnestykket angiver hvor lang tid transitionen "Afslut aktiveringsperiode" tager at udføre og dermed den tid der skal ventes før en ny brik placeres på pladsen "START".

8.5.3. Tid i CPN-modeller

I CPN findes en indbygget metode til at inddrage tid i modeller [CPN1]. Modellering af tid udføres ved at udvide én eller flere af typerne i modellens deklarationer med et tidsstempel. I den konkrete model er typen "E" blevet udvidet med et tidsstempel, da brikker af netop denne type anvendes til modellering af tidslinierne i modellen. Tidsstemplet repræsenteres i form af et heltal, som ledsager alle brikker af typen "E". Det grundlæggende princip i modellering af tid i CPN bygger på, at transitioner kan specificeres med en individuel forsinkelse - intuitivt betyder det, at transitioner kan have en angivelse for hvor lang tid, det tager at udføre transitionen [CPN1]. På Figur 14 ses en sådan angivelse øverst til højre for transitionen "Afslut aktiveringsperiode". Pladsen konsumerer en brik af typen "E" fra den navnløse input-plads og placerer indholdet på output-pladsen "START". En tidsangivelse på "@+x" betyder, at transitionen tager *x* tidsenheder at udføre (i denne model er tidsenheden mikrosekunder). Når transitionen udføres, vil tidsstemplet for "e" blive adderet med værdien *x*. Tidsstemplet for en brik på en plads angi-

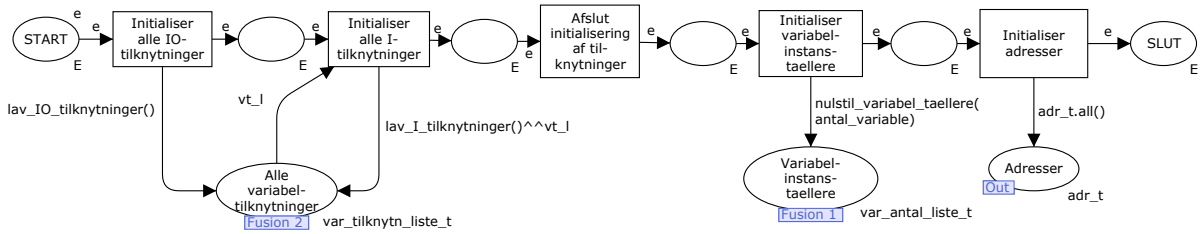
ver tidspunktet for, hvornår denne brik tidligst kan blive konsumeret fra pladsen af en transition. Når tidsstemplet for brikken opdateres, før indholdet placeres som en brik på pladsen "START", betyder det altså at den placerede brik først efterfølgende vil kunne blive konsumeret fra pladsen "START" efter *x* tidsenheder. Tidsstemplet er i forvejen blevet opdateret i forbindelse med modtagelse og afsendelse af beskeder. I det konkrete tilfælde udregnes forsinkelsestiden for transitionen "Afslut aktiveringsperiode" ud fra den samlede aktiveringsperiode og hvor mange beskeder der er blevet modtaget og afsendt. Tidsmekanismen i CPN kan betragtes som en form for karantæneperiode til brikker, hvori brikkerne ikke kan fungere i forbindelse med udførsel af transitioner. Karantæneperioden indgår i den løbende evaluering af hvilke transitioner, der er i stand til at blive udført. Som beskrevet tidligere, skal et specificeret antal brikker være til stede på inputpladserne for at en transition kan udføres. Hvis én eller flere af disse brikker er af en type med tidsstempel, vil brikkerne være usynlige indtil modeltiden er lig med eller over Når en transition udføres, vil modeltiden blive sat til det tidspunkt, hvor det afhængigt af modellens aktuelle mærkning er muligt at udføre den næste transition. Denne fremskrivning af modeltiden sker i forbindelse med udførsel af transitioner både med og uden tidsstemplede brikker på inputpladserne. Den nye værdi for modeltiden vil afhænge af brikkernes tidsstempler, fordi de er med til at afgøre tidspunktet for hvornår den næste transition kan udføres. Modeltiden er enhedsløs, uafhængig af realtiden og global for alle brikker og transitioner i modellen. Det er altså den kronologisk (ift. modeltid) ordnede konsumering af tidsstemplede brikker, som resulterer i den løbende opdatering af modeltiden. Rækkefølgen gør, at tiden opdateres i mindst mulige skridt.

8.6. Underside: Global initialisering

Figur 15 viser den underside, som i forbindelse med opstart sørger for initialisering af modellen. Den globale initialisering af modellen omfatter en gruppe af forskelligartede opgaver. Hver af disse opgaver er repræsenteret med en transition i modellen. Her følger en overordnet beskrivelse af opgaverne:

- Initialisering af tilknytninger imellem noder og variable: Denne del af initialiseringen håndterer generering af tilknytninger til variable, som er fordelt rundt hos systemets noder. Opgaven beskrives i afsnit 8.6.1.
- Initialisering af instanstællere for variable: Der initialiseres en global tabel med tællere for antallet af hver enkelt kernevariabel, som måtte eksistere i form af en instans på én eller flere af systemets noder. Tabellen anvendes til at lave målinger af konsistensen i den kørende model. De nærmere detaljer beskrives i afsnit 9.3.

- Initialisering af adresser: Pladsen "Adresser" initialiseres med et antal unikke brikker af typen "adr_t". Antallet af placerede adressebrikker er det samme som antallet af samtidigt aktive noder i modellen. Når en node bliver initialiseret vil den hente en fri adresse fra denne plads, som deles vha. princippet for substitutionstransitioner. Antallet af placerede brikker afhænger af typedeklarationen for typen "adr_t" - som igen afhænger af en konfigurationsparameter, som angiver det ønskede antal noder. Funktionen "all" anvendes på "adr_t" til at få en brik for hver mulig værdi af typen. Da typen er deklareret som et heltal med en øvre og en nedre grænse, vil funktionen resultere i et endeligt sæt af brikker.



Figur 15 - Side: Global initialisering

8.6.1. Initialisering af variabeltilknytninger

Som beskrevet i afsnit 8.1 er modellen og den efterfølgende analyse fokuseret på modellering af tilstanden for et system, hvor alle tilknytninger imellem node og variable allerede er etableret vha. af DAO-systemets Attach-beskeder (se afsnit 4.2.13). Derfor er initialisering af tilknytninger en vigtig opgave i forbindelse med den globale initialisering af modellen. En DAO-node har to forskellige tabeller, som anvendes i forbindelse med registrering af tilknytninger. Den ene tabel indeholder poster, som registrerer lokale komponenters tilknytninger til variable, som ikke findes lokalt på noden. Denne tabel anvendes til at vurdere, om en indkommende opdateringsbesked for en given variabel skal accepteres. Den anden tabel anvendes til registrering af tilknytninger til lokale kernevariable fra komponenter, som ikke eksisterer lokal på noden. Denne tabel anvendes til at afgøre, for hvilke lokale kernevariable det er nødvendigt at udsende opdateringsbeskeder. Detaljerne for anvendelsen af de to tabeller beskrives i afsnit 8.9 og afsnit 8.10.
Genereringen af tilknytninger er delt op i to faser: Initialisering af IO-tilknytninger og initialisering af I-tilknytninger. IO-tilknytninger er kendetegnet ved, at den komponent, som opretter tilknytningen vil både skrive til og læse fra den berørte kernevariabel. Adgangen til variable i DAO er begrænset således, at maksimalt én komponent kan have IO-tilknytning til en given kernevariabel. For at overholde dette krav, er initialiseringen

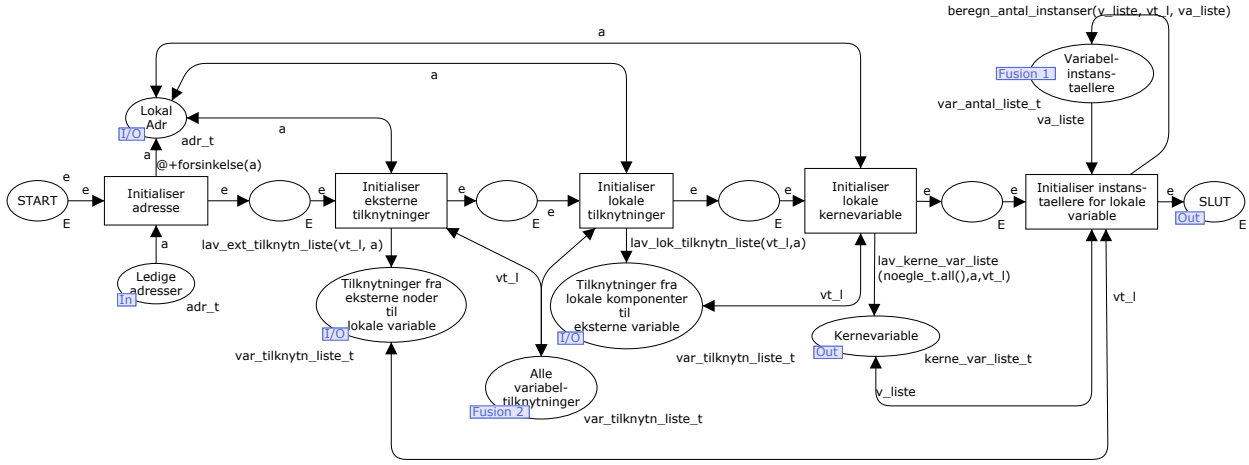
af de to typer af tilknytninger delt op. Begrænsningen findes ikke i forbindelse med I-tilknytninger, som er tilknytninger fra komponenter, som udelukkende læser værdien af kernevariable. Initialiseringen af begge typer tilknytninger sker ved at samle en mængde sæt bestående af en nøgle og en adresse, som identificerer henholdsvis den berørte kernevariabel og noden hvorpå den tilknyttende komponent er lokal. Sættene genereres efter en simpel deterministisk algoritme af funktionerne "IO_noegle_til_adr" og "I_noegle_til_adr". Disse funktioner giver en adresse for den tilknyttende komponent ud fra en nøgle og sikrer dels en ligelig spredning af tilknytninger på systemets noder og dels et deterministisk grundlag for efterfølgende indsamling af analysedata. Placeringen af tilknytningerne kan efterfølgende ske ud fra adresserne i forbindelse med initialisering på node-niveau (se afsnit 8.7). Det totale antal tilknytninger og fordelingen imellem IO- og I-tilknytninger konfigureres vha. en gruppe deklarerede modelkonstanter. I forbindelse med analyse vha. modellen, baseres konstanternes værdier på oplysninger fra Vestas.

8.6.2. Fusionspladser

På modellen i Figur 15 anvendes endnu en teknik til strukturering af CPN-modeller; fusionspladser [CPN1]. På figuren ses det, at flere af pladserne har et fusionsmærkat. Eksempelvis har pladsen "Alle variabeltilknytninger" mærkatet "Fusion 2". Dette mærkat kæder pladsen sammen med alle andre pladser med det samme fusionsmærkat. Mærkatet specificerer at pladsen tilhører fusionssættet "Fusion 2". Alle pladser med det samme fusionsmærkat vil altid have det samme indhold af brikker - antal, typer og værdier. I modsætning til delte pladser i forbindelse med substitutionstransitioner, fungerer fusionspladser ikke som en hierarkisk mekanisme. Sammenkædningen imellem pladser eksisterer på tværs af hierarkiske lag og inddelinger i undersider. Der eksisterer altså ikke en uafhængig plads for hver underside, hvor pladsen findes. Det betyder at

8.7. Underside: Initialiser node

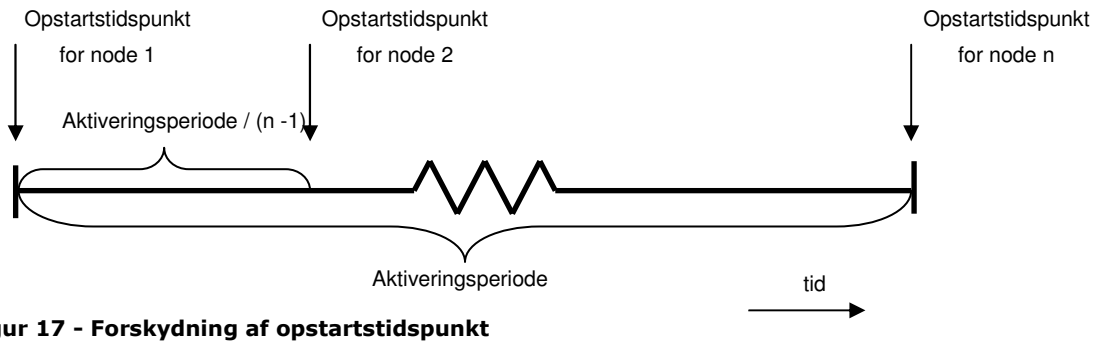
Denne underside specificerer detaljerne for substitutionstransitionen "Initialiser node" på den højereliggende side "DAONode" (afsnit 8.5). Undersiden indeholder en sekvens af operationer. Sekvensen vil blive gennemløbet én gang for hver DAO-node, som bliver initialiseret for at kunne deltage i kommunikationen med de øvrige noder. Operationerne omfatter bl.a. initialisering af nodens adresse, logfil, tilknytningstabeller og lokale kernevariable. De mest centrale operationer beskrives i de efterfølgende underafsnit.



Figur 16 - Side: Initialiser node

8.7.1. Opstartstidspunkt for node

I et virkeligt system med en gruppe af DAO-noder, findes der ingen indbyrdes synkronisering af noderne. For at simulere denne egenskab i modellen, er der i forbindelse med initiliaseringen af DAO-noderne indlagt en tidsforskydning i forhold til modellens samlede opstartstidspunkt, som er individuel for hver enkelt node. Forskydningen udvælges så nodernes opstart spredes over den specificerede aktiveringsperiode med længst mulig forskydning imellem de forskellige noder. Målet med dette er at give en modellering af den værst mulige situation i forhold til den resulterende forsinkelse i forbindelse med opdatering af ændrede variable (se afsnit 6.4.1). Dette er nødvendigt, for at kunne undersøge løsningsforslaget i forhold til kravet om maksimal opdateringstid for ændrede variable. I et virkeligt system vil noderne starte i tilfældig rækkefølge og med tilfældig indbyrdes forskydning. I modellen kan både opstartrækkefølge og indbyrdes forskydning bestemmes ud fra nodernes adresser. Det giver et deterministisk grundlag for indsamling af data fra modellen. Opstartsforskyndingen for en given node med adressen *a* findes vha. funktionen "forsinkelse" med *a* som parameter.



Figur 17 - Forskydning af opstartstidspunkt

Figur 17 skitserer hvordan opstarten af modellens noder placeres i forhold til hinanden i en model med *n* noder.

8.7.2. Initialisering af nodens adresse

Pladsen "Ledige adresser" er igennem modellens hierarkiske struktur kædet sammen med pladsen "Adresser" på topsiden. Som tidligere nævnt initialiseres denne plads under den globale initialisering med et antal forskellige adresser, hvis antal er det samme som det ønskede antal noder i systemet under simulering. Transitionen "Initialiser adresse" kan udføres, hvis det er muligt at hente en adresse-brik fra pladsen "Ledige adresser". Dette er muligt efter gennemførsel af den globale initialisering og så længe der ledige adresser tilbage. Hvis det er muligt tages én af de ledige adresse-brikker og placeres på pladsen "Lokal adr.". Denne brik vil efterfølgende blive brugt til at identificerer den konkrete instans af DAONode-undersiden i forbindelse med initialisering samt afsendelse og modtagelse af beskeder.

8.7.3. Initialisering af nodens tilknytninger

Modellen fokuserer på at danne grundlag for analyse af den del af kommunikationen imellem noderne, som vedrører vedligeholdelse af et konsistent billede af værdierne for de delte variable. Derfor udelades modellering af udveksling af Attach-beskeder imellem noderne. Det giver bedre mulighed for at kontrollere fordelingen af variable og dermed skabe kontrollerede rammer for analysen af systemet på baggrund af modellen. Når DAO's Attach-mekanismer udelades i modellen er det nødvendigt at anvende en alternativ metode til at etablere simulerede tilknytninger imellem komponenter på noder og variable på andre noder. Genereringen af en repræsentativt fordelt samling af tilknytninger er beskrevet i afsnit 8.6.1. I forbindelse med initialisering af den enkelte node er opgaven simplere. Tilknytninger er modelleret vha. af brikker den komplekse type "var_tilknytn_retn_t". Typen er en kombination af flere elementer; en nøgle, en adresse og en retning. Nøglen identificerer den berørte variabel, adressen identificerer en node og retningen beskriver typen af tilknytning (I eller IO). Hver node har to tabeller med tilknytninger; én tabel med tilknytninger fra eksterne noder til lokale kernevariable (dvs. kernevariable som eksisterer på den lokale node) og én tabel med tilknytninger fra lokale komponenter til variable på eksterne noder. De to tabeller er repræsenteret i modellen

med hver deres plads, som kan indeholde lister af tilknytningsbrikker. Indholdet initialiseres under initialisering af noden af de to pladser "Initiliaser eksterne tilknytninger" og "Initiliaser lokale tilknytninger". Disse to brikker tager tilknytningsbrikker fra pladsen "Alle variabeltilknytninger" hvor den samlede mængde af tilknytninger blev placeret under den globale initialisering. Udvælgelsen af relevante tilknytningsbrikker for en aktuel node sker på baggrund af den allerede initialiserede lokale adressebrik. Der anvendes to funktioner i forbindelse med udvælgelsen; "lav_lok_tilknytn_liste" og "lav_ext_tilknytn_liste". Disse to funktioner kan udvælge relevante tilknytninger på baggrund af en viden om, hvorvidt en given kernevariabel eksisterer lokalt på den aktuelle node. I modellen baseres denne viden alene på nodens nøgle. I et virkeligt system, vil den samme viden eksistere i kraft af at de lokale kernevariable vil blive skabt under initialisering af DAO og nodens komponenter. Lokale tilknytninger (fra lokale komponenter til eksterne variable) fra den samlede liste af tilknytninger accepteres, hvis adressen i tilknytningen er den samme som den lokale adresse. Placeringen af disse tilknytninger hos systemets forskellige noder er således bestemt under den globale initialisering. Eksterne tilknytninger (fra eksterne noder til lokale variable) vil blive accepteret, hvis den berørte variabel eksisterer lokalt på den aktuelle node. Alle accepterede tilknytninger tages fra den samlede liste og placeres i de to tilknytningstabeller. Herfra vil de senere blive anvendt i forbindelse med håndtering af indkommende og generering af udgående opdateringsbeskeder.

8.7.4. Initialisering af nodens lokale kernevariable

Som den sidste del af nodens initialisering oprettes nodens kernevariable. Alle kernevariable i systemet vil eksistere i form af én original og et antal kopier hos alle noder med tilknytning til variablen. Kernevariable repræsenteres i modellen vha. brikker af typen "kerne_var_t". Denne komplekse type har et felt, "lokal", som bruges til at angive om en given instans af en kernevariabel er en original eller en kopi. Samlingen af kernevariable på en node består af nodens lokale kernevariable samt kopier af andre noders kernevariable, hvortil lokale komponenter på noden har IO-tilknytninger. Den førstnævnte gruppe af variable kan oprettes alene på baggrund af den lokale adresse, mens den sidstnævnte gruppe oprettes ud fra oplysninger om tilknytninger fra tabellen over lokale tilknytninger. Der er under den globale initialisering taget hensyn til kravet om, at maksimalt én komponent har IO-tilknytning til en given variabel. Under simulering af modellen, vil gruppen af kopier af kernevariable blive suppleret med kopier for de variable, hvortil lokale komponenter har I-tilknytninger. Dette sker efterhånden som opdateringsbeskeder med oplysninger om disse variables værdier kommer ind fra de eksterne no-

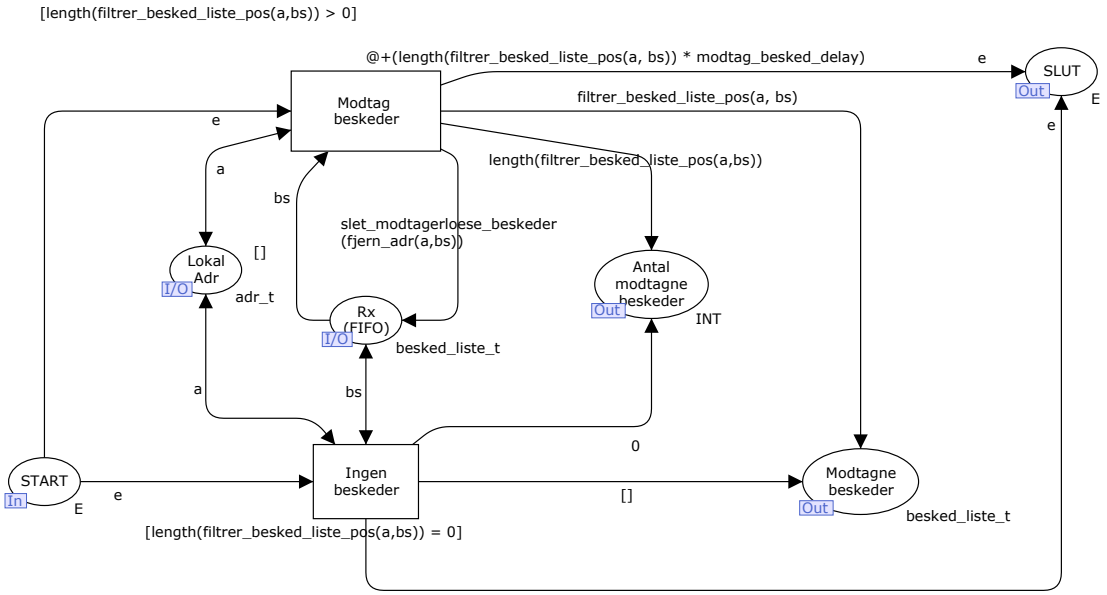
der. De initialiserede kernevariable placeres i form af en liste af brikker på pladsen "Kernevariable", som er kædet sammen med en plads på det højereliggende niveau, hvorfra de senere kan tilgås i forbindelse med den løbende opdatering af variabelværdier.

8.7.5. Afslutning af initialisering = opstart af node

Initialisering af en node afsluttes med, at der placeres en brik på pladsen "SLUT". Denne plads er igennem modellens hierarkiske struktur kædet sammen med pladsen "START" på den "DAONode"-siden. Derfor vil placeringen af brikken i forbindelse med afslutning af initialiseringen resultere i den første aktivering af noden. Herefter vil en brik periodisk blive anbragt på pladsen og resultere i periodisk aktivering af noden. Tidspunktet for placeringen af den første brik på pladsen afgøres af den forsinkelse, som indledte initialiseringen af noden.

8.8. Underside: Modtag beskeder

Figur 18 viser detaljerne for substitutionstransitionen "Modtag beskeder" fra undersiden "DAONode". Denne underside eksisterer, ligesom de øvrige undersider på samme hierarkiske niveau, i en uafhængig instans for hver enkelt node. Undersiden håndterer modtagelse af beskeder fra andre noder via kommunikationskanalen. Undersidens grænseflade til kommunikationskanalen udgøres af pladsen "Rx (FIFO)". Denne plads har via modellens hierarkiske struktur forbindelse til pladsen ved navn "Rx" på topsiden. Undersiden "Kommunikationskanal" har på samme måde forbindelse pladsen på topsiden. Som beskrevet i afsnit 8.4.1, fungerer kommunikationskanalen som en FIFO-buffer for at modellere den direkte forbindelse imellem afsender og modtagere realistisk. Denne mekanisme understøttes yderligere på denne underside, hvor beskederne fra input-pladsen hentes som en ordnet liste. Når en node modtager beskeder, simuleres DAO's forsinkelse i forbindelse med modtagelse af beskeder.



Figur 18 - Side: Modtag beskeder

8.8.1. Modellering af multicast-kommunikation

Alle beskeder i systemet sendes fra én afsender til alle aktive modtagere. Det er tidligere beskrevet hvordan antallet af aktive noder konfigureres ved at justere antallet af ledige adresser under den globale initialisering. Med udgangspunkt i samme parameter, er det muligt for en afsendende node at kende adresserne på alle aktive noder i systemet. Når beskeder sendes via kommunikationskanalen, bliver de modelleret vha. brikker af den komplekse type "besked_t". Denne type indeholder adresser for afsender og modtagere. Modtageradresserne repræsenteres i form af en liste af elementer af typen "adr_t". Hver node er identificeret med en unik værdi af typen "adr_t". Multicasting modelleres ved at lade modtagerlisten indeholde adressen på alle andre noder end afsenderen, når beskeden afsendes. Når en modtager har modtaget beskeden, fjernes modtagerens adresse fra modtagerlisten. Beskederne eksisterer i form af beskedbrikker i kommunikationskanalen, så længe modtagerlisten stadig indeholder adresser. Når modtagerlisten er tom, antages det at alle modtagere har modtaget beskeden. Hver enkelt node fjerner sin egen adresse fra modtagerlisten i hver enkelt modtaget besked i listen af beskeder. Når modtagerens egen adresse er fjernet fra modtagerlisten i alle beskeder i listen, placeres beskedlisten tilbage på pladsen "Rx (FIFO)". Hvis en modtagerliste i en besked i listen efter fjernelse af den lokale adresse er tom, fjernes beskeden fra listen. På denne måde har den sidste node, som modtager en given besked ansvaret for at fjerne den definitivt fra kommunikationskanalen. Når adressen er fjernet, vil den samme besked ikke blive modtaget af den aktuelle node næste gang undersiden "Modtag beskeder aktiveres". Opga-

ven udføres ved at transitionen "Modtag beskeder" tager hele listen af indkommende beskeder fra inputbufferen. Den lokale adresse hentes samtidig fra pladsen "Lokal adr.". Vha. den lokale adresse filtreres listen af beskeder ud fra kriteriet om at den lokale adresse skal eksistere i beskedernes liste over modtageradresser. Alle beskeder hvor dette kriterium er opfyldt placeres på pladsen "Modtagne beskeder". Samtidig placeres en heltalsbrik på pladsen "Antal modtagne beskeder". Der indgår tre funktioner til at udføre opgaven:

- "filtrer_besked_liste_pos"
  Denne funktion tager den samlede liste af indkommende beskeder som input og returnerer den delmængde af listen, hvori den lokale adresse findes i listen over modtagere.

- "fjern_adr"
  Funktionen fjerner den lokale adresse fra modtagerlisten efter filtrering af beskedlisten. Funktionen anvendes til at modificere listen af beskeder før den placeres tilbage på pladsen "Rx (FIFO)".

- "slet_modtagerloese_beskeder"
  Denne funktion anvendes til at den modificerede liste af beskeder efter den lokale nodes adresse er fjernet fra alle beskederne. Funktionen tager den modificerede liste som input og returnerer en liste, som kun indeholder de beskeder, hvor der stadig findes modtageradresser i listen over modtagere.

På figuren ses desuden transitionen "Ingen beskeder", som udføres hvis der ikke findes beskeder til den aktuelle node i listen af beskeder.

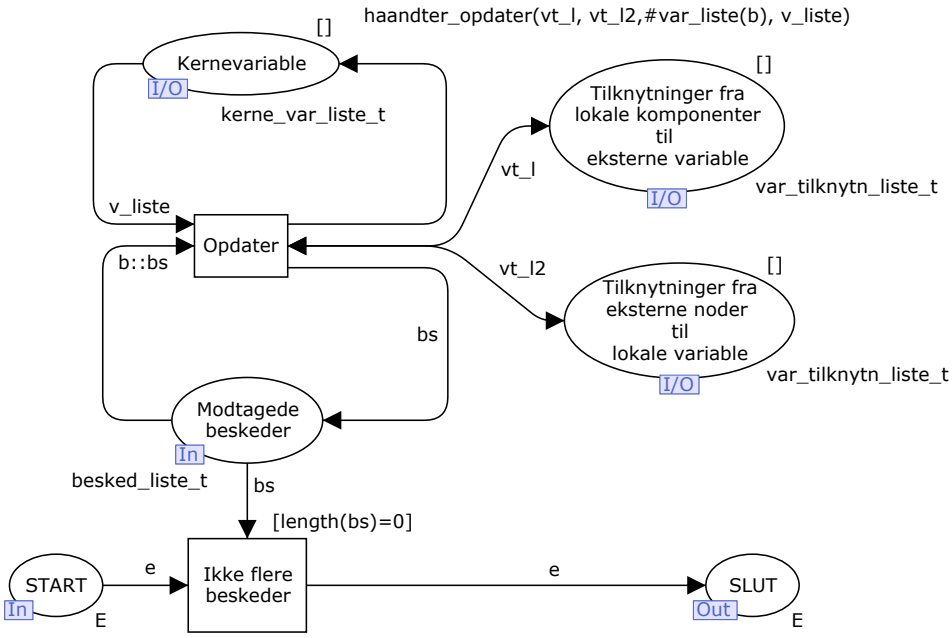
8.8.2. Modellering af forsinkelse

I afsnit 4.2.13 beskrives den forsinkelse der opstår i DAO-kernen i forbindelse med afsendelse og modtagelse af beskeder via kommunikationsstakken. Forsinkelsen modelleres, ved at anvende tidsmekanismen i CPN til at fastsætte en udførselstid for transitionen "Modtag beskeder". Længden af den gennemsnitlige forsinkelse for hver afsendt besked kendes ud fra måling på en kørende node hos Vestas. Denne tidsangivelse indgår i form af en konstant (modtag_besked_forsinkelse) som en input-parameter til modellen. På Figur 18 ses det ved transitionen "Modtag beskeder", hvordan denne konstant ganges sammen med antallet af modtagne beskeder Det gøres for at finde den samlede forsin-

kelse som forårsages af modtagelsen af beskeder for denne aktivering af kommunikationskomponenten. Forsinkelsen bestemmer hvor lang tid der går, før afviklingen af substitutionstransitionen afsluttes. Afslutningen sker ved at der placeres en brik på pladsen "SLUT", som er kædet sammen med en synkroniseringsplads på den side, hvor substitutionstransitionen findes.

8.9. Underside: Udfør kommandoer

Når en node har modtaget opdateringsbeskeder fra kommunikationskanalen, skal nodens kopier af de berørte kernevariable opdateres med værdierne fra opdateringsbeskeden. Figur 19 viser den del af modellen, som håndterer denne opgave. Figuren indeholder detaljerne fra substitutionstransitionen "Udfoer kommandoer", som optræder på undersiden "DAONode".



Figur 19 - Side: Udfør kommandoer

Én opdateringsbesked kan indeholde værdier for flere kernevariable. For hver af disse variable skal ét af to mulige kriterier være opfyldt før en opdatering gennemføres:

1. En lokal komponent har I-tilknytning til variablen
2. En ekstern komponent har IO-tilknytning til variablen

Hvis kernevariablen ikke i forvejen eksisterer i nodens lokale tabel over kernevariable, vil den blive oprettet. Denne situation kan opstå, hvis variablen er blevet slettet fra tabellen, eller hvis den modtagne opdateringsbesked for variablen er den første. Funktionen "haandter_opdater" tager imod den første opdateringsbesked fra den ordnede liste af indkommende beskeder. Ud fra denne besked opdateres listen af kernevariable. Denne liste opbevares på pladsen "Kernevariable". I forbindelse med godkendelse af indkommende opdateringsbeskeder for specifikke kernevariable, benyttes brikker fra pladserne "Tilknytninger fra lokale komponenter til eksterne variable" og "Tilknytninger fra eksterne noder til lokale variable". Disse to brikker indeholder nodens tilknytningstabeller, som benyttes til at undersøge om ét af de ovennævnte kriterier for accept af opdatering er opfyldt. Hvis en opdatering accepteres, vil værdien for den berørte kernevariabel blive opdateret i listen. Da protokollen er baseret på periodisk udsendelse af opdateringsbeskeder, vil en opdateringsbesked ikke nødvendigvis indeholde ændrede værdier for de berørte variable. De indkommende beskeder hentes fra pladsen "Modtagede beskeder" og nodens liste over kernevariable hentes fra og placeres i modificeret form på pladsen "Kernevariable". Begge pladser er via modellens hierarkiske struktur forbundet med pladser på den højereliggende side "DAONode". Når der ikke findes flere beskeder i listen over modtagne beskeder afsluttes undersiden ved at placere en brik på pladsen "SLUT". I denne sammenhæng sikrer tidslinien, at listen af kernevariable hos en node ikke bliver modificeret andre steder i modellen mens de indkommende opdateringsbeskeder bliver behandlet.

8.10. Underside: Lav opdateringsbesked

Undersiden på Figur 20 viser detaljerne for substitutionstransitionen "Lav opdateringsbesked" fra undersiden "DAONode". Undersiden håndterer generering af periodiske opdateringsbeskeder, som sendes til systemets øvrige noder. Nodens kernevariable er samlet i en tabel, hvor hver enkelt kernevariabel har tilknyttet to tællere: En opdateringstæller og en timeout-tæller. Disse to tællere er centrale i princippet for den modellerede Soft State-protokol. Begge tællere bliver løbende nedskrevet. Når en opdateringsbesked modtages for en variabel, bliver begge variablens tællere initialiseret til startværdien. Hvis en tæller for en kernevariabel løber ud (dvs. når til værdien 0) har det to forskellige mulige effekter:

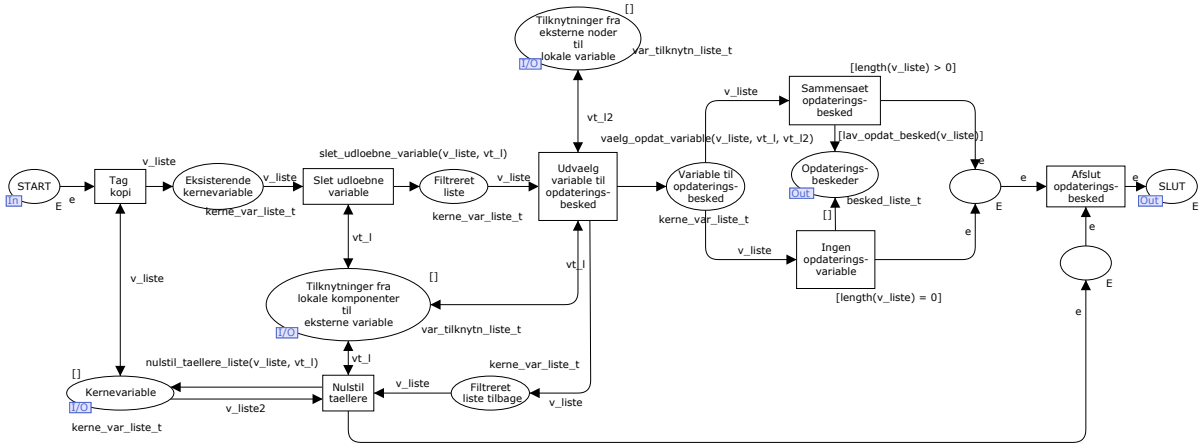
- Udløb af opdateringstæller: Der skal udsendes en opdateringsbesked med kernevariablens aktuelle værdi, hvis ét af følgende to kriterier er opfyldt:
    - Kernevariablen er lokal og der findes I-tilknytninger fra eksterne komponenter til kernevariablen

      eller
    - En lokal komponent har IO-tilknytning til variablen

- Udløb af timeout-tæller: Hvis begge følgende to kriterier er opfyldt, skal kernevariablen slettes fra listen af kernevariable:
    - Kernevariablen er ikke lokal

      og
    - Der findes ingen IO-tilknytninger fra lokale komponenter til kernevariablen

Tællernes startværdier er konstante og individuelle for de enkelte kernevariable. Opdateringstællerens startværdi angiver hvor ofte, der vil blive udsendt opdateringsbeskeder. Startværdien for timeout-tælleren angiver hvor lang tid der går, før en kernevariabel slettes fra tabellen hos en node i tilfælde af udeblivende opdateringsbeskeder.



Figur 20 - Side: Lav opdateringsbesked

De følgende underafsnit gennemgår med udgangspunkt i Figur 20 de opgaver, som udføres af undersiden.

8.10.1. Sletning af udløbne variable

Når udførslen af undersiden startes fra transitionen "START", hentes den aktuelle liste af kernevariable fra den hierarkisk delte plads "Kernevariable". Transitionen "Slet udloebne variable" sletter de variable i listen, hvor timeout-tælleren er udløbet - forudsat de ovenfor beskrevne kriterier er opfyldt. Sletningen foretages af funktionen "slet_udloebne_variable" og resultatet placeres på pladsen "Filtreret liste". I forbindelse med filtreringen af listen, hentes en brik indeholdende listen over lokale komponenters tilknytninger fra pladsen "Tilknytninger fra lokale komponenter til eksterne variable". Indholdet af denne brik bruges til at afgøre, om en kernevariabel er en lokal kopi, som indgår i en IO-tilknytning til en kernevariabel på en anden node. I dette tilfælde skal en udløben timeout-tæller ikke resultere i at kernevariablen slettes.

8.10.2. Udvælgelse af variable til opdateringsbesked

Transitionen "Udvaelg variable til opdateringsbesked" overtager den filtrerede liste af kernevariable fra pladsen "Filtreret liste". Den næste opgave er at udvælge de kernevariable, hvor opdateringstælleren er udløbet. Hvis opdateringstælleren for en kernevariabel er udløbet og et af kriterierne for tilknytning er opfyldt, vil kernevariablen indgå i den genererede opdateringsbesked. Funktionen "vaelg_opdat_variable" foretager udvælgelsen på baggrund af oplysninger om eksterne og lokale tilknytninger. Resultatet af udvælgelsen placeres på pladsen "Variable til opdateringsbesked" i form af en liste af variable. Denne liste tages fra pladsen af transitionen "Sammensaet opdateringsbesked", som generer den samlede opdateringsbesked ved at integrere listen af variable i en brik af typen "besked_t". Denne type indeholder alle nødvendige adresseoplysninger osv. for en besked, der skal afsendes til systemets øvrige noder. Samtidig indeholder den altså oplysninger om værdierne af en gruppe af kernevariable. Beskeden adresseres til alle andre noder end den, hvorfra den afsendes. Det betyder at alle opdateringsbeskeder multicast'es til alle aktive noder. Der tages altså ikke hensyn til den lokale viden om tilknytninger til kernevariable i denne sammenhæng. Hvis en opdateringsbesked modtages på en node, er det den lokale kommunikationskomponents opgave at godkende opdateringerne på baggrund af denne nodes kendskab til tilknytninger.

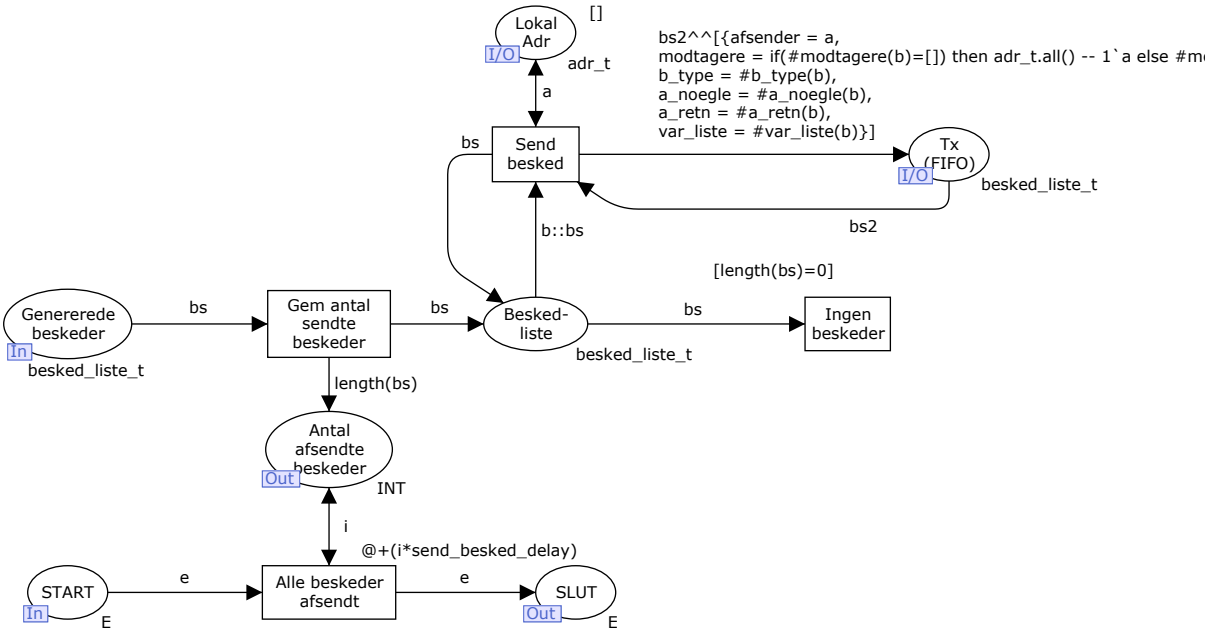
8.10.3. Opdatering af tællere

Før den filtrerede liste af kernevariable sendes tilbage til pladsen "Kernevariable", aktiveres transitionen "Nulstil taellere". Da der nu er udsendt opdateringsbesked for alle

relevante kernevariable, skal opdateringstællerne i de enkelte kernevariable nulstilles. Tællerne bliver nulstillet med deres individuelle startværdier.

8.11. Underside: Send beskeder

Når en opdateringsbesked er blevet genereret med de relevante kernevariable, skal den sendes til systemets øvrige noder. Denne opgave udføres af substitutionstransitionen "Send beskeder", hvis detaljer vises i Figur 21. Den genererede opdateringsbesked indeholder værdier for en liste af kernevariable. Hvis ingen af nodens kernevariable har udløbne opdateringstællere, vil der ikke være genereret en opdateringsbesked, og undersiden "Send beskeder" har ingen funktion. Hvis der til gengæld er blevet genereret en opdateringsbesked, vil den være placeret på den hierarkisk delte plads "Opdateringsbeskeder". Opdateringsbeskeder afsendes ved at placere en besked bagerst i den ordnede liste af beskeder på pladsen "TX (FIFO)". Denne plads er nodens grænseflade til kommunikationskanalen. I forbindelse med afsendelse af en besked anvendes funktionen "lav_adresser" til at udfylde listen over modtageradresser. Opdateringsbeskeder udsendes til alle noder i systemet. Som beskrevet i afsnit 8.8.1 modelleres dette ved at indsætte adresserne for alle andre noder end den afsendende i en liste over modtageradresser i den afsendte besked. Modtagende noder vil herefter tage beskedbrikken fra kommunikationskanalen, fjerne deres egen adresse fra listen og placere brikken tilbage i kommunikationskanalen indtil adresselisten er tom - dvs. modtaget af alle. Dette giver bevidst ikke nogen garanti for pålidelighed, da en fejl i forbindelse med modtagelse af en besked modelleres efter tilbageplaceringen af brikken i kommunikationskanalen. Al modellering af upålidelighed i forbindelse med kommunikation simuleres i forbindelse med modellering af kommunikationskanalen (se afsnit 8.4.1) og modtagerdelen hos noderne.



Figur 21 - Side: Send beskeder

På samme måde som i forbindelse med modtagelse af beskeder, modellerer denne del af modellen det ressourceforbrug i form af processeringstid, som bliver brugt på at håndtere netværkskommunikationen i kommunikationskomponenten. Denne modellering sker i transitionen "Alle beskeder afsendt". Denne transition er vha. tidsmekanismen i CPN angivet til at vare den tid, der svarer til at sende et antal *i* beskeder. Værdien for *i* hentes i form af en brik fra pladsen "Antal afsendte beskeder". Denne brik placeres i forbindelse med afsendelse af genererede opdateringsbeskeder. Brikken fungerer samtidig i forbindelse med synkronisering af undersiden.

8.12. Justerbare parametre i modellen

Med henblik på den efterfølgende analyse, er der stræbt efter at gøre konfiguration af modellen simpel. Konfigurationen foretages vha. en gruppe parametre. Parametrene er deklareret som konstanter i modellen. Her følger en liste over de vigtigste af disse konstanter:

- opdat_p_timeout_p_forhold: Angiver det globale forhold imellem startværdierne for timeout- og opdateringstællerne for kernevariablenes tællere

- sandsynlighed_pakketab: Sandsynligheden for at en afsendte pakke går tabt. Tabet sker med ligeligt fordelt sandsynlighed i enten kommunikationskanalen (ingen noder modtager) eller under modtagelsen hos enkelte noder (se afnit 8.4.1)

- antal_noder: Det totale antal aktive noder
- antal_variable: Det samlede antal kernevariable
- aktivering_p: Aktiveringsperioden for nodernes kommunikationskomponenter
- aendr_forsinkelse_max: Den maksimalt tilladte forsinkelse i forbindelse med videreformidling af en ændring i værdien for en kernevariabel (se afsnit 6.4.1)
- opdat_p_max: Den maksimalt tilladte startværdi for kernevariablenes opdateringstællere (se afsnit 6.4.1)
- modtag_besked_forsinkelse: Den tid det tager at modtage én besked hos en node
- send_besked_forsinkelse: Den tid det tager at afsende én besked hos en node. Tiden er den samme uafhængigt af typen af besked (unicast eller multicast)
- antal_I_tilknytn_pr_node og antal_IO_tilknytn_pr_node: Angiver antallet af lokale tilknytninger af de to typer (I og IO) pr. node
-

9. Modelbaseret analyse

Dette afsnit beskriver hvordan modellen af kommunikationssystemet anvendes til at indsamle målinger, som danner grundlag for en analyse af løsningsforslaget. Afsnittet er inddelt i en række underafsnit, som hver indeholder en beskrivelse af de analyserede egenskaber, den anvendte metode til indsamling af måledata og analyse på baggrund af de indsamlede måledata. Det mest grundlæggende formål med den modelbaserede analyse er at give et grundlag for en evaluering af løsningsforslagets egnethed i forbindelse med DAO. Udover det, tjener analysen en række andre formål:

- At give indsigt i parametrenes indvirkning på systemets virkemåde
- At give viden om optimale parameterværdier for det modellerede system
- At give et billede af løsningsforslagets tolerans overfor kommunikationsfejl
- At undersøge de tilnærmede realtidsegenskaber for løsningsforslaget

Indsamlingen af måledata fra modellen sker ved forskellige kombinationer af variation af de parametre, som er beskrevet i afsnit 8.12.

9.1. Generel målemetode

Værktøjet CPN Tools understøtter indsamling af måledata i forbindelse med simulering af modeller. De indsamlede måledata kan skrives til logfiler og analyseres, når simuleringen er afsluttet. Indsamlingen sker vha. *monitorer* [CPN2]. En monitor kan tilknyttes enten et sæt af pladser eller et kombineret sæt af transitioner og pladser. En monitor, som alene er tilknyttet pladser, vil blive aktiveret for hvert skridt i simuleringen - dvs. hver gang en vilkårlig transition bliver udført. Hvis der derimod indgår én eller flere transitioner i monitorens tilknytning, vil monitoren blive aktiveret hver gang en af disse transitioner bliver udført. Virkemåden for en monitor specificeres af tre forskellige funktioner, som kaldes når modellen simuleres:

- Initialiseringsfunktion: Kaldes når modellen initialiseres i forbindelse med start af simuleringen.
- Prædikatfunktion: Kaldes hver gang en monitor bliver aktiveret og returnerer en sand/falsk værdi, som afgør hvorvidt observeringsfunktionen skal kaldes. Det er muligt at inddrage de tilknyttede pladsers aktuelle indhold af brikker i afgørelsen.
- Observeringsfunktion: Kaldes når monitoren aktiveres og hvis prædikatfunktionen returnerer "sandt". Funktionen bliver kaldt med et sæt af parametre, som består af det aktuelle indhold af brikker på de pladser, hvortil monitoren er knyt-

tet. Funktionen returnerer en heltalsværdi, som skrives til en logfil sammen med information om bl.a. modeltiden.

Når en simulering udføres, vil der altså for hver monitor blive genereret en logfil, som indeholder returværdierne fra observeringsfunktionerne for hver enkelt aktivering af monitorerne. Desuden vil der på baggrund af disse logfiler blive genereret én generel rapportfil, som bl.a. indeholder minimum, maksimum og gennemsnitsværdierne for observeringsfunktionerne for hver enkelt monitor.
En af de store fordele ved at anvende monitorer, er at indsamling af data ikke kræver ændringer i selve modellen. Dataindsamlingen vha. monitorer eksisterer som et lag ovenpå modellen. Alternativet ville være, at indbygge transitioner og pladser i modellen til at håndtere indsamling af data, skrivning til logfiler osv. Det undgås helt, når monitorer anvendes. På den måde sikres det, at modellen udelukkende fremstiller det modellerede system.
Beskrivelsen ovenfor dækker monitorer af typen "Data Collection", som er blevet anvendt i forbindelse med den aktuelle model. Der findes en række andre monitorer, som blandt andet giver mulighed for at overvåge antallet af brikker på pladser, overvåge udførsel af transitioner mm.
I beskrivelsen af resultaterne anvendes operatorerne *avg* og *max*, som beskriver henholdsvis den gennemsnitlige og den maksimale værdi for et sæt af værdier.
Når der indsamles måledata fra modellen, varieres én parameter, mens de øvrige holdes konstante. Hvor andet ikke er nævnt, anvendes følgende værdier for parametrene:

- opdat_p_timeout_p_forhold: 1000
- antal_noder: 3
- antal_variable: 3000
- antal_I_tilknytn_pr_node: 600
- antal_IO_tilknytn_pr_node: 300
- aktivering_p: 50.000 [µs]
- aendr_forsinkelse_max: 50.000 [µs]
- modtag_besked_forsinkelse: 850 [µs]
- send_besked_forsinkelse: 850 [µs]
- sandsynlighed_pakketab: 0,0001

Parametrenes funktion er beskrevet i afsnit 8.12. Valget af parameterværdier er baseret på oplysningerne i afsnit 13.2.

9.2. Afgrænsning af simuleringstid

Vha. en særlig type monitor - "Break point" - er det muligt at specificere et kriterium for hvornår simulering af en model skal afsluttes. Denne type monitor aktiveres hver gang der tages et skridt i simuleringen og indeholder i modsætning til "Data Collection"-monitorer kun en prædikatfunktion. Funktionen returnerer "sandt" eller "falsk", som afgør om den igangværende simulering skal fortsættes ("sandt" afbryder simuleringen). I prædikatfunktionen er det muligt at inddrage den aktuelle modeltid. Det gør det muligt at afbryde simuleringen efter et afgrænset tidsrum (i modeltid). I modellen af DAO vil forholdet imellem simuleringsskridt og modeltid afhænge af parameterangivelsen for bl.a. aktiveringsperioden for kommunikationskomponenten. Inddragelsen af modeltiden til afgrænsning af simuleringsperiode gør det muligt at bortabstrahere denne afhængighed. Når det gøres, vil antallet af skridt i en simulering variere som funktion af parameterværdierne, men simuleringstiden (i modeltid) holdes konstant, hvilket sikrer et ensartet grundlag for indsamling af måledata fra modellen. I forbindelse med indsamling af måledata begrænses simuleringstiden til 10 sekunder (i modeltid).

9.3. Måling af konsistensgrad

Den mest grundlæggende opgave for kommunikationssystemet er at opretholde et konsistent billede af værdierne for de delte kernevariable på tværs af systemets noder. Derfor er det relevant at undersøge hvilken grad af konsistens, der kan forventes på baggrund af modellen.

9.3.1. Målemetode

Systemets konsistens på et givet tidspunkt beregnes som den procentdel af kernevariablene, hvis værdier er konsistente imellem deres forskellige instanser på systemets noder. En kernevariabel er altid repræsenteret i form af én lokal instans og en mængde kopier. For hver I- eller IO-tilknytning til en given variabel eksisterer én kopi af variablen udover den originale instans. Noderne har et konsistent billede af en kernevariabels værdi, hvis værdien hos alle noder, som har enten den lokale instans eller én af kopierne er den samme. Inkonsistens for en kernevariabel kan enten opstå, hvis værdierne er forskellige i de forskellige instanser eller hvis kernevariablen ikke eksisterer i det forventede antal instanser. Det er tidligere beskrevet, hvordan kernevariables kopiinstanser oprettes og nedlægges dynamisk. Disse mekanismer indgår i opretholdelsen af systemets konsistens. Derfor bliver en kernevariabel betragtet som inkonsistent, hvis antallet af instan-

ser ikke modsvarer det forventede antal. Under initialisering af noderne (se afsnit 8.7) opdaterer noden den globale tabel over antal kernevariabelinstanser med udgangspunkt i nodens egne tilknytninger og lokale kernevariable. Den aktuelle grad af konsistens for systemet beregnes som den procentdel af de delte variable, hvor kriterierne for konsistens er opfyldt. Beregningen af konsistensgraden for alle variable i systemet beregnes vha. den rekursive funktion "beregn_konsistens_grad", som vises på Figur 22.

```
fun beregn_konsistens_grad(_, nil) = 0
| beregn_konsistens_grad(kvll, x::xs:var_antal_liste_t) =
if(
alle_ens(find_alle_var_vaerdier(#noegle(x), kvll)) andalso
#antal(x) = tael_instanser(#noegle(x), kvll))
then
 beregn_konsistens_grad(kvll, xs) + 1
else
 beregn_konsistens_grad(kvll, xs);
```

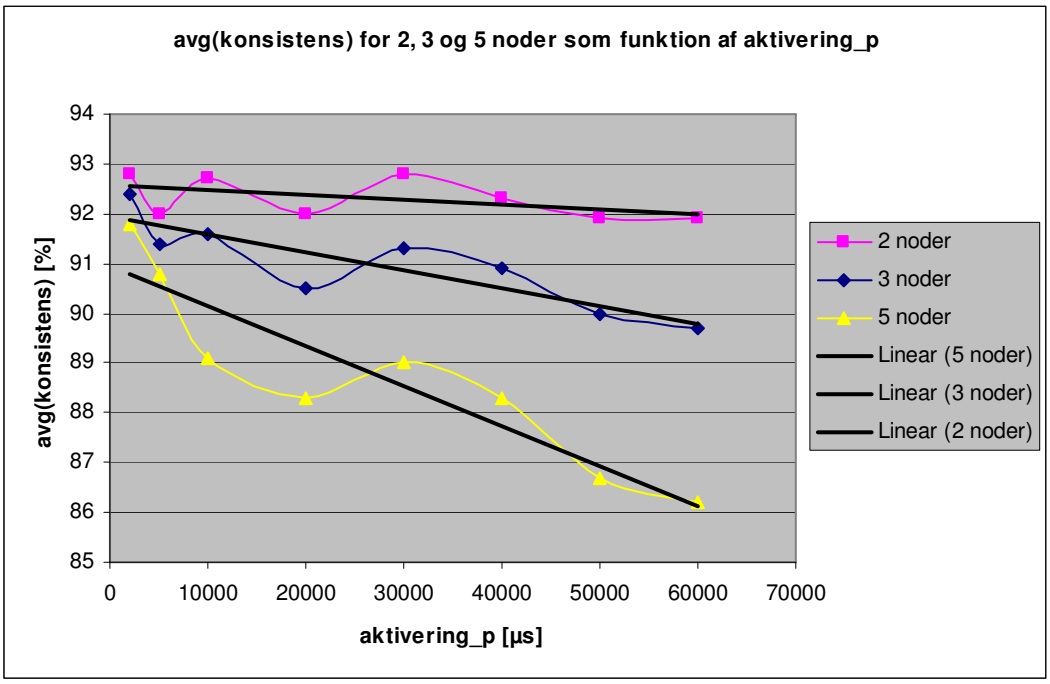
Figur 22 - Funktion til beregning af konsistensgrad

Parameteren "kvll" indeholder kernevariabeltabellerne for alle instanser af undersiden "DAONode" i form af en liste af lister. Når en monitor af denne type anvendes i en hierarkisk struktureret model, vil pladsernes indhold være tilgængeligt på tværs af strukturen. Funktionen returnerer antallet af konsistente variable. Hjælpefunktionen "alle_ens" bruges til at undersøge om alle instanser af en given variabel har samme værdi mens feltet "antal" og funktionen "tael_instanser" i kombination bruges til at afgøre, om variablen findes i det forventede antal instanser. Typen "var_antal_liste_t" er en liste med elementer af den komplekse type "var_antal_t", som indeholder felterne "noegle" og "antal". Listen initialiseres under initialiseringen af modellen og indeholder det forventede antal instanser for hver variabel i systemet. Listen bruges her til at undersøge, om hver variabel eksisterer i det forventede antal instanser.

9.3.2. Resultater

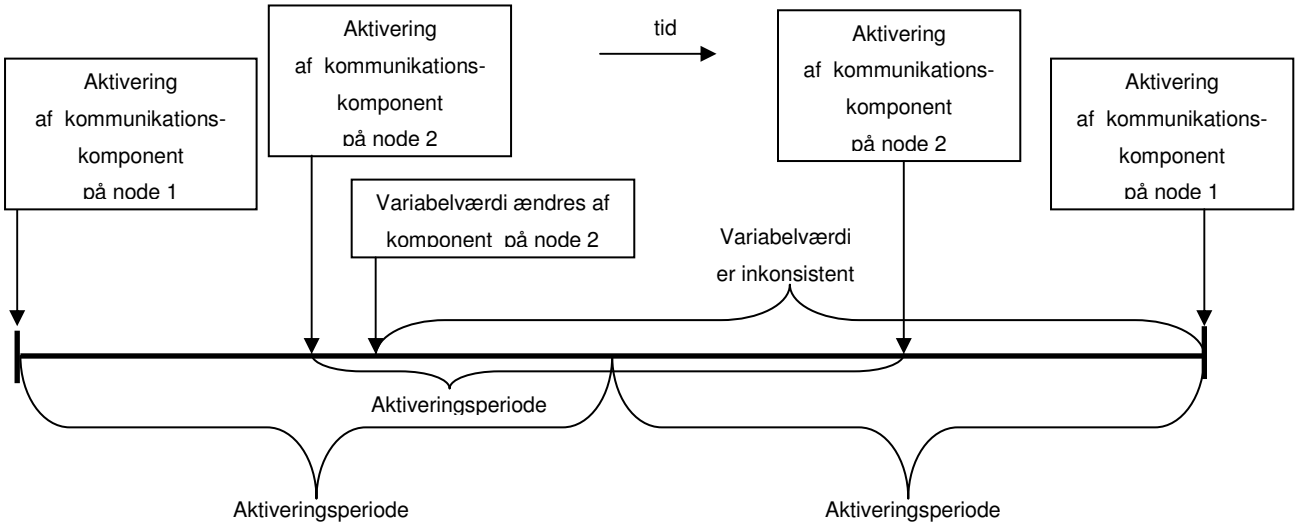
Figur 23 viser gennemsnitsværdierne for konsistensgraden som funktion af en varierende aktiveringsperiode for kommunikationskomponenten ("aktivering_p") for henholdsvis

2,3 og 5 noder ("antal_noder"). Alle andre parametre holdes konstant. For at fremhæve tendensen for de tre målinger er der tilføjet regressionslinier.



Figur 23

Det ses for alle tre målinger, hvordan graden af konsistens falder, når aktiveringsperioden øges. Dette er forventet, eftersom en øget aktiveringsperiode resulterer i større forsinkelse i forbindelse med modtagelse af opdateringsbeskeder og dermed i en lavere grad af konsistens over tid. Forklaringen ligger i, at en øget aktiveringsperiode vil øge de tid der går imellem både udsendelse og modtagelse af periodiske opdateringsbeskeder. Det vil resultere i en øget forsinkelse i forbindelse med ændring af variabelværdier. Den øgede forsinkelse vil have en negativ effekt på den samlede konsistensgrad i systemet, eftersom den resulterer i at forskellige værdier for hver enkelt variabel optræder i en større andel af tiden. På Figur 24 illustreres denne sammenhæng vha. to noder, som deler en variabel. Den ene node (2) har IO-tilknytning til variablen og kan derfor ændre værdien. Den anden node (1) har I-tilknytning og er derfor afhængig af at modtage opdateringsbeskeder med variablens værdi. Når værdien ændres for en variabel, som deles af to noder vil de to noders billede af variablens værdi være inkonsistent i en tidsperiode. Længden af denne periode afgøres af, hvornår kommunikationskomponenten på node 1 aktiveres. Dette tidspunkt er afgørende for, hvornår indkommende opdateringsbeskeder med værdier for delte variable vil blive behandlet.



Figur 24 - Sammenhæng mellem inkonsistensperiode og aktiveringsforskydning

Som en konsekvens af denne sammenhæng, kan det ses at billedet af værdien for den delte variabel vil være inkonsistent og at perioden med inkonsistens stiger, når aktiveringsperioden øges. Denne tendens ses som forventet på måleresultaterne.
Det ses desuden på Figur 23, at graden af konsistens falder, når antallet af noder stiger. Dette fænomen optræder hovedsageligt fordi flere noder i systemet resulterer i flere IO-tilknytninger og dermed hyppigere ændringer af variabelværdierne. Inkonsistens opstår, når variabelværdierne ændres og stiger derfor som en naturlig konsekvens af, at antallet af noder øges og der dermed optræder flere variabelændringer over tid i systemet.

9.4. Måling af forsinkelse

I kombination med målingen af konsistensgraden kan kvaliteten af kommunikationskomponenten vurderes ud fra størrelsen af den forsinkelse, som opstår fra en variabel ændrer værdi og til den nye værdi er kendt på systemets øvrige noder. Forsinkelsens omfang har direkte indflydelse på præcisionen af de reguleringsalgoritmer, som nodernes øvrige komponenter implementerer og er derfor en relevant egenskab at undersøge for løsningsforslaget. Kvaliteten af den regulering, som udføres af reguleringsalgoritmerne er direkte afhængig af rettidigt og regelmæssigt indkommende måledata. På samme måde er det vigtigt, at reguleringsalgoritmernes resultater rettidigt og regelmæssigt bliver leveret videre til alle noder, hvor komponenter er afhængige af resultatet. I begge tilfælde spiller kommunikationskomponenten en central rolle. Kravet om rettidighed gør det nødvendigt at kende de maksimale forsinkelser, som kan opstå i for-

bindelse med delingen af variable. Et af formålene med løsningsforslaget og dets modellering er at undersøge mulighederne for at vurdere den forsinkelse, som uundgåeligt vil opstå i forbindelse med kommunikation imellem noderne.

9.4.1. Målemetode

For at gøre det muligt, at måle forsinkelsen i forbindelse med ændring af variabelværdier, tilknyttes hver enkelt variabel i modellen et tidsstempel. Dette tidsstempel opdateres til den aktuelle modeltid, når variablens værdi ændres af en komponent. Tidsstemplet forbliver uændret indtil variablens værdi igen ændres. Tidsstemplet angiver altså tidspunktet for den seneste ændring af variablens værdi. Når variablen med ændret værdi efterfølgende bliver udsendt i en periodisk opdateringsbesked, vil tidsstemplet blive medsendt. Det gør det muligt at beregne den aktuelle forsinkelse ved at fratrække tidsstemplet, som angiver tidspunktet for ændringen fra den aktuelle modeltid. Denne udregning foretages, når en node med I-tilknytning til den ændrede variabel modtager og processerer opdateringsbeskeden - og altså på det tidspunkt, hvor den modtagende nodes kopi af den ændrede variabel ændres til at have den nye værdi. Resultatet af udregningen svarer til den periode, hvor den berørte variabelkopi på den modtagende node har haft en forskellig værdi i forhold til originalinstansen, som befinder sig på den node, hvorfra opdateringsbeskeden afsendes. Hver gang en opdateringsbesked modtages på én af systemets noder, gemmes den højeste forsinkelsestid i en logfil. Figur 25 viser funktionen "beregn_forsinkelse", som anvendes til at udregne forsinkelsestiden:

```
fun beregn_forsinkelse(v:var_t, kv_l:kerne_var_liste_t) =
if(findes_var(#noegle(v), kv_l)) then
  if(kerne_var_vaerdi(#noegle(v), kv_l) = (#vaerdi(v)) ) then 0
  else
  if(#aendr_ts(v) > 0) then IntInf.toInt(time()) - (#aendr_ts(v))
  else 0
else 0;
```

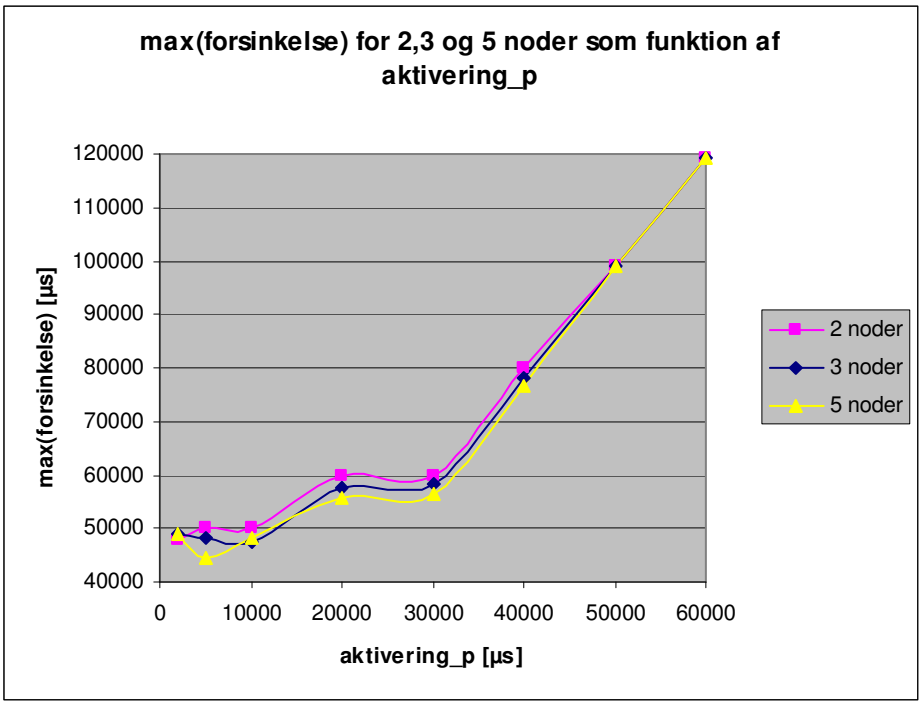
Figur 25 - Funktion til beregning af forsinkelse

Funktionen benytter en række hjælpefunktioner: Funktionen "findes_var" returnerer "sandt" eller "falsk" afhængigt af om variablen findes hos modtageren, "kerne_var_vaerdi" giver den aktuelle værdi for variablen hos modtageren og "time" giver

den aktuelle modeltid. Samtidig anvendes "#"-operatoren til at hente indholdet af felterne "noegle" og "vaerdi" i den komplekse kernevariabeltype.

9.4.2. Resultater

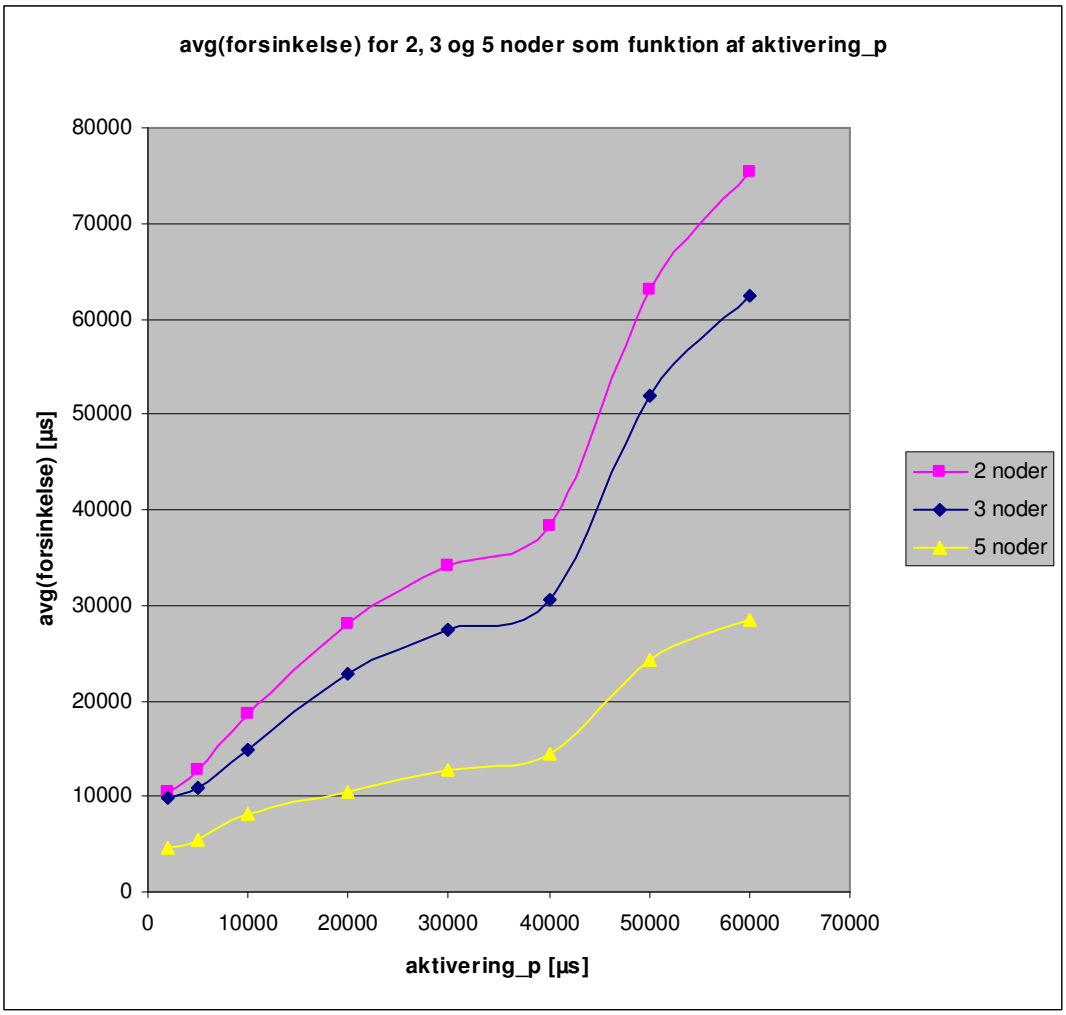
På Figur 26 ses resultatet af indsamling af maksimale forsinkelsestider når aktiveringsperioden varieres. Målingerne er indsamlet for 2, 3 og 5 noder. Den maksimale forsinkelsestid er den længste tid, der er målt under én simulering med et bestemt antal noder og en bestemt aktiveringsperiode.



Figur 26

Det er tydeligt at se, at den maksimale forsinkelsestid er næsten uafhængig af antallet af noder. Det hænger sammen med den indbyrdes forskydning af nodernes kommunikationskomponenter, som beskrives i afsnit 8.7.1. Denne forskydning er valgt, så den uafhængigt af antallet af noder, bliver størst mulig og dermed repræsenterer den værst mulige situation set i forhold til kravet om rettidig opdatering af variabelværdier. Metoden gør, at den maksimale forsinkelsestid afhænger direkte af aktiveringsperioden. Som forventet (se afsnit 9.3.1) ses det, at den maksimale forsinkelsestid stiger, når aktiveringsperioden øges. Graden af konsistens for de delte variable hænger sammen med

forsinkelsestiden i forbindelse med ændringer af variabelværdier og er derfor en afledning af den effekt, som ses på Figur 26.
Til sammenligning vises på Figur 27 gennemsnitsværdierne for forsinkelsestiden som funktion af aktiveringsperioden for 2, 3 og 5 noder. I modsætning til for maksimalværdierne, afhænger måleresultaterne her af antallet af noder.



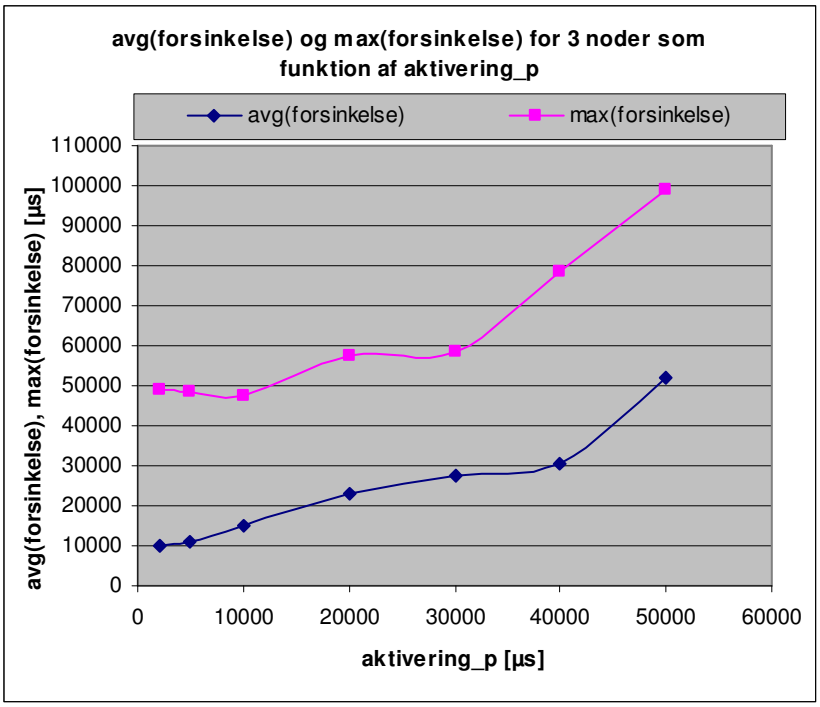
Figur 27

For gennemsnitsværdierne ses effekten af, at flere noder i systemet resulterer i en øget *sammenklumpning* - dvs. aktiveringen af nodernes kommunikationskomponenter sker tidsmæssigt tættere på hinanden, hvilket resulterer i en gennemsnitligt lavere forsinkelsestid i forbindelse med ændring af variabelværdier. Den maksimale værdi ændres ikke, da den - uafhængigt af hvor mange noder der findes i systemet - vil stamme fra kom-

munikation imellem de to noder, hvis forskel imellem aktiveringstidspunkter for kommunikationskomponenterne er størst.

Når den gennemsnitlige forsinkelse betragtes, er det relevant at vide, at der i beregningen af gennemsnittet kan indgå en række forsinkelsesmålinger, hvor resultatet er 0. Det vil sige, at en ændring af en variabelværdi er øjeblikkeligt blevet afspejlet i alle kopiinstanser for variablen. Denne situation kan opstå, hvis alle komponenter med tilknytning til variablen (I og IO) befinder sig på den samme node. I dette tilfælde vil der ikke opstå den forsinkelse, som ellers opstår før alle kopiinstanser er opdateret med den nye værdi. Variablen eksisterer kun i én instans, som deles af alle komponenter og når værdien for denne instans ændres er dette øjeblikkeligt synligt for alle tilknyttede komponenter fordi der ikke går tid med udsendelse af opdateringsbesked osv.

Et typisk billede af forskellen imellem den maksimale forsinkelse og gennemsnitsværdien ses på Figur 28, hvor de to størrelse vises som funktion af aktiveringsperioden i et system med 3 noder.



Figur 28

9.5. Måling af ressourceforbrug

Kommunikationskomponenten på en DAO-node aktiveres periodisk på samme måde som nodens øvrige komponenter. Hvis én eller flere af komponenterne bruger for stor en del

af den samme aktiveringsperiode, kan det være umuligt at aktivere de øvrige komponenter rettidigt. En af de vigtigste egenskaber ved det modellerede løsningsforslag er, at protokollen gør det muligt at forudsige det maksimale ressourceforbrug for kommunikationskomponenten.
Det teoretisk maksimale ressourceforbrug afhænger af antallet af noder og aktiveringsperioden. Antallet af noder er afgørende for, hvor mange beskeder kommunikationskomponenten på en given node maksimalt vil modtage under en aktiveringsperiode. Da aktiveringsperioden er ens for alle noders kommunikationskomponenter, kan der maksimalt komme én opdateringsbesked fra hver af de øvrige noder. Oven i det, vil der altid maksimalt blive afsendt én opdateringsbesked for hver aktiveringsperiode. Sammenholdt med viden om den tid det tager at modtage og afsende beskeder, er det muligt at beregne det teoretiske maksimum for ressourceforbruget med et sæt givne parameterværdier. Herudover giver modellen mulighed for at foretage målinger af det gennemsnitlige ressourceforbrug. I denne sammenhæng spiller flere faktorer ind: Sandsynligheden for pakketab og variablenes opdateringsperioder. Desto flere pakker der tabes, desto færre beskeder bliver modtaget. Det resulterer i en mindskning af det samlede ressourceforbrug. Variablenes opdateringsperioder påvirker det gennemsnitlige ressourceforbrug, da de er direkte afgørende for, hvor ofte der skal udsendes opdateringsbeskeder. Som beskrevet i afsnit 6.4.1 specificerer kravet til maksimal opdateringstid i forbindelse med ændrede variabelværdier indirekte et loft for variablenes opdateringsperioder og den samlede aktiveringsperiode. Før dette loft nås (dvs. for lave aktiveringsperioder), vil opdateringsperioderne og aktiveringsperioden have indflydelse på det gennemsnitlige ressourceforbrug. Det skyldes, at de to typer perioder har indvirkning på, i hvor stor en del af kommunikationskomponentens aktiveringer, det er nødvendigt at udsende opdateringsbeskeder. Når aktiveringsperioden mindskes, øges antallet af aktiveringer, hvor det ikke er nødvendigt at afsende opdateringsbeskeder. På samme tid vil en mindskning af aktiveringsperioden betyde, at den tid der bruges på modtagelse og afsendelse af beskeder kommer til at udgøre en større procentvis andel af den samlede aktiveringsperiode. Opdateringsperioderne har ingen indflydelse for de højere aktiveringsperioder, hvor det vil være nødvendigt at udsende opdateringsbeskeder under hver aktivering. I denne ende af spektret kan de maksimale og gennemsnitlige målinger af ressourceforbruget altså forventes at være ens.

9.5.1. Målemetode

Kommunikationskomponenternes ressourceforbrug registreres ved at overvåge antallet af modtagne og afsendte beskeder for hver aktiveringsperiode. Ud fra antallet af beske-

der og den konfigurerede aktiveringsperiode, kan ressourceforbruget som en procentdel af en given aktivering beregnes vha. funktionen "beregn_cpu_belastning", som vises på Figur 29.

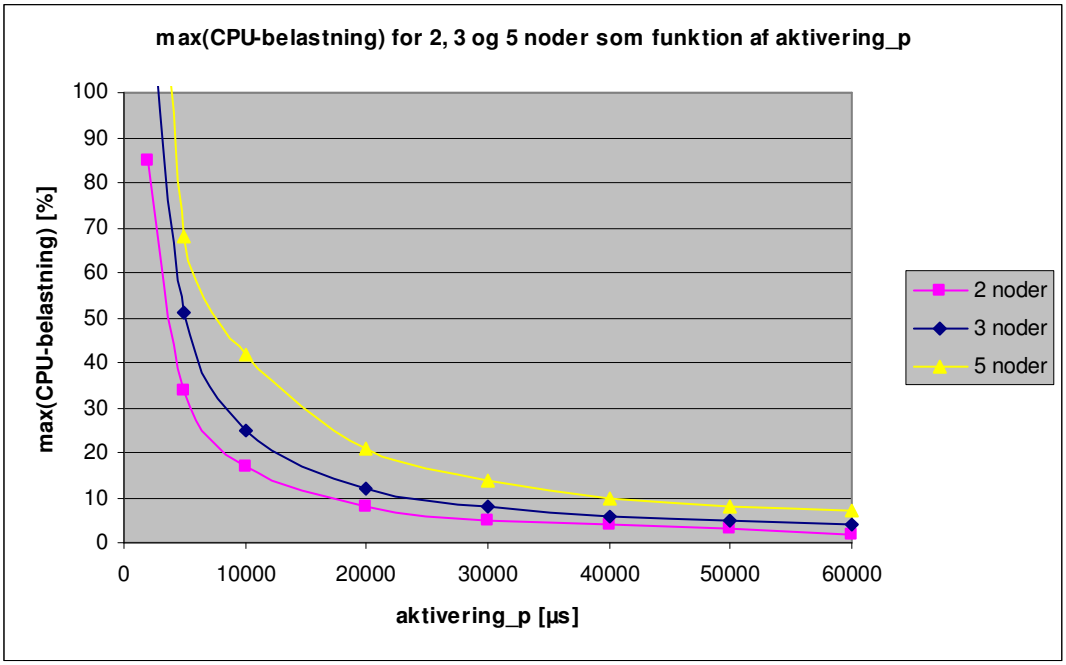
```
fun beregn_cpu_belastning(sendt, modtaget) =
(100 * ((sendt * send_besked_forsinkelse) +
(modtaget * modtag_besked_forsinkelse))) div aktivering_p;
```

Figur 29 - Funktion til beregning af CPU-belastning

Funktionen kaldes hver gang en kommunikationskomponent på én af systemets noder afslutter sin aktivering. Parametrene "sendt" og "modtaget" angiver antallet af sendte og modtagne beskeder i løbet af den aktuelle aktiveringsperiode. De to værdier i forbindelse med hhv. generering af opdateringsbeskeder og modtagelse af opdateringsbeskeder. Ud fra disse to værdier beregner funktionen den aktuelle procentvise CPU-belastning. Udregning sker ud fra en antagelse om, at den øvrige processeringstid i kommunikationskomponenten er meget lille set i forhold til den tid, der bruges på at sende og modtage beskeder og at den derfor kan bortabstraheres i denne sammenhæng. Den beregnede værdi gemmes i en logfil.

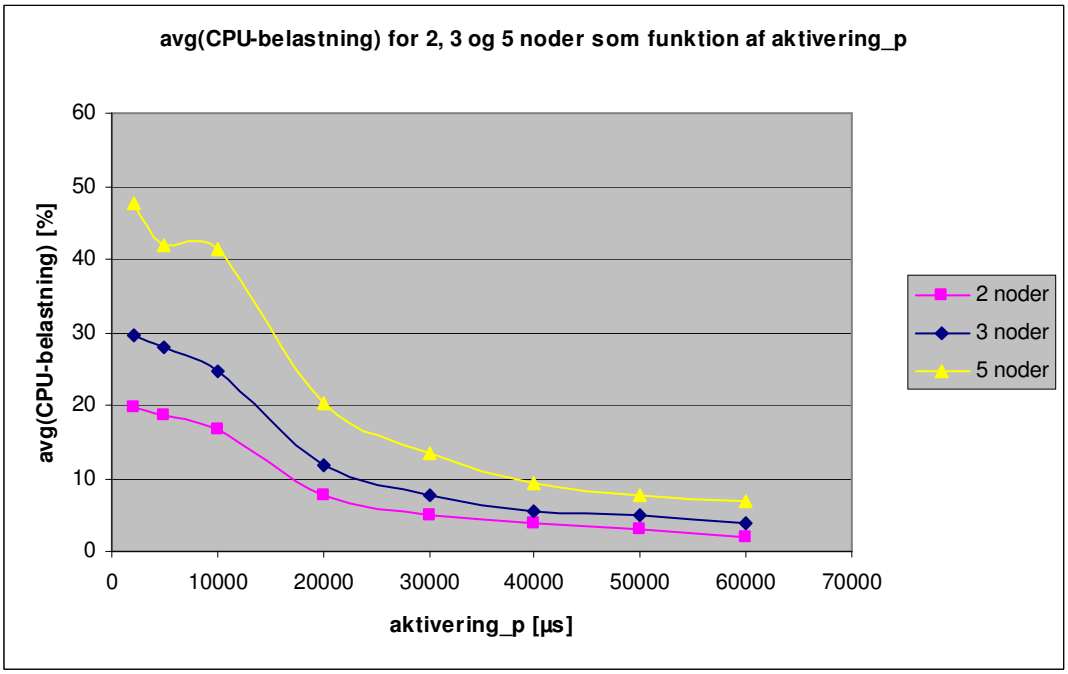
9.5.2. Resultater

På Figur 30 ses resultaterne af indsamling af målinger af maksimalt ressourceforbrug fra modellen. Det ses hvordan ressourceforbruget afhænger af antallet af noder og af aktiveringsperioden.



Figur 30

Figur 31 viser gennemsnittet af de indsamlede værdier. Det kan ses at de to figurer (Figur 30 og Figur 31) ser stort set ens ud, når aktiveringsperioden er over 10.000 µs. Som forklaret skyldes dette, at den gennemsnitlige belastning bliver lavere for de lavere aktiveringsperioder, som resultat af, at det ikke altid er nødvendigt at afsende opdateringsbeskeder.



Figur 31

Det kan samtidig ses på Figur 30, at der for 3 og 5 noder ved den laveste aktiveringsperiode vil opstå en CPU-belastning på over 100 %. Det er naturligvis umuligt, men skal tolkes som et tegn på, at kommunikationskomponenten i denne konfiguration ikke vil være i stand til at håndtere modtagelse og afsendelse indenfor tidsrammen, som specificeres af aktiveringsperioden. Det betyder samtidig, at der ikke vil være nogen processeringstid til rådighed for nodens øvrige komponenter. Den tilladelige grænse for kommunikationskomponenten vil dog typisk gå langt under de 100 %, for at garantere en tilpas mængde ledig processeringstid for alle komponenter på en node. Det grundlæggende resultat af denne måling på modellen er et mål for sammenhængen imellem antal noder, aktiveringsperiode og det resulterende ressourceforbrug. Hvis det antages at modellen er tilstrækkeligt nøjagtig i forhold til det virkelige system i denne sammenhæng, kan det således ud fra målingerne fastslås hvor meget ledig CPU-tid, kommunikationskomponenten vil kræve i en given konfiguration af aktiveringsperioden.

9.6. Måling af tolerance overfor pakketab

Her følger en kort beskrivelse af tre forskellige målinger, som er foretaget for at undersøge løsningsforslagets tolerance overfor pakketab. Det er undersøgt hvordan konsistens, forsinkelse og CPU-belastning ændrer sig som funktion af sandsynligheden for pakketab i kommunikationskanalen. Målingerne kan bruges til at give en indikation af løsningsforslagets robusthed på et netværk, hvor udvekslingen af beskeder er forbundet med en given sandsynlighed for at beskeder går tabt. Samtidig hjælper målingerne til at få et indtryk af, hvornår pakketabet i kommunikationskanalen bliver kritisk for løsningsforslaget.

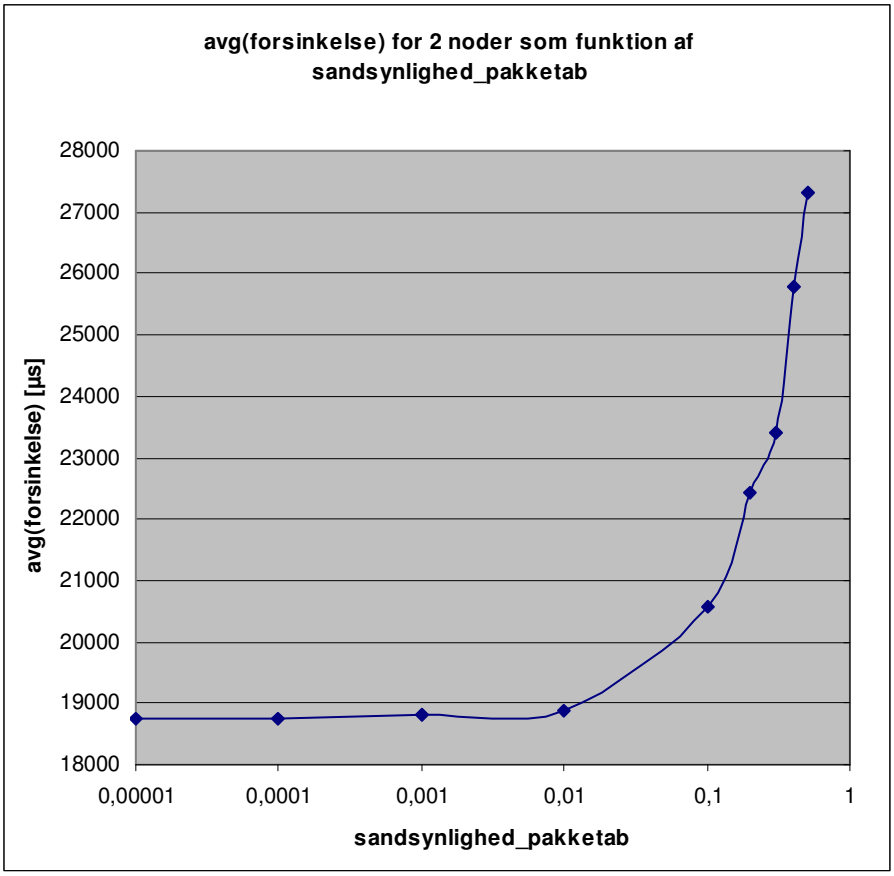
9.6.1. Målemetode

Indsamlingen af måledata foregår som i de tidligere beskrevne undersøgelser vha. monitorer, som indsamler målinger og skriver disse til logfiler. Tre typer målinger indsamles: Forsinkelsen, konsistensen og CPU-belastningen. Metoden til indsamling af disse målinger er beskrevet i de foregående afsnit. Til forskel fra disse afsnit, holdes aktiveringsperioden nu konstant. Derimod varieres parameteren "sandsynlighed_pakketab". Parameteren angiver med en realtalsværdi i intervallet 0 til 1 sandsynligheden for at en pakke går tabt i forbindelse med udvekslingen imellem kommunikationskomponenten hos hhv. afsender og modtager (se afsnit 8.4.1). Indsamlingen af måledata sker for et system med 2 noder. Aktiveringsperioden holdes konstant på værdien 10.000 µs. Til forskel fra de foregående målinger, simuleres der nu over en periode på 100 sekunder i modeltid.

Indsamlingen sker ved at variere sandsynligheden for pakketabet i intervallet fra 0,00001 (0,01 ‰) til 0,5 (50 %). Ifølge oplysningerne fra Vestas er 0,00001 den gennemsnitlige sandsynlighed for den aktuelle netværkskonfiguration (se afsnit 13.2).

9.6.2. Resultater

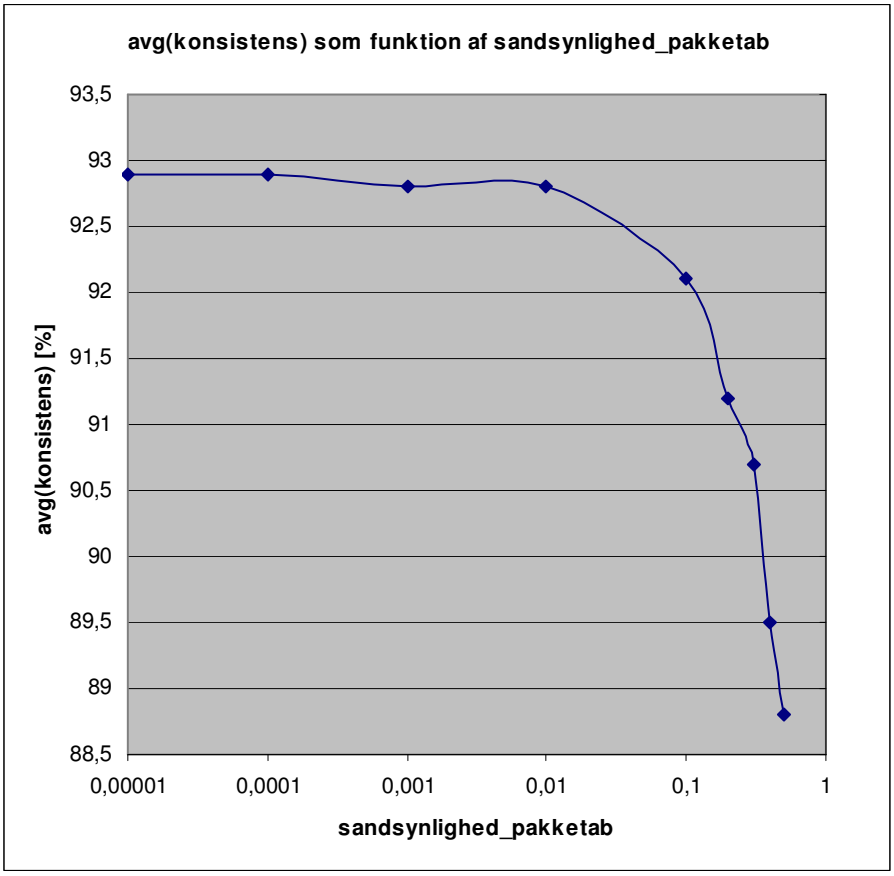
Figur 32 viser gennemsnittet af målinger af forsinkelse i forbindelse med værdiændringer for variable. Det kan ses, at der stort set ikke sker ændringer af den gennemsnitlige forsinkelse, så længe sandsynligheden for pakketab er under 0,01. Herefter ændres resultaterne hastigt som funktion af sandsynligheden; den gennemsnitlige forsinkelse bliver mere end tidoblet, når sandsynligheden stiger fra 0,1 til 0,5.



Figur 32

Figur 33 viser den gennemsnitlige konsistens som funktion af sandsynligheden for pakketab. Har ses den samme tendens; effekten af pakketabet ses fra omkring 0,01. I denne sammenhæng skal det bemærkes, at den gennemsnitlige konsistens med de aktuelle

parameterværdier ligger omkring 80 %, hvis sandsynligheden for pakketab sættes til 1 (dvs. alle pakker går tabt). Grunden til at dette konsistensniveau kan opretholdes, til trods for at ingen pakker når frem, er at kun et mindretal af det samlede antal parametre i systemet har tilknytninger fra eksterne komponenter eller er kopiinstanser af eksterne variable. Det betyder gennemsnitligt, at 80 % af alle variable ikke afhænger af indkommende opdateringsbeskeder.

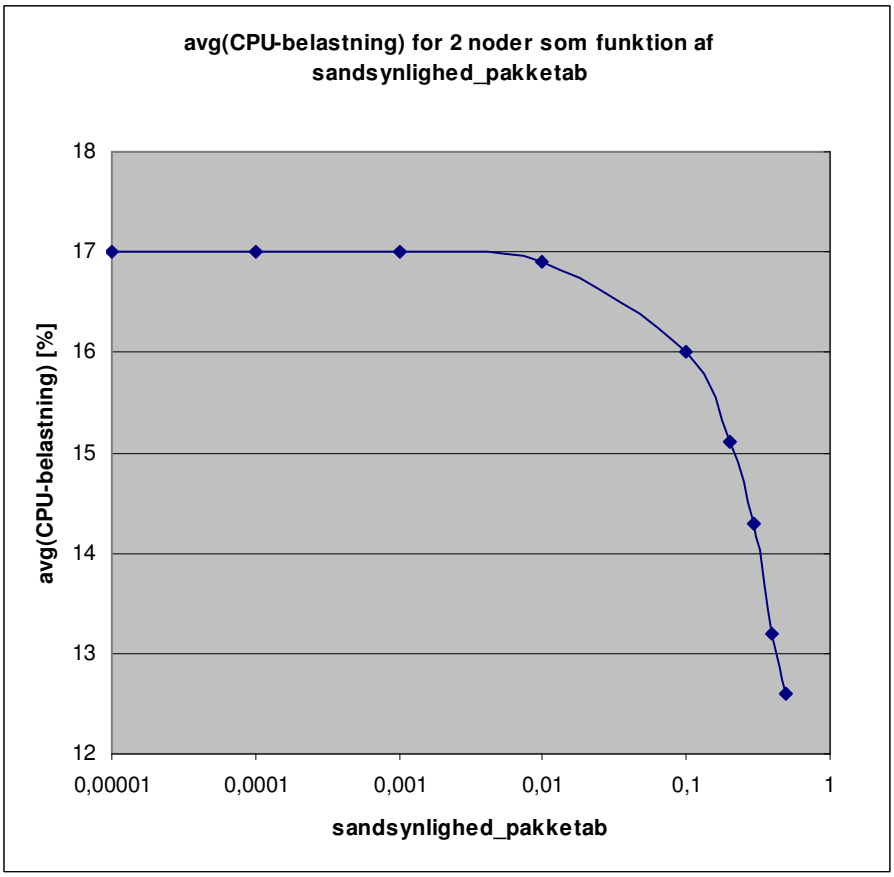


Figur 33

Når pakketabet øges, vil det resultere i en forøgelse af den gennemsnitlige forsinkelse, som opstår i forbindelse med ændring af variabelværdier. Derfor er faldet i konsistensgrad sammenligneligt med det fald, som opleves når aktiveringsperioden og dermed forsinkelsen øgedes i afsnit 9.3.

Som den sidste måling viser Figur 34 hvordan CPU-belastningen påvirkes af pakketab. På samme måde som for de to foregående målinger sker den mest markante ændring her omkring sandsynligheden 0,01. Her begynder den gennemsnitlige CPU-belastning at

falde som funktion af en stigende sandsynlighed for pakketab. Resultatet er forventeligt og er et resultat af, at en mindre andel af de afsendte beskeder når frem til modtagerne. Det betyder, at der bruges færre processeringsressourcer til håndtering af indkommende beskeder hos de modtagende noder. Der vil stadig blive brugt den samme mængde ressourcer på afsendelse af opdateringsbeskeder fra hver enkelt node. Tendensen er en karakteristisk egenskab for den protokol, som modelleres i løsningsforslaget. Eftersom der ikke er implementeret nogen eksplicit pålidelighedsmekanisme, vil tab af pakker ikke blive detekteret. Derfor vil der ikke blive sendt flere pakker som resultat af tabte pakker. Antallet af afsendte pakker per aktiveringsperiode er begrænset til og dækker ikke over retransmission af tabte pakker. På samme måde bliver der ikke udsendt bekræftelsesbeskeder fra modtagerne, som kunne resultere i et uforudsigeligt ressourceforbrug.



Figur 34

Ud fra de tre målinger, kan der ses en generel tendens; der sker først ændringer som resultat af øget sandsynlighed for pakketab, når sandsynligheden overstiger omkring 0,1. For alle tre målinger, begynder ændringer i løsningsforslagets egenskaber at vise sig omkring denne værdi for sandsynligheden. Denne sandsynlighed er 1000 gange så

stor, som den gennemsnitsværdi for sandsynligheden for pakketab, som oplyses af Vestas.

9.7. Opsummering

De beskrevne målinger er foretaget ud fra en model, som er baseret på en række antagelser om virkeligheden. Samtidig repræsenterer parametrenes værdier antagelser omkring gennemsnitsituationen for det modellerede løsningsforslag. Resultatet af de målinger, som uddrages fra modellen afhænger således af flere faktorer; modellens kvalitet i forhold til at fremstille det modellerede system korrekt og præcisionen af de valgte parameterværdier. Derfor er det vigtigt ikke at tolke måleresultaterne som endegyldige svar på, hvordan en implementation af løsningsforslaget vil fungere. Med dette som udgangspunkt, bidrager modellen og måleresultaterne dog til en øget forståelse af løsningsforslaget og dets parametre. Målingerne viser tendensen for hvordan kommunikationssystemet kan forventes at ville reagere på ændrede parametre og forudsætninger. Samtidig giver modellen et udgangspunkt for en højniveauvurdering af realtidsegenskaberne for løsningsforslaget. Modellen er bl.a. anvendt til at undersøge hvordan forskellige målbare egenskaber ændrer sig som funktion af aktiveringsperioden for kommunikationskomponenten. Her sammenfattes de mest centrale konklusioner omkring løsningsforslaget, som kan drages på baggrund af målingerne fra modellen:

- Den forventede grad af konsistens falder, når aktiveringsperioden øges. Effekten forstærkes, når antallet af noder øges.
- Forsinkelsen i forbindelse med opdaterings af ændrede variabelværdier øges, når aktiveringsperioden øges.
- CPU-belastningen falder, når aktiveringsperioden øges. Flere noder betyder større CPU-belastning.
- Når sandsynligheden for pakketab stiger, falder konsistensgraden og CPU-belastningen, mens forsinkelsen stiger.

En af de vigtige konklusioner, som kan drages ud fra målingerne er, at valget af parametre vil resulterer i et kompromis. Aktiveringsperioden for kommunikationskomponenten resulterer i høj CPU-belastning, hvis den vælges lavt. På den anden side resulterer en høj aktiveringsperiode i lang forsinkelse og lav konsistensgrad. Denne centrale parameter skal altså have en størrelse, som på samme tid giver acceptable niveauer for flere modstridende egenskaber; CPU-belastning, forsinkelse og konsistensgrad. Det acceptable niveau for CPU-belastningen afhænger af, hvor ressourcekrævende nodernes øvrige komponenter er. Hvis det er muligt at afgøre, hvor lang processeringstid hver enkelt komponent maksimalt vil anvende pr. aktivering, kan der fastsættes en øvre acceptabel

grænse for kommunikationskomponenten. Denne øvre grænse vil ud fra målingerne fra modellen kunne bruges som udgangspunkt for en nedre acceptabel grænse for aktiveringsperioden. Det acceptable niveau for forsinkelsen og konsistensgraden hænger sammen med følsomheden af de implementerede reguleringsalgoritmer. Ved at fastsætte enten en nedre grænse for konsistensgraden eller en øvre grænse for ressourceforbruget er det muligt at definere en øvre acceptabel grænse for aktiveringsperioden. Når en nedre og en øvre grænse er fastsat for aktiveringsperioden er valget af parameterens værdi afgrænset. Med forbehold for modellens manglende præcision, vil et valg af aktiveringsperiode indenfor disse grænseværdier resultere i en kommunikationskomponent, som overholder de specificerede krav. Der eksisterer stadig et spektrum af værdier indenfor grænseværdierne, som vil resultere i varierende egenskaber for systemet. Målingerne fra modellen anvendes altså til dels at bestemme grænseværdierne for parameteren og samtidig til at give en forståelse af systemet i forbindelse med valg af parameterværdi indenfor grænseværdierne.

10. Diskussion omkring den anvendte metode

I de foregående afsnit er løsningsforslaget blevet modelleret og modellen er blevet brugt som grundlag for den efterfølgende analyse. I dette afsnit diskuteres denne modelbaserede fremgangsmåde. Diskussionen omkring det konkrete modelleringsarbejde kombineres med en generel diskussion om problemstillinger i forbindelse med modelbaseret udvikling. Den generelle diskussion tager udgangspunkt i [MDD1]. Artiklen beskriver fem kriterier, som er afgørende for kvaliteten af softwaremodeller. De fem kriterier beskriver centrale egenskaber, som en model af et software system ifølge forfatteren skal besidde for at udfylde sin opgave. I de næste fem underafsnit vil den anvendte metode til modellering af løsningsforslaget blive evalueret i forhold til disse fem kriterier. Hvert underafsnit består af en forklaring af begrebet med udgangspunkt i [MDD1] efterfulgt af en diskussion om begrebet set i forhold til det modellerede løsningsforslag.
Diskussionen suppleres med en sammenligning med alternative metoder.

10.1. Abstraktion

10.1.1. Selics beskrivelse af begrebet

En model skal repræsentere en abstraktion set i forhold til det modellerede. Dette kriterium beskrives i [MDD1] som det vigtigste af de fem kriterier. En model skal have et passende abstraktionsniveau, som gør at uvæsentlige detaljer skjules. På den måde hjælpes udvikleren til at fokusere på de vigtigste aspekter i det system, der modelleres. Abstraktionen kan enten eksistere i form af en generel hævning af detaljeringsniveauet i forhold til det modellerede system eller som en metode til at fremhæve vise detaljer frem for andre. Hævning af abstraktionsniveauet i modeller svarer til det skridt, der tages når man i traditionel programmering bevæger sig fra fx maskinkode til et objektorienteret programmeringssprog [MDD2]. Det modellerede system kan præsenteres på en abstrakt form vha. et modelleringssprog (fx CPN eller UML) og på en konkret måde vha. et programmeringssprog (fx C eller Java), hvis det modellerede system implementeres. Indenfor begge områder eksisterer endnu et valg af abstraktionsniveau. Når systemet modelleres, vil der blive truffet en række valg for detaljeringsniveauet af modellen. Når systemet implementeres, vil selve valget af programmeringssprog omfatte et valg af detaljeringsniveau; i C er det muligt at påvirke den underliggende platform mere direkte end i Java, hvor platformen bortabstraheres vha. virtuel-maskine-paradigmet [COMP].

10.1.2. Relevans for CPN-model af løsningsforslaget

I modellen af løsningsforslaget, resulterer den hierarkiske struktur i forskellige niveauer af abstraktion i forhold til det modellerede system. På modellens øverste niveau, opereres der med et højt abstraktionsniveau, hvor entiteter som *noder* og *kommunikationskanal* indgår. På niveauet herunder opereres med mere detaljerede abstraktioner. Her modelleres indholdet af hver enkelt node og indholdet af kommunikationskanalen. Også på dette niveau dækker modellen over en lang række abstraktioner i forhold til detaljerne i det modellerede system. I forbindelse med modellering af en DAO-node, er detaljerne bag alle komponenter udover kommunikationskomponenten bortabstraheret. Disse komponenters indvirkning på systemet er indkapslet i form af en enkelt funktion, som simulerer komponenternes ændring af variabelværdier. Til gengæld er selve kommunikationskomponenten modelleres væsentligt mere detaljeret i form af en sekvens af handlinger, som hver er modelleret i yderligere detaljer på undersiderne. Dette er et eksempel på, hvordan valg af abstraktionsniveau kan anvendes til at udvælge fokus for en model. Samtidig viser eksemplet, at flere forskellige abstraktionsniveauer med fordel kan eksistere i en model. Systemets overordnede virkemåde forstås nemmest ved at kigge på et højere abstraktionsniveau, men detaljeret viden om løsningsforslagets virkemåde kan indhentes ved at betragte et lavere abstraktionsniveau.

10.2. Forståelighed

10.2.1. Selics beskrivelse af begrebet

En model af et system skal gøre det nemmere at forstå, hvordan det modellerede system fungerer. I [MDD1] understreges vigtigheden af, at bortabstraktionen af detaljer i forbindelse med udvikling af modellen resulterer i en beskrivelse, som letter forståelsen. Det beskrives hvordan der kræves en større teknisk indsigt for at forstå et system ud fra implementationens kildekode end fra en model af systemet. I [MDD3] understreges vigtigheden af, at en gruppe af interessenter i et givet systemdesign skal kunne sætte sig ind i systemets virkemåde. Denne motivation supplerer den generelle fordel ved at en udvikler opnår størst mulig forståelse for det system, der modelleres. Desto mere forståelig en model er, desto større er sandsynligheden for at fejl og misforståelser bliver opdaget. Når modeller anvendes som en del af specifikationsarbejdet, kan de fungere som et fælles kommunikationsmedium imellem projektets interessenter. På den måde kan anvendelse af modeller medvirke til at etablere en bedre fælles forståelse omkring det system, der specificeres.

10.2.2. Relevans for CPN-model af løsningsforslaget

I forbindelse med modellering af løsningsforslaget, er der anvendt en række forskellige metoder for at gøre modellen forståelig. I denne sammenhæng modelleres en kombination af noget kendt og noget nyt - nemlig den eksisterende DAO-kerne i kombination med løsningsforslaget til kommunikationskomponenten. Derfor har det været vigtigt, at fremstille modellen på en måde, som både sikrer genkendelighed og forståelighed på samme tid; det skal være muligt at genkende den eksisterende implementation af DAO-kernen i modellen og samtidig skal det være muligt at forstå, hvordan den foreslåede kommunikationskomponent integreres med det eksisterende. I forbindelse med et relativt komplekst distribueret system, som det aktuelle, har anvendelsen af modeller en konkret fordel i kraft af, at det er muligt at betragte hele det samlede system i én model. I modellens højeste abstraktionsniveauer, fremstilles alle noder sammen med kommunikationskanalen. Hvis udgangspunktet for forståelse af systemet havde været implementationens kildekode, ville det derimod kun have været muligt at betragte detaljerne for en nodes implementation. På den måde eksisterer der ikke direkte information om nodernes relation til den kontekst, de indgår i for at skabe det samlede system.
I forbindelse med modelleringen af systemet er der desuden fokuseret på at gøre brug af en domænespecifik terminologi til navngivning af entiteter i modellen. Dette er gjort dels i forbindelse med modelleringen af den eksisterende DAO-kerne og det nye løsningsforslag. Fra DAO-terminologien hentes begreber som "kernevariable", "attach-beskeder", "aktiveringsperiode", "komponent" osv. På samme måde er modelleringen af løsningsforslaget baseret på den terminologi, som anvendes indenfor litteraturen om signalprotokoller. Indenfor begge områder er det nødvendigt at have en hvis domæneindsigt, for at opnå forståelse for det modellerede. Dette speciale burde give tilstrækkelig indsigt indenfor begge områder, til at gøre modellernes terminologi forståelig.
Endelig er tidsliniebegrebet (se afsnit 8.5) søgt anvendt, for at lette forståelsen af de sekventielle forløb i modellen. Denne metode er anvendt, for at gøre det muligt at observere den centrale eksekveringsrute igennem modellen. Modellen kan på den måde *læses* på samme måde som kildekode, men med den store fordel, at eksekveringen visualiseres og forståelsen af løkker og sideveje i eksekveringsruten derfor lettes. I [MDD1] beskrives besværligheden af at forstå et system ud fra kildekoden alene. Linie efter linie af sekventielt kode er måske nemt at forstå, men i de fleste programmer, er der netop på samme måde som i det modellerede løsningsforslag mange løkker og sideveje i form af funktionskald osv. I denne sammenhæng giver den hierarkiske struktur i modellen den fordel, at det bliver muligt at vælge hvilket abstraktionsniveau modellen læses på. De nævnte metoder er i kombination anvendt for at sikre bedst mulig forståelighed for modellen.

10.3. Nøjagtighed

10.3.1. Selics beskrivelse af begrebet

Modellen skal være en nøjagtig og realistisk fremstilling af det modellerede. Denne egenskab er central for enhver model uafhængigt af type. Hvis det skal give mening at analysere et system på baggrund af en model af systemet, er det vigtigt at det rent modellen rent faktisk er en nøjagtig model af systemet. Derfor er det en stor fordel at have mulighed for at validere en model. En triviel tilgang til validering af en model kunne være, at lade model og implementation generere hver sin logfil med det samme udgangspunkt og herefter sammenligne de to logfiler. Hvis det kan valideres, at en model fremstiller et system på en nøjagtig måde, er det muligt at garantere, at nøjagtighedskriteriet er opfyldt. Validering af en model kan enten ske manuelt eller automatisk af et computerværktøj. For begge typer validering er det nødvendigt at have eksekverbare modeller. Det kan dog ofte være kompliceret eller umuligt, at foretage en sådan validering. Én situation hvor dette i alle tilfælde er umuligt, er hvis modellen fremstiller et system, som endnu ikke er implementeret. Hvis modellen kan oversættes direkte til implementationen (implementationen autogenereres), vil dette give garanti for nøjagtigheden af modellen - på samme måde som et program skrevet i C++ er en nøjagtig beskrivelse af det program, der genereres, når kildekoden kompileres. I [MDD1] beskrives det, hvordan ét af de klassiske problemer i forbindelse med modelbaseret udvikling er, at netop muligheden for generering af implementation på baggrund af modeller ofte ikke er til stede. Derfor sker oversættelse fra model til implementation, når kildekoden skrives af en programmør. Denne situation beskrives som en typisk kilde til unøjagtigheder imellem model og det modellerede system. Mangel på nøjagtighed fremhæves i [MDD1] som én af de mulige grunde til, at modelbaseret udvikling i mange sammenhænge har manglet succes.

10.3.2. Relevans for CPN-model af løsningsforslaget

I forbindelse med modellering af løsningsforslaget til DAO, har der grundlæggende været fokuseret på to typer nøjagtighed. De to typer kunne kaldes *logisk* og *numerisk* nøjagtighed. Den logiske nøjagtighed har at gøre med modellens evne til at fremstille den måde, hvorpå det modellerede system er opbygget, hvordan noderne er forbundet osv. For at optimere den logiske nøjagtighed for modellen, har det været vigtigt at opnå bedst mulig forståelse for, hvordan DAO-frameworket fungerer, hvordan og hvornår komponenter aktiveres osv. Den type nøjagtighed kan kun garanteres fuldt ud, hvis der

anvendes en form for automatisk oversættelse fra model til implementation. I det konkrete tilfælde har dette ikke været muligt, så tiltroen til modellens logiske nøjagtighed kan udelukkende bero på en manuel sammenholdning mellem viden om det modellerede system og selve modellen. Den anden type nøjagtighed - numerisk nøjagtighed - har at gøre med de parameterværdier, som anvendes i modellen. Parameterværdierne anvendes eksempelvis til at konfigurere, hvor lang tid DAO-kernen bruger på at modtage én besked. I den konkrete model er denne parameterværdi en gennemsnitsværdi, som er baseret på målinger i det virkelige system. Alene det at anvende gennemsnitsværdier, vil være kilde til unøjagtigheder i modellen. Over tid vil en fejl i værste fald akkumuleres op, så effekten af unøjagtigheden forøges. Et alternativ ville være, at anvende en værdi, som repræsenterede den tid man maksimalt kunne forestille sig, det ville tage - igen på baggrund af målinger på det virkelige system. Det kan dog være en problematisk løsning, da det stiller store krav til forståelsen for hvordan ændring af parameterværdier ændrer modellens opførsel. Eksempelvis kunne man forestille sig, at en forøgelse af den tid, det tager at modtage en besked vil kunne resultere i et lavere ressourceforbrug på noderne. Således kan forkerte parameterværdier give et misvisende billede af den gennemsnitlige systemopførsel over tid. Løsningen på denne slags problemer, kunne være at modellere forsinkelsen mere præcist. I den aktuelle model, simuleres forsinkelsen med én enkelt statisk konstant. For at øge præcisionen for denne parameter, kunne den i stedet repræsenteres i form af en funktion, som simulerede de statiske egenskaber for forsinkelsen i et virkeligt system ud fra indsamlet viden om dynamikområde, varians osv. for værdien. Denne teknik kunne anvendes for flere af modellens parametre. For at kunne simulere parametre på denne måde, ville det kræve en større indsats i forbindelse med indsamling af målinger fra det virkelige system. Samtidig vil det have den effekt, at udfaldsrummet for målingerne på modellen øges væsentligt i kompleksitet. Resultatet kan være, at det bliver nødvendigt at simulere over væsentligt længere perioder i modeltid, for at få afdækket det modellerede løsningsforslags dynamiske egenskaber. I modsætning til den logiske nøjagtighed, kan den numeriske nøjagtighed ikke garanteres som en direkte konsekvens af, at automatiseret generering af implementation på baggrund af model anvendes. Denne form for nøjagtighed ville til gengæld i en vis udstrækning kunne opnås, hvis dele af modellen blev genereret vha. *reverse engineering* af det aktuelle system. I det aktuelle tilfælde, kunne man altså forestille sig, at den del af modellen, som omhandler kommunikation med DAO-kernen, afsendelse af beskeder osv. blev autogenereret på baggrund af den konkrete implementation. Herefter kunne modellering af løsningsforslaget ske som en viderebygning på den autogenererede model. Det er i denne sammenhæng relevant at være opmærksom på hvordan kriteriet om abstraktion er opfyldt, hvis nøjagtigheden øges væsentligt på denne måde.

10.4. Forudsigelighed

10.4.1. Selics beskrivelse af begrebet

En model skal gøre det muligt at forudsige ikke-trivielle egenskaber for det modellerede system. Viden om systemets egenskaber på baggrund af modellen kan indsamles i forbindelse med simulering eller formel analyse på baggrund af modellen. I begge tilfælde afhænger kvaliteten af den indsamlede viden i høj grad af modellens nøjagtighed. Evnen til at forudsige egenskaber for et system, før det er implementeret, er et vigtigt argument for anvendelse af modeller i forbindelse med udvikling af software. Argumentet er kun gyldigt for nøjagtige modeller. En unøjagtig model fremstiller ikke det forventede system, og forudsigelser på baggrund af modellen vil derfor (sandsynligvis) ikke gælde for det forventede system.

10.4.2. Relevans for CPN-model af løsningsforslaget

Under forudsætning af at kriteriet om nøjagtighed er opfyldt, viser den modelbaserede analyse på baggrund af det modellerede løsningsforslag et på eksempel på forudsigelighed. Det vises, hvordan egenskaber for et endnu ikke fuldt implementeret system bestående af den eksisterende DAO-kerne, eksisterende DAO-komponenter og det foreslåede løsningsforslag vil være i stand til at opfylde kravene til systemet. Flere af disse krav har ikke absolutte numeriske størrelser - eksempelvis er det ikke et krav at konsistensen over tid skal være 100 %, men den målbare størrelse skal optimeres til at være størst mulig. Modellen kan i denne sammenhæng bruges til at forudsige hvor stor konsistensgrad, der kan forventes med forskellige kombinationer af parameterkonfigurationer. I denne sammenhæng anvendes modellen bl.a. til at forudsige sammenhængene imellem de parametre, som bruges til at konfigurere system. Nogle af disse parametre er konfigurerbare parametre i det endelige system (fx aktiveringsperioden) mens andre repræsenterer egenskaber for systemet (fx antal noder eller sandsynligheden for pakketab).
Modellen af løsningsforslaget gør det muligt at forudsige dynamiske egenskaber for det modellerede system. I forbindelse med simulering af den eksekverbare model, er det muligt at fastfryse modeltiden og på den måde opnå indsigt i det samlede systems tilstande over tid. Denne slags forudsigelse af systemets tilstande vil i mange tilfælde være umuligt at foretage på et virkeligt implementeret system. Særligt i det aktuelle tilfælde med DAO's kommunikationskomponent, hvor løsningsforslaget er underlagt en række realtidskrav. Kommunikationen og opdateringen af variable sker så hurtigt, at blot det at opsamle målinger som grundlag for efterfølgende analyse ville påvirke systemets virke-

måde. Dette problem løses, når viden om systemets egenskaber indsamles fra en model. Samtidig kan alle systemets noder i modellen overvåges samtidigt og indsamling af måledata kan synkroniseres. Det ses fx. i forbindelse med måling af kernevariablenes konsistensgrad, hvor målingen sker ud fra et tidsmæssigt fastfrosset billede af alle nodernes variabelinstanser. I denne sammenhængen gør modellen det muligt at forudsige den grad af konsistens, som kan forventes for et givet sæt af parametre - under forudsætning af, at modellen er nøjagtig i forhold til det modellerede.

10.5. Rentabilitet

10.5.1. Selics beskrivelse af begrebet

Det skal kunne svare sig tidsmæssigt og dermed økonomisk at udvikle en model af et system. Det kan være rentabelt at anvende modeller af flere forskellige årsager:

- Hvis implementationen autogenereres ud fra en model, kan modellens forhøjede abstraktionsniveau medvirke til at forkorte udviklingstiden
- Hvis modellen hjælper udvikleren med at indse problemer i systemets design tidligere i forløbet, end hvis der ikke havde været anvendt modellering
- Det kan være kompliceret og dyrt (eller umuligt), at indsamle måledata fra en implementation af systemet

Der findes med sikkerhed flere eksempler på situationer, hvor anvendelse af modeller ideelt set kan være en økonomisk fordel. På samme måde findes der en række situationer, hvor det modsatte gør sig gældende; hvor anvendelse af modeller ikke bidrager til en bedring af økonomien i forbindelse med et projekt.

10.5.2. Relevans for CPN-model af løsningsforslaget

Modellen af løsningsforslaget har givet viden om systemet, som kunne have været svær at opnå uden en model. Hvis et sådant system var blevet udviklet uden den foregående kendskab til parametrenes effekt, realtidsegenskaberne osv., kunne konsekvensen være, at systemet i sidste ende ikke var i stand til at opfylde kravene til rettidighed i forbindelse med opdatering af variable, de maksimale grænser for ressourceforbrug for kommunikationskomponenten osv. I [MDD4] beskrives det, at fejl i forhold til de specificerede krav typisk opdages som de sidste – og at denne type fejl samtidig er den dyreste type at rette. Problemet opstår, fordi en fungerende udgave af det system, der er under udvikling, typisk først er til rådighed meget sent i projektforløbet. I forhold til den aktuelle

opgave med at udvælge en ny kommunikationsprotokol til kommunikationskomponenten i DAO er denne problemstilling meget relevant. Under forudsætning af, at modellen lever op til kravet om nøjagtighed, kan måledata indsamlet fra simulering vha. modellen erstatte måledata fra et implementeret system i forbindelse med analyse af løsningsforslagets evne til at opfylde kravene. En model af løsningsforslaget kan forhindre, at der bruges ressourcer på at udvikle en løsning, som ikke kan opfylde kravene. Samtidig giver en model et grundlag for at vælge et egnet design af løsningen.

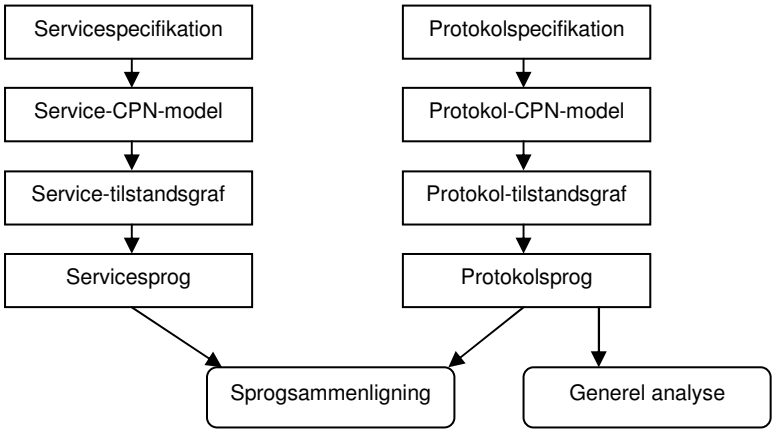
10.6. Sammenligning med andre metoder

I dette speciale er løsningsforslagets egenskaber vurderet i en analyse, som er foretaget ud fra måledata, som er indsamlet i forbindelse med simulering af en model af løsningsforslaget. Her beskrives to alternative metoder til modellering af distribuerede systemer og kommunikationsprotokoller. Metoderne beskrives i hovedtræk og sammenlignes med den metode, som er anvendt i dette speciale.

10.6.1. Formel protokolverifikation

I [PV] beskrives analysen af to forskellige kommunikationsprotokoller ved hjælp af CPN. [PV] præsenterer en metode til verifikation af protokollers korrekthed. CPN anvendes til at beskrive protokollerne og til at påvise eksistens af eller mangel på krævede egenskaber for den protokol, som undersøges. Metoden er målrettet protokoller, som optræder i protokolstakke og derfor leverer en service til et overliggende lag og samtidig forudsætter en leveret service fra et underliggende lag. Formålet med metoden er at verificere om en given protokol opfylder dels en række generelle krav og dels en række protokolspecifikke krav.
Figur 35 viser de elementer, som indgår i metoden. Set fra toppen eksisterer der to forskellige specifikationer af protokollen: Kravene til protokollen og specifikationen af protokollen repræsenteres af hhv. "Servicedefinition" og "Protokoldefinition". Protokolspecifikationen stammer typisk fra en standardspecifikation af protokollen (fx [TCP]). Servicespecifikationen beskriver de egenskaber for protokollen, som overliggende lag i protokolstakken forventer. De to specifikationer danner grundlag for to CPN-modeller: "Service-CPN-model" og "Protokol-CPN-model". Et af målene med protokolverifikationen er at undersøge, om disse to modeller giver anledning til samme funktionalitet. Et andet mål er at undersøge protokollen for tilstedeværelse af en række generelle egenskaber. For at kunne gennemføre begge disse undersøgelser, genereres der tilstandsgrafer (occurrence graphs) ud fra de to CPN-modeller. En tilstandsgraf kan betragtes som en til-

standsmaskine, som definerer et sprog bestående af hændelser, som kan optræde i CPN-modellen. Grafens noder svarer til individuelle opnåelige mærkninger af CPN-modellen og pilene imellem noderne svarer til hændelser, som medfører transitioner imellem disse mærkninger. En tilstandsgraf for en CPN-model kan blive meget stor, hvis modellen er kompleks. Derfor kan det ofte betale sig, at holde de dele af modellen, som oversættes til tilstandsgrafer så simple som muligt. De to tilstandsgrafer definerer altså hhv. servicesproget og protokolsproget. Hændelserne er kommunikationsprimitiver og kriteriet for, om en hændelse optræder i tilstandsgrafen er, at den optræder på protokollens grænseflader. På den måde svarer de to sprog til mulige sekvenser af beskeder, som kan sendes og modtages af protokollen.



Figur 35 - Elementer i metoden. Kilde: [PV]

For at undersøge om en given protokol overholder specifikationen, sammenlignes de to sprog. Hvis sprogene repræsenterer de samme sekvenser af kommunikationsprimitiver, er protokolspecifikationen overholdt. Hvis der er forskel imellem de to sprog er der fejl eller mangler i servicespecifikationen i forhold til protokolspecifikationen (under antagelse af at sidstnævnte er korrekt).

De to tilstandsgrafer anvendes altså i kombination til at verificere protokollens korrekthed i forhold til specifikationerne. Protokol-tilstandsgrafen alene danner grundlag for den anden type undersøgelse, som omfatter identifikation af situationer, hvor protokollen går i baglås afslutter uforudset osv. Denne type undersøgelse kan baseres på en række generelle principper, som er typiske for mange protokoller.

Den vigtigste forskel imellem metoden, som beskrives i [PV] og metoden, som anvendes i dette speciale er måden hvorpå modellen analyseres. Metoden i [PV] fokuserer på de logiske og kvalitative egenskaber for protokoller i form af rækkefølge for udvekslede be-

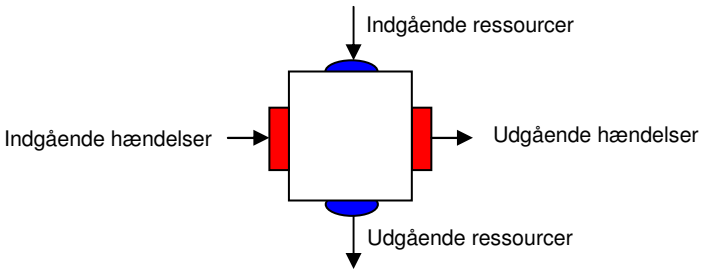
skeder osv. I den anvendte metode i dette speciale er analysen til gengæld baseret på de realtidsmæssige egenskaber for løsningsforslaget. Her har der været fokus på at analysere de kvantitative egenskaber i løsningsforslaget med udgangspunkt i modellen. Den kvantitative analyse tager udgangspunkt i målbare størrelser såsom konsistensgrad, forsinkelse osv.

De to metoder anvendes begge indenfor det samme domæne; modelbaseret protokolanalyse og i begge metoder anvendes CPN som værktøj til modellering. Til gengæld repræsenterer de to metoder altså to helt forskellige tilgange til analysen. Valget af metode må derfor baseres på en vurdering af hvilke egenskaber, der ønskes undersøgt i forbindelse med analysen. I dette speciale har det været relevant at undersøge realtidsegenskaberne for løsningsforslaget, hvorfor den kvantitative analysemetode har forekommet mest egnet.

En anden egenskab, som adskiller de to metoder, er anvendelsen af tilstandsgrafer i [PV]. På grund af sammenhængen imellem CPN-modeller og deres tilstandsgrafer, er analyse på baggrund af tilstandsgrafer kun praktisk håndterbart for relativt små modeller [CPN1]. Det skyldes at tilstandsgrafer har en tendens til at blive meget store, når CPN-modellen vokser i kompleksitet. I forbindelse med det konkrete løsningsforslag giver alene tidsaspektet et enormt udfaldsrum for hvornår beskeder afsendes. Det vil blive afspejlet i form af voldsom kompleksitet i modellens tilstandsgraf. Denne egenskab er generel og gælder ikke kun for den metode, som beskrives i [PV].

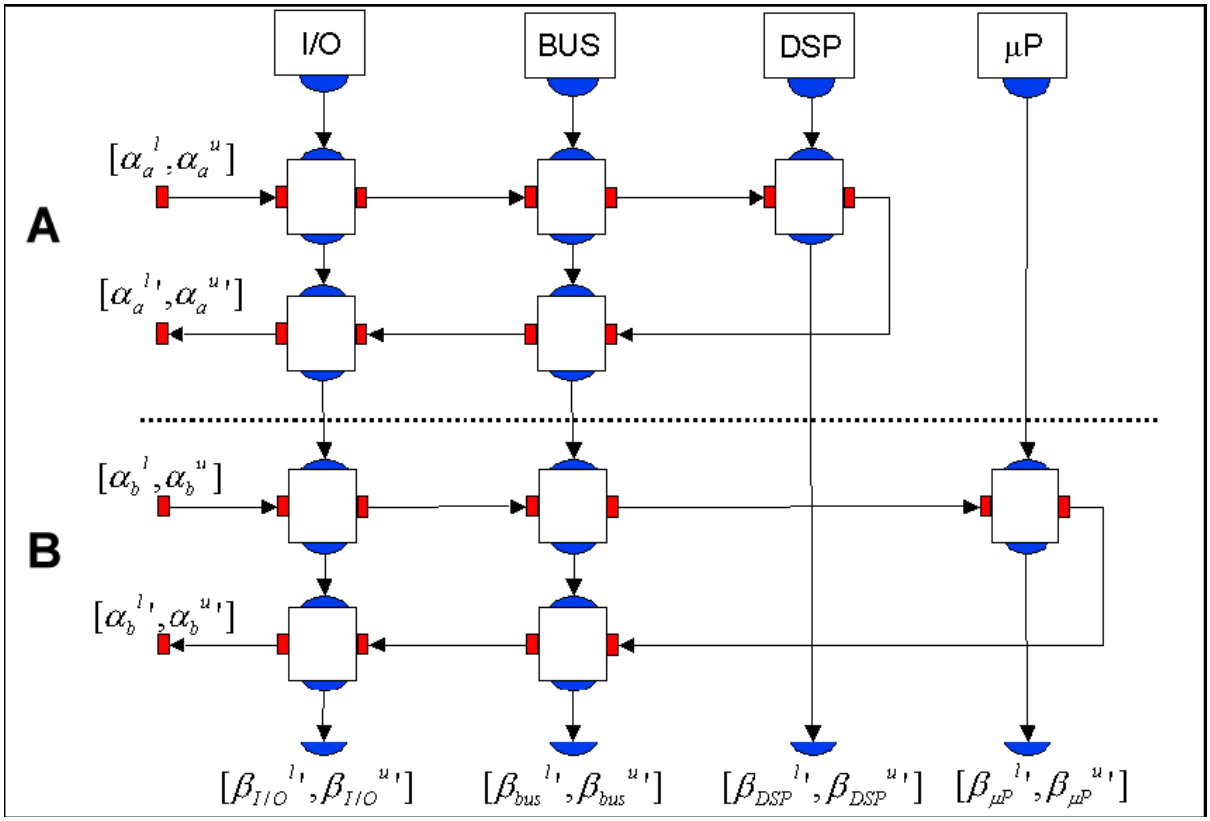
10.6.2. Modular Performance Analysis

Metoden "Modular Performance Analysis" (MPA) beskrives i [MPA] og er en metode til estimering af ressourceforbrug i forbindelse med design af distribuerede systemer. Metoden er velegnet til at identificere flaskehalse og kan på den måde danne grundlag for valg af en egnet arkitektur for et givet system.



Figur 36 - En komponent

I MPA modelleres et helt system som et net af komponenter (performance components). Komponenterne er byggeklodser, som kan repræsentere en algoritme, en hel hardwareenhed eller lignende. På Figur 36 vises en komponent. En komponent er i stand til at modtage og håndtere indkommende hændelser og genere udgående hændelser. I forbindelse med håndtering og generering af hændelser, kan komponenten forbruge ressourcer. Med mindre komponenten forbruger alle ressourcer fra en given ressourcekilde, stilles den del af de tilgængelige ressourcer, som ikke er blevet forbrugt til rådighed for andre komponenter. En CPU eller en netværksforbindelse kan repræsenteres i form af komponenter, som er i stand til at stille hhv. processerings- og kommunikationsressourcer til rådighed. På samme måde kan eksterne entiteter repræsenteres i form af komponenter, som er i stand til at generere hændelser. Et helt system modelleres ved at sammensætte et net af komponenter. På Figur 37 ses et eksempel på et sammensat net. Den øverste række af komponenter er ressourcekilder, som repræsenterer hardwareenheder og netværket, som forbinder dem. Ressourcekilderne stiller processerings- og kommunikationsressourcer til rådighed for nettets øvrige komponenter. I kolonnen til venstre ses to hændelseskilder (med højrerettede pile), som genererer hændelser til nettets komponenter.



Figur 37 - Et eksempel på et sammensat net af komponenter. Kilde: [MPA]

Et sammensat net af komponenter kan anvendes til beregning af egenskaberne for det modellerede system. Beregningerne udføres vha. metoden "Real Time Calculus" (RTC), som beskrives i [RTC]. Den samme metode anvendes til at beskrive hver enkelt komponent i nettet. Beskrivelsen omfatter en funktionel specifikation af hver enkelt komponents ind- og udgående strømme af hændelser og ressourceforbrug. Samtidig danner RTC grundlag for en formel analyse af det samlede net af komponenter, med udgangspunkt i de enkelte komponenters specifikationer. Denne form for analyse resulterer i estimater for det maksimale ressourceforbrug, beskedforsinkelser osv.
MPA er tiltænkt anvendelse på et langt højere abstraktionsniveau end den CPN-baserede metode, som anvendes til modellering af løsningsforslaget. I den CPN-baserede metode, modelleres selve funktionaliteten af systemet. Dette er ikke tilfældet i MPA, hvor funktionalitet bortabstraheres på et højt niveau og indkapsles af funktioner, som udelukkende beskriver det forventede maksimale ressourceforbrug for komponenter, den hyppighed af generering af hændelser osv. Denne metode gør MPA velegnet i forbindelse med valg af hardwareplatforme, netværkstopologi osv. på et tidligt stadie i udviklingsforløbet. Metoden gør det muligt at identificere overforbrug eller spild af ressourcer ud fra en overordnet viden om systemet. Til gengæld giver bortabstraktionen af funktionalitet et mere grovkornet grundlag for den efterfølgende analyse. Samtidig vil den manglende modellering af funktionalitet gøre det umuligt at indsamle den type måledata, som er anvendt i forbindelse med den modelbaserede analyse for løsningsforslaget i dette speciale. I dette speciale er funktionaliteten modelleret relativt detaljeret. Metoden til bortabstraktion i MPA kunne i en lignende form anvendes i forbindelse med en CPN-model, for at højne abstraktionsniveauet i forbindelse med modellering af funktionalitet.

11. Konklusion

Dette afsnit opsummerer det arbejde, der er lavet i forbindelse med specialet. Første underafsnit beskriver de resultater, der er opnået. Derefter vurderes mulighederne for at arbejde videre med de resultater, der er beskrevet i dette speciale.

11.1. Beskrivelse af arbejdet og dets resultater

Arbejdet i dette speciale blev indledt med en indgående beskrivelse af DAO-systemet. Denne beskrivelse er i sig selv et værdigfuldt resultat, eftersom der ikke foreligger nogen opdateret dokumentation af DAO-frameworket. Samtidig danner beskrivelsen grundlag for den efterfølgende udvælgelse og modellering af løsningsforslaget. Arbejdet med beskrivelse af DAO viste sig at være en meget effektiv måde at opnå indsigt i detaljerne i systemet. Samtidig dannede det skrevne materiale et godt fundament for diskussion omkring problemstillingerne. Selve afsnittet som beskriver DAO var flere gange til gennemlæsning hos Vestas, og én af de erfaringer der blev gjort i den sammenhæng var, at det var nemmere at blive enige om fejl i det skrevne og på den måde forfine beskrivelsen end i én omgang at ramme en præcis beskrivelse. Afsnittet om DAO blev afsluttet med en forklaring af problemstillingerne i forbindelse med kommunikationen imellem noderne i DAO. Disse problemstillinger har dannet udgangspunktet for det efterfølgende arbejde med udvælgelse af løsningsforslag, modellering, analyse osv.
Efter arbejdet med at beskrive DAO, blev arbejdet med at finde en egnet protokoltype til brug for kommunikationsprotokollen indledt. Det blev hurtigt klart, at en form for signaleringsprotokol kunne være en oplagt løsning. Derfor blev afsnittet om DAO efterfulgt af et afsnit, som introducerer de grundlæggende aspekter indenfor området for signaleringsprotokoller. Der blev defineret en generisk systemmodel, som blev anvendt til at forklare en gruppe varianter af signaleringsprotokoller. Indholdet i dette afsnit var resultatet af arbejdet med læsning af en samling kerneartikler indenfor området. Der blev bevidst sigtet imod at levere en tilpas generel beskrivelse af begreberne. Dels for at højne forståelsen for domænet og dels for at øge sandsynligheden for at resultaterne i dette speciale kan finde anvendelse i andre sammenhænge.
Efter beskrivelsen af konceptet bag signaleringsprotokoller og en gennemgang af en række generiske varianter, fulgte opgaven med at specificere kravene til et løsningsforslag i den konkrete sammenhæng. I dette afsnit blev der argumenteret for de ønskede egenskaber for kommunikationssystemet og denne argumentation blev sammenkædet med de generelle egenskaber for signaleringsprotokoller for at give et udgangspunkt for valg af løsningsforslaget. Afsnittet blev afsluttet med et konkret valg af protokoltype. For

at sætte dette valg i perspektiv til det eksisterende system, blev løsningsforslaget og den eksisterende implementation sammenlignet.
Efter at have afsluttet beskrivelsen af dels DAO og dels løsningsforslaget, var grundlaget for udvikling af modellen lagt. Det var klart *hvad* der skulle modelleres og *hvilke* egenskaber, der kunne være relevante at analysere med udgangspunkt i modellen. Næste skridt var at gå i gang med at undersøge *hvordan* en sådan model i CPN konkret skulle udvikles. Arbejdet med modellen blev indledt med litteraturlæsning om de helt grundlæggende principper indenfor CPN. Samtidig blev der kigget på artikler, som beskriver lignende anvendelser af CPN. Denne læreproces omkring CPN resulterede i første omgang i skrivning af et generelt afsnit om begreberne indenfor CPN. Samtidig blev læsningen af teoretisk litteratur tidligt understøttet af praktiske eksperimenter med værktøjet CPN Tools. Fra første færd blev der arbejdet med at udvikle modeller med relation til DAO. Disse modeller dannede så grundlag for diskussion og blev løbende forfinet. Som noget af det første, blev det klart, at det var vigtigt at introducere en overskuelig hierarkisk struktur. Herefter kunne modellen udvikles iterativt og med stadigt voksende detaljeringsgrad. Det endelige resultat er den model, som i form af dens forskellige undersider blev præsenteret i et separat afsnit i specialet. I dette afsnit blev modellen beskrevet i detaljer. Under arbejdet med modellen blev der meget bevidst fokuseret på at opretholde en stærk binding til den forudgående beskrivelse af DAO og løsningsforslaget. Denne binding blev etableret på både terminologisk og logisk niveau med det formål at sikre bedst mulig forståelse og højst mulig nøjagtighed for modellen. Undervejs med udviklingen af modellen blev der gjort en lang række erfaringer med forskellige aspekter af CPN og protokolmodellering generelt. En central samling af disse erfaringer blev beskrevet i sammenhæng med beskrivelsen af selve modellen. Sammen med den forudgående beskrivelse af hhv. DAO og CPN blev dette afsnit skrevet for at knytte de to verdener sammen og give en tekniknær indsigt i CPN-modellerne og teorien bag. Beskrivelsen af modellen blev afsluttet med en gennemgang af de parametre, som anvendes til at konfigurere modellen og dermed kom til at danne grundlag for den efterfølgende indsamling af måledata.
Efter arbejdet med at udvikle modellen var næste skridt at bruge modellen til at indsamle en stor mængde måledata, som kunne bruges som grundlag for en analyse af egenskaberne ved løsningsforslaget. I denne sammenhæng var det nødvendigt at indføre en form for instrumentering af modellen, som gjorde det muligt at registrere modellen og de enkelte pladsers løbende tilstande i forbindelse med simulering. I første omgang blev denne instrumentering etableret ved at introducere en række dedikerede pladser og transitioner i modellen, hvis formål var at skrive statistiske oplysninger til logfiler, som efterfølgende kunne analyseres. Dette viste sig dog at være en uhensigtsmæssig metode. Metoden betød at der blev oprettet elementer i modellen, som ikke repræsenterede

elementer i det modellerede, hvilket resulterede i en manglende sammenhæng imellem model og det modellerede. Disse metaelementer forstyrrede det samlede billede, slørede modellens fokus og gjorde generelt modellen mindre forståelig. Heldigvis viste der sig en løsning på dette problem i form af en ny funktionalitet i CPN Tools; monitorer. Denne funktionalitet eksisterede endnu ikke i den offentliggjorte udgave af værktøjet, men vha. en betaversion var det muligt at anvende monitorer i forbindelse med modellen af DAO og løsningsforslaget. Monitorerne gjorde det muligt at specificere detaljerne for indsamling af måledata helt uafhængigt af selve modellens opbygning. Det medførte en bedre mulighed for at opretholde en domænespecifik fokusering i modellen og lettede desuden opgaven med indsamling af måledata fra simuleringer væsentligt. I forbindelse med indsamling af måledata opstod der flere gange den situation, at der blev opdaget mindre fejl eller misforståelser i modellen – fejl som først blev synlige da der forelå måledata. Det betød, at modellen måtte rettes til og indsamlingen af måledata måtte gentages. Opdagelsen af fejl betød uventet ekstra arbejde, men samtidig viste observation fordelen ved muligheden for at indsamle målbare data fra modellen.

Efter indsamling af måledata var det nu muligt at gennemføre en analyse af løsningsforslaget. Analysen blev inddelt i fire underområder, hvor fire forskellige relevante aspekter ved løsningsforslaget blev undersøgt. For hver enkelt underområde blev målemetoden beskrevet i detaljer hvorefter selve de indsamlede måledata blev præsenteret og dannede grundlag for en diskussion. Formålet med analyseafsnittet var at give indsigt i sammenhængene imellem det modellerede løsningsforslags parametre og på den måde gøre det muligt at vælge en passende konfiguration for kommunikationskomponenten, hvis den på et tidspunkt skulle implementeres på den fysiske platform. Samtidig repræsenterer diskussionerne en generel undersøgelse af kombinationen imellem det eksisterende DAO-system og en signaleringsprotokol.

Igennem arbejdet med modellering og analyse blev der gjort en lang række erfaringer. Disse erfaringer blev brugt i forbindelse med den efterfølgende diskussion omkring selve den anvendte metode. Diskussionen baseres på fem generelle kriterier for kvalitet af modelbaseret udvikling. Disse kriterier blev beskrevet på baggrund af en lille samling artikler med fokus på én artikel, hvor kriteriernes inddeling stammer fra. Hvert enkelt kriterium blev efterfølgende anvendt som udgangspunkt for en diskussion af det konkrete arbejde med modellering af løsningsforslaget. Diskussionen gav en god mulighed for at vurdere det gennemførte arbejde og sammenfatte de erfaringer der var gjort løbende. Diskussionen blev afsluttet med en undersøgelse af den anvendte metode set i forhold til andre alternative metoder inden for modelbaseret udvikling. Resultatet blev en sammenligning med to modelbaserede metoder. Fælles for de tre metoder (den anvendte og de to alternative) var, at de var udviklet med fokus på modellering af distribuerede systemer. Denne sammenligning gav et godt indtryk af den anvendte metode set i forhold til

alternativerne. Perspektiveringen tjente to formål; dels at diskutere alternativer til den valgte metode og dels at indplacere den valgte metode imellem allerede beskrevne metoder for at muliggøre en generalisering af resultaterne i dette speciale.

11.1.1. Løsningsforslag

Specialet giver Vestas et gennemanalyseret løsningsforslag til en konkret og aktuel problemstilling. Resultatet er en meget domænespecifik model af løsningsforslaget, som hænger nært sammen med de allerede kendte begreber indenfor den eksisterende software i firmaet. Modellen kan konfigureres vha. en lang række parametre, som svarer til de parametre, som i forvejen indgår i overvejelserne i forbindelse med den eksisterende kommunikationsprotokol. Løsningsforslaget er ledsaget af en gennemgående analyse af dets egenskaber baseret på måledata indsamlet fra modellen. Modellen af løsningsforslaget og beskrivelsen af arbejdet med udarbejdelse af modellen og den efterfølgende modelbaserede analyse viser Vestas et omfattende eksempel på, hvordan modelbaserede metoder kan finde anvendelse indenfor firmaet tekniske domæne. På denne måde giver specialet dels en indsigt i hvordan en modelbaseret metode kan anvendes og dels hvilke resultater en sådan metode kan give i forbindelse med udvikling af software til firmaets vindmøller. Modellen er opbygget med stærk fokus på nøjagtighed i forhold til det modellerede. Derfor er måleresultaterne ikke kun eksempler på typerne af analysemuligheder, men også rent faktiske resultater. Denne slags resultater kan finde direkte anvendelse i forbindelse med udvælgelse af parameterværdier, hvis den foreslåede løsning kommer til at danne grundlag for videre arbejde.

11.1.2. Generaliserbare resultater

Et system, hvor DAO anvendes er kendetegnet ved en række egenskaber: Direkte kontakt til omgivelserne via sensorer og aktuatorer, systemet består af en gruppe kommunikerende noder med begrænsede processerings- og netværksressourcer, disse noder er afhængige af at kunne dele outputværdier og måledata og endelig er delingen af information underlagt skrappe realtidskrav. Hvis det antages, at disse egenskaber er typiske for mange andre systemer end netop vindmøllesystemer hos Vestas kan det ligeledes med passende forudsætninger antages, at de problemstillinger som opleves i forbindelse med det konkrete system er typiske for systemer med lignende egenskaber. Hvis problemstillingen betragtes som værende generel ved at hæve abstraktionsniveauet en anelse, vil også løsningsforslaget have generel værdi i forbindelse med løsning af lignen-

de problemstillinger. Selve analysen af løsningsforslaget er specifikt rettet mod det konkrete DAO-system, men argumentationen for signalprotokollers anvendelighed i denne sammenhæng kan have relevans i lignende sammenhænge. Det samme kan den efterfølgende vurdering af den anvendte metode. Denne diskussion kan give grundlag for at vurdere metodens generelle egnethed i forbindelse med systemer af denne type.
Selve løsningsforslaget kan overføres til lignende projekter og det samme kan metoden, som er anvendt til at analysere løsningsforslagets egenskaber. Metoden viser hvordan en model kan anvendes som grundlag for vurdering af en skræddersyet signaleringsprotokol og dennes målbare egenskaber i form af konsistensgrad, forsinkelse og ressourceforbrug osv.

11.2. Fremtidigt arbejde

Arbejdet med dette speciale har blotlagt et stort og interessant forskningsområde indenfor anvendelse af modelbaserede metoder. Der er mange muligheder for at arbejde videre, med de problemstillinger, som er berørt undervejs. Arbejdet med vurdering af den anvendte metode har bidraget med identifikation af en lang række interessante problemstillinger indenfor det modelbaserede domæne. Problemstillinger er beskrevet på overordnet niveau og er i den sammenhæng sammenholdt med det konkrete projekt. Det kunne være interessant at gå videre med undersøgelse af disse og andre problemstillingers relevans for meget praktisk relaterede projekter som det aktuelle. Samtidig ville det være relevant at hæve abstraktionsniveauet for resultaterne i dette speciale, således at de i højere grad gøres anvendelige i en generel sammenhæng.

11.2.1. Implementation af løsningsforslag

Dette speciale har præsenteret argumentation for et løsningsforslag til en konkret problemstilling suppleret med en efterfølgende modelbaseret analyse af den foreslåede løsning. Den modelbaserede analyse har dannet et beslutningsgrundlag for valg af et alternativt design for kommunikationskomponenten i DAO. Samtidig foreligger der i dette speciale en analyse af sammenhængen imellem løsningsforslagets parametre. Det næste naturlige skridt ville være at implementere løsningsforslaget i det eksisterende DAO-system. Hvis løsningsforslaget fører til en implementation, vil det være hensigtsmæssigt at lade implementationen følge det design, som beskrives af modellen af løsningsforslaget. På den måde højnes sandsynligheden for, at analyseresultaterne for løsningsforslaget kan finde anvendelse i forbindelse med valg af parameterværdier for implementationen.

11.2.2. Validering af model ud fra implementation

Hvis løsningsforslaget blev implementeret i sammenhæng med det eksisterende DAO-system, ville det være interessant at undersøge sammenhængen imellem modellen og implementationen. Hvis implementationen bevidst blev lavet som en nøjagtig udgave af det modellerede løsningsforslag, ville det være relevant, at undersøge modellens evne til at forudsige egenskaber før systemet modelleres. Dette kunne gøres ved, at genskabe den beskrevne analyse på baggrund af måledata indsamlet fra implementationen i stedet for fra modellen, som det har været tilfældet i dette speciale. Dette ville give en værdigfuld indikation af kvaliteten af den anvendte model.

11.2.3. Mere detaljeret manuel validering af model

I [MDD] foreslås manuel validering af modeller. Dette er en oplagt mulighed, for at indlemme en domænespecialist i udviklingen af modellen. I forbindelse med dette speciale vil det være oplagt at bruge en dag i fællesskab med en specialist fra Vestas på at gennemgå modellen for at sikre bedst mulig grad af virkelighedstro modellering i forhold til det eksisterende DAO-system. Dette ville bidrage til at forbedre modellens præcision og ville samtidig være en oplagt måde at få overleveret specialets resultater til videre brug hos Vestas.

11.3. Generel konklusion

Valget af emne for specialet har givet en spændende mulighed for at arbejde med en virkelig problemstilling, hvis løsning har relevans for Vestas. Det har gjort arbejdet interessant og samtidig krævende eftersom det har været nødvendigt at opnå en detaljeret forståelse for et allerede implementeret system hos Vestas. Det er mit håb at de resultater, som dette speciale bidrager med kan være til gavn dels for Vestas og dels i en generel sammenhæng i forhold til forståelse for muligheder og begrænsninger i forbindelse med modelbaseret udvikling og analyse af realtidskritiske systemer.

12. Litteraturliste

[VMI1] http://www.windpower.org/da/stats/turnover.htm

[VMI2] http://www.windpower.org/da/stats/averagepowerrating.htm

[VWS] http://www.vestas.com/dk/Home/index.asp

[MULT] G. Couloris, J. Dollimore: *"Distributed Systems, Concepts and Design"*, Addison-Wesley, 2001

[PETRI] C. A. Petri: "*Communication with Automata*", Griffis AFB, 1966 (tysk original: *"Kommunikation mit Automaten"*, 1962)

[CPN1] K. Jensen: *"Coloured Petri Nets - Basic concepts, analysis methods and practical use (vol. 1, 2. udgave)"*, Monographs in theoretical computer science, Springer, 1996

[CPN2] http://wiki.daimi.au.dk/cpntools

[CPNGR] http://www.daimi.au.dk/CPnets

[RMS] C.L. Liu, James W. Layland: *"Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment"*, Journal of the Association for Computing Machinery (ACM), Vol 20, No. 1, januar 1973

[AO] R. Greg Lavender, Douglas C. Schmidt: *"Active Object: An Object Behavioral Pattern for Concurrent Programming"*, Proc. Pattern Languages of Programs, 1995

[JAWS] James C. Hu, Douglas C. Schmidt: *"JAWS: A Framework for High-performance Web Servers",* Wiley & Sons, 1999

[UDP] J. Postel: *"User Datagram Protocol"*, RFC 768, USC/Information Science Institute, 1980

[RSVP] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, Daniel Zappala: *"RSVP: A New Resource ReSerVation Protocol"*, IEEE Network Magazine, september 1993

[SS2] P. Ji, Z. Ge, J. Kurose, D. Towsley: *"A comparison of hard-state and soft-state signaling protocols*", SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, 2003

[ATM] A. Alles: *"ATM Internetworking"*, Cisco Systems, 1995

[SIP] M. Handley, H. Schulzrinne, E. Schooler: *"SIP: Session Initiation Protocol"*, RFC 3161, Network Working Group, The Internet Society, 2002

[SS1] Suchitra Raman, Steven McCanne: *"A model, Analysis, and Protocol Framework for Soft State-based Communication"*, SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, 1999

[INT] R. Braden and D. Clark and S. Shenker: *"Integrated Services in the Internet Architecture: an Overview*", RFC 1633, Internet Engineering Task Force, 1994

[DARPA] David D. Clark: *"The Design Philosophy of the DARPA Internet Protocols"*, SIGCOMM '88: Symposium proceedings on Communications architectures and protocols, 1988

[SS3] B. Ling, E. Kiciman, A. Fox: *"Session State: Beyond Soft State"*, Stanford University, 2004

[TCP] J. Postel: "Transmission Control Protocol", RFC 793, USC/Information Science Institute, 1981

[SPEC] R. J. Wieringa: *"Design Methods for Reactive Systems: Yourdon, Statemate and the UML"*, Morgan Kaufmann, 2003.

[RM] J. Crowcroft, K. Paliwoda: *"A multicast transport protocol"*, SIGCOMM '88: Symposium proceedings on Communications architectures and protocols, 1988

[CPN3] N. Mulyar, W. M. P. van der Aalst: *"Towards a Pattern Language for Colored Petri Nets"*, Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 2005

[SML1] R. Milner, M. Tofte, R. Harper: *"The definition of Standard ML"*, Michigan Institute of Technology, 1990

[FUNK1] R. A. Finkel: *"Advanced programming language design"*, Addison-Wesley, 1996

[MDD1] B. Selic: "*The Pragmatics of Model-Driven Development*", IEEE Software, 2003

[MDD2] S. J. Mellor, A. N. Clark, T. Futugami: *"Model-Driven Development*", IEEE Software, 2003

[COMP] N. Nisan, S. Schocken, *"The Elements of Computing Systems"* (s. 100-102), MIT Press, 2005

[MDD3] C. Atkinson, T. Kühne: *"Model-Driven Development: A Metamodeling Foundation"*, IEEE Software, 2003

[MDD4] H. Gomaa: *"designing concurrent, distributed, and real-time applications with UML"* (s. 91-93), Addison-Wesley Longman, 2000

[PV] J. Billington, G. E. Gallasch, B. Han: *"A Coloured Petri Net Approach to Protocol Verification"*, Lectures on Concurrency and Petri Nets (s. 212-220), Springer, 2003

[MPA] E. Wandeler, L. Thiele, M. Verhoef, P. Lieverse: *"System Architecture Evaluation Using Modular Performance Analysis - A Case Study"*, ISOLA 2004, 2004

[RTC] L. Thiele, S. Chakraborty, M. Naedele: *"Real-time calculus for scheduling hard real-time systems"*, Proc. IEEE ISCAS 2000, 2000

13. Bilag

13.1. Projektdagbog

Dato	**Beskrivelse**
8/2-2005	Første møde hos Vestas med Jesper Schmidt og Jes Rasmussen Indledende snak om DAO-systemet og problemstillinger i forhold til kommunikation og ressourceforbrug
9/2-2005	Første møde vejleder, Jens Bæk Jørgensen Indledende snak med udgangspunkt i det forudgående møde hos Vestas. Beslutning om at fokusere på modellering frem for prototype
9/3-2005	Møde hos Vestas med Jesper Schmidt Snak om DAO
18/3-2005	Vejledermøde Review af afsnit 3 (indledning)
6/4-2005	Vejledermøde Review af afsnit 1 (case study afsnit)
4/5-2005	Møde hos Vestas med Jesper Schmidt Detaljesnak om DAO med udgangspunkt i det skrevne case study-afsnit. Det var meget udbytterigt at snakke om systemet på baggrund af mit eget case study afsnit. Det resulterede i at mange uklarheder i min forståelse af systemet blev afdækket. Jeg besluttede efterfølgende at arbejde videre med en ny udgave af case study afsnittet.
4/8-2005	Vejledermøde Review af nyt case study afsnit
18/8-2005	Vejledermøde Review af afsnit om signaleringsprotokoller Det blev aftalt at bytte om på rækkefølgen i de to afsnit om CPN, så modellering kan gå forud for den teoretiske beskrivelse.
29/8-2005	Endelig accept af case study-afsnit fra Vestas (med smårettelser)
13/9-2005	Vejledermøde Snak om det igangværende modelleringsarbejde. Første udgave af modellen blev diskuteret. Det blev aftalt, at sørge for mere overskuelig hierarkisk strukturering af modellen.
27/9-2005	Vejledermøde Fremvisning af ny struktur for model. Diskussion om det forestående arbejde med beskrivelse af modellen. Snak om dansk terminologi for CPN-begreber. Desuden blev det overvejet at basere den teoretiske introdukti-

	on af CPN på eksempler fra modellen.
23/11-2005	Vejledermøde Review af afsnit om modelbaseret analyse. Der blev lavet aftale for den sidste fase af arbejdet.
5/11-2005	Vejledermøde Gennemgang af samlet version af specialet.

13.2. Detaljerede oplysninger fra Vestas

Dette bilag indeholder en opsummering af alle de konkrete oplysninger om DAO-systemet, som er indsamlet fra Vestas. Nogle oplysninger stammer fra målinger mens andre er værdiangivelser for systemparametre, realtidskrav osv.

1. For hver 100.000 afsendte beskeder imellem to noder vil 1 i gennemsnit gå tabt hos enten afsenderen eller modtageren og ikke nå frem til kommunikationskomponenten hos modtageren. Sandsynligheden for hver af de to situationer er ligeligt fordelt. I et system med mere en to noder (dvs. mere end én modtager) er situationen den samme. I dette tilfælde vil betyder et tab hos modtageren, at kommunikationskomponenten hos én modtager ikke modtager beskeden korrekt. Situationen kan opstå flere end én node for samme besked.
2. Det tager 850 µs for kommunikationskomponenten på en DAO-node at håndtere en besked; dvs. afsende eller modtage et UDP-datagram via protokolstakken.
3. Kommunikationskomponenten på en DAO-kerne aktiveres periodisk med en periodetiden 50 ms. Aktiveringsperioden bestemmes af en konfigurerbar parameter.
4. Kernevariablene ændrer værdi med forskellig hyppighed. Herunder vises den fordeling, som kan antages at passe for et typisk system:

   5 % af alle kernevariable ændres med en frekvens på 100 Hz
   10 % ændres med 50 Hz
   15 % ændres med 10 Hz
   20 % ændres med 1 Hz
   50 % ændres med >1 Hz

5. Når en kernevariabel ændrer værdi, skal alle instanser af kernevariablen opdateres til den nye værdi inden der er gået 50 ms fra ændringstidspunktet.
6. Antal noder i et system:
   Minimum: 2
   Typisk: 3
   Maksimum: 5

7. Antal unikke kernevariable i et system:
   Minimum: 2000
   Typisk: 3000
   Maksimum: 5000

8. Antal input-tilknytninger til eksterne kernevariable for hver node:

Minimum: 400

Typisk: 600

Maksimum: 1000

9. Antal input/output-tilknytninger til eksterne kernevariable for hver node:

   Minimum: 200

   Typisk: 300

   Maksimum: 500