Utvikling av system for tilstandsbasert overvåking og driftstøtte

Hovedoppgave
ved
sivilingeniørutdanning i
informasjons- og kommunikasjonsteknologi

av

Jonas Eriksson

Grimstad mai, 2002

SAMMENDRAG

I dagens industri er behovet stort for å produsere så effektivitet som mulig. Dette medfører at bedriften har behov for kostnads optimal drift, mest mulige oppetid og minst mulig vedlikeholdskostnader på sitt prosessutstyr. Den beste måten å møte dette behov på er ofte å implementere tilstandsbasert overvåking. Ved å implementere tilstandsbasert overvåking vil man oppnå en rekke fordeler som

- økt oppetid for utstyr i prosessen
- forbedret pålitelighet og sikkerhet
- redusert sannsynlighet for havarier
- bedre planleggningsmuligheter

På markedet finnes det i dag en rekke forskjellige, ofte svært omfattende og dyre, systemer for tilstandsovervåking. De baserer seg på tunge prosessmodeller med stor nøyaktighet når alle parametre og driftsforhold er kjent. Den gode nøyaktigheten er det som regel ikke bruk for, men en enkel indikasjon er ofte nok til tilstandsbasert vedlikehold. Dette gjør at det for kunden ikke eksisterer noen kost/nytte verdi å implementere tunge løsninger til vedlikeholdsformål.

Med dette som utgangspunkt ønsket CARDIAC å se på muligheten for å lage et system for enkel og robust tilstandsovervåking, sammen med et brukervennlig driftsstøtteverktøy som skal implementeres på IMATIS (Integrert Modulbasert Administrativ Teknisk System) plattformen.

Systemet som er utviklet er et rammeverk der man lett kan implementere nye tilstandsbaserte modeller for ulike typer utstyr. Det består av to deler der den første delen tar seg av beregninger av nye utgangsverdier som er basert på konstanter og andre variable utgangsverdier i en prosess. Man vil nå kunne få utgangsverdier som bedre egner seg for tilstandsovervåking.

Til den første delen av systemet er det laget Active Server Pages (ASP) websider hvor brukeren kan velge mellom forskjellig typer av utstyr som skal parameteriseres. All data tilknyttet den nye variable utgangen lagres så i en Microsoft SQL server database. I fra denne hentes alle parametre som skal brukes for å beregne den nye utgangsverdien i en LabVIEW applikasjon. Den nye verdien lagres så i en ny Citadel Historikk database hvor brukeren kan se på trender og historikk fra IMATIS Observer som er en del av CARDIACs IMATIS plattform.

I del to er det laget ASP sider for registrering av de utgangsverdier som skal knyttes mot vedlikehold. Til denne vedlikeholdsverdien knyttes det forskjellige parametre som forenkler beslutninger om vedlikehold. Det er også parameter for hvilken type handling som skal utføres ved en alarm. Alle utgangsverdier som er knyttet til vedlikehold blir kontinuerlig sjekket i en LabVIEW applikasjon for alarm status. Det er nå mulig for brukeren å ta frem ulike statusrapporter over prosessutstyrets vedlikeholdsbehov.

Systemet er implementert og testet ut på IMATIS plattformen med gode resultater.

FORORD

Denne rapporten er et resultat av en 10-vektalls hovedoppgave i faget IKT6400 ved sivilingeniørstudiet i informasjon og kommunikasjons teknologi (IKT) ved Høgskolen i Agder, avdeling Grimstad.

Hovedoppgaven er gitt av CARDIAC i Porsgrunn der også meste parten av arbeidet er utført. CARDIAC er en teknologibedrift innefor industriell og medisinsk IT med en målsetting om å være brobygger mellom tekniske og administrative systemer.

Rapporten tar for seg spesifisering, design og utvikling av et generelt vedlikholdssystem for å kunne håndtere tilstandbasert vedlikehold av prosessutstyr.

Det kreves basiskunnskaper i LabVIEW, Active Server Pages (ASP) programmering og databasehåndtering for å få full forståelse av rapporten og en mer utdypende kunnskap for full forståelse av vedleggene. Litteraturhenvisninger er gjort ved at nummer i litteraturlisten er satt i klammeparantes [XX].

Jeg vil gjerne rette en takk til alle ved CARDIAC for all hjelpsomhet. Spesielt vil jeg takke mine veiledere, sivilingeniørene Sigurd J Juvik, Lasse Klovning og Marius Kvitnes for ideer og teknisk bistand.

Grimstad, 27. mai 2002.

Jonas Eriksson

INNHOLDSFORTEGNELSE

1 INNLEDNING

Bakgrunn for denne oppgaven er at det i dagens industri er et stort behov for å produsere så effektivt som mulig. Dette oppnås ved å ha lave drifts- og vedlikeholdskostnader. For å møte dette behovet har det kommet en rekke såkalte tilstandsbaserte vedlikeholdssystem på markedet. Det viser seg at disse systemene er svært omfattende og dyre. De baserer seg ofte på tunge prosessmodeller som er lite robuste og med en nøyaktighet som det normalt ikke er bruk for. I mange tilfeller er enkle indikasjoner ofte nok til tilstandsbasert vedlikehold. Dette gjør at det for bedriften ikke eksisterer noen kost/nytte verdi å implementere tradisjonelle løsninger til vedlikeholdsformål.

Med dette som utgangspunkt ønsker CARDIAC å se på muligheter for å lage et enkelt system for vedlikehold som baserer seg på tilstandsovervåking. Systemet skal være et rammeverk der man lett kan implementere nye tilstandsbaserte modeller. Det skal kunne integreres og være en del av CARDIAC's plattform IMATIS (Integrert Modulbasert Administrativ Teknisk System).

Mål med oppgaven

Hovedmålene med oppgaven er å "spesifisere, designe og utvikle deler av et generelt driftstøttesystem for enkelt å kunne håndtere tilstandsbasert vedlikehold av prosessutstyr"[1].

Oppgaven skal innholde følgende momenter[1]:

1. Litteraturstudie for å se hva slags funksjonalitet som er implementert i andre mer tradisjonelle prosess-simuleringssystemer. Aktuelle systemer kan være Emerson AMS, Honywell Alert Manger, AspenTech InfoPlus, Kongsberg Simrad ASSET og MP2.
2. Sette opp en kravspesifikasjon til et drifsstøttesystem for tilstandsbasert vedlikehold.
3. Design og implementasjon av database og eksekveringssystem for å kjøre modeller med sanntidsmålinger tilgjengelig fra IMATIS® PIMS som input og output til/fra modellene.
4. Design og implementasjon av verktøy for enkelt å kunne registrere beregningsmodeller for ulike typer utstyr, eksempelvis varmevekslere, pumper og ventiler.
5. Design og implementasjon av verktøy for enkelt å kunne opprette beregningsobjekter basert på innregistrerte beregningsmodeller og koble dem mot målinger/tag fra IMATIS® PIMS. For de målinger/tag som brukes til tilstandsbasert vedlikehold, må en utover de tagegenskaper som finnes i dag, kunne definere egenskaper som kritikalitet, kostnader ved feiltilstand osv i tillegg til hendelse som definerer når tilstanden er slik at vedlikehold bør utføres på et utstyr.
6. Implementere et par enkle modeller for utstyr til uttesting av systemet.
7. Implementere noen enkle Web rapporter, der bruker kan liste ut f.eks. vedlikeholdslister eller alarmlister, sortert etter status, kritikaliteter, navn, dato, osv.

Omprioriteringer og avgrensninger

Det ble tidlig klart at oppgaven med å lage et verktøy for innregistrering av beregningsmodeller ikke skal prioriteres, punkt 4 over. Dette siden et slikt verktøy ikke i første omgang er så viktig for en brukere. Det ble heller valgt å prioritere rapporteringsmuligheter og integrering med IMATIS. Derfor blir beregningsmodeller hardkodet inn i en database.

Det vil heller ikke bli lagt noen større vekt på å lage sikkerhet rundt websider som for eksempel hindrer brukeren fra å taste inn feil format.

Alle omprioriteringer og avgrensninger er gjort i fellesskap med veiledere på CARDIAC.

Oppbygning av rapport

Kapittel 2 (Forstudie) skal gi en forståelse av hva et system (rammeverk) som skal ta seg av tilstandbasert vedlikehold bør ha av funksjoner og muligheter.

Kapittel 3 (Kravspesifikasjonen) er en konkretisering av systemets funksonaliteter.

Kapittel 4 (Teknologier) gir en innblikk i alle de forskjellige teknologier som blir brukt gjennom prosjektet.

Kapittel 5 (Design & Implementasjon) beskriver først overordnet hvordan systemet er designet for så å beskrive de tekniske løsningene. Det er ikke tatt med noen kode i rapporten. Applikasjoner er isteden beskrevet med flytskjema. Websider som er laget og implementert i IMATIS portalen vises det for å spare plass bare utsnitt av.

Kapittel 6 (Uttesting av noen enkle modeller) beskriver et scenario der systemet testes fra begynnelse til slutt.

Kapittel 7 (Videre arbeid) peker på eventuelle svakheter og ting det kan jobbes videre med.

Kapittel 8 (Konklusjon) trekkes konklusjoner av arbeidet som er utført.

2 FORSTUDIE

Hensikten med dette forstudie er å definere hva tilstandsovervåking og vedlikehold innebærer. Deretter blir det sett på hvilke funksjoner et generelt driftstøttesystem bør ha. Det gjøres ved å se på hvilke funksjoner og muligheter andre vedlikeholdssystem har innebygget. Dette vil så ende opp i en kravspesifikasjon i kapittel 3.

2.1 Vedlikehold

Med vedlikehold menes alt det som blir gjort for å holde utstyr i en spesiell tilstand eller for å føre det tilbake til en spesiell tilstand. Det er flere forskjellige metoder å planlegge og utføre vedlikehold på. Her følger en kort beskrivelse med fordeler og ulemper av fire ulike typer av vedlikehold [1][12]:

Sammenbruddbasert vedlikehold

Dette er den enkleste formen for vedlikehold. Den lar utstyr kjøres til et sammenbrudd inntreffer.

Fordeler: effektiv for enkle reparasjoner og trenger ingen planlegging.

Ulemper: forstyrrer produksjon og kan få uakseptable konsekvenser.

Konstruksjonsforbedrende vedlikehold

Denne typen vedlikehold innebærer at ytelsen/holdbarhet forbedres ved at man redesigner utstyr ved feil, enten gjennom materialbytte eller konstruksjons forandring.

Fordeler: man vil finne en endelig god løsning og uakseptable feil kan elimineres.

Ulemper: kan ta lang tid før endelig løsning kommer på plass og kan bli kostbart.

Periodisk vedlikehold

Denne typen vedlikehold krever at ettersyn og vedlikehold blir utført i faste planlagte intervaller uten hensyn til utstyrets tilstanden.

Fordeler: effektiv metode når tilstanden til utstyr er direkte relatert til tid. Man får også en veldig strukturert og lett organisert vedlikeholdsplan.

Ulemper: ikke effektiv mot feil som oppstår uregelmessig, bruker mye tid til å stoppe utstyr som skal vedlikeholdes og fremfor alt maksimerer ikke levetiden til utstyr.

Tilstandsovervåking og vedlikehold

Tilstandsovervåking baserer seg på registrering av målinger for å kunne forutsi tilstanden til produksjonsutstyr. Utfra disse resultatene vil man så utføre nødvendig vedlikehold.

Fordeler:

- *Øker oppetiden på prosessutstyr*. Dette siden det bare utføres vedlikehold når det er nødvendig og antall uforutsette havarier reduseres betraktelig.
- *Reduserer sannsynlighet for havari*. Man kan forutsi alvorlige feil og utføre vedlikehold før det er for sent.
- *Forbedre pålitelighet og sikkerhet*. Det er mulig å erstatte utstyr før farlige situasjoner oppstår. I tillegg er det ikke nødvendig med så mange rutinemessige utskiftninger av deler som i sin tur kan introdusere feil.

- *Bedre planleggingsmuligheter*. Dette siden man tidligere ser indikasjoner på feil og kan planlegge vedlikehold i god tid. Man trenger heller ikke avbryte planlagt arbeid pga uforutsette hendelser.

Ulemper:

- Gevinsten kan være marginal slik at kost/nytte verdien blir liten.
- Problematisk organisering og oppstart av system.
- Problematisk å integrere i eksisterende vedlikeholdssystem.

2.2 Tilstandsovervåking

Siden denne oppgaven omhandler tilstandsovervåking er det vurdert kriterier rundt dette tema.

Et system for tilstandsovervåkings bør ha følgende egenskaper [1]:

- Et klart forhold mellom målinger og tilstanden til utstyret.
- Overvåkingssystemet må ha rask nok respons slik at ikke utstyret tar skade før passende vedlikehold blir iverksatt.
- Vurdering av tilstanden til utstyr må være gjort ved å sammenlikne avlesninger med eksisterende målinger og/eller fördefinerte og absolutte standarder.
- Fordelene med og utføre tilstandsovervåking må oppveie implementasjons- og løpende kostnader.
- Et system for måling og datainnsamling må eksistere for å kunne forutsi tilstanden til utstyr.

En viktig faktor i tilstandsovervåking er samplingsfrekvensen som bør være minst 4 ganger raskere en den tiden det tar for en feil å utvikle seg, ettersom det er nødvendig med minst 4 til 5 avlesninger for å få en klar trend.

Når tilstandsovervåking skal etableres er det viktig å ta hensyn til følgende:

- Identifisere utstyr som har behov for tilstandsovervåking
- Utrede hvordan utstyret kan svikte.
- Finne årsak, effekt og konsekvens av svikten
- Velge passende tilstandsovervåkingsteknikk
- Bestemme hvor ofte avlesninger skal gjøres.
- Sette utgangsverdier og alarmgrenser.

Videre er det tre måter å vurdere de avleste verdiene på:

1. nivåsjekking
2. trend i forhold til tid
3. sammenligning med andre data

Det er forskjellige teknikker som kan tas i bruk når et tilstandsovervåkingssystem skal implementeres. Alt ettersom hvilken type utstyr som skal overvåkes. De mest vanlige teknikkene som brukes er:

- vibrasjons analyse

- olje/avfalls analyse
- manuell inspeksjon
- elektrisk strømovervåking
- ledningsevne testing
- ytelseovervåking
- termisk overvåking
- korrosjonsovervåking

2.3 Andre vedlikehold og driftstøttesystem

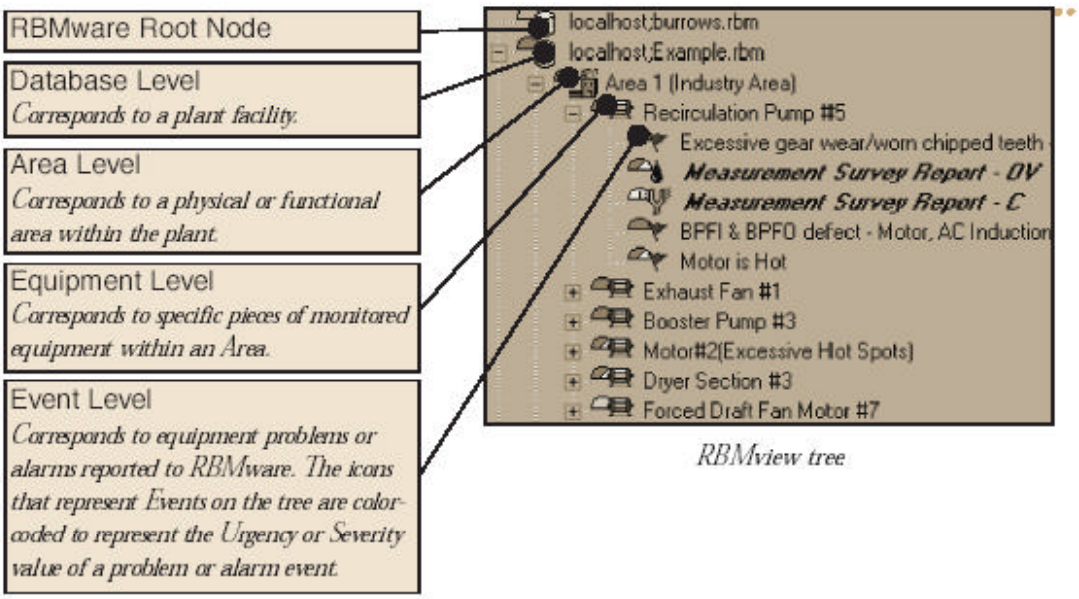
For å få en oversikt over hvilke funksjoner et tilstandsovervåkingssystem bør ha er det gjort en studie av forskjellige vedlikeholdssystemer. Det er i hovedsak brukt Internett for å innhente informasjonen, se vedlegg 6.

2.3.1 AMS, Emerson

AMS/RBMware Software Interface AMS (Asset Management Solutions), fra Emerson, er et sett med software (snap-on) løsninger for å samle alle vedlikeholdsaktiviteter i en fabrikk. AMS gir online aksess til instrumenter i prosess og fanger automatisk opp vedlikeholds-informasjon. RBMware (Reliability Based Maintanance) software, fra CSI, er designet for å støtte overvåking av produksjonsutstyr (roterende og statisk), data behandling, analyser, rapportering av resultater fra predektivt vedlikehold. Til sammen gir disse et grensesnitt mot data fra roterende utstyr og prosess instrumentering. Noen av mulighetene som gis med dette systemet er[A]:

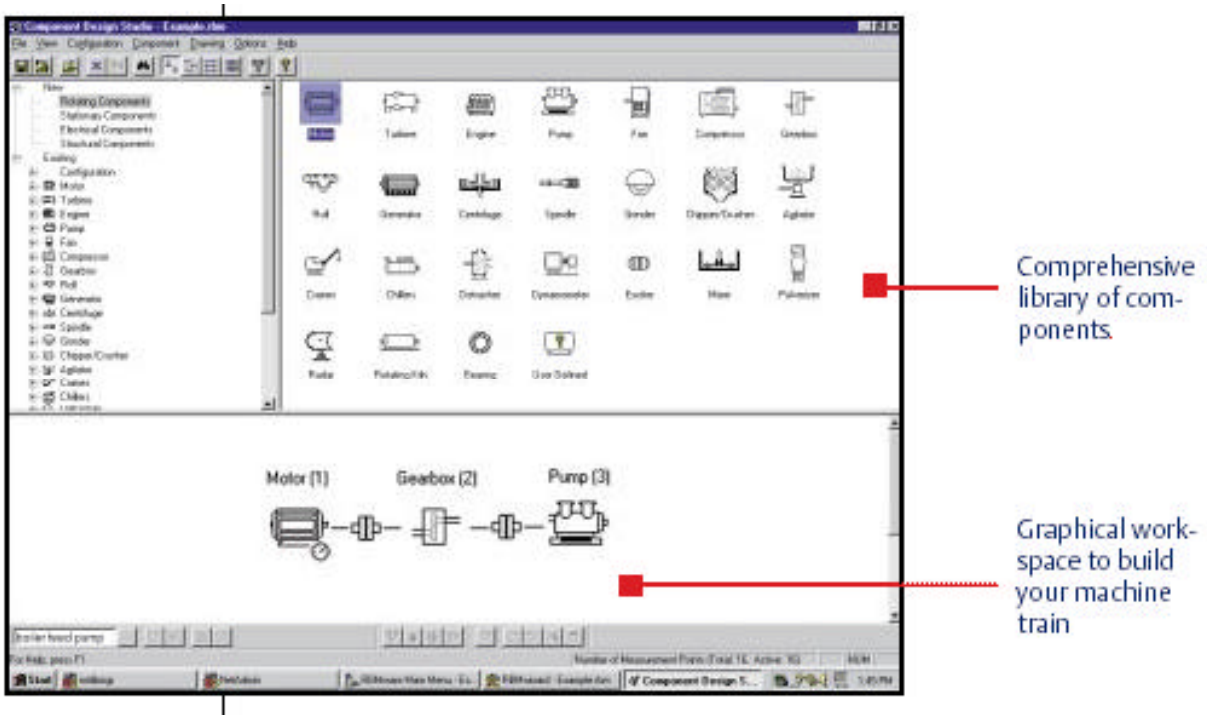
- Verktøy for tilstandsovervåking. Med mulighet for overvåking som bygger på ulike frekvensanalyser, oljeanalyser, infrarøde målinger og bilde analyser, ultralyd og frekvensanalyse av motorers forsyningsstrøm.
- Rapporterings- og dokumentasjonsverktøy for og lage kostanalyser, historikk og service etterspørsel.
- Verktøy for å kommunisere med andre ”plant” dataverktøy.

Alle instrumenter som er registrert blir plassert i en trestruktur som begynner med fabrikksavsnitt, utstyrsnivå og til slutt hendelse nivå. Forskjellige farger på ikoner indikerer problem, alarm eller feil på et prosessutstyr. Ved å klikke på et ikon får man opp mer utfyllende informasjon, se Figur 1.



Figur 1 RBMware instrument trestruktur[A].

For lett å kunne registrere nye prosessutstyr i systemet brukes en RBMWizard. Her kan man plukke det utstyr man skal sette inn fra et grafisk bibliotek. Wizarden lager automatisk passende målepunkter, parameterisering og alarm grenser, se Figur 2.

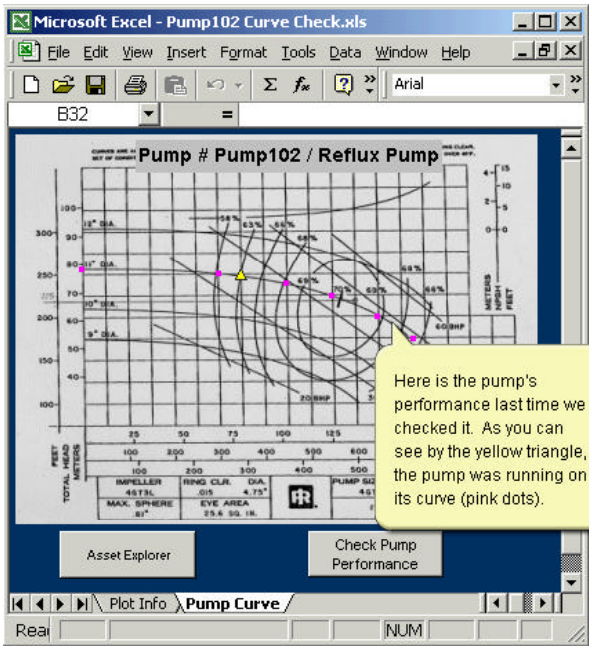


Figur 2 RBMWizards grafiske bibliotek over komponenter som kan registreres[A].

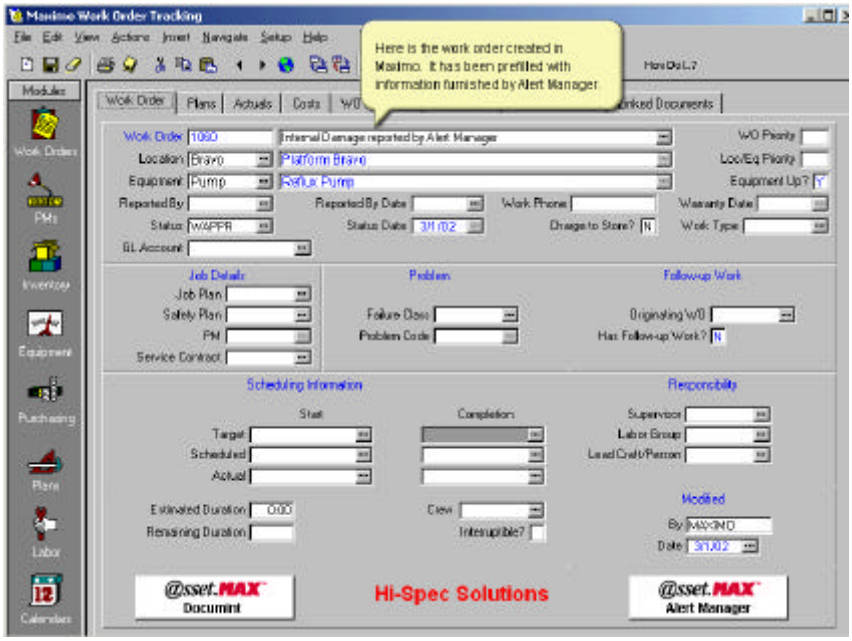
2.3.2 Alert Manger, Honywell

Dette er et omfattende vedlikeholdssystem med mange forskjellige funksjonaliteter tilgjengelige for brukeren. De funksjoner som beskrives på deres internettsider og i en egen pump demo er som følger[B]:

- Samle og organisere data fra en mengde kilder ved hjelp av blant annet OPC (OLE for Process Control, beskrives i kap. 4.5 OPC).
- Monitorering av måleinstrumenter, ventiler, varmeveksler, kompressor, pumper og smøre- og tetningsoljesystem.
- Avanserte analyser av dataen for så å identifisere symptomer og feil på prosessutstyr.
- Ved en feil identifikasjon kan en alarm settes igang, vedlikeholdspersonell automatisk tilkalles, mail sendes med spesifikke tiltaksforslag og arbeidsordre genereres.



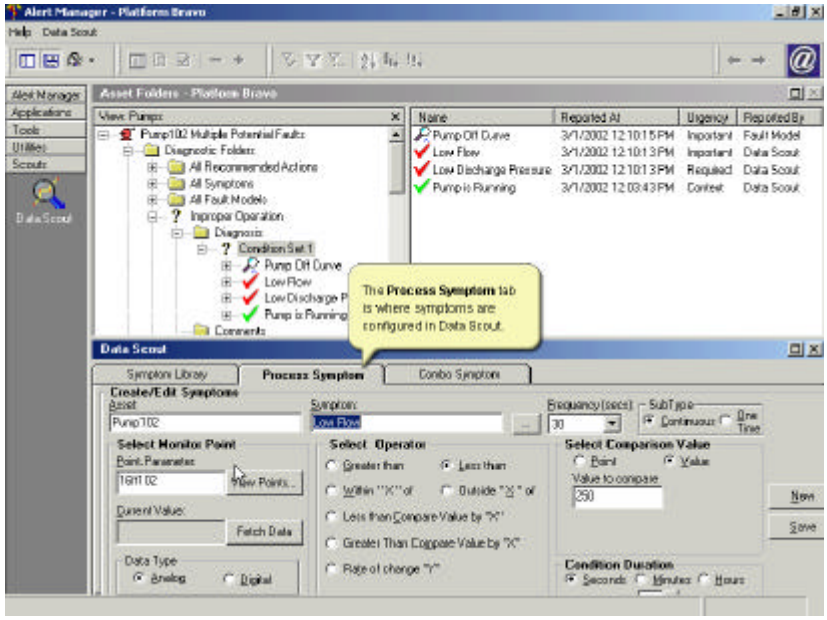
Figur 3 Her er en pumpes tilstand plottet i pumpediagram lagt inn i Excel[B].



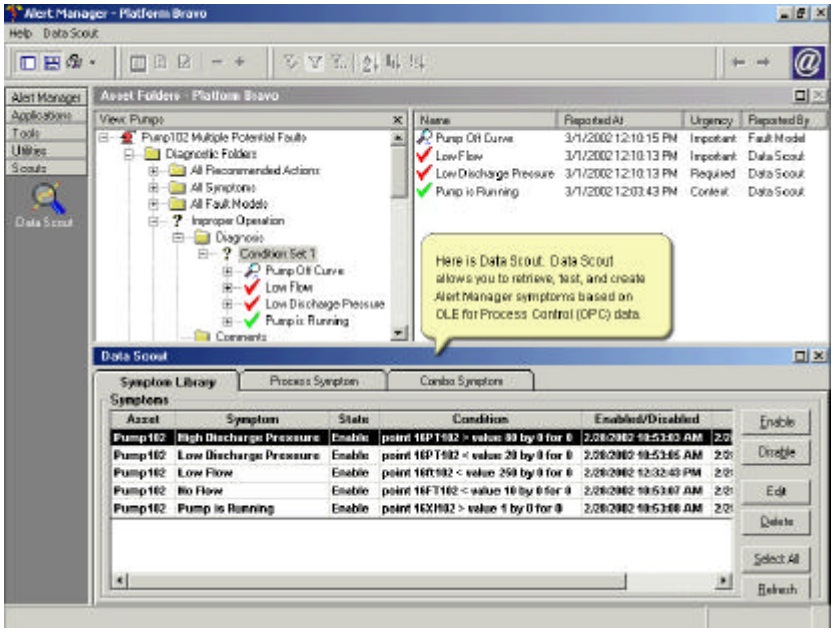
Figur 4 Her er det generert arbeidsordre med utgangspunkt i en feil tilstand på en pumpe[B].

Alert Manager innbefatter tre hoved software moduler:

1. ”Asset Builder” som er et verktøy for å opprette nye typer av utstyr med selvdefinerte egenskaper.
2. ”Asset Folders” gjør informasjon om en spesifikk type tilgjengelige for brukerne. All informasjon som omhandler en type kan nås fra et sted. Informasjonen kan f.eks være test- og inspeksjonshistorikk, tegninger, prosedyrer, arbeidsordre historikk, reservedeler, arbeidsstatus. Det lagres også symptomer og feil, tiltaksforslag og relaterte feilrapporter.
3. ”Diagnostic builder” med mulighet for å spesifisere og aktivisere varsling av symptomer og feil for utstyr. Evaluering av symptomer og reparasjons aktiviteter kan også defineres.



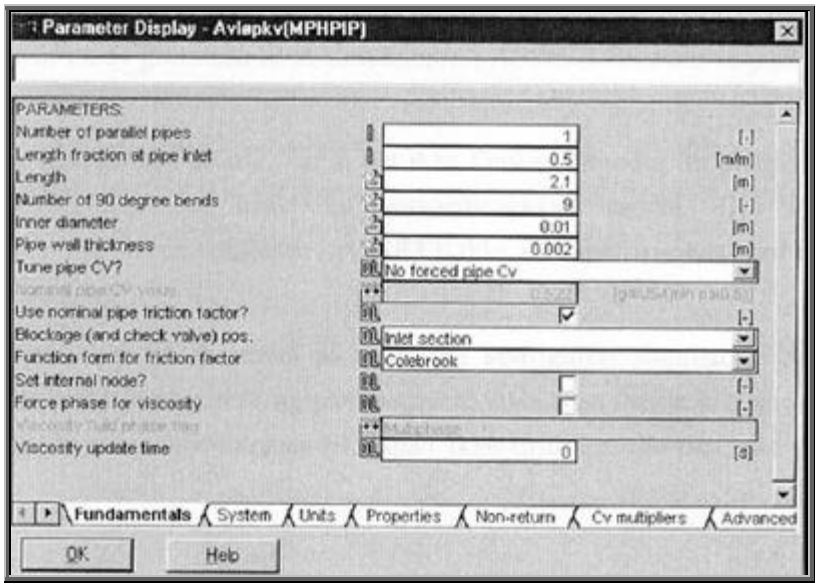
Figur 5 Det er en egen dialog som brukes for å konfigurere alarmer til et prosessutstyr[B].



Figur 6 Her hentes verdier opp fra OPC for så å testes og leses i et ”data scout” vindu[B].

2.3.3 ASSET, Kongsberg Simrad

ASSETT (Advanced Simulation, Studies, Engineering & Training Tool) er et simuleringsverktøy der forskjellige moduler er byggesteinene. Modulene kan være for eksempel tankmoduler, rørmoduler eller pumpmoduler. Først settes alle moduler som inngår i prosessen sammen deretter skal de paramertriseres. Parametriseringen er ganske omfattende for å få så nøyaktig simulering som mulig. Eksempel på parametre kan være rørlengder, rørtykkelse, høydeforskjeller, trykk og temperaturer [10].



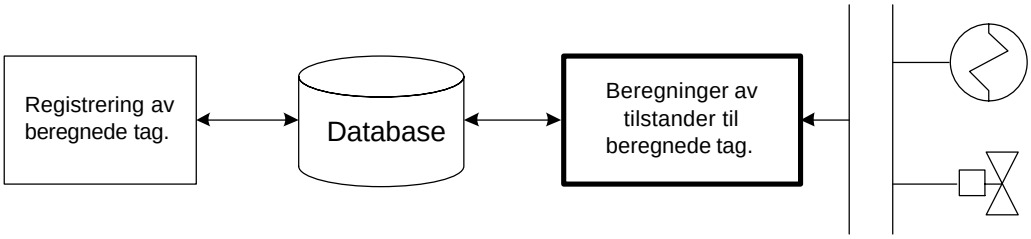
Figur 7 Parametrisering av rørlengder i ASSET[10].

2.4 Resultat forstudie

Systemene det er sett på er ganske omfattende hver på sitt område. Asset fra Kongsberg Simrad er et ren odlet simuleringshjelpemiddel mens de andre bygger på vedlikehold og administrering av det. Det som ønskes i vårt tilfelle er fremfor alt et enkelt system hvor brukeren lett kan definere hva som er vedlikehold og så få ut tilgjengelig data lettvint. Samtidig som fleksibiliteten må være tilstede fremfor alt når det gjelder hvordan og hva som skal beregnes for å kunne gi en vurdering om vedlikeholdsbehov. Med dette som grunnlag ble det valgt å lage et system bestående av to deler, heretter navngitt som del I beregnede tag og del II vedlikeholdstag.

Del I, beregnede tag

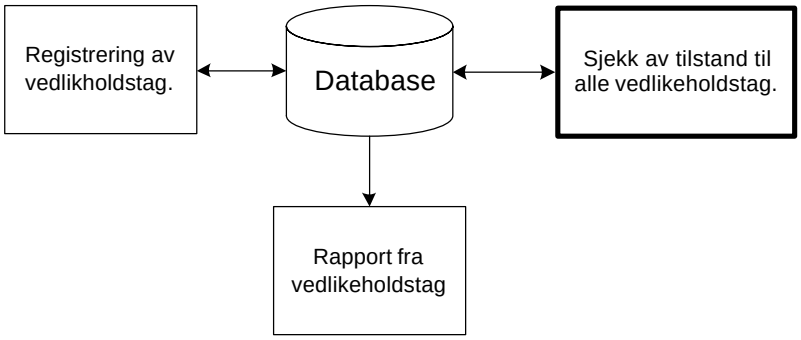
Dette skal være den delen som tar seg av registrering av nye vedlikeholds objekter. Nye vedlikeholds objekter skal basere seg på eksisterende verdier fra I/O i et prosessavsnitt i tillegg til nye definerte verdier for å kunne beregne nye vedlikeholdsparametrer. Disse nye parametrene som utledes av likninger skal selvfølgelig ha en betydning i forhold til vedlikehold. Når man prater om verdier (eg. signaler) som kommer ifra I/O i prosessen er de betegnet med en såkalt fysisk ”tag id”. Denne tag id er koblet til et spesifikt fysisk signal, og det som gjøres i del I er å bruke en eller flere av disse for å beregne en ny verdi. Denne nye verdien blir så koblet til hva som videre i rapporten benevnes som en ”beregnet tag”.



Figur 8 Skisse over elementer i del I.

Del II, vedlikeholdstag

Denne delen skal være der brukeren bestemer hvilke tag, beregnede fra del I eller eksisterende tag, som skal defineres som vedlikeholds tag. Her skal i prinsippet alle tag kunne brukes, men en forutsetning vil være at det til tagen er registrert alarmer med grenseverdier. Når en vedlikeholdstag er registrert vil bakomliggende applikasjoner kontrollere at man er innenfor satte grenseverdier. Hvis ikke skal det registreres i en spesiell vedlikeholds database/tabell slik at brukeren lett kan få opp sanntids eller historisk informasjon om sine vedlikeholdstag.



Figur 9 Skisse over elementer i del II.

Etter et forslag fra Eivind Bjørge Sandsmark, sivilingeniør ved Statoil, skal det til hver vedlikeholdstag knyttes forskjellige kritiske parametrer. Disse skal i første omgang være basert på stopp, kostnad, sikkerhet og miljø. Avhengig av viktigheten settes en verdi mellom 0 og 10 (10 er mest kritisk). Disse kritiske parametrene, fra nå av benevnt ”kritikaliteter”, kan så brukes ved planlegging av vedlikehold for eksempel hvor man ønsker å se på de med mest kritiske verdiene med hensyn på sikkerhet.

3 KRAVSPESIFIKASJON

Kravspesifikasjonen er utarbeidet i samarbeid med CARDIAC. Det overordnede kravet er at det skal lages et driftstøttesystem for enkelt å kunne håndtere tilstandsbasert vedlikehold av prosessutstyr.

Systemet skal ha følgende funksjonalitet og innehold:

Del I, beregnede tag

1. Det skal være egne databasetabeller for håndtering av beregnede tag.
    a. Det skal opprettes et type-bibliotek med ferdig definerte typer. En type kan for eksempel være en væske ventil, varmeveksler eller lignende.
    b. Det skal være mulig å knytte forskjellige beregningsmetoder til en type.
    c. Utfra hver type skal en brukere kunne lage ubegrenset med objekter som alle har sine egne parametrer knyttet til seg.
2. Det skal lages websider for å opprette nye og editere eksisterende beregnede tag.
3. Det skal lages en hovedapplikasjon som kan starte beregninger til de forskjellige objektene i databasen.
4. Det skal lages en applikasjon for hver type som kan gjøre beregninger til den typens objekter.
5. Det skal lages en enkel tagvalue generator for å produsere verdier inn på de variable inngangene ved uttesting.

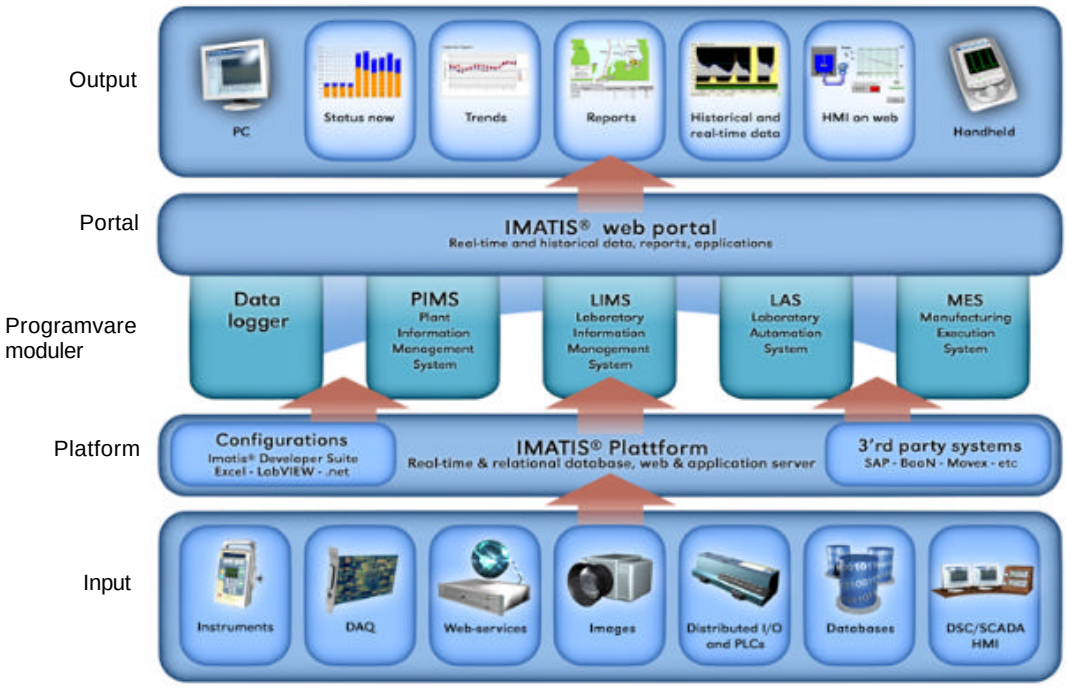
Del II, vedlikeholdstag

6. Det skal lages egne databasetabeller for håndtering av vedlikeholdstag og tilhørende historikk.
7. Det skal lages websider der brukeren kan opprette, endre og slette vedlikeholdstag med utgangspunkt i eksisterende tagliste. Til en vedlikeholdstag skal det det være knyttet forskjellige kritikaliteter med en skala fra 0-10.
8. Det skal lages en applikasjon som kontinuerlig sjekker status på alle vedlikeholdstag og skriver til en vedlikeholds historikkdatabase ved en alarm.
9. Det skal lages websider med mulighet for rapportering av sanntids- eller historiske data fra en vedlikeholdstag.

4 TEKNOLOGIER

For å kunne realisere de forskjellig momentene i kravspesifikasjonen må det brukes teknologier som kan implementeres i CARDIACs plattform. Dette kapittel vil derfor gi en kort introduksjon til teknologier som brukes gjennom prosjektet.

4.1 IMATIS plattform



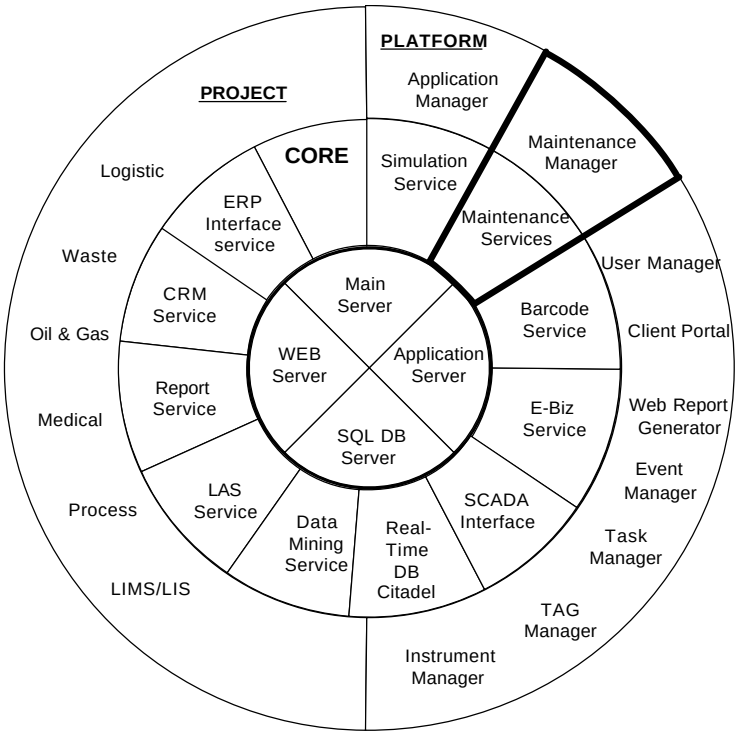
Figur 10 IMATIS platformstruktur.

IMATIS er en applikasjonsplattform som integrerer administrativ datasystem og prosess kommunikasjon. IMATIS består av en plattform med blokker av programvare moduler og en standardisert teknologi for å koble dem sammen med andre systemer [4].IMATIS er tilpasset forskjellige bransjer fra den farmasøytiske til olje- og gassindustrien med moduler som PIMS (Plant Information Managment System), LIMS (Laboratory Information Managment System), og LAS (Laboratory Automation System).

- PIMS er ment og være en brobygger mellom prosesskontrollsystemer og administrasjonensforretningssystemer (ERP) hvor data overføres til personell med riktig tilgang og presenteres i en web-browser. Man kan hente ut både sanntids- og historisk data om produktkvalitet, prosessverdier, ressursbruk og andre nøkkeltall.
- LIMS er fokusert på registrering og samordning av analyser utført på laboratorier. I tillegg finnes funksjoner for å administrere laboratorieinstrumenter, enheter, metoder, analysepakker, historie og kalibrering, samt resultatanalyse og kvalitetskontroll.
- LAS brukes for å automatisere datafangst i laboratorier fra instrumenter og IO enheter. Hensikten med dette er å effektivisere prosedyrer for å starte analyser av prøver, kontrollere/verifisere analyser og legge dem inn i et hvilket som helst LIMS system.

Figur 11 IMATIS webportal med applikasjoner

IMATIS portalen er hovedapplikasjonen i systemet. Den brukes til å starte moduler, se på rapporter, browse tag, browse Internett. Uten noen moduler koblet til portalen vil den være en standard web browser. Alle brukere må anvende en egen login når portalen startes. Dette for å kunne styre hvilke moduler, avhengig av behørighetsgrad, en bruker kan starte. Trestrukturen bygges opp hver gang en bruker starter portalen med de moduler en administrator har gitt tilgang til. Noen av disse modulene brukes i prosjektet og beskrives nærmere i kap. 5.2, IMATIS moduler.



Figur 12 Arkitektur av IMATIS plattformen med hvor det markerte området indikerer vedlikeholdsdelen.

I Figur 12 er det tegnet en overordnet struktur over inneholdet i IMATIS plattformen. Den består av noen kjernetjenester (Core, de to innerste ringene) som brukes til etablering av forbindelser i systemet, gjenopprette applikasjoner, utføre oppgaver og å øke effektiviteten i dataflyten. I den ytterste ringen til høyre ligger såkalte system applikasjoner som tar seg av administrasjon og oppsyn av systemet. De fete linjene rundt ”Maintenance” indikerer hvor vedlikeholdsdelen befinner seg i systemet.

4.2 Active Server Pages (ASP)

For å lage websider som klarer å lese og skrive fra en database må det benyttes Active Server Pages fra Microsoft (ASP) koding i tillegg til den vanlige HTML koden. ASP koden gjør det mulig å utføre operasjoner på serversiden. Ved og installere IIS (Internett Information Server) lokalt på egen maskin kan man teste og se resultater under utvikling av websider. Det finnes andre løsninger som kunne gjort jobben like godt som ASP, men valget falt på ASP siden CARDIAC bruker det i andre løsninger[9].

4.3 Databaser

Det brukes to typer databaser en som er laget i Microsoft SQL Server Enterprise Manager og en Citadel Historical database. I Microsoft SQL databasen ligger alle IMATIS tabeller som brukes for å kjøre applikasjonene. For å klare håndtering av store data mengder fra prosessen fort nok benyttes en sanntidsdatabase. Etter det blir verdiene permanent lagret i Citadel historikk databasen[7].

4.4 LabVIEW

LabVIEW er et grafisk programmeringsspråk som bruker ikoner isteden for vanlige linjer til å bygge opp en applikasjon. I motsetting til andre programmeringsspråk hvor instruksjoner bestemmer eksekveringen bestemmes den her av dataflyten.

LabVIEW applikasjoner kalles for ”virtual instruments” heretter betegnet som VI. En VI består av tre hoveddeler et frontpanel, et blokk diagram og et ikon med tilkoplingspanel. Disse lages ved hjelp av et sett med verktøy og objekter[5].

4.5 OPC

OPC (OLE for Process Control) er en industri standard utviklet av en mengd internasjonale selskaper innenfor hardware/software og automasjon i samarbeid med Microsoft. Organisasjonen som administrerer denne standarden er OPC Foundation. Det er over 270 medlemmer over hele verden der iblant CARDIAC AS.

Teknologien baserer seg på Microsoft’s Active X, COM (component object model) og DCOM (distributed component object model) teknologi. Active X/COM teknologien definerer en standard for hvordan individuelle software komponenter påvirker hverandre og deler data. OPC har et felles interface for kommunikasjon mellom diverse prosess-kontroll utstyr uavhengig av overliggende applikasjoner[C].

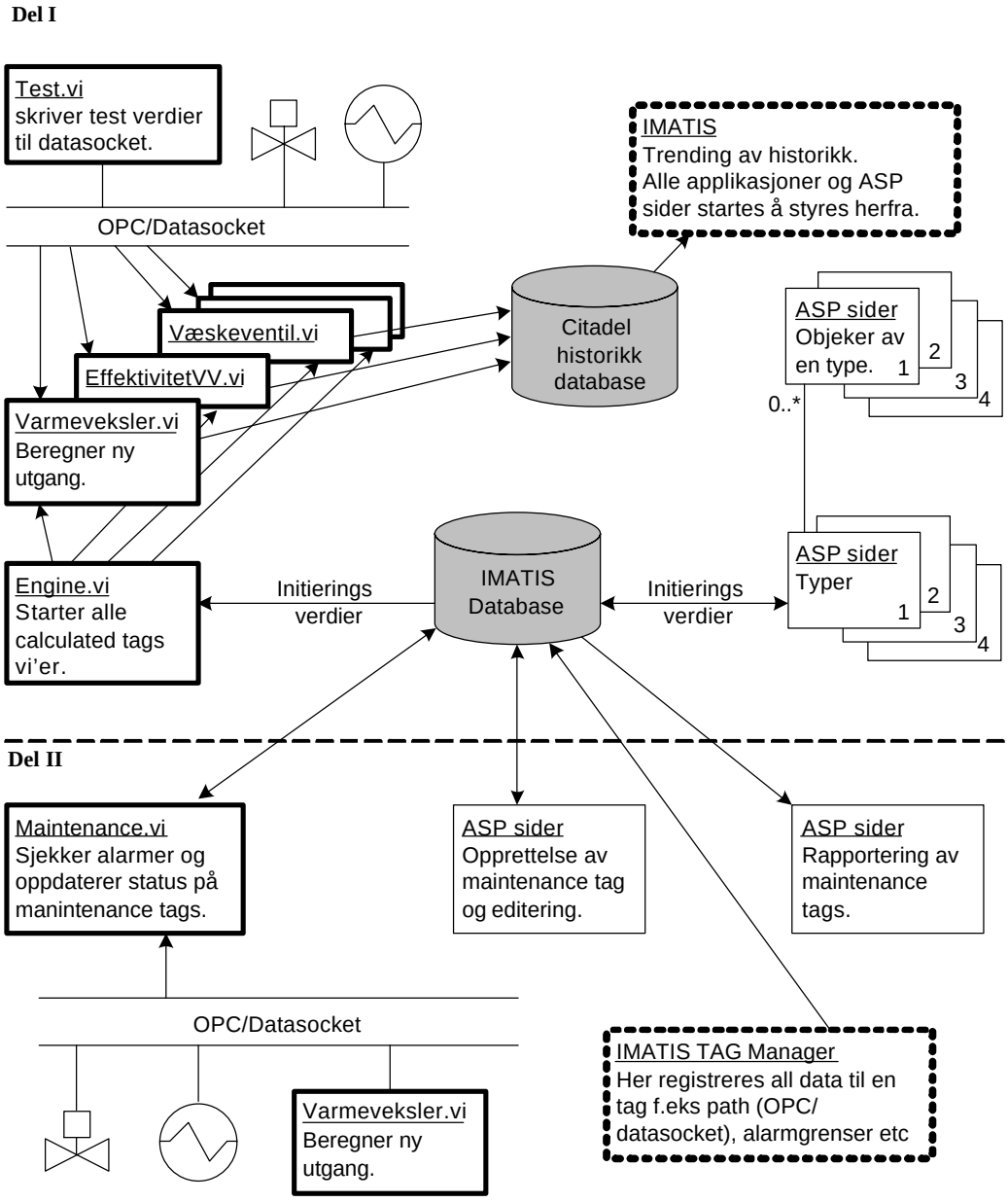
4.6 Datasocket

Datasocket er en teknologi som baserer seg på TCP/IP og forenkler ”live data” utveksling mellom ulike applikasjoner i en datamaskin eller mellom datamaskiner forbundet via nettverk. Teknologien består av to deler – Datasocket API og en Datasocket server. Datasocket API er et interface for kommunikasjon med multiple datatyper fra multiple dataprogrammeringsspråk. Den konverterer automatisk brukerens data til en ”stream” av bit som sendes over nettverket. Mottakaren konverterer automatisk datastreamen tilbake til sin opprinnelige form. Datasocket serveren forenkler Internett kommunikasjonen ved å ta seg av TCP/IP programmeringen. Selve adresseringen av dataen foregår på samme måte som web adressering (http://) med den forskjell at man setter ”dstp” foran slik at datasocketen åpner Data Socket Transfer Protokollen isteden. En adressering kan da se slik ut: dstp://labview/tagy [D].

5 DESIGN & IMPLEMENTASJON

5.1 Systemoversikt

Figur 13 viser hvordan de forskjellige applikasjonene og databasene er koblet sammen.



Figur 13 Systemoversikt

Del I, over den stiplede linjen er den delen som brukes til å opprette nye beregnede tag. Her er ASP siden vist med tynne rammer. De kommuniserer kun med IMATIS databasen å brukes der til å legge til og forandre verdier i vedlikeholdstabellene, dette beskrives nærmere i kap. 5.3.1. Boksene med fet stil er LabVIEW applikasjoner, VI'er, der den første er selve motoren (Engine.vi). Denne starter opp alle vedlikholdstag som er registrert fra ASP sidene. Dette gjør den ved å sende riktige parametre til respektive type-VI. Nå vil en type-VI, for eksempel

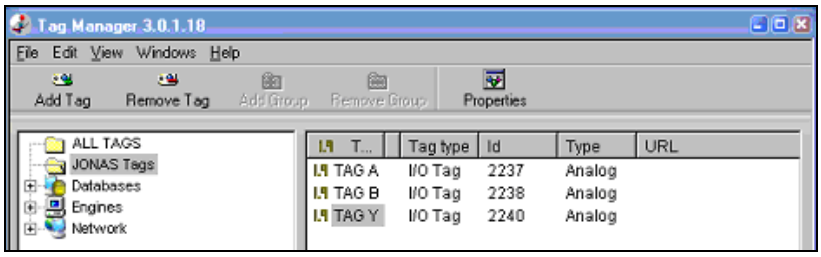
varmeveksler typen, utføre beregninger og sende resultat videre til Citadel Historikk databasen. Derifra kan verdiene hentes og vises ved hjelp av IMATIS portalen for så videre å analyseres.

Del II er den delen der vedlikeholdstag først registreres i en vedlikeholdstag tabell i IMATIS databasen. En forutsetning for at brukeren kan registrere en vedlikeholdstag er at alarmgrenser først blir lagt inn ved hjelp av IMATIS Tag Manager. Når vedlikeholds VI'en (Maintenance.vi) starter vil alle vedlikeholdstagene kontinuerlig sjekkes om det er noen alarm utløst. Hvis det er slik blir alarminformasjonen (se kap 5.4.1) lagret i historikk tabellen. Nå kan all historikk som er registrert fås opp på en ASP side. På denne siden finnes det forskjellige søke muligheter slik at brukeren kan se på ønskede prioriteringer. Man kan også se på alarmer som er aktive.

5.2 IMATIS moduler

En forutsetning i oppgaven var at noen av de eksisterende IMATIS modulene skulle brukes[4].

5.2.1 Tag Manager



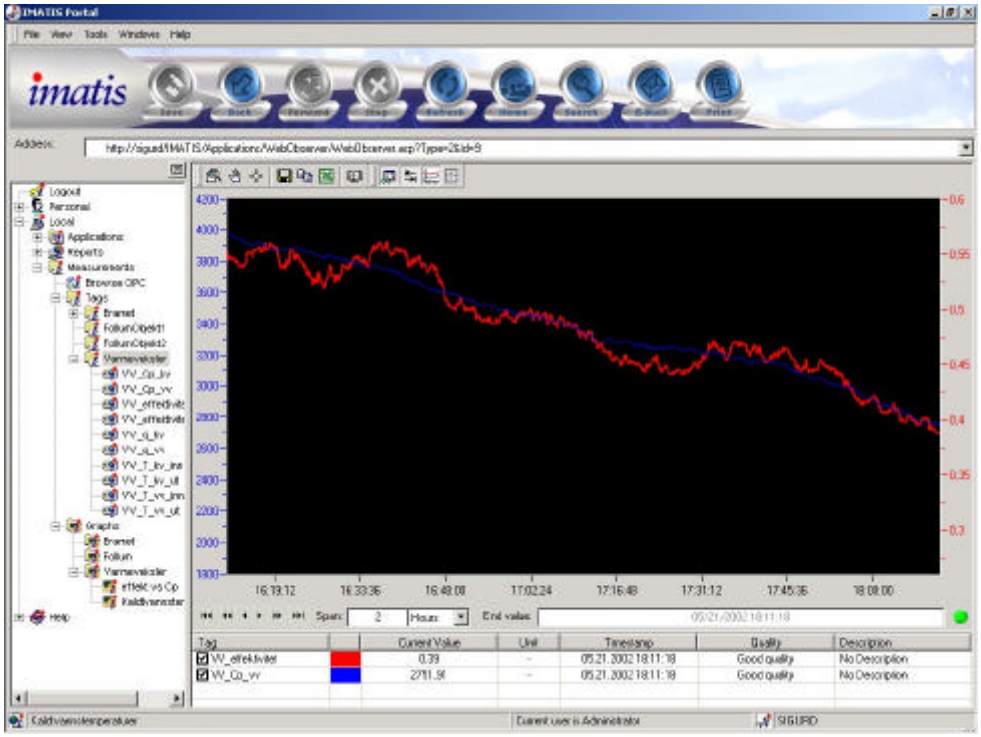
Figur 14 Tag Manager til IMATIS.

Alle tag i IMATIS registreres i Tag Manageren. Dette er en applikasjon for å konfigurere og administrere tag i IMATIS systemet. Hovedfunksjonene i Tag Manageren er:

- Legge til nye tag i systemet.
- Fjerne eksisterende tag.
- Konfigurere tag i systemet.
- Lage grupper av tag.

Når en tag konfigureres legges det blant annet inn alarmgrenser og adressering (OPC eller datasocket). Her "enables" også en alarm til en tag noe som har betydning for om den kan brukes til vedlikehold, dette forklares nærmere i kap 5.4.1.

5.2.2 WEB Observer



Figur 15 IMATIS Observer for å se på trend av en tag.

Med IMATIS Observeren hentes tag opp fra IMATIS systemet for å se på en trend. Noen av funksjonene som er tilgjengelige er:

- Se trend i sanntid for en tag gjennom OPC eller Datasocket.
- Vise historiske data fra en eller flere tag samtidig.
- Spare og hente in egne trend innstillinger.
- Eksportere til en fil.

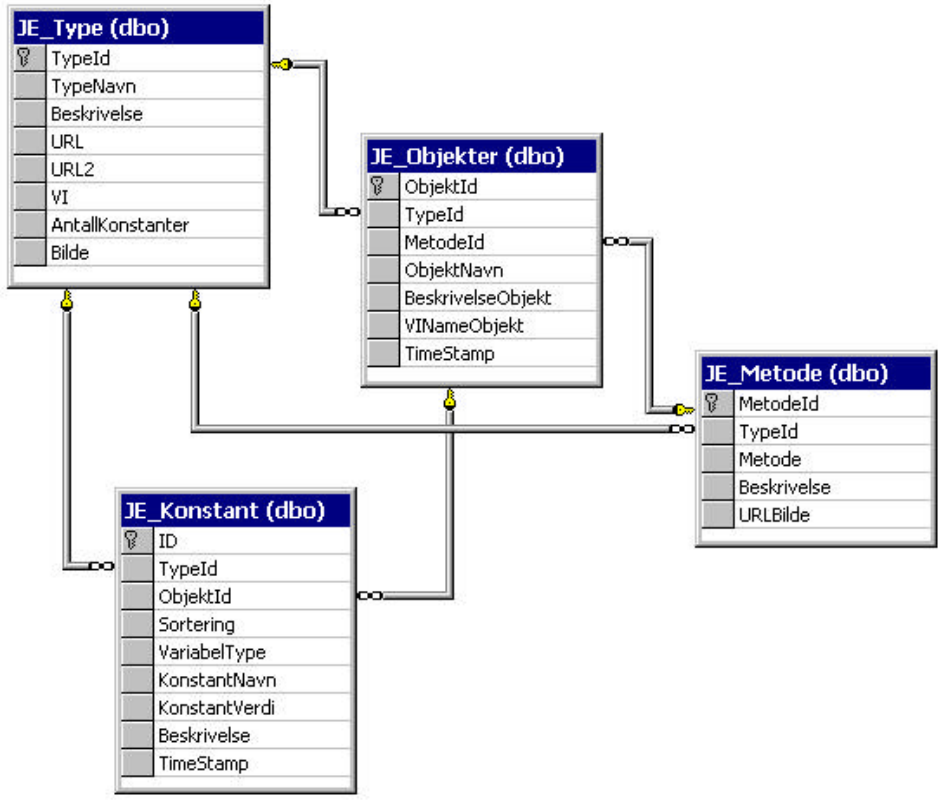
5.3 Del I, beregnede tag

Her beskrives det hvordan de forskjellige delene i løsningen fungere. I database kapitlet henvises det til en kolonne i en tabell ved å bruke *kursiv* stil. I ASP og LabVIEW kapitlene er det noen flytskjema hvor ”boksene” er nummerert og henvist i teksten slik ”(1)”.

For å enkelt kunne endre parametre og adresser tilhørende en beregnet tag ble det valgt å lage grensesnittet i web format. Websidene er laget i HTML og ASPkode for å kunne gjøre kall mot databasen. Parametrene lagres i tabeller i IMATIS databasen hvor de så hentes opp av LabVIEW applikasjoner som gjør beregninger.

5.3.1 Database tabeller

Oppbygning av tabellene i databasen tar utgangspunkt i ønskede funksjoner fra kravspesifikasjonen. Alle kolonner til de forskjellige tabellene blir beskrevet fullstendig i vedlegg 2.



Figur 16 Database struktur for beregnede tag.

JE_Type

Denne tabellen inneholder alle ulike typer av utstyr som kan brukes for eksempel varmeveksler og væskeventil. En type kan egentlig sies være en likning for å beregne en verdi. Det kan for eksempel være en verdi som ikke lar seg måle men beregnes eller en verdi som har et mer brukervennlig format. Et eksempel er, isteden for å vise ulike temperaturer for en varmeveksler kan man regne ut effektiviteten og vise den. En likning kan snus og vendes på for å regne ut forskjellige variabler i likningen. Det gjørs her ved at man knytter ulike metoder til en type, JE_Metode. Hver type har fem egne ASP sider (se kap 5.3.2), urlene for å kalle disse ligger i denne tabellen, *URL, URL2*. Pathen til VI'en som beregner verdier for typen ligger i *VI* kolonnen og kopieres inn i JE_Objekter tabellen ved opprettelse av et nytt objekt.

JE_Objekter

Til hver type kan det finnes mange objekter som ligger i JE_Objekter tabellen. Det kan for eksempel være at man vil registrere enten flere ventiler med samme beregningsmetode eller en ventil med flere beregningsmetoder. Metoden vil ligge i *MetodeId* kolonnen. Annen viktig informasjon her er, som nevnt tidligere, pathen til LabVIEW VI'en som skal kjøre objektet.

JE_Metode

For å kunne bruke samme likning, men få forskjellige utgangsverdier brukes tabellen "JE_Metode". Her får hver metode et navn, *Metode*, som brukes i LabVIEW koden til å starte forskjellige beregninger. Den får også en URL, *URLBilde*, til et bilde som skal instruere brukeren hvordan verdier skal legges inn i JE_Konstant tabellen.

JE_Konstant

For at en beregnet tag skal kunne starte i det hele tatt i en LabVIEW VI må dens tilhørende parametre ligge i denne tabellen. En verdi, *KonstantVerdi*, kan være av tre ulike typer, (*VariabelType)*:

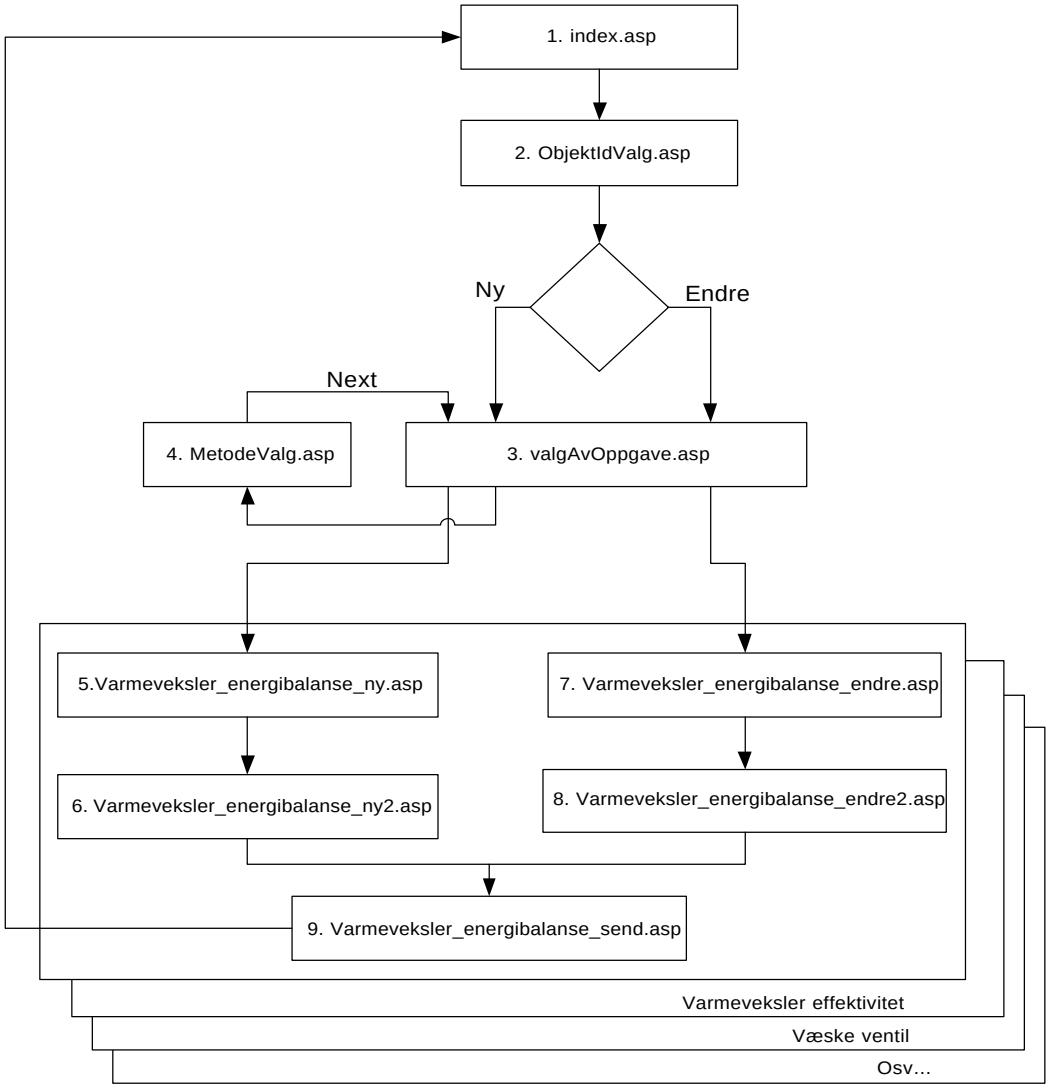
- 0 = konstant
- 1 = variabel (Navn på tag som verdien hentes fra)
- 2 = Utgang (Navn på tag som verdien skrives til)

For å få verdiene i riktig rekkefølge inn i en LabVIEW VI brukes en sorterings kolonne, *Sortering*. Det er i tillegg mulighet for å legge inn navn, beskrivelse og et tidspunkt for registrering.

Citadel Historikk database

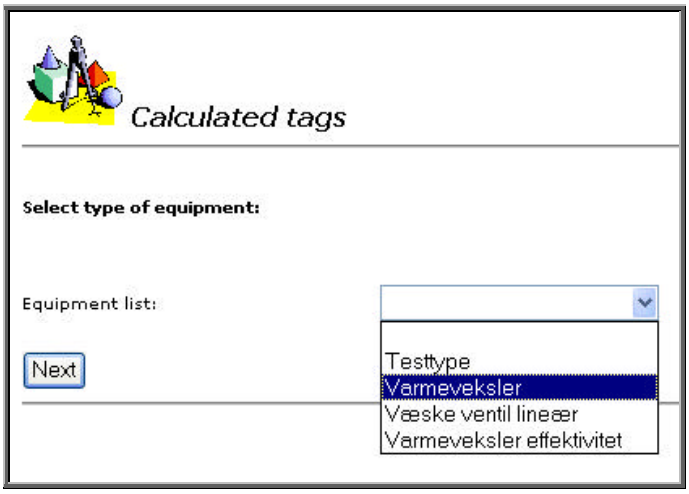
Dette er en database som lagrer alle resultatdata tilhørende en tag. Denne database ligger fra før i National Instruments LabVIEW pakken. Dataen til en tag kan ses på med for eksempel IMATIS Observeren. En forutsetting for å logge data er at en såkalt ”Tag Engine” fra National Instruments er installert på datamaskinen. Eller at filen som data på skal lagres kan skrives til fra datamaskinen med Tag Engine installert[7].

5.3.2 ASP sider



Figur 17 Flytskjema over ASP sider del I.

De første fire ASP sidene er felles for all registrering av beregnede tag. For å kunne lage brukervennlige websider ble det valgt å lage unike sider for hver type i tabellen JE_Type. I dette kapittel blir det kun presentert sider tilhørende en ”varmeveksler energibalanse”, men det er også laget sider for ”Varmeveksler effektivitet”, ”væske ventil lineær” og en ”testtype”. På alle sider blir det gjort diverse SQL kall mot database, for eksakt kode på disse se vedlegg 4.

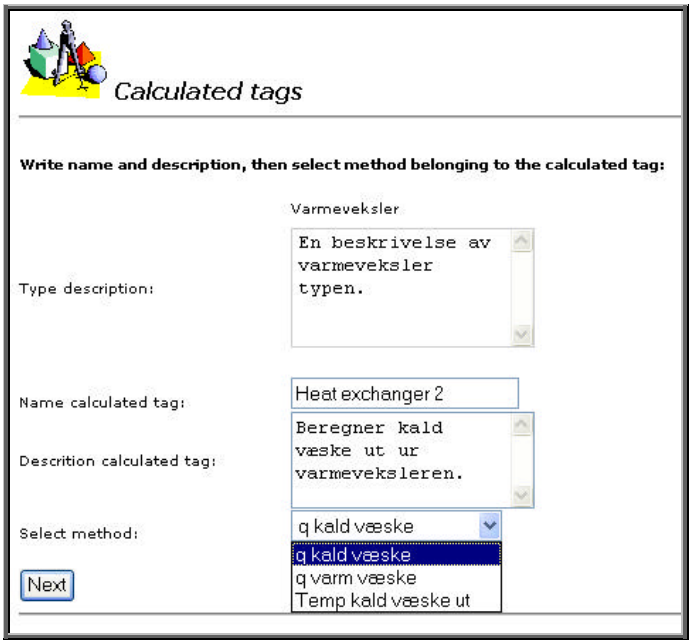


Figur 18 Valg av type utstyr, index.ASP(1)

På denne første side skal man velge hvilken type utstyr som ønskes. Alle typene ligger i JE_Type tabellen og er hardkodede inn der. Her lagres en ASP variabel for den typen som blir valgt og tas med videre til de neste sidene.

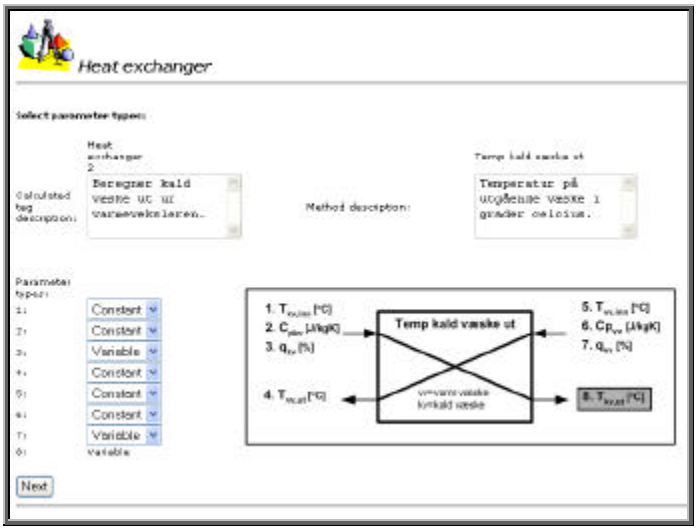
Figur 19 Her har brukeren to valg, enten opprette et nytt objekt eller forandre et eksisterende, ObjektIdValg.asp(2)

Her har brukeren to muligheter, enten å opprette en ny beregnet tag (ALT1)eller forandre på en eksisterende (ALT2). Hvis man velger og forandre må man først velge den som skal forandres på i listen over alle eksisterende tag for denne typen. En eksisterende tag her tilsvarer et objekt i JE_Objekt tabellen. På den ASPsiden skjer det også et SQL kall for å hente opp verdier i JE_Type tabellen knyttet til typen. Det er *TypeId, URL, URL2, VI* og *Beskrivelse* som så lagres i globale variabler og tas med videre. Etter at en oppgave er valgt blir siden ”valgAvOppgave.asp” (3) generert hvor man blir sendt videre avhengig av valg.



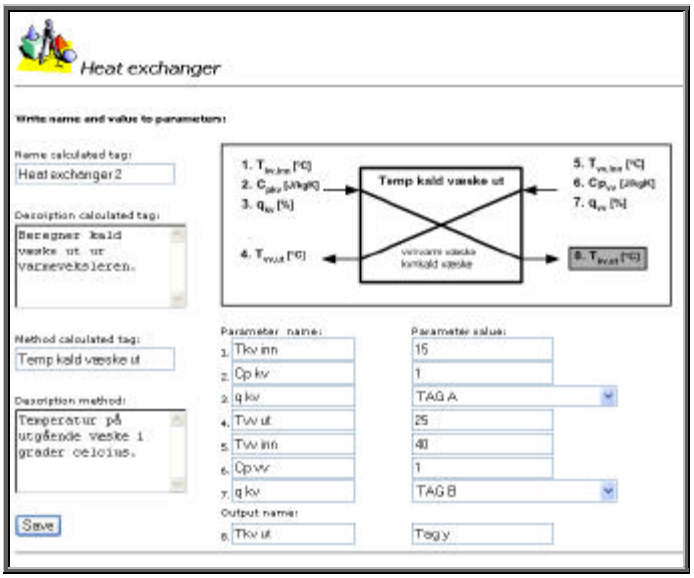
Figur 20 Valg av metode til et objekt, MetodeValg.asp (4)

Hvis man har valgt å legge inn en ny beregnet tag (ALT1) kommer man først til denne siden, Figur 20. Her skriver brukeren inn navn og en beskrivelse av den nye beregnede tagen. Til hver type hører det en mengde forskjellige metoder, dvs at hver metode er en mulig framstilling av en likning. Et eksempel kan være at y=kx+m er en metode mens x=(y-m)/k er en annen metode av samme type likning. Brukeren må her velge en fra listen.



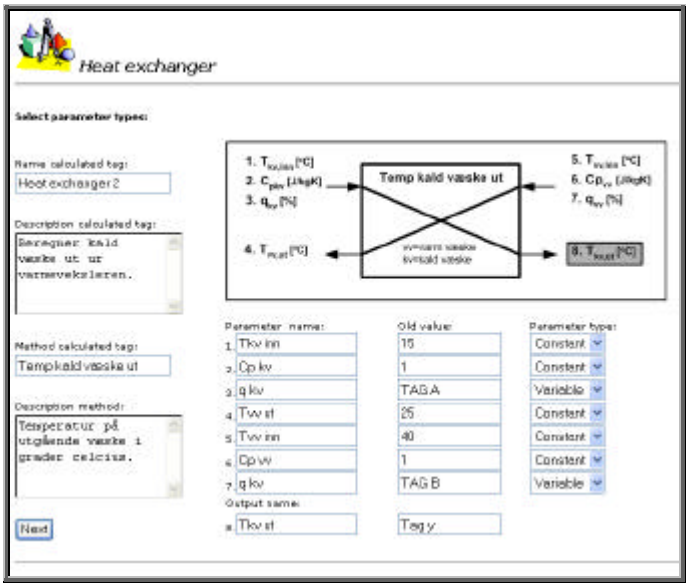
Figur 21 Valg av parameter type, konstant eller variabel, Varmeveksler_energibalanse_ny.asp(5)

Nå kommer brukeren inn på en av de fem sidene som tilhører en varmeveksler, Figur 21. Grunnen til at denne siden ble laget er at det skal være mulighet til å velge hvilken type en parameter skal være av. Den kan enten være konstant eller en variabel, se også kap 5.3.1 JE_Konsant. Til hver metode er det laget et unikt bilde som beskriver hvordan parametre skal skrives inn. Siden det er like mange parametre til hver metode av typen trenger man kun å forandre på bildene. Nummereringen på bildet tilsvarer nummer under ”Parameter types”, Figur 21.



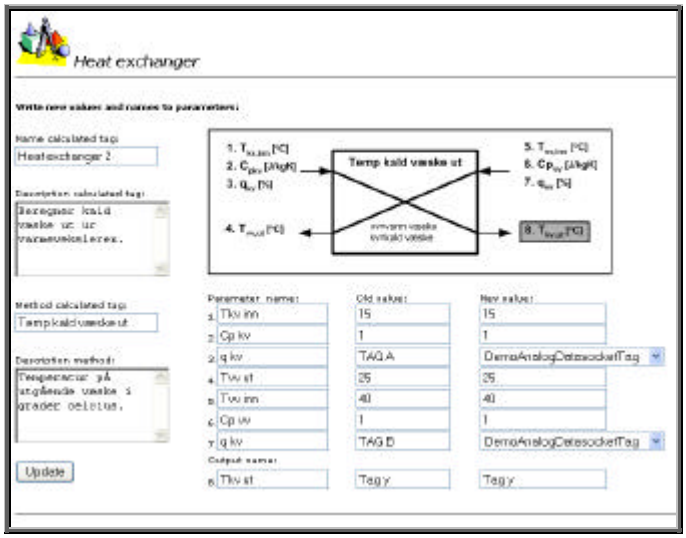
Figur 22 Her skriver brukeren inn alle data tilhørende objektet, Varmeveksler_energibalanse_ny2.asp(6)

På denne siden (Figur 22) fyller brukeren inn parameter navn og verdier. Der hvor brukeren valgte ”variable” på forrige siden (Figur 21) hvil man få opp en ”drop-down box”. Denne er koblet til IMATIS sin tagliste tabell (TAG se Figur27) slik at alle tag som ligger i den kommer opp her. Feltet for utgangen vil alltid være en variabel derfor kan man ikke velge type parameter på den. Når alle felt er fylt ut trykker brukeren på ”Save” knappen og verdiene lagres. det blir først lagret et nytt objekt i JE_Objekt tabellen og deretter parametre i JE_konstant tabellen (Varmeveksler_energibalanse_send.asp(7)).



Figur 23 Brukeren må velge hvilken type en parameter skal være, Varmeveksler_energibalanse_endre.asp(7)

Hvis det ble valgt å endre på en eksisterende beregnet tag (ALT2 i Figur 19), kommer man til denne siden (Figur 23). Her hentes alle verdier som er registrert i JE_Objekter og JE_konstant tabellene frem for riktig objektet. Brukeren må nå velge hvilken type parametrene skal være av, konstant eller variabel.



Figur 24 Her endres de verdier som ønskes, Varmeveksler_energibalanse_endre2.asp(8)

Brukeren trenger nå kun å endre på de parametrer som ønskes. De parametrer som skal være variable har fått opp en ”drop-down box” med tag fra IMATIS TAG tabellen (se Figur27). Til slutt trykker man på ”Update” knappen og JE_Konstant tabellen blir oppdatert med de nye verdiene (Varmeveksler_energibalanse_send.asp(7)).

5.3.3 LabVIEW kode

Alle de forskjellige typene har en egen VI som utfører beregninger. Dette siden det er ulikt antall parametre som skal behandles i hver type. En type-VI kan ha ubegrenset med objekter koblet til seg. LabVIEW koden til del I består også av en ”motor” som har oppgaven med å sende verdier til de forskjellige typene. Verdiene hentes fra JE_konstant, JE_Objekter og JE_Metode tabellene. LabVIEW koden til del I er utarbeidet med assistanse fra Lasse Klovning ved CARDIAC.

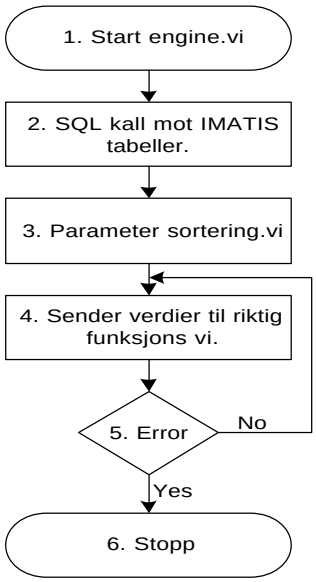


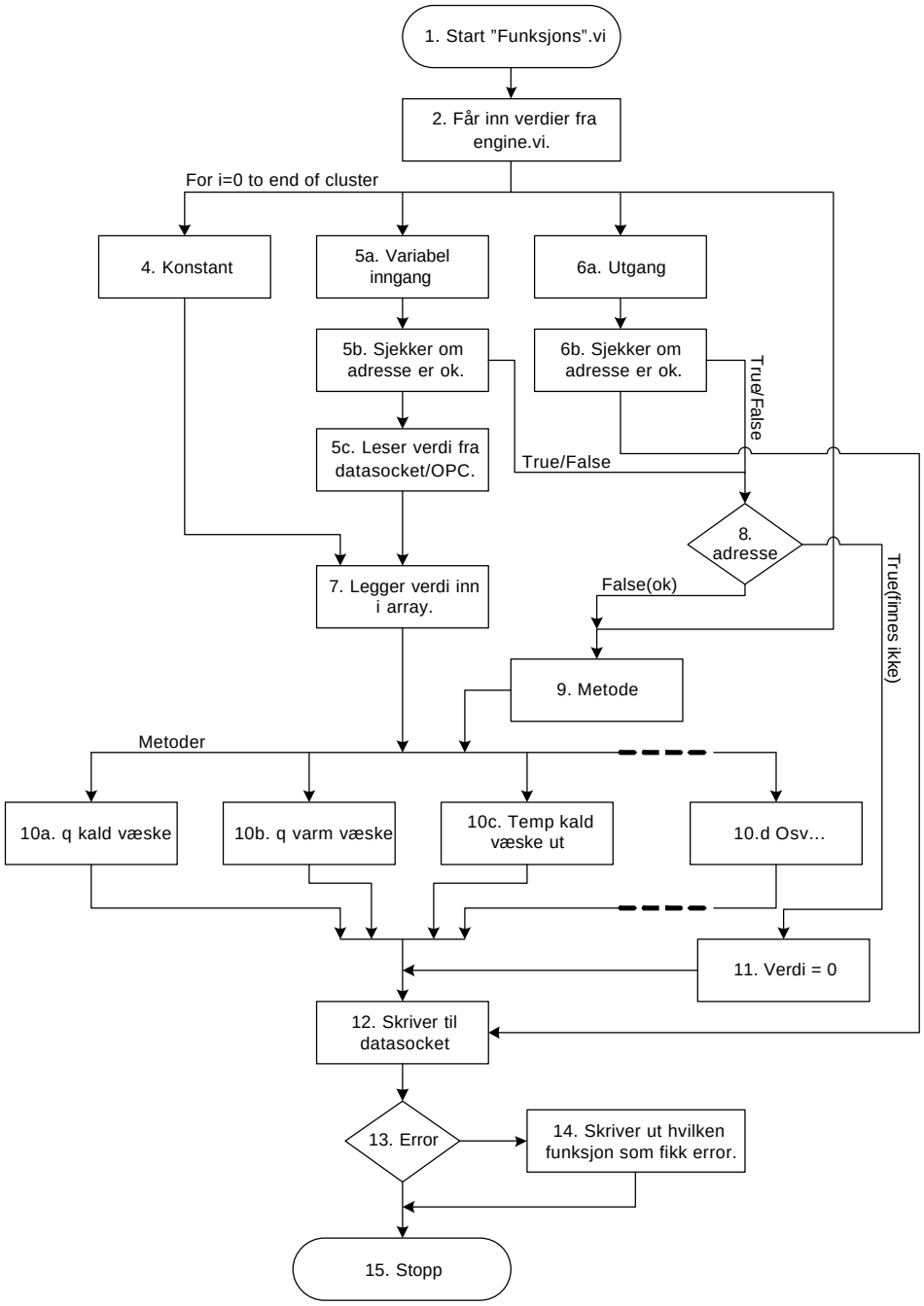
Figure 25 Flytskjema over engine.vi

Når engine er startet (1) blir det først gjort et SQL kall mot IMATIS databasen (2). Her blir de verdier som trengs for å kjøre riktig type og metode hentet i tillegg til beregningsparametrer. Alle verdier blir så lagt inn i cluster som så legges i en ny array. Et cluster er gruppe med data av forskjellig typer (String, int, real etc). Verdier tilhørende et objekt samles i et cluster og får en index array verdi fra 0- uendelig (3). Tabell 1 tilsvarer et cluster og vil ha en array index avhengig av hvor i rekkefølgen den ble hentet i SQL kallet.

	ObjektId	KonstantVerdi	VINameObjekt	VariabelType	Metode
	36	10	varmeveksler.vi	0	q kald væske
	36	5	varmeveksler.vi	0	q kald væske
	36	TAG A	varmeveksler.vi	1	q kald væske
	36	TAG B	varmeveksler.vi	1	q kald væske
	36	5	varmeveksler.vi	0	q kald væske
	36	50	varmeveksler.vi	0	q kald væske
	36	20	varmeveksler.vi	0	q kald væske
	36	Tag y	varmeveksler.vi	2	q kald væske

Tabell 1 Utsnitt (cluster) av parameterliste sendt ut fra engine.vi

Nå blir clusterne pakket opp en etter en. Kolonnen *VINameObjekt* i Tabell 1 brukes nå for å besteme hvilken VI verdiene skal sendes til. De verdier som sendes er *KonstantVerdi*, *VariabelType* og *Metode*. Hvis det har oppstått en feil vil applikasjonen stoppe og vise en feil dialog ellers går den i løkke med intervall på et sekund. Hver gang en type-VI mottar verdier utføres en ny beregning.



Figur 26 Flytskjema over varmeveksler.vi som er egen type.

Alle beregninger til en type, for eksempel varmeveksler, blir gjort i en egen VI tilhørende den typen. Rammeverket for disse VI'er er like mens selve beregningsprosedyren og antall initierings verdier er forskjellig. Etter at en VI er startet (1) kommer initieringsverdier inn i fra engine.vi (2).

Avhengig av hvilken "VariabelType" en verdi er (0=konstant, 1=variabel, 2=utgang) blir de behandlet forskjellig. Hvis det er en konstant (4) blir verdien lagt rett i en array (7) der totalt 7 verdier skal inn (gjelder kun varmeveksler).

Hvis det er en variabel (5a)(Navn på tag som verdien hentes fra, se Tabell 1 TAG A og TAG B typiske) blir den sjekket mot registrerte tag i Tag Manageren. Om den ikke er registrert blir verdien "True" sendt (5b,8) og ingen verdi blir lest. Er den registrert leses verdien fra datasocket/OPC (5c) og legges inn i samme array som de andre konstant verdiene (7).

Hvis det er en utgang (6b) (Navn på tag som verdien skrives til)(6a) blir den sjekket på samme måte som en variabel og om den ikke er registrert sendes ”True” videre. Hvis utgangen er gyldig blir den sendt videre til en funksjon som til slutt legger utgangsverdien på datasocket(12).

Hvis det ikke kommer noen ”True” verdier fra tag sjekken blir ”Metode” brukt (9) til å utføre den riktige beregningen (10a, 10b,10c eller 10d) bassert på de nye verdiene i arrayen (7).

Når beregningen er utført skrives verdien til datasocket (12) og en eventuell feil blir skrevet ut med feilbeskrivelse og hvilken type VI som forårsaket feilen (13,14).

Til slutt stopper VI’en (15) og starter på nytt når nye verdier kommer inn fra engine.vi.

5.3.4 Drøfting del I

For å få brukervennlige ASPsider ble det valgt å lage sider for hver enkel type i systemet. Men tanken på å bruke en felles side for alle typer var også tilstede. Derfor ble kolonnen *AntalKonstanter* lagt inn i tabellen. Tanken var at dynamisk tegne opp riktig antall innskrivningsfelter og ha et bilde som instruerer brukeren hvordan felten skulle fylles ut. Da hadde man kun trengt å lage et nytt bilde for hver type og tilhørende metode. På grunn av tekniske problemer ble den løsningen lagt til siden.

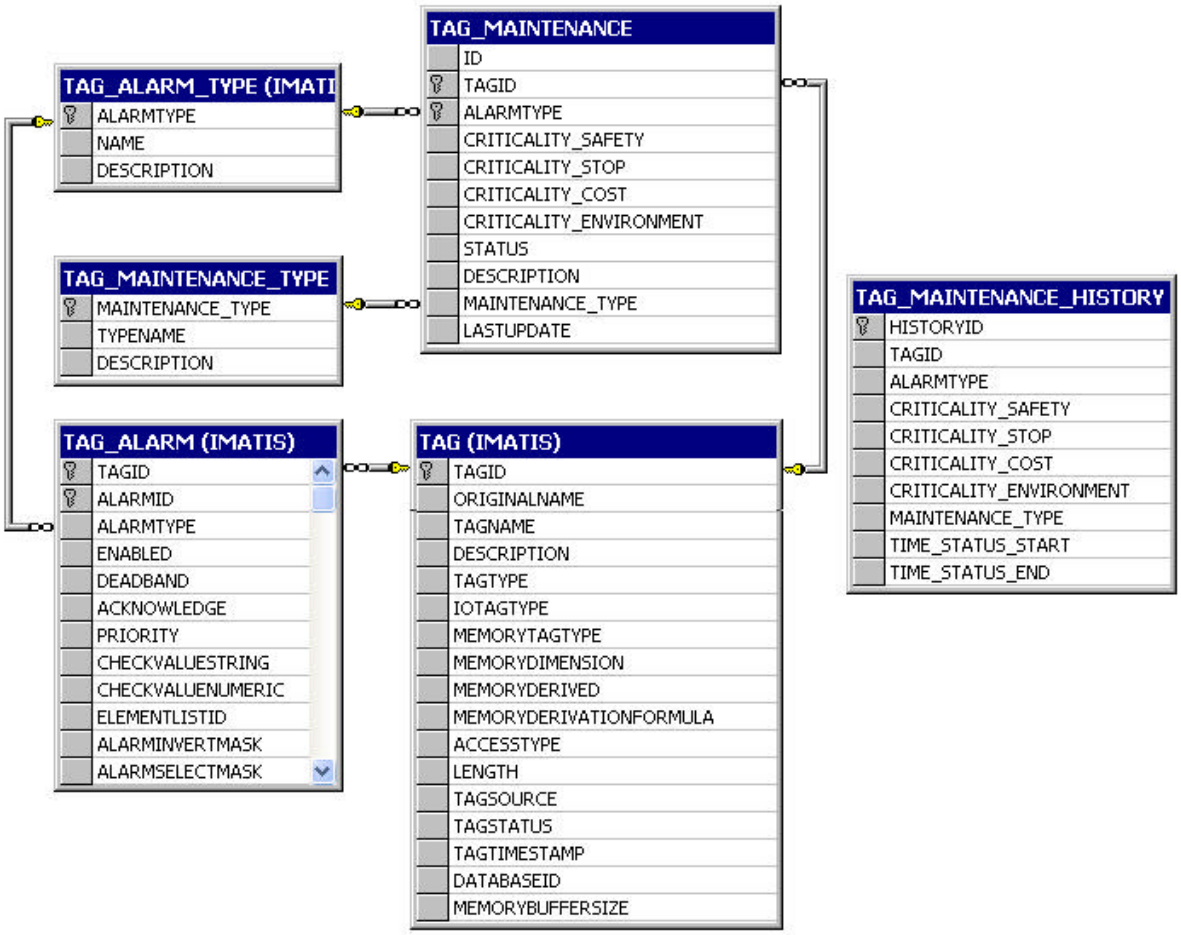
Det måtte legges inn en kolonne for adressering til et bilde tilhørende metoden (*URLBilde*). Det var på grunn av at det er så vanskelig å legge inn et bilde i en database. Bildet lagres på bitformat og det må lages en applikasjon i f.eks LabVIEW som lager bitformat av et bilde og omvendt. Dette medførte at den lettere løsningen ble valgt med å lagre bildene på fil isteden.

Siden det er forskjellig antall konstanter og variabler som brukes til beregninger i de ulike typen er man tvungen til å ha en VI for hver type. Fordelen er at rammen til disse er lik dvs at det bare er selve beregningsdelen i applikasjonen som må fornyes. Dette gjør det svært enkelt å opprette nye type-VI’er til systemet.

5.4 Del II, vedlikeholdstag

5.4.1 Database tabeller

Tabellene for vedlikeholdsdelen bygger på noen nye tabeller, og noen som fra før ligger i IMATIS databasen. Her følger en kort beskrivelse av viktige kolonner og hensikten med dem. En fullstendig beskrivelse av alle kolonner til de nye tabellen ligger i vedlegg 3.



Figur27 Database struktur over vedlikeholdstabeller.

TAG_MAINTENANCE
I denne nye tabellen plasseres alle tag som blir definert som vedlikeholdstag. For å plukke ut en unik vedlikeholdstag brukes i utgangspunkt *TAGID* sammen med *ALARMTYPE*, men for å gjøre en del ASP funksjoner enklere ble det lagt inn en unik *ID* også. Alle *TAGID* blir hentet fra TAG tabellen som er en gammel IMATIS tabell. Det er kolonner til de forskjellige kritikalitetene som knyttes opp mot en vedlikeholdstag. *STATUS* kolonnen sier om en vedlikeholdstag står i en alarm (0 = av og 1 = på) eller ikke. Denne kolonnen brukes primært av LabVIEW applikasjonen som styrer vedlikeholdstagene. MAINTENANCE_TYPE kolonnen viser hvilken type handling som bør/skal foretas ved en alarm. For å vite når en alarm slår inn brukes *LASTUPDATE* kolonnen. Verdien vil hver gang *STATUS* forandrer seg flyttes inn i TAG_MAINTENANCE_HISTORIE tabellen.

TAG_MAINTENANCE_TYPE
Her legges alle forskjellig handlinger inn som skal foretas ved alarmer. De som er registrert til nå er:

- No action
- Warning
- Immediate action

TAG_ALARM_TYPE
Dette er en tabell som ligger i IMATIS databasen fra før. Det som ligger i tabellen er de forskjellig navnen på mulige alarmer som LoLo, Lo, HiHi og Hi.

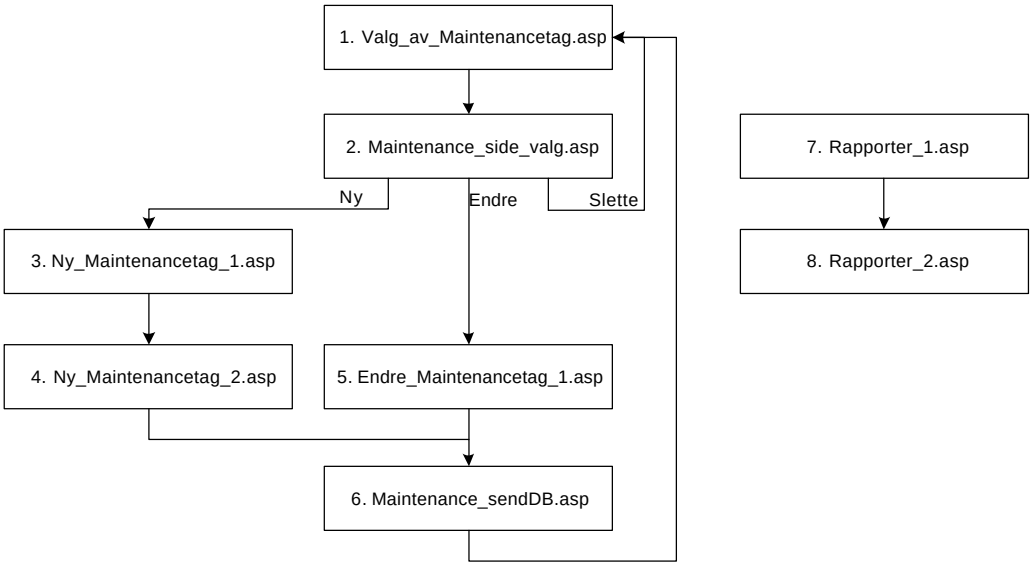
TAG_ALARM
Dette er også en eksisterende IMATIS tabell der kolonnen *ENABLED* brukes for å se om det er satt en alarm på tagen. Dette er en forutsetning for å kunne registrere den som vedlikeholdstag. I kolonne *CHECKVALUENUMERIC* ligger grenseverdien til alarmen som brukes i LabVIEW applikasjonen for å sjekke om en alarm skal aktiveres eller deaktiveres.

TAG
Dette er hovedtabellen i IMATIS databasen. Når en beregnet tag er registrert med Tag Manageren vil den legges inn her å kan da bli en vedlikeholdstag. Her ligger alle tag definert i en rekke forskjellig kolonner som i sin tur brukes i ulike IMATIS applikasjoner. Den knytter også sammen tabellene tilhørende vedlikeholdstag. En viktig parameter her som brukes i LabVIEW kode er *TAGSOURCE*. Det er adressen som brukes når en verdi skal innhentes fra datasocket/OPC.

TAG_MAINTENANCE_HISTORY
I denne tabellen lagres alle verdier tilhørende en vedlikeholdstag når en alarm *STATUS* går fra verdien 1 til 0. I tillegg skrives nå-tiden i kolonnen *TIME_STATUS_END* slik at man kan se hvor lenge en alarm har vært aktivert. Dette styres fra LabVIEW applikasjonen som beskrives senere i kapittelet. Fra denne tabellen hentes all informasjon som brukes til å lage web-rapporter som omhandler vedlikehold.

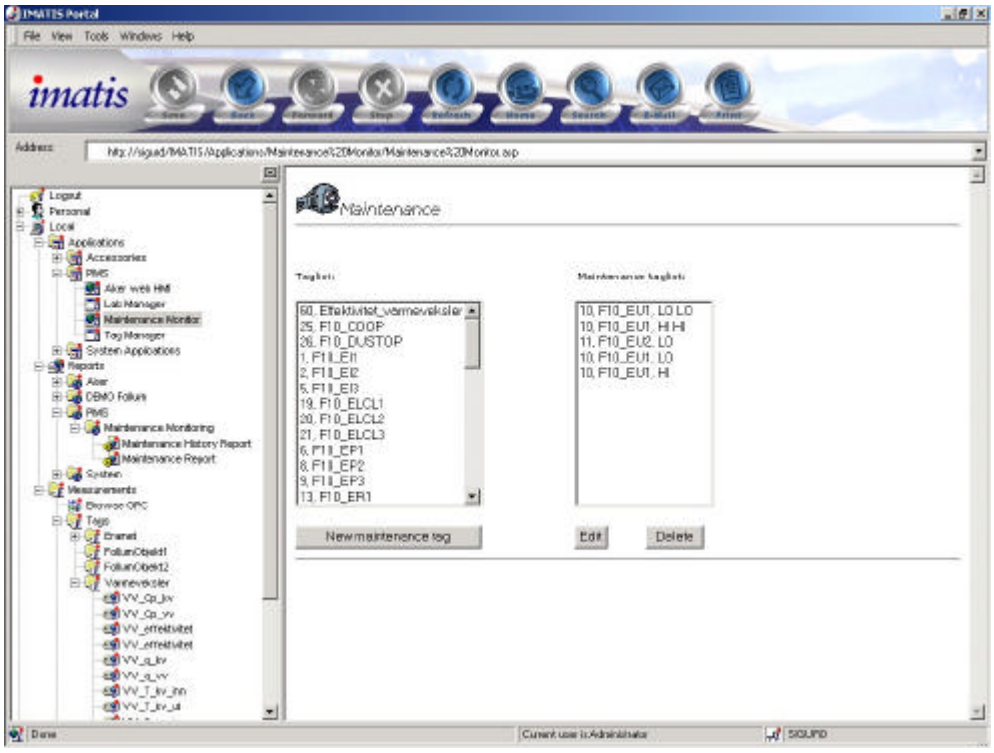
5.4.2 ASP sider

Brukeren har mulighet til å opprette nye vedlikeholdstag fra tag som ligger i IMATIS TAG tabellen. Det er også mulig å forandre på eksisterende vedikeholdstag. En forutsetting er at tagen som skal brukes har alarmer knyttet til seg som gjøres i Tag Manageren. Den fullstendige koden til ASP sidene ligger i vedlegg 4.



Figur 28 Flytskjema over ASP sider del II.

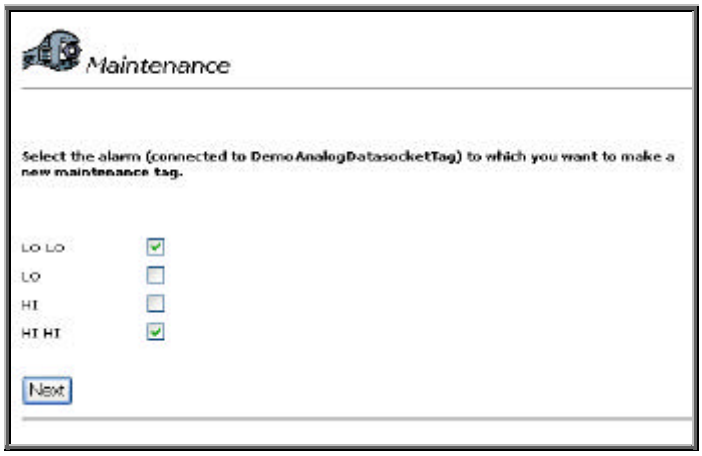
Manipulering og rapportering av vedlikeholdstag gjøres i to separate trinn. Først registreres en vedlikeholdstag inn på sidene 1-6 etter det kan man få den opp på rapporteringssidene 7-8.



Figur 29 Hovedside for innregistrering og forandring av vedlikeholdstag(1).

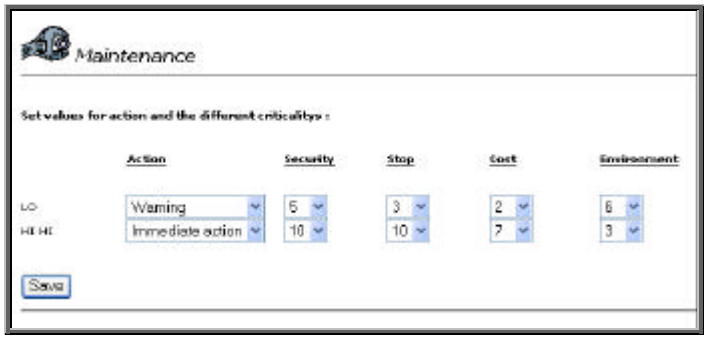
Figur 29 viser innregistreringssidene for vedlikehold lagt inn under ”Maintenance Monitor” i IMATIS portalen. Før det kommer noen vedlikeholdstag i den høyre ”drop-down” boksen må de registreres fra eksisterende tag i venstre drop-down boks. Etter at en vedlikeholdstag er registrert første gangen er det mulig å forandre og slette den fra vedlikeholdstaglisten.

Avhengig av valg blir bruker nå sendt til riktig side via Maintenance_side_valg.asp (2). Fra den siden slettes også en vedlikeholdstag i databasen hvis det er valget.



Figur 30 Valg av alarm som man ønsker knytte til vedlikehold(3).

I Figur 30 vises kun de alarmer der feltet *ENABLED* i TAG_ALARM tabellen (Figur27) er lik ”1”. Brukeren kan nå velge hvilken eller hvilke alarmer som det skal lages vedlikeholdstag av.



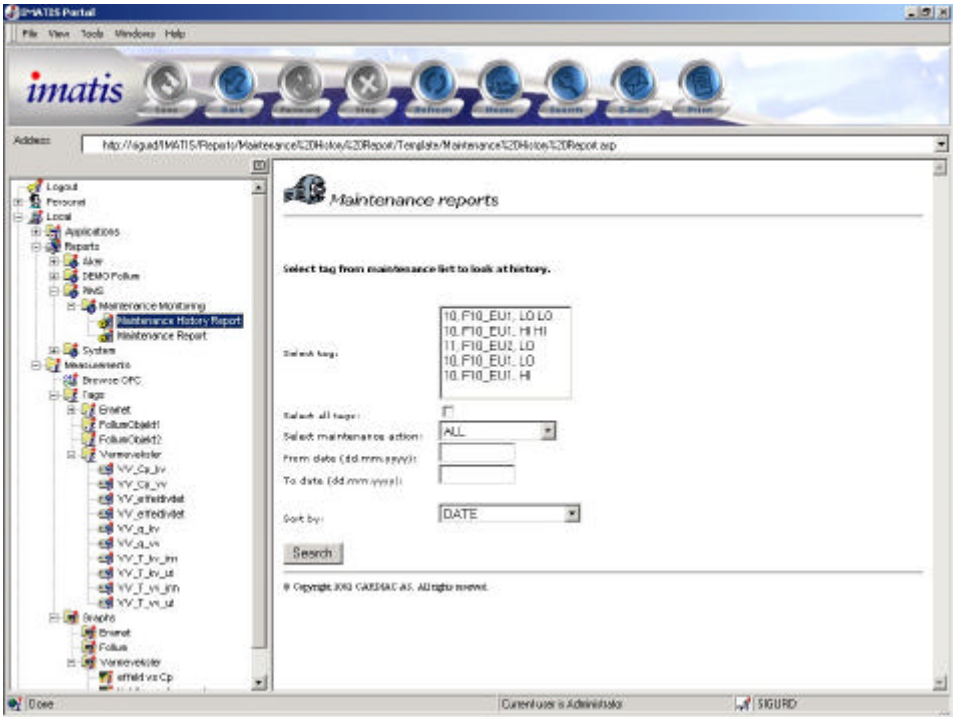
Figur 31 Parametersetting av vedlikeholdstag(4).

I Figur 31 har de alarmer som ble valgt i forrige skjermbilde kommet opp og brukeren kan legge inn verdier. I ”Action” feltet velges hvilken type handling som skal utføres ved en alarm (se kap. 5.4.1) mens det i de andre felten velges en verdi mellom 0-10 for kritikaliteten. De ulike kritikalitetene er som beskrevet i kapittel 2.4, Security, Stop, Cost og Environment.

Figur 32 Forandring av parametrer til en vedlikeholdstag(5).

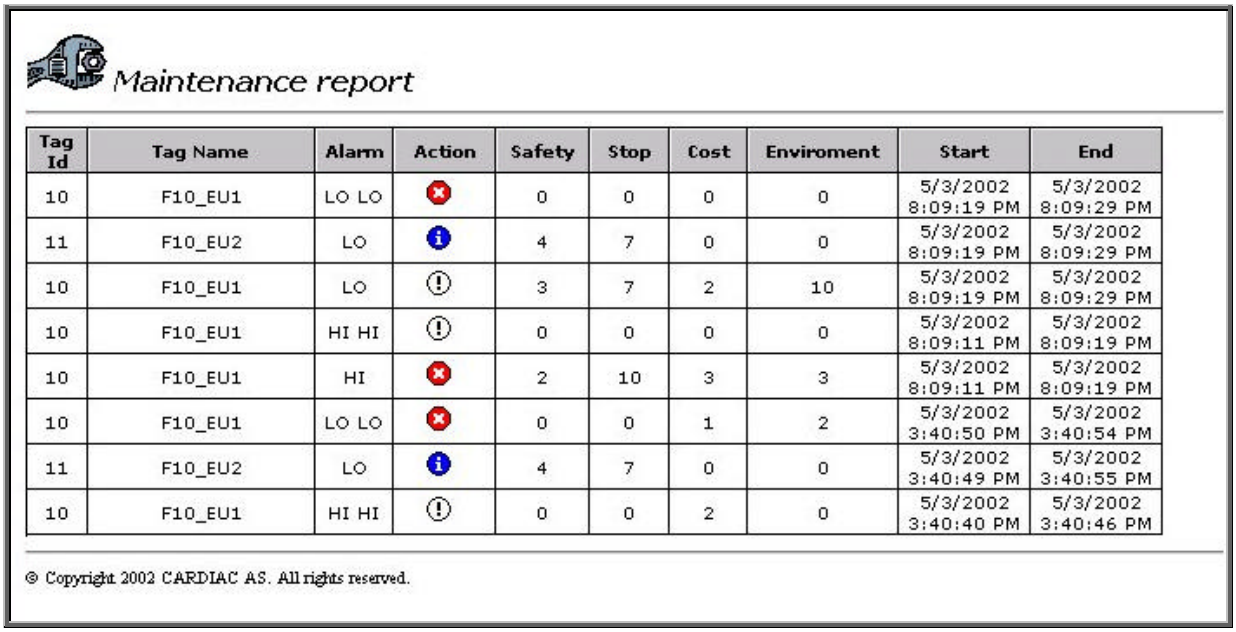
Hvis det ble valgt en vedlikeholdstag og ”Edit” på ASP side 1 (Figur 29) kommer tagen opp med de gamle verdiene og mulighet til å forandre dem her i Figur 32.

All lagring eller oppdatering av verdier gjøres på siden Maintenance_sendDB.asp (6).



Figur 33 Valg av vedlikeholdstag som det skal lages rapport på(7).

Når brukeren ønsker en rapport fra en vedlikeholdstag går man til denne siden (Figur 33) ved å trykke på ”Maintenance history repport” i IMATIS portalen. På denne siden velges enten ”en” eller ”alle” vedlikeholdstag som man ønsker å se en rapport av. Videre velger man den handling (action) som skal vises eller om man ønsker alle handlinger. Til slutt kan man skrive inn mellom hvilke dato som skal vises. Det er også mulighet for å sortere etter kriteriene dato, Tag Id eller noen av de fire kritikalitetene. Hvis ingen velges sorteres det automatisk etter dato. Hvis brukeren trykker på knappen ”Maintenance repport” vil man få en status over alle vedlikeholdstag med aktive alarmer.



Maintenance report

Tag Id	Tag Name	Alarm	Action	Safety	Stop	Cost	Enviroment	Start	End
10	F10_EU1	LO LO		0	0	0	0	5/3/2002 8:09:19 PM	5/3/2002 8:09:29 PM
11	F10_EU2	LO		4	7	0	0	5/3/2002 8:09:19 PM	5/3/2002 8:09:29 PM
10	F10_EU1	LO		3	7	2	10	5/3/2002 8:09:19 PM	5/3/2002 8:09:29 PM
10	F10_EU1	HI HI		0	0	0	0	5/3/2002 8:09:11 PM	5/3/2002 8:09:19 PM
10	F10_EU1	HI		2	10	3	3	5/3/2002 8:09:11 PM	5/3/2002 8:09:19 PM
10	F10_EU1	LO LO		0	0	1	2	5/3/2002 3:40:50 PM	5/3/2002 3:40:54 PM
11	F10_EU2	LO		4	7	0	0	5/3/2002 3:40:49 PM	5/3/2002 3:40:55 PM
10	F10_EU1	HI HI		0	0	2	0	5/3/2002 3:40:40 PM	5/3/2002 3:40:46 PM

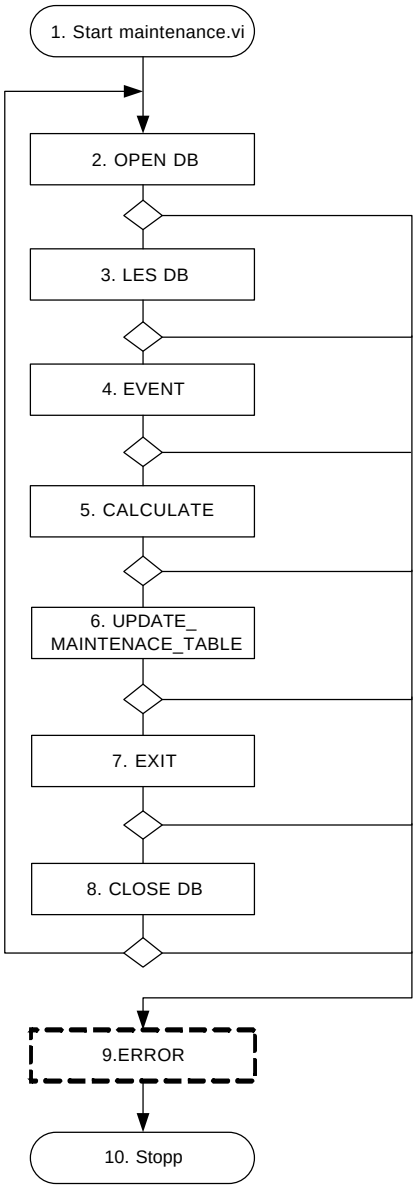
Figur 34 Rapport utskrift(8)

Selve rapporten som brukeren får opp i Figur 34 skal nå vise ønskede verdier riktig sortert. Tiden mellom ”Start” og ”End” er tidsintervallet alarmen har vært aktiv. Symbolene i ”Action” kolonnen blir valgt etter hvilken verdi det er i *MAINTENANCE_TYPE* kolonnen i Figur27. Betydningen av de er som følger:

- = No action
- = Warning
- = Immediate action

5.4.3 LabVIEW

Denne applikasjonen utfører sjekker og databaseoppdateringer til alle vedlikeholdstag. Det sjekkes kontinuerlig mot datasocket/OPC om verdien er utenfor tillatt grenseverdi. Hvis det er tilfelle oppdateres databasen TAG_MAINTENANCE_HISTORY. Applikasjonen bygget opp etter ”state machine” modellen der verdien i en case lagres i et shift register slik at det kan brukes i noen av de neste casene etter behov, se vedlegg 5. Blokkene i figurene under er nummerert med heltall som henviser til Figur 35 og med et desimaltall når det henvises til et utvidet flytskjema for en av blokkene i Figur 35.LabVIEW koden til del II er utarbeidet med assistanse fra Sigurd Juvik ved CARDIAC.



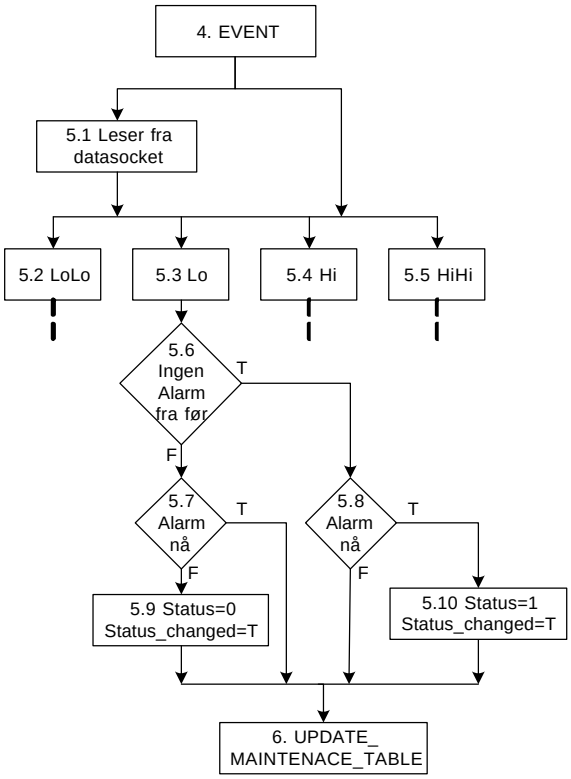
Figur 35 Flytskjema maintenance.vi

Etter at applikasjonen har startet (1) og IMATIS databasen er åpnet (2) leses alle vedlikeholdstag fra TAG_MAINTENANCE tabellen inn (3). I tillegg leses *TAGSOURCE* verdien som er adresseringen for å hente verdien til tagen på datasocket. Grenseverdien til tagens alarm hentes i *CHECKVALUENUMERIC* kolonnen i TAG_ALARM tabellen, se Tabell 2.

	TAGID	ALARMTYPE	STATUS	TAGSOURCE	CHECKVALUENUMERIC
	70	2	0	\\johno\LabVIEW\DemoAnalogDatasocketTag	47,5
	70	5	0	\\johno\LabVIEW\DemoAnalogDatasocketTag	52
	74	3	0	\\johno\LabVIEW\DemoAnalogDatasocketTag1	50
	70	3	0	\\johno\LabVIEW\DemoAnalogDatasocketTag	48
	70	4	0	\\johno\LabVIEW\DemoAnalogDatasocketTag	51,5

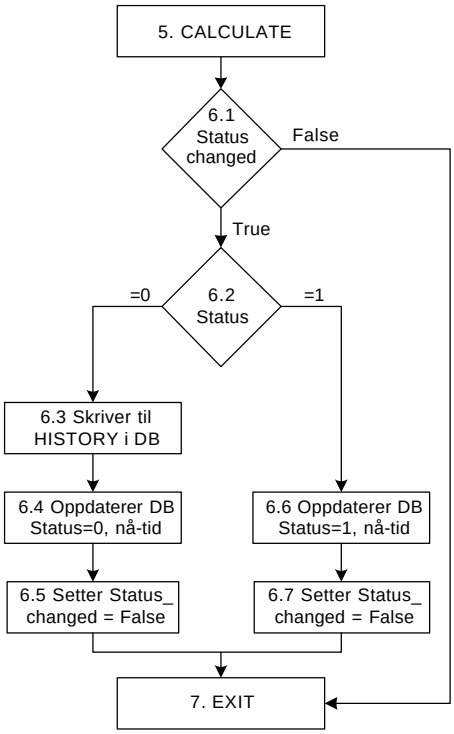
Tabell 2 Utdrag av verdier som hentes med SQL-kall til maintenance.vi

I EVENT (4) blokken er det mulighet for å legge inn diverse funksjoner som skal utføres i applikasjonen.



Figur 36 Utvidet flytskjema over "CALCULATE" blokken.

Når CALCULATE (5) blokken starter blir det først lest inn en verdi tilhørende tagen. Her brukes i utgangspunkt adressen i Tabell 2, men under uttesting kan man manuelt forandre på verdien (5.1). Avhengig av hvilken ALARMTYPE (se Tabell 2) blir nå riktig ”case” kjørt (5.3). Flytskjema som følger nå ser likt ut for alle alarmtypene. Det første som sjekkes er om statusen er 0 eller 1, hvis den er 0 (F) er det alarm på fra før (5.6) og er den 1 (T) er det ingen alarm på fra før. Neste sjekk er nå i begge tilfeller (5.7, 5.8) om alarmen er aktiv nå. Da sammenliknes innlest verdi (5.1) og grenseverdien hentet fra *CHECKVALUENUMERIC* kolonnen i Tabell 2. Sagt på en annen måte, hvis det er alarm fra før, men ingen nå, skal status settes lik 0 (5.9) og hvis det ikke er noen alarm fra før, men en ny alarm nå, skal status settes lik 1 (5.10). Status_changed settes lik True i begge tilfeller.



Figur 37 Utvidet flytskjema over "UPDATE_MAINTENANCE_TABLE" blokken

Denne blokken tar seg av all lagring til IMATIS databasen. Hvis Status_changed er true og Status har blitt satt til 0 i CALCULATE blokken (6.1, 6.2) skal verdiene knyttet til tagen i TAG_MAINTENANCE tabellen flyttes over til TAG_MAINTENANCE_HISTORY tabellen. Da må de først hentes opp i LabVIEW applikasjonen og siden legges inn i den nye tabellen. For å vite hvor lang tid en alarm har stått på lagres det samtidig nåtid i *TIME_STATUS_END* kolonnen i TAG_MAINTENANCE_ HISTORY tabellen (6.3). Etter det oppdateres tagen i TAG_MAINTENANCE tabellen med STATUS = 0 og *LASTUPDATE* med nåtid (6.4). Før applikasjonen går til EXIT (7) setter den Status_changed = False (6.5).

Hvis Status har blitt satt til 1 i CALCULATE blokken (6.2) innebærer det at en alarm har slått inn og det trengs bare en oppdatering av kolonnene STATUS og LASTUPDATE i TAG_MAINTENANCE tabellen med 1 respektive nå-tid (6.6). Før applikasjonen går i EXIT setter den Status_changed = False (6.7).

I EXIT (7) blokken gjøres eventuelle avsluttende oppgaver før databasen stenges i CLOSE DB (8). Hvis det skulle oppstå en feil underveis i noen av blokkene er det lagt opp til at man skal gå til en ERROR (9) blokk, men denne er ikke implementert i denne omgang.

5.4.4 Drøfting del II

Denne delen er mye mer integrert med eksisterende IMATIS tabeller. Her brukes både tag- og alarminformasjon til ASP sidene og applikasjonen.

LabVIEW applikasjonen er oppbygget etter en modell som er vanlig å bruke når det er sekvensielle handlinger. Verdiene fra en handling lagres i skiftregister tils de skal brukes i en ny sekvens. Det er ikke tatt hensyn til samplingfrekvens i denne delen, uten det er brukt en generell samplingsfrekvens for å sjekke alarmer.

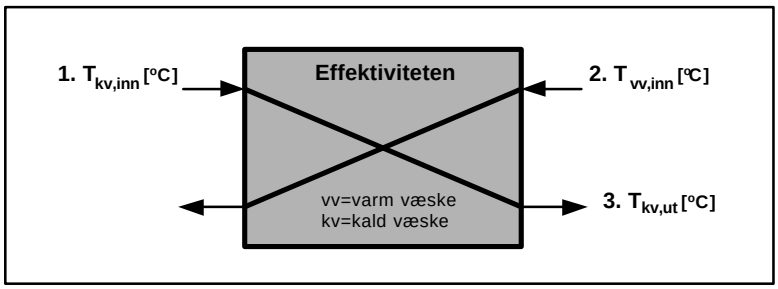
Det viste seg at det var behov for å kunne få en sanntidsstatus på alarmer også. Derfor måtte det legges inn en knapp for det og oppdatere rapporteringssiden med et nytt SQL kall som henter opp alle vedlikeholdstag med status=1 i TAG_MAINTENANCE tabellen.

6 UTTESTING AV NOEN ENKLE MODELLER

For å vise hvordan alle funksjoner henger sammen er det i dette kapittel laget et scenario fra begynnelse til slutt. Scenariet viser effektiviteten til en varmeveksler der noen parametrer er variable og noen konstante. Verdier og likninger som brukes i uttestingen er hentet fra Sigurd J. Juvik hovedoppgave 1996 for Hydro Agri Porsgrunn om deres inndamperannlegg. Det er laget en enkel simulator for å simulere de variable verdiene fra en prosess. Den fullstendige dokumentasjonen med skjermbilder til uttestingen ligger i vedlegg 7. Det er også beskrevet i kapittel 5.3.2 og 5.4.2 hvordan innregistrering foregår.

Likningen som er implementert i VI'en for varmeveksler effektivitet er:

$$\text{Effektiviteten} = \frac{T_{kv,ut}[^oC] - T_{kv,inn}[^oC]}{T_{vv,inn}[^oC] - T_{kv,inn}[^oC]}$$



Figur 38 Effektivitet varmeveksler

$T_{kv,ut}$ = Temperatur kaldt vann, ut i grader celsius.

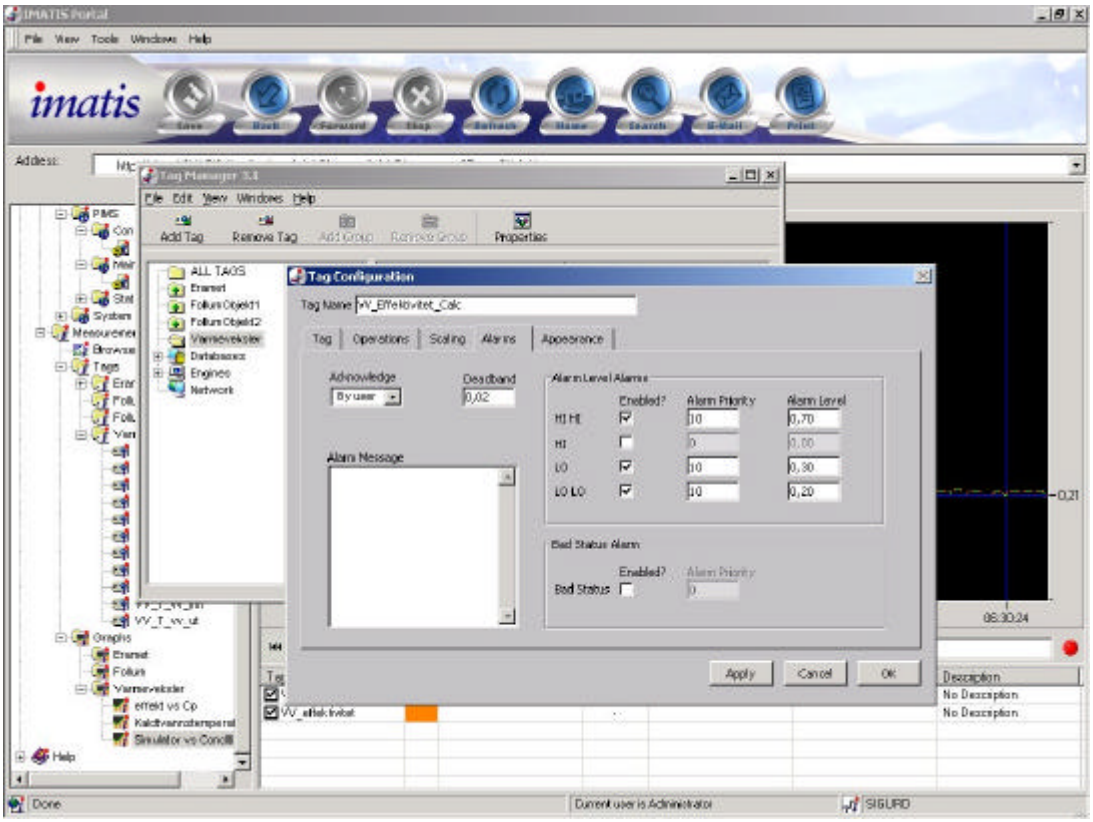
$T_{kv,inn}$ = Temperatur kaldt vann, inn i grader celsius.

$T_{vv,inn}$ = Temperatur varmt vann, inn i grader celsius.

Effektiviteten = verdi ut fra 0 – 1.

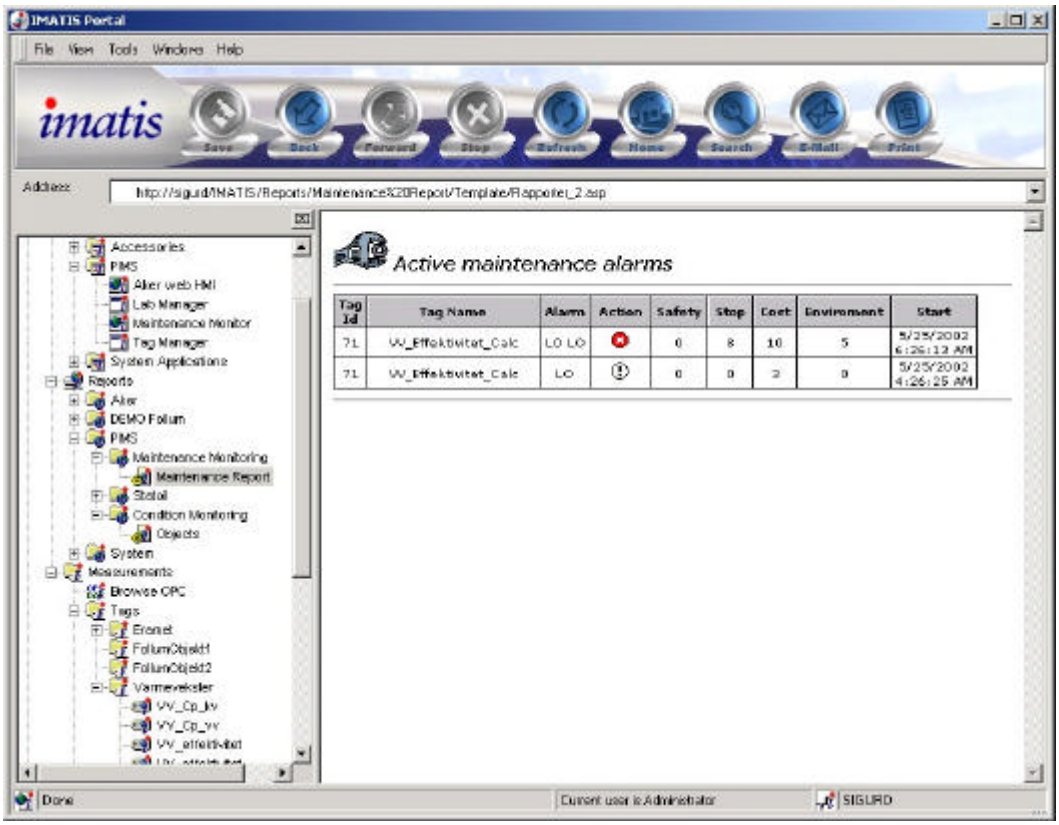
1. Først må brukeren registrere inn et nytt objekt av typen "Varmeveksler effektivitet" (Figur 1 i vedlegg 7).
2. Det velges nå å registrere ny tag med knappen "New calculated tag" (Figur 2 i vedlegg 7).
3. På den siden som kommer opp skal det skrives inn navn og beskrivelse av den nye beregnede tagen (Figur 3 i vedlegg 7).
4. Nå får brukeren opp et bilde av en varmeveksler og man må velge hvilke type parametere som skal brukes på de forskjellige inngangene (Figur 4 i vedlegg 7).
5. Nå får brukeren opp lister der det er valgt variabel i forrige punkt. Der kan det velges inngang fra eksisterende tagliste i IMATIS systemet. Den siste verdien er konstant så verdien skrives rett inn. På feltet for utgangs parameter verdi skrives tag adressen inn (Figur 5 i vedlegg 7).
6. Nå vil den nye beregnede tagen komme opp i listen over "calculated tags" og kan etter behov forandres ved å trykke på edit knappen (Figur 6 i vedlegg 7).

7. Nå skal den nye beregnede tagen registreres som en vedlikeholdstag. Det første man må gjøre er da er å registrere den ny tagen inn i IMATIS taglisten med adresse og alarmgrenser. Det gjøres i Tag Manageren i IMATIS se Figur 39. Adressen som det skal leses fra er den samme som man skrev inn i punkt 5 (datasocket/OPC) som utgangsadresse og alarmgrensen er (Figur 7 i vedlegg 7):
    a. LoLo alarm = 0.20 (20% effektivitet)
    b. Lo alarm = 0.35
    c. Hi alarm = 0.7 (Denne settes slik at man kan oppdage en eventuell målefeil side effektivitet aldri vil kunne bli så høy)



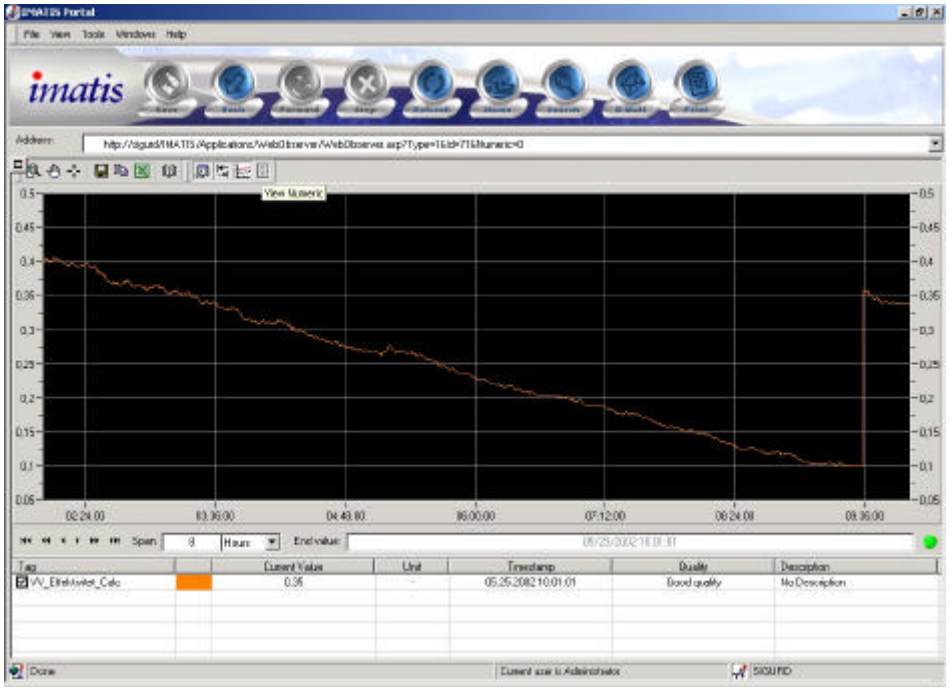
Figur 39 Her vises hvordan alarmgrenser blir satt opp i Tag Manager.

8. Nå skal man registrere en vedlikeholdstag. Brukeren velger da ”New Maintenance tag” knappen (Figur 8 i vedlegg 7) etter at den tagen som skal registreres er valgt i IMATIS taglisten.
9. På neste side krysse de alarmer av som man ønsker knytte til vedlikehold (Figur 9 i vedlegg 7).
10. Etter det velges handlings type og viktighet av kritikalitene (Figur 10 i vedlegg 7).
11. Nå vil brukeren få opp de nye vedlikeholdstagen i en egen drop-down boks herifra kan de velges hvis man ønsker editer på de (Figur 11 i vedlegg 7).
12. Hvis man nå går til selve rapport sidene så vil man se at de registrerte vedlikeholdstagene kommer opp her. Da kan det velges hva som skal rapporteres (Figur 12 i vedlegg 7).
13. Hvis man trykker på knappen ”Active alarms” får man opp hvilke alarmer som er aktive akkurat nå. Det man ser i Figur 40 er at klokken 4:26:25 AM har en Lo alarm slått på og klokken 6:26:13 AM har LoLo alarmen slått på (Figur 13 i vedlegg 6).



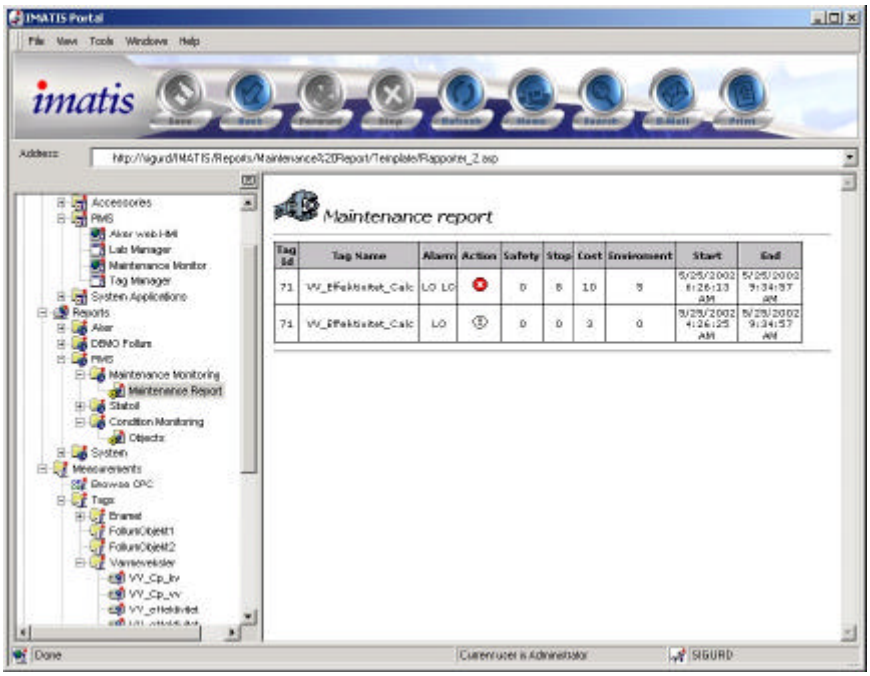
Figur 40 Her ser brukeren hvilke alarmer som er aktive i nåtid.

14. Ved å gå inn i IMATIS Observer kan disse verdier leses ut fra kurven som simulerer en varmeveksler se Figur 41. Der den beregnede verdien for effektiviteten kommer inn og logges (Figur 14 i vedlegg 6).



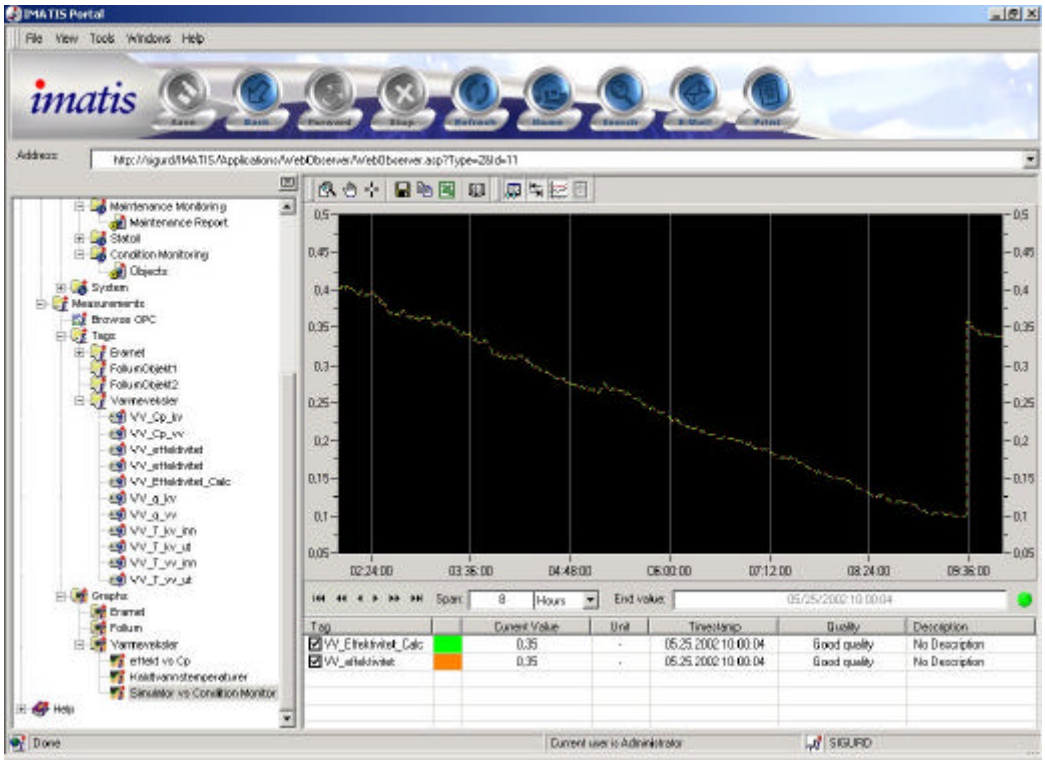
Figur 41 Simuleringsverdier for en varmeveksler, Y-aksen viser effektiviteten og X-aksen tid.

15. Til slutt kan brukeren se på en historikkrapport, se Figur 42. Der ser man alarmene som har vært aktive og når de deaktivertes klokken 9:34:57 (Figur 15 i vedlegg 6).



Figur 42 Historikk rapport der man ser alarmer som vært aktive.

Ved å sjekke at verdier som har blitt skrevet til vedlikeholds historikken (Figur 42) stemmer overrens med kurven i IMATIS Observeren (Figur 41) ser man at alle applikasjoner fungerer som de skal.



Figur 43 Test med verdier fra simulator som regner effektivitet satt opp mot varmveksler effektivitets VI'en.

Det ble kjørt en test der en LabVIEW simulator til Sigurd Juvik simulerte varmeveksler effektivitet. Verdiene fra simulatoren og beregningene gjort i varmeveksler effektivitets VI'en ble hentet inn i samme trend vindu. Der ser man at verdiene følger hverandre helt likt dvs at beregningen i VI'en stemmer.

7 FORSLAG TIL VIDERE ARBEID

På grunn av tidsbegrensning har det ikke vært mulig å få til alle forbedringer som har oppdagets under prosjektets gang. Her er en punktvis oppsummering av forslag til videre arbeid og forbedringer av systemet.

- Det kan lages en generell likningseditor der en man selv kan velge hvilke og hvordan parametre skal beregnes, slik at det blir full frihet for brukeren.
- Det kan lages flere typer av utstyr.
- Det kan gjøres en del med sikkerhet rundt inntasting av parametre og kall til database, slik at man ved feil inntasting får en feilmelding.
- Rapporter kan videreutvikles med flere søke kriterier, sorterings muligheter og opprettelse av ferdig definerte rapporter.
- ASPsidene for registreringen av nye beregnede tag kan det gjøres noe både med design og funksjoner. De kan gjøres mer dynamiske slik at omfang av ASPsider reduseres.
- Det kan lages verktøy for å legge inn egne typer slik at brukeren kan gjøre det selv.
- I Tilstandsbasert vedlikehold er samplingsfrekvensen en viktige parameter (kap 2.2). Når alarmgrenser sjekkes fra vedlikeholds VI'en (maintenance.vi) tas det ingen hensyn til samplingsfrekvensen, det er bare satt en generell samplingsfrekvens for alle vedlikeholdstag. Her kan VI'en forandres etter at det er laget ny kolonne og inntastingsfelt for samplingfrekvensen.
- Hvis det ikke er satt alarmgrenser til en tag bør det lages en funksjon som tar opp Tag Manageren automatisk, slik at grenser kan settes.

8 KONKLUSJON

I dette prosjektet er det designet og utviklet deler av et generelt driftstøttesystem for enkelt å kunne håndtere tilstandsbasert vedlikehold av prosessutstyr.

Utfra litteraturstudie ble det i fellesskap med veiledere på CARDIAC besluttet at systemet skulle ha to deler, et for å beregne nye utgangsverdier (tag) og et for tilstandsovervåkingen av vedlikeholdstag. Deretter ble det satt opp en spesifikk kravspesifikasjon over systemet.

I henhold til kravspesifikasjonen er det for del I implementert egne Microsoft SQL databasetabeller for håndtering av beregnede tag. ASP websider benyttes for å opprette nye og editere eksisterende beregnede tag. For å starte å utføre beregninger av alle innregistrerte beregnede tag benyttes LabVIEW applikasjoner.

Til vedlikeholdstagene i del II brukes Microsoft SQL databasetabeller både for lagringa av taginformasjon historikk til tagen. For å håndtere all data tilhørende en vedlikeholdstag benyttes ASP sider til innregistring av nye vedlikeholdstag og editering av eksisterende. For å håndtere selve tilstandsovervåkingen benyttes en egen LabVIEW applikasjon. Det er til slutt brukt ASP sider for å kunne se på sanntids- og historisk data knyttet til en vedlikeholdstag.

Alle databaser, ASP sider og LabVIEW applikasjoner er implementert på en IMATIS testserver hos CARDIAC.

Systemet er testet ut og verifisert mot en simulert varmeveksler. Alle implementerte applikasjoner samt integrasjon mot IMATIS fungerte i henhold til kravspesifikasjon og intensjonen i oppgaven.

REFERANSER

Litteratur

1. Barron, Ron: Engineering condition monitoring, Practice, Methods and Applications. Addison Wesley Longman Inc.,New York, 1996.
2. Homer Alex, Sussman Dave, Francis Brian: Active Server Pages 3.0 professional, Wrox Press Ltd.Arden House, 1999.
3. Juvik, J Sigurd: Operatørstøttesystem for inndamperanlegg, HiT/TF- Sivilingeniør utdanningen, 1996.
4. Kvitnes, Marius: White paper IMATIS™, CARDIAC AS, Porsgrunn, 2001.
5. National Instruments: LabVIEW Basics I Course Manual, October 2000 Edition, National Instruments Corporation, 2000.
6. National Instruments: LabVIEW Basics II Course Manual, September 2000 Edition, National Instruments Corporation, 2000.
7. National Instruments: LabVIEW Datalogging and Supervisory Control Module, October 2000 Edition, National Instruments Corporation, 2000.
8. Phillips, Lee Anne: HTML 4 practical, Que Corporation, Indianapolis, 1999.
9. Reselman, Bob: Active Server Pages 3.0 by example, Que Corporation, Indianapolis, 2000.
10. Styring av varmeveksler i AIM2000 og opperatørstøttesystem i ASSETT, Rapport fra 3. semesters undervisningsprosjekt høsten 2002, SIV-9-01 ved HiT, TF.
11. Thore,-: © CARDIAC as 1-1 Imatis Evaluation Guide, 2001.
12. Warberg, Helge: Prosessutstyr og vedlikehold, Generell beskrivelse av vedlikehold i prosessindustrien, 1. utgave 1. opplag, Yrkesopplæring ans, Oslo, 1996.

Informasjon hentet fra Internett linker under er lagt i vedlegg 6 på en CD plate.

Internett linker

A. AMS http://www.emersonprocess.com/optimize/
B. Honeywell Alert Manager http://www.iac.honeywell.com/
C. OPC Foundation http://www.opcfoundation.org/
D. National Instruments http://www.ni.com/datasocket/
E. ASP101, http://www.asp101.com
F. W3schools, http://www.w3schools.com/

VEDLEGGSLISTE

1. Hovedoppgaveteksten fra CARDIAC.
2. Databasetabeller til del I
3. Databasetabeller til del II
4. ASPkode
5. LabVIEWkode
6. CD med kopier fra Internettsider til andre vedlikeholdssystemer
7. Skjermbilder fra uttesting av system