Instrumentation for High Availability Systems

Av

Alf Tore Håland

Vermund Lea

Hovedoppgave til Mastergraden i

Informasjons- og Kommunikasjon Teknologi

Høgskolen i Agder

Grimstad, august 2004

Sammendrag

I denne oppgaven har vi sett på ulike industri standarder for instrumentering. Vi har utviklet en instrumenteringsteori for instrumentering av systemer som krever høy tilgjengelighet. Igjennom et teknologistudium har vi sammenlignet en rekke teknologier mot instrumenteringsteorien.

Vi har laget en prototyp for å vise konseptene i instrumenteringsteorien. Denne prototypen bruker ”Enterprise Instrumentation Framework” som er et applikasjonsgrensesnitt, som utvider ”Windows Management Instrumentation” eksisterende hendelse, logging og sporings mekanismer allerede innebygd i Windows. ”Enterprise Instrumentation Framework” gjør det mulig for utviklere å produsere egne forvaltningsapplikasjoner, som kan overvåke og styre prosesser og ressurser, basert på interne instrumenteringsdata i Windows.

Vi har prøvd å bevise følgende påstander for oppnå høyere tilgjengelighet:

- Ved å innføre instrumentering er det mulig å oppnå en høyere tilgjengelighet, ved at gjennomsnittstiden det tar å få et system opp igjen går ned.
- Ved å innføre instrumentering gjør oss i stand til å planlegge og utføre mottiltak som kan unngå avbrudd.

Den beste instrumentering løsningen er en kombinasjon av ekstern og intern instrumentering. Intern instrumentering kan eksportere prosessinformasjon fra interne prosesser eksternt gjennom ”Enterprise Instrumentation Framework” til egne utviklete forvaltningssystemer.

Forord

Denne oppgaven var skrevet i samarbeid med Sense Intellifield og Høgskolen I Agder avdeling Grimstad som en del Mastergraden i faget IKT6400 (Informasjon og Kommunikasjon Teknologi). Arbeidet er utført i perioden januar 2004 til august 2004.

Vi vil takke våre veiledere visepresident Rune Skarbø og senior programvare ingeniør Vigebo Endresen ved Sense Intellifield for oppgaveideen og veiledningen gjennom oppgaven. Vi vil også takke våre veiledere høgskolelektor Arne Wiklund og professor Andreas Prinz ved Høgskolen I Agder for verdifull hjelp og støtte gjennom oppgaven.

Grimstad, august 2004

Alf Tore Håland

Vermund Lea

Innholdsfortegnelse

Figurliste

Tabelliste

1 Introduksjon

Instrumentering er viktig for å gi programoperatørene en bedre støtte til forvaltningen av de kjørende prosessene i et applikasjonsmiljø ved å gi bedre oversikt over status og tilstand til prosessene. Dette gjøres ved å tilegne seg informasjonsdata om prosessene som kjøres i applikasjonen, behandle disse instrumentasjonsdataene, gjøre instrumentasjonsdataene tilgjengelig, og til slutt utføre forhåndshandlinger. Instrumenteringen vil hjelpe operatørene til å se kritiske situasjoner i programmiljøet, og kan da ta avgjørelser og aksjoner for applikasjonsprosessene før programvaren eventuelt feiler og forårsaker avbrudd. Instrumentering kan bli brukt til å støtte forvaltning ved å gjøre det mulig å endre kjørende applikasjoner. I systemer der det er viktig å oppnå høy tilgjengelighet på programvaren er det viktig at instrumenteringen og forvaltningen av applikasjonsprosessene er så effektiv som mulig, og bruker så lite resurser som mulig for ikke å forstyrre hovedfunksjonen til applikasjonen.

1.1 Klassisk eksempel på bruk av instrumentering

I biler er det en hastighetsmåler som er forbundet med kardangen som forteller føreren av bilen hvor fort han kjører. Ved å bruke pedalene for hastighet og brems er føreren i stand til å kontrollere hastigheten. Dette gjør føreren i stand til å forvalte bilen ved å justere hastigheten. Hastighetsmåleren er en instrumentering som gir direkte opplysninger om hastigheten. En ekstern hastighetsmåler ved bruk av dopplerradar enten på bil, ved vegkant eller i hånden på en politimann, er et klassisk eksempel på instrumentering som vil gi en eksakt fysisk hastighet. Føreren kan selv forutsi hastigheten ved å betrakte omgivelsene rundt seg når han kjører. Instrumentering kan brukes ved at bilen er tilført hastighetsregulering som selv sørger for ønsket eller konstant hastighet. Høyere tilgjengelighet kan oppnås ved at bilen er utstyrt med drivstoffmåler, oljetrykkmåler, bremseklossnivå og så videre, for å unngå driftstans på øde landeveier. Dette

vil hjelpe føreren å unngå lange driftstanser, og istedenfor ta korte kontrollerte og planlagte driftstanser. Instrumentering har sørget for høyere tilgjengelighet på bilen gjennom innsamling, bearbeiding og presentering av tilstander om bilen til føreren.

1.2 Sense Intellifield

Sense Intellifield, som vi i utgangspunktet fikk oppgaven fra, er en global leverandør av integrerte automasjon og informasjon systemer for borerigger. Sense Intellifield har utviklet et system som heter SiteCom for å innhente, distribuere, lagre og overvåke data i forbindelse med borevirksomhet. SiteCom er påtenkt å være et sentralt sted hvor andre systemer kan koble seg til for å sende og motta sanntidsdata og historikkdata fra boreriggene. SiteCom har krav om å være tilgjengelig kontinuerlig 24 timer per dag 7 dager i uken. Dette gir høye krav til instrumenteringen for å ikke forstyrre hovedprosessene i SiteCom. I dag har Sense Intellifield en egenutviklet instrumenteringsløsning SiteCom Central for SiteCom applikasjonen. SiteCom Central bruker standarder som WITSML (Well site Information Transfer Standard Markup Language), SOAP (Simple Object Access Protocol - som er web basert og plattformuavhengig), og XML (Extensible Markup Language).
[Ref: www.sensetech.no]
SOAP bruker språket XML for å definere formatet og protokollen HTTP (HyperText Transfer Protocol) for adresseringen av applikasjonene.
[Ref: http://www.hardware.no/nyheter/juni03/soap_1_2.html]
Sense Intellifield tok kontakt med Høgskolen I Agder for å få én studie av instrumentering i hensyn på deres instrumenteringsløsning i forhold til EIF [7].

1.3 Oppgave definisjon

Oppgaven er definert i engelsk som følgende.

Instrumentation is an important aspect of commercial grade software systems requiring high availability. Service failures can be crucial for the business and must be avoided. This can be done by adding run-time monitoring. Instrumentation is needed to avoid system failure and downtime. Instrumentation capability could help tuning the system to run more reliable.

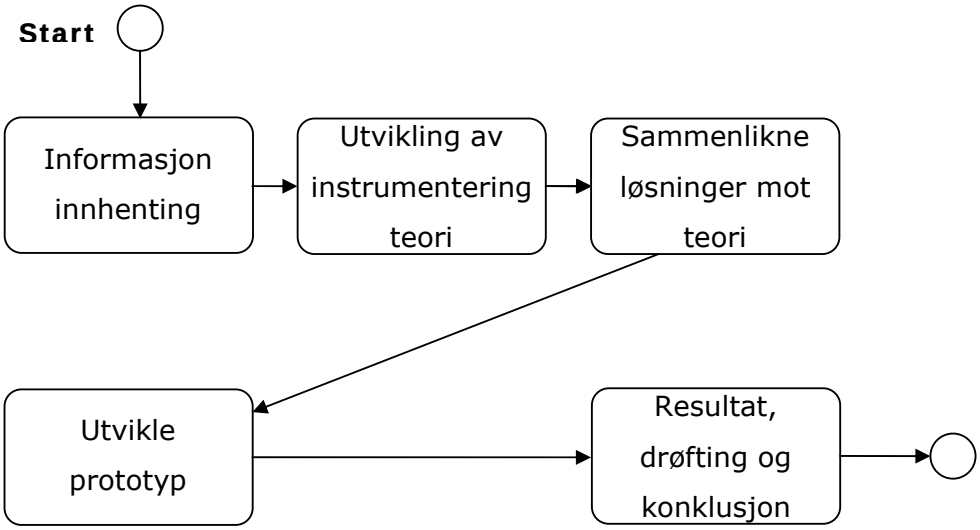
A study about common practices and methodologies on how to instrument high availability systems will be done. This may include multiple platforms and different programming languages. The study will result in a theory how ideal instrumentation should look like.

We will discuss some relevant existent techniques for instrumentation, including Enterprise Instrumentation Framework which is a part of the Visual Studio .NET 2003 release. These techniques should be compared to the theory developed during the study.

If possible a prototype will be developed to demonstrate some of the concepts involved.

1.4 Arbeidsmetode

I denne oppgaven vil vi først starte med begrepsforståelse ved å innhente informasjon om instrumentering, hva som menes med høy tilgjengelighet, og hvordan man kan oppnå høy tilgjengelighet ved hjelp av instrumentering. Vi vil se på hvordan vanlig praksis og metoder er brukt i eksisterende løsninger i dag inkludert ”Enterprise Instrumenting Framework” EIF. I denne fasen skal vi utvikle vår egen instrumenteringsteori. Deretter vil vi sammenligne de instrumenteringsteknikkene vi har sett på sammen med teorien. Til slutt vil vi utvikle en prototyp som demonstrerer noen av instrumenteringsteknikkene vi har sett på. Arbeidsmetoden er illustrert i figuren under.



Figur 1-1 Arbeidsmetode

1.5 Begrensninger i oppgaven

I denne masteroppgaven vil vi se på instrumentering og viktigheten med instrumentering for å oppnå høy tilgjengelighet. Vi vil kun se på programvare og ikke maskinvare. Da det finnes så mange operativsystemer vil vi i hovedtrekk holde oss til Windows når vi ser på løsninger og prototyp.

1.6 Rapport struktur

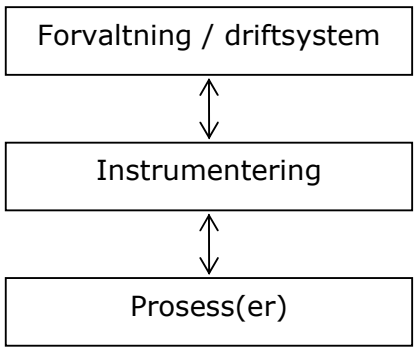
Det første kommende kapittel 2 vil vi skrive om instrumentering og høy tilgjengelighet. Under Kapittel 3 vil vi sette opp en instrumenteringsteori. I kapittel 4 vil vi sammenligne tilgjengelig teknologi vi har sett på, mot teorien som ble utviklet i kapittel 3. Kommende kapittel 5 vil vi utvikle en prototyp som skal ta for seg eksempler for de 6 underpunktene til instrumenteringsteorien. I kapittel 6 vil vi skrive om resultater. Under kapittel 7 vil vi diskutere instrumenteringsteorien og argumentere for de valgene vi tok i kapittel 5 om prototypen. Siste kapittel 8 vil inneholde konklusjonen.

2 Instrumentering og høy tilgjengelighet

I dette kapittelet vil vi belyse instrumenteringsstruktur, ulike instrumentering teknikker som ekstern, intern, automatisk og manuell instrumentering og forskjellen på disse. Deretter vil man se på tilgjengelighet, og alternativ måte å regne ut tilgjengelighet på. Til slutt vil vi se på hva som kreves av instrumenteringen for å opprettholde eller oppnå høy tilgjengelighet.

2.1 Instrumenteringsstruktur

Instrumentering inngår som et mellomledd mellom et styringssystem (ofte kalt forvaltning eller driftsystem) og en eller flere prosesser. Se figur under. Instrumenteringen har da som hensikt å være et bindeledd mellom et driftsystem og ulike prosesser. Informasjon fra en eller flere prosesser skal da formidles vha instrumentering fra prosessene til driftsystemet. Driftsystemet henter inn vha instrumentering informasjonen fra prosessene, analysere informasjonen og kan til slutt bruke denne informasjonen til å ta avgjørelser angående prosessene. Graden av instrumentering vil da avgjøre hvor godt et driftsystem kan forvalte de ulike prosessene.



Figur 2-1 Instrumenteringsstruktur

2.2 Instrumenteringsteknikker

Vi har sett på fire ulike instrumenteringsteknikker som ekstern, intern, automatisk og manuell instrumentering. Vi har funnet ut at ekstern og intern instrumentering kan være manuell eller automatisk instrumentering. Tabellen under viser eksempler på instrumenterings kombinasjoner.

Instrumenterings-teknikker	Automatisk instrumenteringsteknikk	Manuell instrumenteringsteknikk
Ekstern instrumentasjon	- Windows oppgavebehandling. - Sun Management Center	- Eksterne driftsystemer som for eksempel Open View fra Hewlett-Packard.
Intern Instrumentasjon	- Implementeres vha en kodefortolker. - Implementeres vha en er kodekompilator.	- Egenutviklet kode internt i applikasjonen for instrumentering. - Kode i applikasjon kaller standardisert kode som for eksempel EIF.

Tabell 2-1 Eksempler på instrumenteringsteknikker

2.2.1 Ekstern Instrumentering

Ekstern instrumentering gjøres ved å overvåke miljøet til applikasjonen. Det er ikke mulig å overvåke de interne prosessene i applikasjonen, men det er mulig å overvåke resurser som applikasjonen bruker. Dette er mulig da applikasjonen kjører i et miljø eller skall som er styrt av et operativsystem, og har en rekke forhåndsdefinerte funksjoner å jobbe med. Dette gjør operativsystemet i stand til å samle inn begrenset informasjon av de kjørende prosesser. Ved denne begrensningen er det ikke mulig å oppdage interne feil i applikasjonen. Driftpersonellet må ta aksjon manuelt dersom noe unormalt skjer med disse parametrene. Eksempler kan være heng i prosessen, høy prosessorbruk, eller at minneforbruket til prosessen øker uten rimelig grunn. Et tredje parts verktøy kan brukes til å få tilgang til de samme dataene da Windows har et grensesnitt til dette. Og ved å overvåke og analysere disse parametrene er det mulig å sende meldinger i form av e-post, tekstmelding osv.

2.2.2 Intern instrumentering

Intern instrumentering er når applikasjonen selv eksporterer informasjon om interne prosesser. Dette gjøres ved at det tilføres instrumenteringskode som avdekker hendelser, statuser og variabler i henhold til de interne prosessene. Instrumenteringskoden har da til hensikt å gjøre denne informasjonen tilgjengelig utenfor applikasjonen. Disse dataene eller informasjonen er da viktig under overvåking av en applikasjon, for å kunne fastslå tilstanden til de interne prosessene som kjører i applikasjonen. Dette gir da oss en unik mulighet til å oppdage og fastslå mer presist når feil oppstår. Med intern instrumentering er det mulig å tilføre mer tilstandsinformasjon om hendelsene og dermed støtte ytelsesanalyse, feilanalyse, problemdiagnostikk og applikasjonskonfigurasjon. Intern instrumentering vil da bedre besvare spørsmål angående: applikasjonsstatus, hva gikk galt, hvor og ofte det går galt, prosessbelastning, når prosessen jobber mest og minst, flaskehals i applikasjonen og hvor trang flaskehalsen er.
Ved implementering av intern instrumenteringskode i applikasjoner kan man utvikle en egen standard med egne definerte målevariabler eller man kan bruke ett applikasjonsgrensesnitt API (fra engelsk: Application Program Interface) fra andre eksisterende driftsystemer. Instrumenteringskode kan også implementeres automatisk vha en fortolker eller kompilator. Fordelen ved å bruke standard applikasjonsgrensesnitt er at noen utviklingsverktøy har ferdig definerte klasser med instrumenteringskode. Dermed har de også klasser som er forberedt til bearbeiding av disse instrumenteringsdataene. En annen fordel er at driftprogramvare som er ferdig utviklet og tilpasset disse standardene kan bruke disse instrumenteringsdataene. Disse fordelene gjør instrumenteringen mer fleksibel og pålitelig. Ulemper ved å bruke en eksisterende grensesnitt er at man blir låst til standarden, og dermed hvilke informasjon som kan eksporteres. Fordelen med egen standard er at man selv bestemmer hvilken informasjon som skal eksporteres. Ulempen med egen standard er lang utviklingstid i forhold til bruk av et ferdigdefinert grensesnitt.

2.2.3 Automatisk instrumentering

Automatisk instrumentasjon er når instrumentasjonen er ferdig tilrettelagt. Windows oppgavebehandling i Microsoft Windows er et veldig godt eksempel på automatisk ekstern instrumentasjon. Automatisk instrumentering kan også bli utført internt når en applikasjon er klargjort av en kompilator, eller når applikasjonen blir startet. Dette skjer uten innblanding fra brukeren. Feilsøking av en applikasjon under utvikling er en form for automatisk instrumentering, fordi et moderne feilsøkingsprogram gjør det mulig for utvikleren å gå trinnvis gjennom kildekoden under kjøring. Trinnvis gjennomgang av kildekoden er viktig for å oppdage problematiske områder i kildekoden. Denne feilfinningen er svært omfattende, tar veldig lang tid og gir ikke et helhetsoverblikk av prosessen. Derfor er automatisk instrumentering ofte ikke nok til å brukes alene.

2.2.4 Manuell instrumentering

Manuell instrumentering er når programutvikleren lager all instrumenteringskode selv. Dette er en stor utfordring for programutvikleren da ofte brukeren av driftprogrammet og programmereren av programvaren har forskjellig syn på hva som er viktig. Manuell instrumentasjon gir programutvikleren total kontroll over hva som skal instrumenteres. Utfordringen til programutvikleren er å oppdage hva som er viktig å ha med i en instrumenteringsløsning. En annen ulempe er dersom programmet skal endres under utviklingstiden så må også instrumenteringskoden endres.

2.3 Høy tilgjengelighet

Høy tilgjengelighet på systemer betyr systemer med lav avbruddstid. Høyere tilgjengelighet kan oppnås med god instrumentasjon av en applikasjon for å oppnå lavere avbruddstid. Med god instrumenteringsteknikk menes instrumenteringsteknikker som forstyrrer minst mulig de sensitive kjørende prosessene i applikasjonen. Det er en stor utfordring da systemer med høy

tilgjengelighet krever at avbruddstiden er så liten som mulig. Den optimale tilgjengelighet er høyst mulig en eller en kombinasjon av de fire instrumenteringsteknikkene nevnt over. Mennesker klassifiserer ofte tilgjengelighet som en prosent av den tiden et system er oppe og kjører. Å oppnå fullstendig oppetid på 24 timer i døgnet 7 timer i uka (herfra kalt 24/7) er mest ønskelig og nesten umulig. Det vil alltid være planlagt avbruddstid av applikasjonen på grunn av driftsmessige årsaker.

2.3.1 Beregning av tilgjengelighet

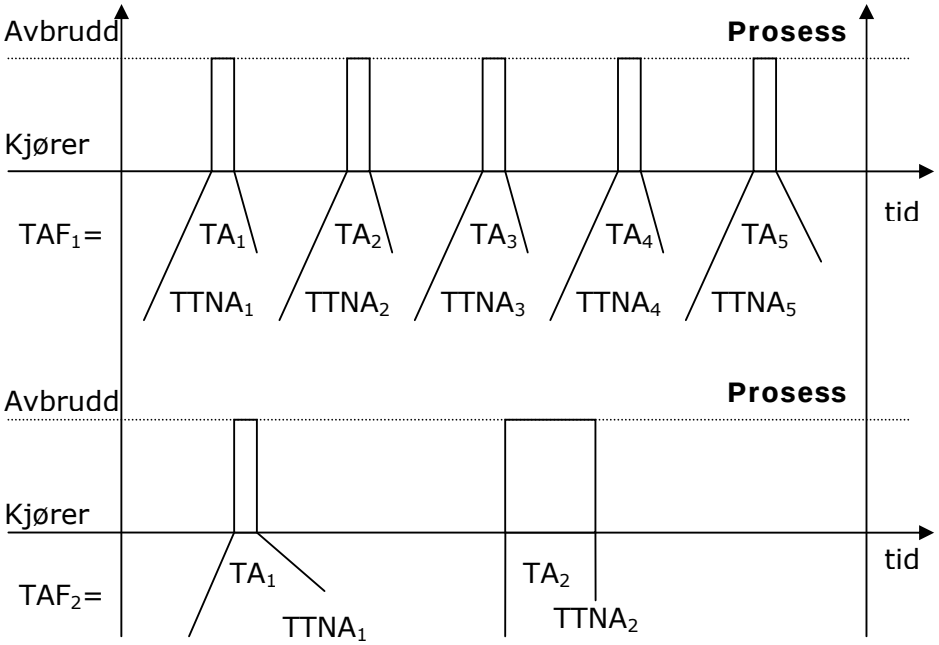
Forskjellige datasystemer har forskjellige krav til tilgjengelighet. På bakgrunn av den kalkulerte oppetiden til en applikasjon kan man si hvilken tilgjengelighetsklasse applikasjonen tilfredsstiller. Dette gjøres ved å telle antall 9'ere i den utregnete oppetiden i prosent. Tabellen under viser i en periode på et år sammenhengen mellom kalkulert oppetid, avbruddstid, tilgjengelighetsprosent og tilgjengelighetsklasse.

Tilgjengelighets-klasse [niere]	Tilgjengelighet i prosent [%]	Total oppetid [dager]	Total avbruddstid [tidsenhet]
2 niere	99 %	361,35 dager	3,65 dager
3 niere	99,9 %	364,64 dager	8,76 timer
4 niere	99,99 %	364,96 dager	52,6 minutter
5 niere	99,999 %	364,9964 dager	5,26 minutter
6 niere	99,9999 %	364,9996 dager	31,5 sekunder

Tabell 2-2 Tabell for tilgjengelighetsklasse

Total oppetid er den akkumulerte tiden over en periode, og ved å dividere den akkumulerte oppetiden på den totale tidsperioden får man tilgjengelighet i prosent. Og ved å telle antall niere får man tilgjengelighetsklassen. I motsatt tilfelle kan man regne ut hvor mye avbruddstid som kan aksepteres fra en tilgjengelighetsklasse. Generelt er tilgjengelighet kalkulert basert på total avbruddstid til et system over en periode, men kan også bli utrykt ved å kalkulere hvor lang tid det tar å få systemet opp igjen. Dette har mye å si for en bruker av systemet siden et langt avbrudd er mye mer irriterende enn mange veldig korte avbrudd over tid som ikke registreres av brukeren, selv om total akkumulert avbruddstid er like lang når avbruddene akkumuleres.

Figuren under viser løpet til 2 prosesser over en periode på 100 tidsenheter med samme akkumulerte avbruddstid over tid.



Figur 2-2 Figuren viser oppetid og avbruddstid for 2 prosesser

- TAF_1 er sum antall feil prosess 1, og TAF_2 er sum antall feil prosess 2.
- TA_n er tiden for en enkelt feil.
- $TTNA_n$ er tiden mellom to feil.

Prosess 1 har 5 like lange tidsavbrudd (TA) på 0,2 tidsenheter til sammen 1 tidsenhet. Prosess 2 har et tidsavbrudd (TA_1) på 0,2 tidsenhet og et tidsavbrudd (TA_2) på 0,8 tidsenhet til sammen 1 tidsenhet. Dette gir en total avbruddstid (ΣTA) til 1 tidsenhet på begge prosessene og gir da lik tilgjengelighet på:

Tilgjengelighet *= Oppetid / Totaltid, ΣTA = 1, Totaltid = 100*
= (Totaltid - ΣTA)/ Totaltid
= (100 - 1) / 100 = *99,000 %*

Derfor er det viktig å også vite hvor mange avbrudd det er over en periode. Flere avbrudd med samme akkumulerte avbruddstid vil gi en betre tilgjengelighet for en bruker.

2.3.2 Alternativ utregning av tilgjengelighet

Oppetid er en vanskelig variabel å regne, da du hele tiden må legge sammen den totale avbruddstiden som igjen trekkes fra totaltid. Derfor vil vi bruke en alternativ måte som inkluderer avbruddstiden i den totale tiden. Dermed får vi formelen:

$$Tilgjengelighet_{ALTERNATIV} \approx Totaltid \,/\, (\, Totaltid + Avbruddstid \,)$$

Denne formelen vil gi oss følgende resultat fra figur 2.2

$$Tilgjengelighet_{ALTERNATIV} \approx 100 \,/\, (\, 100 + 1) \approx \underline{99{,}010\ \%}$$

Vi ser at resultatet er omtrent det samme
Vi skal nå bevise at denne formelen kan brukes for utregning av tilgjengelighet for systemer med høy tilgjengelighet. Det vil si at avbruddstiden går mot null. Vi tar først og dividerer formelen på AF (antall feil) for å kunne bruke gjennomsnittsverdier:

$$\approx (Totaltid \;\; / \; Antall\ feil) \,/\, (Totaltid \,/\, Antall\ feil + Avbruddstid \,/\, Antall\ feil \,)$$

, Totaltid / Antall feil er Gjennomsnittstid mellom avbrudd (GMA) og
Avbruddstid / Antall feil er Gjennomsnittsavbrudd (GA)

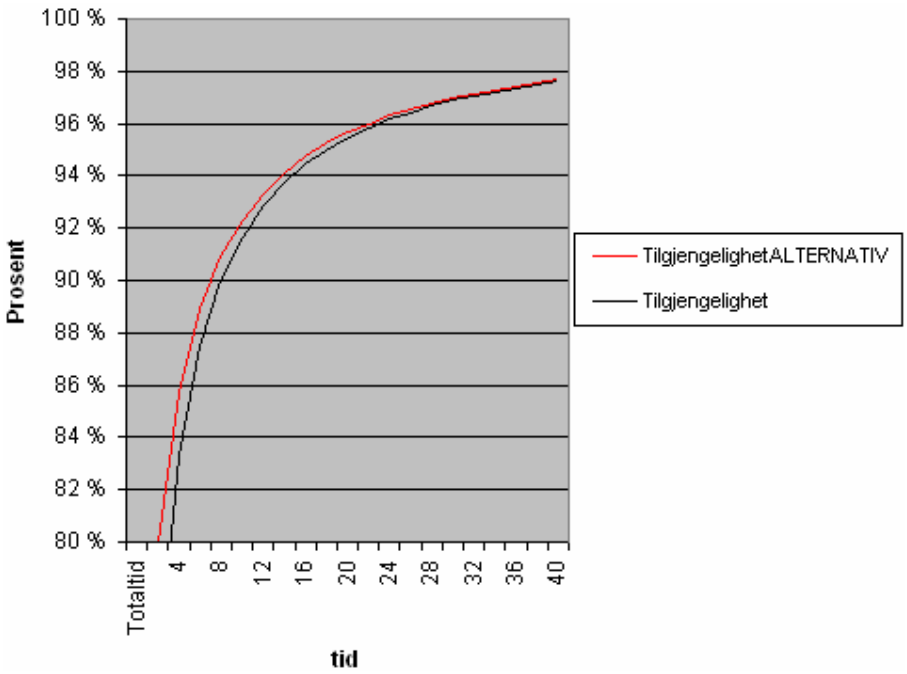
Dette gir oss formelen:

$$Tilgjengelighet_{ALTERNATIV} \approx GMA \,/\, (GMA + GA)$$

Fordelen med denne formelen er at vi kan unngå å omberegne total oppetid og total avbruddstid, og bare omberegne GMA og GA når en ny feil oppstår. GMA og GA kan utregnes følgende når ny feil oppstår:

GMA_{n+1} = ($Totaltid_n$ + $TTNA_1$) / (AF_n + 1), (n+1) er ny feil og n er siste feil.

La oss nå tenke oss at tiden går og ingen nye feil oppstår. Hva vil da skje med tilgjengeligheten?



Figur 2-3 Opprinnelig og alternativ beregningen av tilgjengelighet.

Fra figuren over ser vi at den alternative formelen for beregning av tilgjengelighet er litt høyere enn opprinnelig formel. Dette er som sagt tidligere da summen av avbrudd er medberegnet i totaltiden, og vil da gi en litt høyere tilgjengelighet. Den alternative formelen tangerer allerede ved 97 % tilgjengelighet. Vi kan da konkludere at den alternative formelen for tilgjengelighet kan brukes for tilgjengelighetsklasser 2 niere og høyere.

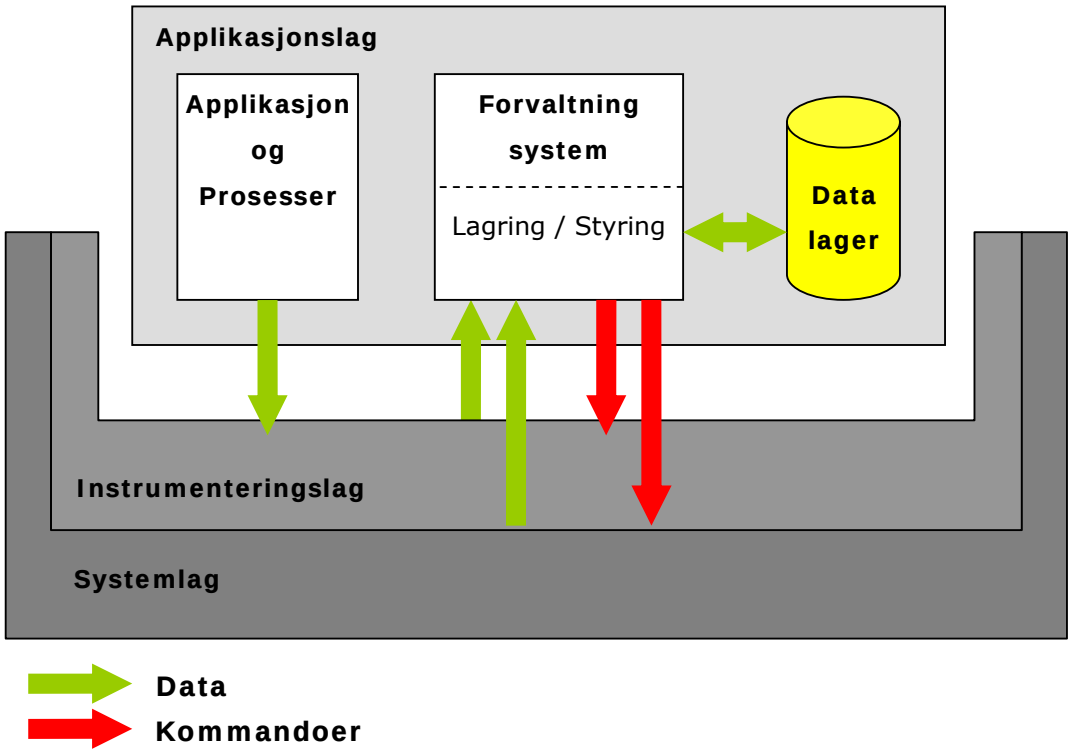
3 Instrumenterings teori

I dette kapittelet vil vi bygge våres egen instrumenteringsteori som vil innbefatte begreper fra kapittel 2. Utfordringen i dagens applikasjoner er at applikasjoner i dag er kompliserte, webbaserte, distribuert og har krav om høy tilgjengelighet. Den ideelle instrumentering vil benytte seg av både ekstern og intern instrumenterings metoder. Hvor ekstern instrumentering vil gi oss applikasjonens status, og hvordan den kjører i operativsystemets miljø. Intern instrumentering vil gi oss informasjon om de interne prosessene i applikasjonen. Målet med instrumenteringen er å forutse driftstans og ta aksjoner for unngå driftstans for å oppnå høyere tilgjengelighet. Det er også et ønske eller krav å konfigurere applikasjonen uten eller et minimum av driftstans. Som en tommelfingerregel kan en si at instrumentering ikke bør oppta mer en 10 % av den totale prosesseringskraften.

3.1 Generell instrumenterings arkitektur

Instrumenteringsarkitekturen vist på figur 3.1 nedenfor ble utviklet ved å se på mange eksisterende instrumenterings løsninger. Alle applikasjoner, prosesser, databaser og lignende som utgjør applikasjonslaget ligger øverst i figuren, og disse kjører i systemlaget som består av operativsystemet og maskinvaren. Instrumenteringslaget ligger mellom systemlaget og applikasjonslaget. Instrumenteringslaget har som oppgave å være et grensesnitt for forvaltningssystemet mot enten applikasjonslaget eller systemlaget. Instrumenteringslaget henter inn instrumenteringsdata enten fra applikasjonslaget eller systemlaget, og sender all instrumentasjonsdata til forvaltningssystemet som skal bearbeide og lagre instrumentasjonsdataene. Forvaltningssystemet kan også hente data fra filer i systemlaget. Alle instrumentasjonsdata skal hjelpe forvaltningssystemet i driften av prosesser og instrumentasjonslaget, for å fastslå statuser til prosessene i applikasjonslaget. Intern instrumentering i applikasjoner kan legges til prosesser i applikasjoner. Dette gjør applikasjoner i stand til å utføre ekstra oppgaver, som å sende instrumentasjonsdata om interne prosesser til

instrumentasjonslaget. Instrumenteringslaget kan få kommandoer fra forvaltningssystemet som sier hvilke instrumenteringsdata som skal sendes til forvaltningssystemet. Forvaltningssystemet har deretter som oppgave å ta aksjoner på de lagrete instrumenteringsdata, og presentere instrumenteringsdata til bruker av forvaltningssystemet. Aksjoner kan være; å endre på instrumentasjonslaget ved å gi beskjed om hva som skal overvåkes, endre på konfigurasjonen til prosesser via instrumentasjonslaget, starte og stoppe prosesser gjennom operativsystemet, eller sende meldinger til driftspersonell. Driftspersonell kan også endre på instrumenteringslaget eller styre prosesser gjennom presentasjonsdelen av forvaltningssystemet.



Figur 3-1 Generell instrumenterings arkitektur

3.2 Fremgangsmåte for å innføre instrumentering

Instrumentering innføres som del av et system fordi det eksiterer et behov for å vite hvordan en applikasjon prosesserer under drift. Først og fremst for å forhindre uønsket avbrudd som kan vise seg å være svært kostbare for

bedriften. Som eksempel ble det estimert i en undersøkelse [20] utført i år 2000 av Contingency Planning Research at Amazon ville tape 180 000 dollar per time driftstans. For å innføre den riktige instrumenteringsmetoden for instrumenteringen er det viktig å ha i bakgrunnen problemstillingen, hva vi vil med instrumenteringen, og hvilke forventninger og krav som er bundet opp til instrumenteringen. Dette kan vi finne ut ved å overvåke systemet og finne ut når det begynner å oppføre seg problemetisk. Deretter kan man fastslå hvilke tiltak som må innføres for å unngå driftstans, og dermed øke tilgjengeligheten til applikasjonen. Dersom det skulle oppstå et avbrudd skal instrumenteringen gjøre det lettere å få systemet opp igjen. Avbrudd kan skyldes stor trafikk i applikasjonen eller feil. Trafikken er det vanskelig å gjøre noe med da dette ligger i programdelen av applikasjonen, men vi kan innføre varslinger om status på trafikken. Feil finnes i alle større applikasjoner uansett hvor lang tid det er blitt brukt på utvikling og testing. Derfor er det viktig å samle inn nok informasjon slik at når en feil oppstår vil det være så enkelt som mulig å finne årsaken til feilen, og dermed få applikasjonen raskt opp og kjøre igjen. For å hindre uønsket avbrudd ønsker vi å vite om systemet er friskt eller ikke. Med friskt menes om systemet oppfører seg normalt i forhold til hva som er forventet. Hensikten med å skape seg et bilde av normal oppførsel til applikasjonen er å identifisere unormal oppførsel. Utfordringen vil være å definere normal oppførsel. Ved innføring av en instrumenteringsløsning må følgende punkter gjennomføres:

1. *Identifisering*
2. *Innsamling*
3. *Instrumentering*
4. *Systematisere instrumentasjonsdata*
5. *Aksjonere på instrumentasjonsdata*
6. *Presentere instrumentasjonsdata*

Det er veldig viktig under hele utviklingsfasen av instrumenteringsløsningen å ha fokus klart i minne på hva vi vil med instrumenteringen. Dette gjør det enklere å velge ut den informasjonen vi har behov for, og hvilken

informasjon vi med rimelig grunn kan avvise. De følgende underkapitlene går nærmere inn på punktene over.

3.3 Identifisering

I identifiseringsfasen må vi skape oss et helhetsblikk over hva som skal instrumenteres. Dette gjøres først ved å identifisere systemlaget og applikasjonslaget i figur 3.1. Vi vil da starte med å undersøke overvåkingskilder i systemlaget og instrumentasjonslaget i figur 3.1.

1. *System lag*
2. *Applikasjonslag*

3.3.1 System Lag

Her må man identifisere operativsystemet og maskinvare. Deretter å identifisere hvilke instrumentasjonsdata som automatisk blir generert av operativsystemet, og hvilke instrumentasjonsdata vi manuelt må implementere. Videre hvilke eksterne instrumenteringsløsninger vi har mulighet til å betrakte prosessene utenfra i miljøet. I et moderne Windows operativsystem vil du ikke få direkte tilgang til maskinvarebaserte ressurser. Microsoft har valgt å gjøre det slik for å forhindre krasj forårsaket av dårlige skrevet programmer. For å få tilgang til ressursene benyttes HAL (fra engelsk: hardware abstraksjons lag). Men dette gjør igjen at en er avhengig at HAL støtter alle de funksjonene som er tilgjengelig i maskinvaren. Instrumenteringsdata vi kan få fra systemlaget er maskinvare baserte resurser, og referanseverdier om prosesser som prosessnavn, brukerobjekter, tellere og så videre.

3.3.2 Applikasjonslaget

I Applikasjonslaget må vi identifisere hvilke applikasjoner, prosesser, tjenester, biblioteker, filer, databaser og så videre, som skal overvåkes og eventuelt instrumenteres. Deretter undersøkes om hvilke, om det er noen, prosesser som har tilgjengelig kildekode. Dermed kan vi fastslå i hvilke prosesser det er mulig å innføre intern instrumentering. Det er en stor utfordring da applikasjonen kan bestå av mange ulike prosesser og

prosesstyper. Vi har prosesser som går i bakgrunnen, synlige prosesser med brukergrensesnitt, prosesser som kan ligge latent, og prosesser som sørger for dataflyten mellom prosessene. Utfordringen er å finne rett instrumenteringsteknikk for rett prosess. Siden noen prosesser må betraktes som en sort boks som vi ikke har mulighet til kildekode, mens andre prosesser har åpen kildekode.

3.4 Innsamling

Selv en liten applikasjon kan generere store mengder informasjon og da er det viktig å tenke på hva som vil være nyttig for overvåking. Eksempelvis ved en feil vil det være nyttig å vite hvor og når feilen oppstod, og hva som forårsaket at feilen oppstod. Det er essensielt å være klar over hvilke data som skal samles inn, hvor data kan samles inn og når data skal samles inn før de systematiseres. Her er alt avhengig av hvor lite eller mye vi vil vite om prosessene. Se momenter nevnt under:

1. *Hvilke data skal velges*
2. *Hvor kan data samles inn*
3. *Når data skal hentes*
4. *Hvordan data skal hentes*

3.4.1 Hvilke data skal velges

Målet med å instrumentere er å få en dypere forståelse av applikasjonens tilstand. Vi kan identifisere tilstanden til et program ved å se på bruk av ressurser. En applikasjon som oppfører seg normalt vil konsumere en bestemt mengde ressurser basert på hvor stor last den har i øyeblikket og oppgaven den utfører. Derfor vil det være viktig å overvåke prosessorbruk, minnebruk, nettverksbruk og diskkapasitet. Responstiden er viktig for å vite tiden det tar for en prosess å utføre en bestemt oppgave eller funksjon. Dersom en applikasjon ikke frigjør ressursene etter bruk er det ofte et tegn på minnelekkasjer eller låsing av ressurser. Derfor bør minnebruk måles. Frekvens på feilmeldinger og årsak til feilmeldinger bør måles. Belastning kan måles ved å finne antall koblinger mot applikasjonen og hvor mye data som

overføres. I forbindelse med sikkerhet kan overvåking av brudd på sikkerhet måles.

3.4.2 Hvor kan data samles inn

Det finnes mange kilder hvor vi kan hente ut data om applikasjonens tilstand og miljøet rundt applikasjonen. Instrumenteringsdata kan hentes fra systemlaget eller instrumentasjonslaget. Noen data er tilgjengelig fra flere kilder som for eksempel prosessor tid, som kan hentes direkte fra intern instrumentasjon i applikasjonene via instrumentasjonslaget, selve instrumentasjonslaget eller systemlaget. Hvor det er mest hensiktmessig å hente data er vanskelig å si på forhånd ettersom de fleste systemer er ulike. Dette er noe som må avgjøres under innføring av instrumentering. Tabellen under viser steder hvor man kan samle inn data.

Lag	Instrumenteringsteknikk	Data
Systemlag	Datafiler, konfigurasjonsfiler, bibliotekfiler og systemkall	Prosessdata, systemdata.
Instrumentasjonslag	Intern instrumentasjonsdata fra applikasjoner.	prosessdata
	Ekstern instrumentasjonsdata fra systemlag.	Prosessdata, systemdata.
Applikasjonslag	Databaser	Prosessdata, systemdata.
	Prosessbelastning	Prosessdata

Tabell 3-1 Kilder for instrumentasjonsdata

3.4.3 Når data skal hentes

Tidspunktet for henting av data er prioritert avhengig av behov for aksjon, gyldighetstid og behov. Det vil si data som er av kort gyldighet må prioriteres først dersom disse skal tas aksjon på først. Men som oftest er data med kort gyldighetstid best å integreres over tid med gjennomsnittsverdier. Eksempler på kort gyldighetstid kan være prosessorbruk, som først er viktig over et mindre tidsintervall. Data med lang gyldighetstid er ikke tidspunktet så viktig, men dersom verdiene krever kort aksjonstid må de likevel prioriteres høyt. Kommer tilbake til dette under kapittel 3.7.

3.4.4 Hvordan data skal hentes

Instrumentasjonsdata kan hentes under forskjellige teknikker som manuell og automatisk. Manuell i denne sammenheng er hvor Forvaltningsapplikasjonen selv henter og kalkulerer de data som trengs. Automatisk menes prosesser som automatisk sender data i form av rådata eller ferdig kalkulerte data. Forvaltningsapplikasjonen kan også abonnere på meldinger fra lagene:

- Applikasjonslaget som sender interne meldinger.
- Systemlaget som sender eksterne meldinger.
- Instrumentasjonslaget som genererer meldinger fra innkommende meldinger. Disse kalles kalkulerte meldinger.

Forvaltningssystemet kan også kunstig stimulere prosessen med en kjent belastning for a anskueliggjøre prosesskapasitet ut fra en forventet respons. Dette krever kunnskap til inn og utverdier for prosessen.

3.5 Instrumentering

Instrumenteringslaget er litt flytende, da instrumenteringen kan foregå inne i prosessene, mellom applikasjonslaget og systemlaget, og internt i operativsystemet. Men hensikten her er å klargjøre hvilke eksterne instrumenteringsteknikker som finnes og hva de tilbyr av overvåking. Instrumenteringsgraden er avhengig fra hvor og i hvilken grad forvatningsapplikasjonen har behov for informasjon. Operativsystemet har som oftest ulike instrumenteringsstandarder bygget inn i operativsystemet. Dersom instrumenteringsdataene fra operativsystemet ikke er tilstrekkelig kan det implementeres et instrumentasjonslag mellom systemlaget og applikasjonslaget som vist i figur 3.1. Dersom heller ikke instrumentasjonslaget tilbyr nok instrumentasjonsdata, er det mulig å modifisere applikasjonen med intern instrumentasjon i prosessene. Men det sistnevnte er kun mulig dersom kildekoden til prosessene gjøres tilgjengelig. Det bør også undersøkes nærmere om instrumenteringsteknikken:

1. *Bruk av standarder*
2. *Skalerbar instrumentasjon*

3. Dynamisk instrumentasjon

3.5.1 Bruk av standarder

Ved innføring av instrumenteringsteknikker kan dette enten gjøres manuelt eller ved å bruke en standard. Manuelt menes i denne sammenheng å ”finne opp kruttet på ny”. Det vil si man definerer vår egen standard med egne definerte måledata. Fordelen med egen standard er at den kan skreddersys til behovet angående hurtighet og nøyaktighet, men tar lang tid å utvikle og vil være en egen løsning for den valgte applikasjon. Ved å velge en eksisterende standard kan man få de dataene en ønsker, men kan være for omfattende å behandle data etter behovet. Fordelen er fleksibiliteten ved bruk av standard. Se kap 2 angående standarder.

3.5.2 Skalerbar instrumentasjon

Over tid kan applikasjonen med de tilhørende prosesser endre seg og det blir nødvendig å legge til mer instrumentasjon. En applikasjon kan være fordelt over flere datamaskiner for belastningsfordeling og feiltoleranse, og representerer nye krav om skalerbar instrumentasjon. Derfor skal det være mulig å legge til instrumentering dersom applikasjonen senere blir fordelt over flere datamaskiner.

3.5.3 Dynamisk instrumentasjon.

Det bør være mulig å endre instrumenteringen senere under aksjonsdelen i forvaltningsapplikasjonen, uten å endre på kildekoden til applikasjonen. Samtidig bør det være mulig å konfigurere instrumenteringen mens applikasjonen kjører uten å ta en omstart av applikasjonen. Dynamisk instrumentering er hvor instrumenteringen er koblet løst opp mot applikasjonen. Instrumenteringen ønskes da å være så lite statisk som mulig.

3.6 Systematisere instrumentasjonsdata

Instrumenteringsdata som er innsamlet må systematiseres før de kan brukes senere. Her er alt avhengig av hvor lite eller mye vi vil vite om prosessene. Se momenter nevnt under:

1. *Beregne og prioritere data*
2. *Lagre data*

3.6.1 Beregne og prioritere data

Dataene må ofte bearbeides og kalkuleres før dataene kan brukes av forvaltningsapplikasjonen. Deretter er det viktig å prioritere data da noen data kan være mindre viktige, og noen data kan være basert på andre data. Vi må ha klart i minne hvilke data vi er ute etter. Dette for å få færrest data som sier mest mulig, uten at det går utover den kvalitetsmengden av informasjon vi forventer å beholde. Dette er svært viktig momenter under informasjonshentningen senere i aksjonsdelen og presentasjonsdelen om det er kvalitet eller kvantitet vi er ute etter. Vi må huske på at mye data tar lang tid å behandle og prosessere, og kan gå ut over prosessorkraft til andre applikasjoner. Derfor er det viktigst å regne ut og strukturere de data som skal brukes under aksjonsdelen, som det trenger hurtig aksjon på. Og kan heller lagre data mindre ustrukturert som rådata, som kan senere beregnes under behov i presentasjonsdelen.

3.6.2 Lagre data

De ulike instrumentasjonsdata lagres i forskjellige kilder avhengig av lagringstid, tilgjengelighet og krav om strukturert lagring. Instrumentasjonsdata av kort levetid som brukes til umiddelbar beregning av aksjonsdata bør mellomlagres i minne. Videre instrumenteringsdata for analyse og bearbeidelse bør lagres strukturert, mens data for senere analysering og etterbehandling kan lagres ustrukturert. Strukturert lagring er egnet for analysering og bearbeidelse, mens ustrukturert lagring (dumping) krever lengre analysering og etterbehandling. Ustrukturert lagring kan

brukes for feilsøking og feilretting, da bruker kan gå tilbake og hente ut informasjon senere. Tabellen under viser steder hvor man kan lagre data.

Sted	Fordeler	Ulemper	Merknad
Minne	Superraskt ved beregning av data.	Datatap ved strømbrudd	Egnet for bearbeiding av data
Filer	Rask. Egnet for store datamengder.	Vanskelig å bearbeide.	Egnet for ustrukturert lagring
Database	Forholdsvis rask og enkel lagring av data.	Data må fordeles ved store mengder.	Egnet for strukturert og ustrukturert lagring
Over nettverk	Data kan distribueres til andre brukere av nettverk.	Tregt ved bruk av kommunikasjon protokoller. Fare for tap dersom nettverk faller ned.	Egnet for distribuering av data til andre brukere av nettverket.
Over Internet	Data kan distribueres til andre brukere av Internett.	Veldig tregt ved bruk av kommunikasjonsprotokoller over flere nettverk. Stor fare for tap av data.	Egnet for distribuering av data til andre brukere av Internett.

Tabell 3-2 Lagring av instrumentasjonsdata

3.7 Aksjonere på instrumentasjonsdata

Noen situasjoner vil kreve en umiddelbar reaksjon fra forvaltningssystemet for å unngå driftstans. Raskere aksjonstid på hendelser vil gi en høyere tilgjengelighet på applikasjonene. Følgende faktorer inngår i styringsdelen av forvaltningssystemet for å gi riktige ønskete aksjoner.

1. *Grenseverdier*
2. *Endre instrumentering*
3. *Sende meldinger*

3.7.1 Grenseverdier

Hvilke instrumentasjonsdata med tilhørende grenseverdier som det skal aksjoneres på er en veldig stor utfordring. Enkelte grenseverdier som stans er enkelt å forholde seg til, da er det å starte prosessen igjen. Når det gjelder tidsbegreper, belastning og flyt av datamengder er det grenseverdier som må justeres etter innstilling av applikasjonen. Med innstilling menes det

å justere grenseverdier ved å betrakte hvordan prosessene kjører under drift ut i fra hurtighet, belastning og krav om prioritering av meldingsflyt.

3.7.2 Endre instrumentering

Endring av instrumenteringen for prosesser bør skje automatisk før man eventuelt sender meldinger til bruker, siden situasjoner lett kan forverre seg dersom handling ikke skjer umiddelbart. Handlinger kan være å bearbeide prosesser gjennom instrumenteringslaget, endre på mengden av informasjonsflyt fra instrumenteringslaget, graden av instrumentasjonsdata som skal beregnes og i hvilken grad meldinger skal prioriteres og sendes. Under utvikling og innstilling av applikasjonen kan det være nødvendig med en høyere grad av instrumentasjonsdata for å kunne belyse svake sider og flaskehalser i applikasjonen. Under full drift kan graden av instrumentasjonsdata begrenses til det mest viktigste.

3.7.3 Sende meldinger

Før man sender meldinger som informasjon, varsler og alarmer, er det viktig å dele meldinger opp i grupper eller prioritet. Kritiske meldinger krever øyeblikkelig oppmerksomhet og skal derfor bli behandles med høyest prioritet. Det er mange ulike måter å sende meldinger til brukere. Mindre viktige meldinger kan sendes som e-post, mens viktigere meldinger kan poppe automatisk opp på skjerm eller sendes til mobiltelefon. Et eksempel på prioritering av meldinger er vist i tabell under.

Prioritet	Tekst (eng/nor)	Forklaring
1	*Debug / Feilsøking*	Meldinger som blir brukt under feilsøking.
2	*Info / Info*	Infomeldinger som inneholder generell informasjon som kan være til nytte for forvaltningsapplikasjonen.
3	*Minor / Mindre*	Mindre viktige melding som ikke er advarsel men er mer viktig enn informasjonsmelding.
4	*Warning / Advarsel*	Advarsler blir sendt når noe kommer til å oppstå eller at regler er blitt brutt.
5	*Exception / Unntak*	Unntaksmeldinger blir sendt når det oppstår ett unntak innenfor en prosess

		i applikasjonen.
6	*Critical / Kritisk*	Kritiske meldinger blir sendt når noe skjer som umuliggjør applikasjonen å fungere normalt. Eksempler er unntak som foresaker feil i applikasjonen.

Tabell 3-3 Prioritering av meldinger

3.8 Presentere instrumentasjonsdata

Presenteringsdelen av forvaltningsapplikasjonen er i hovedtrekk ment som en hjelp for driftpersonell for å forvalte applikasjonene og prosesser på datamaskinen. Men er en nødvendighet dersom det trengs manuell justering på innstillinger i instrumentasjonslaget, eller det er annet på datamaskinen som må justeres manuelt. Hva som skal presenteres og hvilke håndteringer som skal være mulig bør avtales mellom utvikler og den som skal drift systemet. Følgende punkter må tas i betraktning:

1. *Målgruppe*
2. *Sikkerhet og informasjonsbehov*
3. *Brukervennlighet og presenteringsform*
4. *Hvilke håndteringer som kan foretas av bruker på prosesser (starte, stoppe osv) og instrumentering.*

3.8.1 Målgruppe

Hvilken målgruppe som skal bruke forvaltningsapplikasjonen er svært essensielt med tanke på sikkerhet, informasjonsbehov, brukervennlighet, hva i hvilken grad og hvordan instrumenteringsdata skal presenteres.

3.8.2 Sikkerhet og informasjonsbehov

De ulike målgruppene har ulike informasjonsbehov men graderes vanligvis etter kunnskap til brukerne. Utviklere har tilgan til alt, mens driftpersonell og nettansvarlige har reduserte tilganger. Dette graderes fordi brukere ikke skal foresake feil i applikasjon på grunn av uvitenhet, og selvsagt sikkerhetsmessige årsaker om sensitive data. Her er det viktig å styre i hvilken grad de forskjellige brukerne skal ha tilgang til instrumentasjonsdata og tilgang til instrumenteringslaget.

3.8.3 Brukervennlighet og presenteringsform

Brukervennlighet og presenteringsform henger sterkt sammen. Grafisk fremstilling brukes gjerne dersom man trenger stork overblikk. Da kan man også bruke grafiske prosesskart som viser belastningen ved hjelp av belastningsstolper. Enkelte instrumenteringsdata som ressursbruk i form av prosessorbruk, minnebruk, diskbruk osv som endres raskt er hensiktmessig å fremføre i grafiske diagrammer over en tidsperiode. Verdier som varer eller endres sent over tid kan være presentert i tekstform.

3.8.4 Tilgang til håndteringer

Tilgang til håndteringer av prosesser og instrumenteringslag kan inndeles i flere kategorier for de forskjellige brukerne. Men noen håndteringer er ment til å bare kunne justeres automatisk enten av forhåndsdefinering eller at tilstandene endrer seg så raskt at en menneskelig justering ikke er tilstrekkelig. Samtidig at andre håndteringer kun kan justeres manuelt. En bruker trenger ikke ha tilgang til å stoppe en prosess fordi han har tilgang til å starte eller restarte en prosess.

4 Tilgjengelig teknologi

I løpet av denne masteroppgaven utførte vi en studie i vanlige praksis og teknikker med tanke på instrumentering. Studiet viste oss at det eksisterer flere teknologier for instrumentering som alle har sine mål. I dette kapittelet vil vi summere noen av de vanligste teknikkene vi har sett på. Det eksisterer flere løsninger enn denne masteroppgaven vil dekke. Først vil vi sette opp momenter vi først må undersøke foren evalueringsmetode som inneholder punkter fra instrumenteringsteorien utarbeidet i kapittel 3. Deretter vil vi se på ulike standarder, biblioteker som blir brukt, og se på noen av de forskjellige forvaltningssystemene som er tilgjengelig på markedet. Tilslutt vil vi sammenligne teknologien for å få et bedre overblikk.

4.1 Evalueringspunkter for tilgjengelig teknologi

Teknologien vi skal se på dekker hele eller deler av instrumenteringsteorien utviklet i kapittel 3 som tok for seg:

1. *Identifisering*
2. *Innsamling*
3. *Instrumentering*
4. *Systematisere instrumentasjonsdata*
5. *Aksjonere på instrumentasjonsdata*
6. *Presentere instrumentasjonsdata*

4.2 Verktøy innebygd i operativsystemet

Operativsystemet inneholder ofte mange nyttige programmer som kan gi et overblikk over tilstanden til systemet. I dette kapittelet vil vi se på noen av de vertøyene Microsoft Windows tilbyr.

4.2.1 Windows oppgavebehandler

Oppgavebehandleren gir et raskt innblikk i hva som skjer i systemet. Den viser en liste over hvilke prosesser som kjører, hvilken status de befinner seg i og hvor mye minne og prosessor kraft de konsumerer. Ut ifra denne informasjonen kan du velge å håndtere prosesser som påvirker systemet negativt.

4.2.2 Windows hendelsesliste

I Microsoft Windows finner vi noe som heter hendelsesliste. I listen kan du finne informasjon om maskinvare, programvare og system problemer. Det er også mulig å overvåke Windows sikkerhets meldinger. Listen er som standard delt opp i tre ulike logger:

- Program
  Program loggen inneholder meldinger fra applikasjoner og programmer. Utviklerne velger hvilke meldinger de ønsker å overvåke.
- Sikkerhet
  Sikkerhet loggen kan brukes til å logge forsøk på å logge seg inn på datamaskinen og andre hendelser som forsøk på å opprette, åpne eller slette filer.
- System
  System loggen inneholder meldinger fra komponenter som er relatert til operativsystemet. Eksempelvis vil en driver som feiler under oppstart generer en melding i denne loggen.

Hendelseslisten deler hendelsene inn i fem ulike kategorier: Feil, advarsel, informasjon, overvåkning av vellykkede forsøk og overvåking av feilende forsøk.

Problemet med dagens versjon av hendelsesliste er at den ikke støtter lagring av store mengder informasjon. Kvaliteten på informasjonen er også et problem. Dette har Microsoft tenkt å gjøre noe med, derfor vil det i neste generasjon av Windows, Windows Longhorn, komme en ny versjon av hendelse liste som er XML basert. XML gir muligheten til å strukturere dataene på en mer fornuftig måte og ved å bruke XPATH språket vil det være mulig å utføre rimelig sofistikerte søk. Samtidig vil den nye versjonen være bakover kompatibel med tidligere versjoner.

4.2.3 Performance monitor

Performance Monitor er et kraftig verktøy innebygd i nyere versjoner av Microsoft Windows. Verktøyet gir et detaljert innblikk i ytelsen i systemet ved

å overvåke ulike ytelse tellere. Du kan sette den opp til å overvåke samle inn informasjon fra den lokale eller eksterne maskiner. Det finnes også mulighet til å generere rapporter og sette opp alarmer som kan varsle deg hvis en teller skulle gå over eller under en angitt verdi. Performance monitor startes ved å trykke på "Start->Kjør" og skrive inn perfmon.exe. Trykk deretter på Enter tasten.

4.2.4 Netstat

Netstat gir en oversikt over hvilke porter og aktive TCP tilkoblinger som finnes mot maskinen, Ethernet statistikk, IP ruter tabellen, IPv4 statistikk for IP, ICMP, TCP og UDP protokollene og IPv6 statistikk. Verktøyet kan brukes til å se om en prosess kobler seg opp og lytter på en bestemt port.

4.2.5 Ping

Ping bruker Internet Control Message Protocol (ICMP) ”Echo” “Request” og “Echo Reply” meldinger. Ping er et svært hendig lite verktøy som kan brukes til å sjekke om en nettverks enhet svarer på forespørsel og detektere eventuelle feil i nettverksoppsettet. Brukes ofte til å finne ut om en ekstern nettverksenhet er tilgjengelig over nettverket. Svarer ikke enheten på Ping er det ofte et tegn på at det er problemer med nettverksoppsettet eller at den er uten strømtilførsel. Dessverre tar mange i dag i bruk brannmurer som sperrer bruken av Pling for å forhindre "Dental og Service" angrep (ping of death).

4.3 Standarder og rammeverk

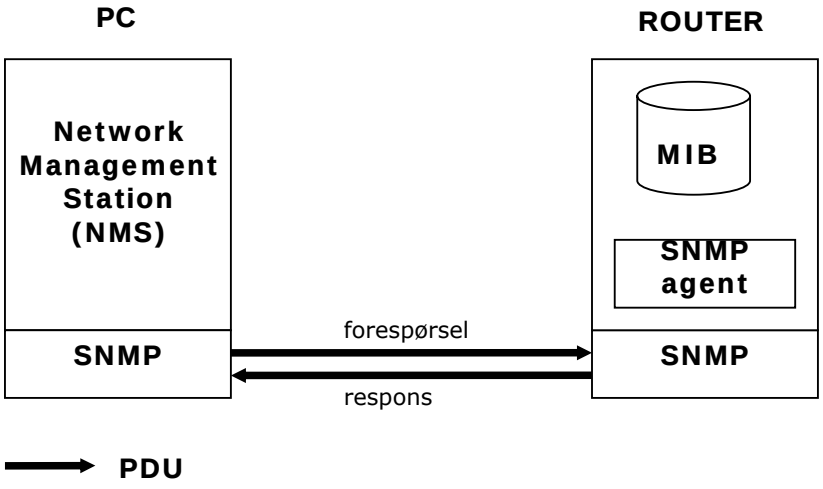
I dette kapittelet vil ta for oss noen av de mest utbredte standardene og rammeverkene innenfor instrumentering. Vi vil også se på EIF som er et helt nytt rammeverk fra Microsoft for instrumentering av .NET applikasjoner.

4.3.1 Simple Network Management Protocol (SNMP)

SNMP er en videreutvikling av Internet Engineering Task Force (IETF) [1] sin SGMP protokoll. Den ble utviklet i mot slutten av 1980 årene for å hjelpe til med å forvalte forskjellige nettverks enheter fra et sentralt punkt. SNMP [5] var egentlig ment å være en midlertidig løsning til OSI gjorde ferdig deres CMOT (Common Management over TCP/IP), men på grunn av SNMP sin enkle oppbygning skulle den vise seg å bli en større suksess en noen hadde turt å spå på forhånd.

Identifisering

SNMP består av tre hoveddeler. En ”Network Management Station” (NMS), SNMP agenter og ”Management Information Base” (MIB) objekter. NMS er en dedikert maskin som sørger for å samle inn viktig informasjon fra nettverket gjennom bruk av agenter. En agent har mulighet til å betjene flere NMS-maskiner samtidig som to NMS-maskiner kan snakke sammen.



Figur 4-1 Interaksjon mellom NMS og en SNMP agent

Innsamling

NMS bruker som sagt agenter til å samle inn informasjon om nettverket. Informasjonen blir pakket i noe som kalles for ”Protocol Data Unit” (PDU). Inni disse PDUene ligger det objekt identifikatorer som referer til verdier som både NMS og SNMP agenten forstår.

SNMP bruker UDP protokollen til å overføre informasjon mellom de ulike nettverksenhetene. UDP protokollen er en protokoll laget for å raskt overføre data med minst mulig ”overhead”. Det sørger for en rask prosessering og et mindre krav til ressurser.

Instrumentering

Må spesialtilpasses den enkelte implementering av standarden.

Systematisere instrumentasjonsdata

SNMP utveksler data mellom agent og NMS i form av meldinger. Disse meldingene inneholder versjon nr

SNMP benytter seg av et oppbevaringssted kalt ”Management Information Base” (MIB) til å sende forvaltning informasjon. MIB objektene inneholder en hierarkisk oppbygd trestruktur, hvor toppen av treet inneholder den mest generelle informasjonen om nettverket. Grenene innholder mer detaljert informasjon om områder av nettverket, mens bladene innholder den mest detaljerte informasjonen.

Aksjonere på instrumentasjonsdata

Det er mulig å sette at en agent skal sende en melding dersom det oppstår en feil eller dersom en verdi går over eller under en gitt verdi grense.

Presentasjon av instrumentasjonsdata

Standarden inneholder bare regler for hvordan data samles inn og struktureres.

4.3.2 Desktop Management Interface (DMI)

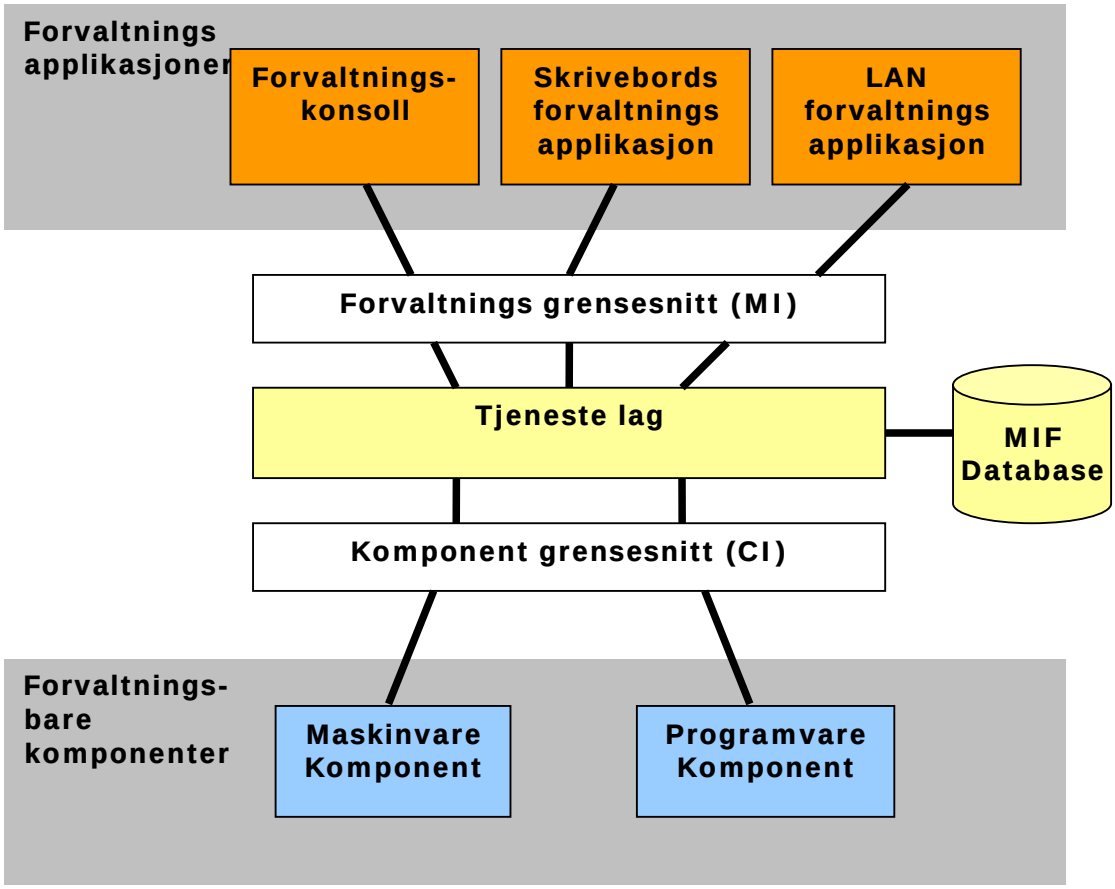
Desktop Management Interface [12] ble designet av Distributed Management Task Force (DMTF) [3]. DMTF er et samarbeid mellom flere bedrifter som jobber sammen for å utvikle nye standarder for å vedlikeholde og forvalte personlige datamaskiner. DMTF ble grunnlagt i mai 1992 ut i fra et samarbeid mellom 8 bedrifter: Digital Equipment Corporation, Hewlett-Packard, IBM, Intel, Microsoft, Novell, SunSoft og SynOptics.

Målene ved utviklingen av DMI var:

- At standarden skulle bli uavhengig av alle operativsystemer, maskinvare plattformer og forvaltnings protokoller.
- Gjøre det enkelt for leverandører å adoptere løsningen.
- At den skulle være skalerbar, det vil si brukbar på alt fra svært enkle til komplekse og utvidbare produkter. I alt er DMI kapabel til å beskrive mer enn 80 000 PC-produkter.
- Tilknyttet eksisterende forvaltningsprotokoller.

Identifisering

DMI består av fire hoveddeler. Management Information Format (MIF), et tjenestelag, et komponentgrensesnitt (CI) og et forvaltningsgrensesnitt (MI).



Figur 4-2 Arkitektur DMI

Innsamling

Innsamling av data skjer gjennom komponent grensesnittet (CI). Komponent grensesnittet er en API som sender instrumentasjons data til MIF databasen ved hjelp av tjeneste laget. Dette skjer ved bruk av diverse Get og Send funksjoner. Ved kritiske hendelser bruker den også en Event funksjon for å varsle forvaltningsapplikasjonen.

Instrumentering

Ikke evaluert ettersom det må spesialtilpasses den enkelte implementering av standard.

Systematisere instrumentasjonsdata

Informasjonen om de forskjellige komponentene blir lagret i en tekstfil (MIF database). Som standard samles det inn informasjon om hvilken ID-gruppe komponenten tilhører, produktnavn, versjon, serienummer og tid og dato for den siste installasjonen.

Filen er sårbar for endringer og derfor er det viktig at operativsystemet har et filsystem som beskytter den mot endringer.

Aksjonere på instrumentasjonsdata

Komponentene kan bruke ”Event” (meldings) funksjonen i komponent grensesnittet til å varsle forvaltningsapplikasjonen når det skjer noe kritisk.

Presentere instrumentasjonsdata

Forvaltning grensesnittet er en API som definerer hvordan forvaltnings applikasjoner kommuniserer med tjeneste laget. Sentralt i dette grensesnittet er funksjoner som Get, Set og List, hvor sistnevnte lister opp alle enheter som kan forvaltes gjennom DMI. Hvordan disse data igjen blir presentert for operatøren er opp til de som utvikler forvaltnings applikasjoner.

4.3.3 Common Information Model, Web-Based Enterprise Management (CIM / WBEM) og Windows Management Instrumentation (WMI)

Lansert i 1996 var Web-Based Enterprise Management (WBEM) [8] et initiativ for å koble sammen eksisterende standarder med nye teknologier som CIM og XML. WBEM er med andre ord ikke en standard i seg selv. Common Information Model (CIM) [9] definerer en modell som representerer alle forvaltningsbare objekter i et nettverk. CIM-modellen støtter fysiske og logiske presentasjoner av nettverkskomponenter som PCer, skrivere, applikasjoner, topologier og komponenter som vifter og strømforsyninger.

På Microsoft Windows er Windows Management Instrumentation (WMI) [13] Microsoft sin implementasjon av CIM/WBEM.

Identifisering

WBEM består av tre hoveddeler. Common Information Modell (CIM), xmlCIM koding spesifikasjon [11] og CIM Operations over HTTP [10]. CIM gir et felles format, språk og metodologi for innsamling og beskrivelse av instrumentasjonsdata. xmlCIM spesifikasjonen definerer XML elementer, skrevet i Document Type Definition (DTD), som kan brukes til å representere CIM klasser og instanser av disse. CIM Operation over http spesifikasjonen definerer en kobling mellom CIM operasjoner på http som tillatter implementasjoner av CIM å samspille i en åpen, standardisert måte og fullbyrder teknologiene som støtter WBEM.

Innsamling

Innsamling av data skjer over HyperText Transfer Protocol (HTTP) og overføres som XML. Agenter konventerer informasjonen fra eksisterende standarder som SNMP og DMI til et format som er leselig for CIM datalager.

WMI støtter dessuten innsamling av en rekke Win32 spesifikke komponenter.

Instrumentering

Ikke evaluert.

Systematisere instrumentasjonsdata

Ved hjelp av objekt-orienterte programmerings teknikker tilbyr CIM en konsistent definisjon og struktur av data, inkludert uttrykk for elementer slik som objekt klasser, egenskaper, assosiasjoner og metoder.

Managed Object Format (MOF) er formatet som blir brukt til å definere strukturen og innholdet av CIM skjema i leselig form. En MOF tekst fil beskriver en klasse av en eller flere instrumenterte objekter. Informasjonen i MOF kan importeres inn i CIM datalageret.

Aksjonere på instrumentasjonsdata

WBEM/CIM gjør det mulig å aksjonere på instrumentasjonsdata ved at metoder eller funksjoner lar seg anrope fra forvaltningsapplikasjonen.

Presentere instrumentasjonsdata

Instrumentasjonsdataene er tilgjengelig over HTTP og XML og er dermed tilgjengelig som tekst i en menneskelig og maskin leselig form. Hvordan instrumentasjonsdataene presenteres rent grafisk i forvaltningsapplikasjonen er opp til utviklerene.

Når det gjelder WMI følger det med en del verktøy med installasjonen av WMI. Blant annen WMI studio som er en verktøy som åpnes i nettleseren.

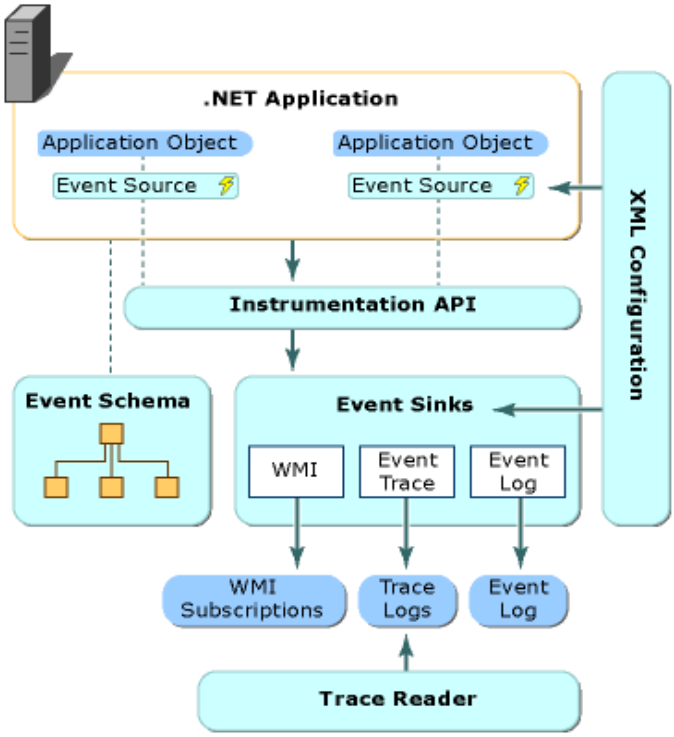
4.3.4 Enterprise Instrumentation Framework (EIF)

Enterprise Instrumentation Framework [7] ble først introdusert ved lansering av Visual Studio .NET 2003. I første omgang var EIF kun tilgjengelig for abonnementer av MSDN [2] Universal og MSDN Enterprise, men i oktober 2003 ble EIF gjort offentlig tilgjengelig.

EIF er at API som utvider eksisterende hendelse, logging og sporings mekanismer allerede innebygd i Windows.

EIF kan integreres med eksisterende forvaltningssystmer som Microsoft Application Center og Microsoft Operations Manager (MOM). Trolig er det også mulig å bruke forvaltnings systemer som HP Open View gjennom å ta i bruk WMI [13] brønnen.

Arkitektur



Figur 4-1 EIF arkitekturen

Figur 4-1 er hentet fra EIF dokumentasjonen og viser et overordnet bilde av EIF arkitekturen. Grunnprinsippet bak EIF er at du definerer event sources for å poste eventer og event sinks for å samle dem inn.

Event Sources

Event Sources blir brukt til å reise hendelser til hendelse brønner, men blir også brukt til å separere ulike applikasjons lag fra hverandre. Som standard når en hendelse blir reist bruker den en hendelse kilde ved navn Application. Noen ganger er det nødvendig og nyttig å separere hendlelser fra forskjellige applikasjon komponenter ved å benytte seg av distinkte hendelse kilder. Som et eksempel kan du tenkte deg et system som genererer en feil. La oss si at vi kjenner til i hvilken hendelse kilde feilen ble generert. Ved å filtrere

hendelser til å kun omfatte denne hendelse kilden vil det være langt kjappere å finne årsaken enn hvis vi måtte gått gjennom alle hendelsene.

Event Sinks

EIF kommer ferdig med tre forskjellige Event Sinks. Ingen kode er nødvendig for å ta i bruk disse brønnene.

- TraceEventSink
  Denne brønnen bruker Windows Trace Session Manager til å skrive hendelser til hendelses logg. Ettersom Windows Trace Session manager kjører under kjerne nivå er den ekstremet rask og kan brukes til å generere tusenvis av hendelser vært sekund.
- LogEventSink
  Denne brønnen bruker Windows hendelses logg til å skrive hendelser. Brønnen kan bli konfigurert til å skrive på en annen maskin, men denne måten å gjøre det på har blitt rapportert til å være ustabil. Derfor bør den bare bli brukt til overordnede hendelser.
- WMIEventSink
  Ettersom denne brønnen er den tregeste av alle brønnene bør det kun brukes til sjeldne eller kritiske hendelser.

Valg av brønn er besluttet på:

- Type hendelse (viktighet på hendelsen).
- Hvor ofte hendelsen oppstår (frekvens).

En tilpasset brønn kan lages spesielt for en applikasjon. Dette blir gjort ved å lage en klasse som arver fra den abstrakte klassen EventSink og ved å skrive kode som håndterer prosesseringen av hendelses sending. En tilpasset brønn må kompileres med et befestet navn og installeres i Global Assembly Cache (GAC).

Event Schema

Hendelses skjema inneholder alle hendelsene som er kjent for systemet. Skjemaet er utvidbart og det er mulig å lage sine egne hendelser. Det er bare viktig å forsikre seg om at hendelsen lar seg lagres. ”serialized”

Request tracing

Request tracing eller sporing ved forespørsel er antageligvis en av de beste delene ved EIF. Sporing ved forespørsel gjør det mulig å spore en forretnings prosess på kryss av applikasjonen sine grenser.

XML konfigurasjons fil

EIF bruker en XML konfigurasjons fil å binde ulike Event Sources til Event sinks og filtrere hendelsene.

Performance counters

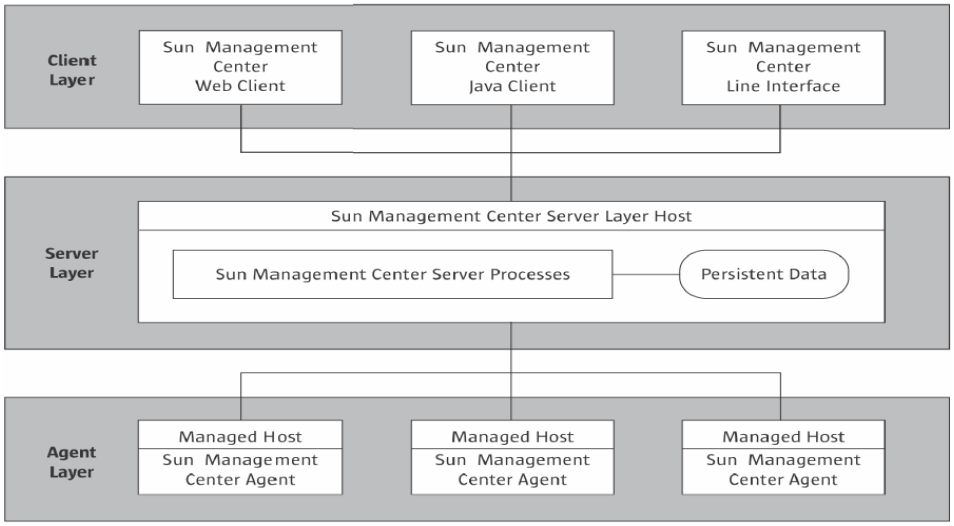
Ved bruk av EIF tilgjengeliggjørs et sett med ytelse tellere uten ekstra arbeid. Gjennom å bruke disse tellerne er det mulig å måle flyten av hendelser fra en hendelses kilde og gjennom en hendelses brønn.

4.4 Forvaltnings systemer.

Forvaltningssystemer er ferdige instrumenteringsløsninger til å bli installert på et operativsystem på lik linje med andre ferdige applikasjoner. Disse er ofte dyre, bruker proprietære løsninger og er beregnet på større systemer. Hvor mye instrumentering som er bygget inn i instrumenteringslaget er veldig forskjellig i de forskjellige forvaltningssystemene.

4.4.1 Sun Management Center

SMC [17] er et forvaltningssystem som er beregnet på Solaris operativsystem over en UNIX plattform. SMC er delt inn i tre komponentlag vist i figuren under:



Figur 4-3 Arkitektur til Sun management

Identifisering

Systemlaget kan ikke ses i figuren over, men består av Solaris på en UNIX plattform. Instrumenteringslaget består av agentlaget og serverlaget. Applikasjonen består av to komponentlag, som er inndelt i et serverlag og klientlag. Det vil si at etter vår instrumenteringsteori går skillet for instrumenteringslaget inne i serveren. Programvaren til SMC er delt i tre lag, server, klienter og agenter. Se figur over.

Serveren er hjernen i forvaltningsapplikasjonen. Serveren har som oppgave å kjøre prosessene og tjenestene i forvaltningsapplikasjonen, og sende videre ordrer fra klientene til agentene. Ellers har serveren oppgaver som:

- ”Multiplekser” for klient forbindelser og agenter.
- Topologi tjeneste som å være områdeadministrator og vise forvaltningstjenester.

Klientene som er brukernes grensesnitt mot forvaltningsapplikasjonen, brukes til å initiere forvaltningsoppgaver. Klienter kan kjøres på forskjellige plattformer siden de er java baserte eller kan bli kjørte via en nettleser. Klientene kan også kommunisere via CLI (Kommando linje grensesnitt) som er tilgjengelig via en programterminal.

Agentene har som oppgave å samle inn instrumentinformasjon, overvåke lokale resurser og svare på forvaltningsordre. Agenter finnes det flere av for forskjellige oppgaver.

Innsamling

Det er agentene som samler inn data til instrumenteringslaget. Ulike agenter har hvert sitt område å samle inn data fra.

Instrumentering

SMC støtter SNMP og har en SNMP sportjeneste mottaker som logger og sender meldinger videre til inntresanne komponenter. Agentene I instrumentasjonslaget er dynamiske, skalerbare, utvidbare og SNMP baserte. Instrumenteringsmengden er da avhengig av hvor mange agenter som blir lagt inn i instrumenteringslaget. Det er også mulig å lage egne JAVA baserte grensesnitt til å implementere i agentene.

Systematisere instrumenteringsdata

SMC tilbyr meldingstjeneste for lagring av meldinger og forvaltning av meldinger.

Aksjonere på instrumenteringsdata

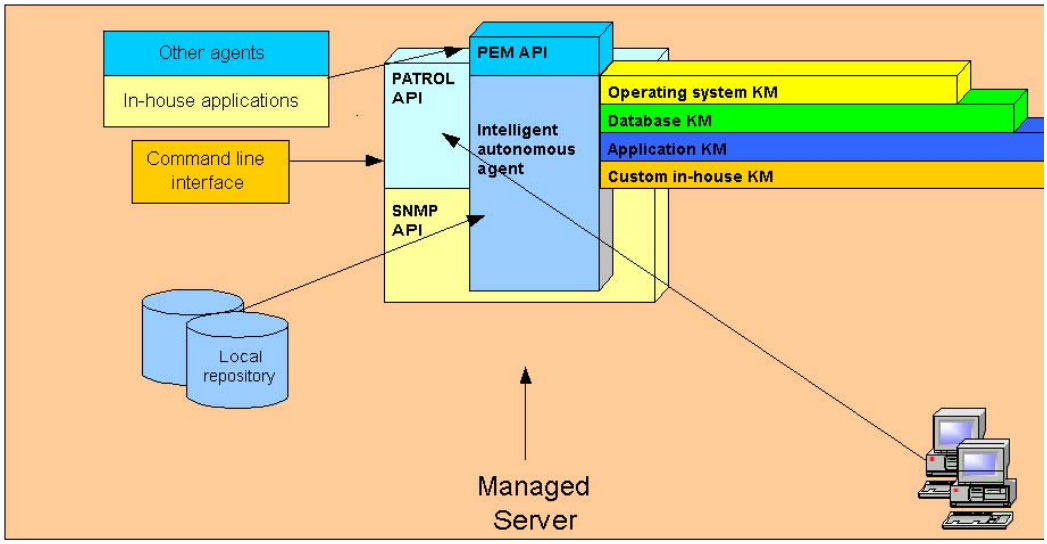
SMC tilbyr grupperingstjeneste som lager, planlegger og utfører oppgaver gjennom varierte filtreringskriterier.

Presentere instrumentasjonsdata

SMC har en konfigurasjonstjeneste for autentisering for klienter. SMC tilbyr informasjon om metamodell av forvaltningstilganger for interne prosesser. Brukerne kan bruke forhåndsdefinerte klienter eller egenutviklet klienter til å overvåke instrumentasjonsdata og prosesser.

4.4.2 BMC Patrol

Patrol [16] fra BMC er et forvaltningssystem som er beregnet på Windows operativsystem. Vi har tatt for oss et arkitekturdiagram (se figur nedenfor) som et utgangspunkt for Patrol.



Figur 4-4 Arkitektur til Patrol

Modulene i Patrol er inndelt i 3 kategorier som agentmoduler, kunnskapsmoduler (KM) og konsoll for brukeren.

Identifisering

Systemlaget kan ikke ses i figuren over, men Patrol er beregnet for operativsystemet til Windows. Sammenlignet med arkitekturen i instrumenteringsteorien er agenten og KM'er (Kunnskapsmodul) deler av instrumenteringslaget, der KM'ene er grensesnittene for de forskjellige systemresursene mot agenten. Patrol kan overvåke og forvalte applikasjoner, databaser, mellomvare produkter, Internett applikasjoner og operativsystem.

Innsamling

Det er agenten som samler inn instrumentasjonsdata gjennom kunnskapsmodulene til instrumenteringslaget. Kunnskapsmodulene er

hovedsakelig pakket rundt ulike applikasjoner eller systemressurser, og blir administrert og tjener som grensesnitt for agenten. Patrol overvåker automatisk verdier om prosessorbruk, minnebruk, harddiskbruk, tjenester, COM+ og MSMQ (Microsoft Message Queue, meldings køer). Andre verdier kan bli overvåket manuelt ved å konfigurere filteret til meldingsloggen (Event Log filtering). Patrol har innebygd tryllestav (Wizzard) for ”Microsoft Performance Monitor” og WMI.

Instrumentering

Patrol benytter ekstern instrumentering gjennom WMI og MPM (Microsoft Performance Monitor). Derfor er det kun mulig å rapportere informasjon som er tilgjengelig fra operativsystemet. Hva som instrumenteres er avhengig av antall kunnskapsmoduler som blir installert. SNMP (Simple Network Management Protocol) brukes mellom agenten og kunnskapsmodulene.

Systematisering av instrumentasjonsdata

Hvordan Patrol beregner og kalkulerer instrumentasjonsdata har vi ikke funnet ut da Patrol ikke har tilgjengelig kildekode. Kunnskapsmodulene lagrer instrumentasjonsdata som lokale binære filer i katalogen til Patrol. Agenten lagrer i et ukjent lokalt lager.

Aksjonere på instrumentasjonsdata

Hvordan Patrol aksjonerer på instrumentasjonsdata har vi ikke klart å finne ut basert på den informasjonen vi har funnet på nettet.

Presentere instrumenteringsdata

Konsollen til Patrol har et grafisk brukergrensesnitt. Brukeren kan fra denne konsollen få informasjon vist i form av alarmer, grafer og rapporter. Det er også mulig å sende ordrer til agenten.

4.5 Sammenligning av tilgjengelig teknologi

Vi skal i dette underkapitlet sammenligne de forskjellige teknologiene vi har sett på for å vurdere teknologiene opp mot hverandre. Med så mange forskjellig instrumentering løsninger kan det være vanskelig å vite hvilken teknologi vi bør velge. Noen løsninger er bedre egnet en andre for en spesiell oppgave, alt avhengig av kravene applikasjonen stiller og hvilket fokus en har på instrumenteringen.

4.5.1 Sammenligning av verktøy innebygd i operativsystemet

Fra tabellen neste side ser vi at de alle benytter seg av systemlaget for identifisering, innsamling og instrumentering. Når det gjelder systematisering er det kun Windows hendelsesliste og performance monitor som lagrer data, og da som strukturert til fil. Netstat og Ping kan lagre på kommandolinje dersom utskriften omdirigeres til for eksempel en fil (>filnavn.txt).

4.5.2 Sammenligning av forvaltnings systemer

Fra tabellen neste side ser vi at forvaltningssystemene støtter alle hovedpunktene i instrumenteringsteorien. BMC Patrol benytter seg kun av ekstern instrumentasjon og er veldig begrenset til integrerte instrumenteringsteknikker i BBC Patron. Sun Management Center er det motsatte. Intern instrumentering støttes da både agenter og klientene kan modifiseres. Dette gjør Sun Management Center til en dynamisk, skalerbar og utvidbar. Ulempen er at Serveren er utviklet for Solaris og ikke Windows.

	Instrumenteringsteori					
Teknologi	Identifisering	Innsamling	Instrumentering	Systematisering	Aksjonere	Presentere
Verktøy innebygd i operativsystemet.						
Windows oppgavebehandler	Systemlag	Prosesser	OS	Ingen	Start/stopp av prosesser	Grafisk
Windows hendelsesliste	Systemlag	Maskinvare og programvare	OS	Strukturert lagring til fil	Ingen	Grafisk
Performance monitor	Systemlag	Ytelse tellere	OS, WMI	Strukturert lagring til fil	Alarmer som kan aktivere aksjoner.	Grafisk og rapport til fil
Netstat	Systemlag	Nettverk	OS	Ingen	Ingen	Tekst på kommandolinje
Ping	Systemlag	Nettverk	ICMP	Ingen	Ingen	Tekst på kommandolinje
Standarder og rammeverk						
SNMP	NMS, SNMP agenter og MIB	Gjennom SNMP agenter	Ikke evaluert	Lagring i MIB og data overføres som PDU objekter	Mulighet for varsling	Ikke evaluert
DMI	MIF, CI og MI	Gjennom CI	Ikke evaluert	Lagring til MIF	Event funksjon	Ikke evaluert
CIM/WBEM	WBEM, xmlCIM, CIM Operations over HTTP	XML over HTTP	Ikke evaluert	Lagring til CIM repository	Mulighet til å kalle operasjoner	Ikke evaluert
EIF	Event Source, Event Sinks, Event schema og API	EventLog, WMI og Trace Log	Ikke evaluert	Lagring til hendelses brønner	Ingen	Ikke evaluert

Forvaltnings-applikasjoner						
Sun Management Center	Server og agenter på Unix – Solaris Klient på JAVA Web, CLI	Agenter	SNMP sportjeneste, Dynamisk, Skalerbare og Utvidbare.	Lagrer meldinger	Gruppering-tjeneste	Autentisering, Metamodel. Kan utvikle egne klienter
BMC Patrol	Windows	Agenten via flere KM'er	Kun ekstern WMI,MPM	KM lagrer ustrukturert fil. Agent lagrer i ukjent lokalt lager.	Ukjent	Grafisk. Kan sende ordrer.

Tabell 4-1 Evalueringspunkter for tilgjengelig teknologi

5 Prototyp

I dette kapittelet vil vi presentere prototypen vi utviklet i løpet av denne hovedoppgaven. Deretter vil vi sette prototypen opp mot teorien vi utviklet i kapittel 3. Prototypen vil bli brukt til å demonstrere konseptene av teorien vi utviklet i kapittel 3.

5.1 Valg av utviklingsmiljø

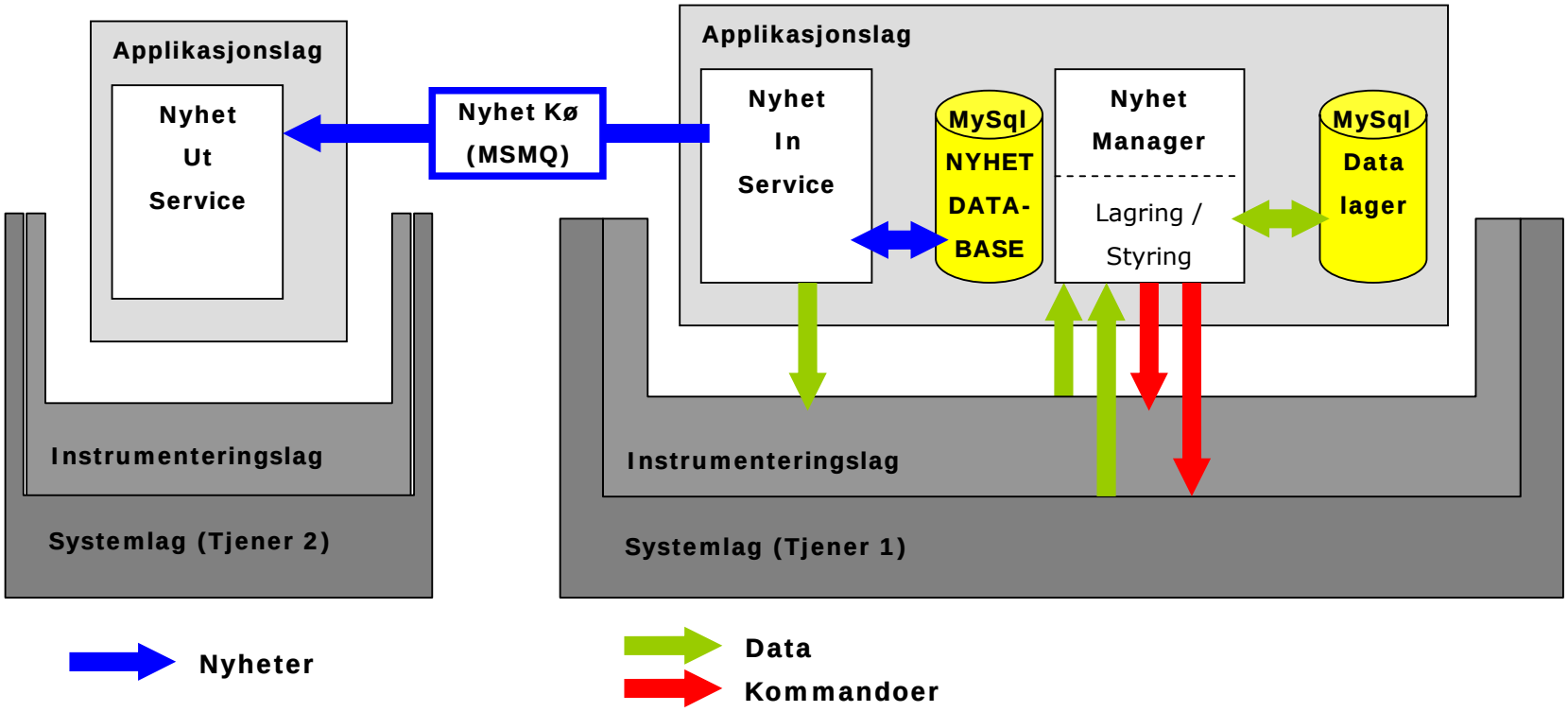
Vi valgte å bruke Visual Studio .NET 2003 og .NET rammeverket som vårt utviklingsmiljø. Vi kunne ha valgt Java eller en annen plattform men valgte dette fordi det er det samme som Sense Intellifield har brukt på deres SiteCom system. Vi har også valgt å bruke EIF som det primære instrumentering valget i denne prototypen.

5.2 Et enkelt nyhetssystem

For å demonstrere konseptene ved instrumentering måtte vi først utvikle en applikasjon å instrumentere. Vi brukte SiteCom systemet Sense Intellifield hadde utviklet, som en inspirasjon faktor, og endte opp med et enkelt nyhetssystem som vi gjorde distribuert over to forskjellige tjenere. En for innkommende nyheter og en for utgående nyheter. Ideen bak nyhet systemet er at det vil fungere som et sentralt punkt for lagring og mottaking av nyheter, tilsvarende det SiteCom gjør for måledata på en oljeplattform.

5.2.1 Arkitektur

Figur 5.1 neste side viser arkitekturen til nyhet systemet. Systemet går på 2 tjenere. Tjener 1 består av består av en web service, en database, et bibliotek og en MSMQ kø. Tjener 2 består av en Windows service og et bibliotek. Begge tjenerne har diverse konfigurasjons filer.



Figur 5-1 Arkitektur Nyhetssystem

Nyhetene kommer inn i systemet i tjener 1 gjennom Web-tjenesten Nyhet In Service. Nyhet In Service sørger for at nyheten blir lagret til en database MySql [19]og deretter blir nyheten puttet i en kø for videre prosessering i tjener 2 av Nyhet Ut Service. Køen er basert på Microsoft Message Queuing (MSMQ) som innholder mekanismer for sikker overføring av meldinger mellom to prosesser på samme eller to forskjellige maskiner. Etter nyhetene har blitt satt i kø vil de etter hvert bli plukket opp, videre prosessert og distribuert av Nyhet Ut Service. En mer detaljert beskrivelse av Nyhetssystemet finnes i Appendiks A.

5.3 Instrumentering av nyhetssystemet

Vi vil nå sette prototypen opp mot teorien vi utviklet i kapittel 3.

1. *Identifisering*
2. *Innsamling*
3. *Instrumentering*
4. *Systematisere instrumentasjonsdata*
5. *Aksjonere på instrumentasjonsdata*
6. *Presentere instrumentasjonsdata*

Valgene vi har foretatt her i kapittel 5.3 vil bli argumentert i kapitel 7.3 angående prototypen.

5.3.1 Identifisering

Første steg er å identifisere de forskjellige lagene til systemet.

1. *System lag*
2. *Applikasjonslag*

Systemlag

Vi har valgt å bruke Windows XP som operativsystem både til tjener 1 og tjener 2,som vi likevel har på våre bærbare datamaskiner som vi brukte til å skrive denne rapporten.

Applikasjonslag

Applikasjonene vi har i nyhetssystemet er kjent. De ser vi i figur 5.1.

Tjener 1 består av består av en web service, en database, et bibliotek og en MSMQ kø.

Tjener 2 består av en Windows service og et bibliotek.

Begge tjenerne har diverse konfigurasjons filer.

5.3.2 Innsamling av instrumentasjonsdata

I dette kapitlet skal vi se på innsamling av instrumenteringsdata

1. *Hvilke data skal velges*
2. *Hvor kan data samles inn*
3. *Når data skal hentes*
4. *Hvordan data skal hentes*

Vi vil dekke følgende:

Kritiske områder:

Konfigurasjonsfiler og andre resurser

Hvis ikke konfigurasjonsfilene eller de nødvendige ressursene er tilgjengelig vil ikke systemet fungere.

Innkommende nyheter.

Mottar Nyhet In Service nyhetene? Er tjenesten tilgjengelig for klientene

Lagring i databasen.

Fungerer databasen? Får Nyhet In Service kontakt med db? Blir dataene lagret?

MSMQ

Er køen tilgjengelig fra Nyhet In Service? Klarer Nyhet Ut Service og prosessere nyhetene raskt nok slik at køen ikke fylles opp?

Videre distribusjon fra Nyhet Ut Service.
Får Nyhet Ut Service kontakt med klientene og får klientene kontakt med Nyhet Ut Service? Hvor mange klienter er koblet opp mot tjenesten? Tar det lang tid å overføre nyhetene til klientene? Dette vil være kritisk fordi før den ikke det vil applikasjonen henge og prosessering vil ta lenger tid.

5.3.3 Instrumentering

Etter instrumenteringsteorien skulle vi utføre følgende punkter under instrumentering.

1. *Bruk av standarder*
2. *Skalerbar instrumentasjon*
3. *Dynamisk instrumentasjon*

Bruk av standarder
Vi har valgt å bruke standardene EIF og WMI i instrumenteringsløsningen vår. For å samle inn hendelsene som blir sendt fra EIF gjennom WMI brønnen er det nødvendig å legge til en WMI abonnement. Dette er gjort mulig i "News Manager" ved å fortelle den hvilket "namespace" og hva slags hendelse som ønskes. Nyhet systemet krevde fire abonnementer, to på hver tjener. En som lyttet etter hendelser som ble sendt ved forespørsel og en som lyttet på generelle hendelser. Videre valgte vi å utvikle et egendefinert SOAP kall i Nyhet In Service som et eksempel på manuell intern instrumentering.
Vi valgte følgende applikasjoner som vi skulle instrumentere vist i tabellen under.

Applikasjon	Instrumenteringsteknikk	Kommentar
Web Service – Nyhet In Service (News Input Service)	Egendefinerte SOAP kall og EIF (se også IIS og ASP.NET)	EIF er brukt til å instrumentere hva som skjer internt i tjenesten. Det blir sendt meldinger hver gang tjenesten mottar en nyhet, ved innsetting i database og ved

		innsetting i kø. Det er også laget en egendefinert Web-metoder kalt IsAlive(). Denne kan brukes til å sjekke om tjenesten svarer på respons, en slags hjerterytme funksjon.
Windows Service - Nyhet Ut Service (News Output Service)	*EIF og WMI*	I Nyhet Ut Service har vi lagt inn EIF kode som sender melding til forvaltningsapplikasjonen hver gang det plukkes en melding fra køen og når det sendes en melding til en klient. I tillegg har vi brukt WMI til å overvåke statusen til tjenesten. WMI er også brukt for å legge til funksjonalitet for å stoppe og starte tjenesten.
Nyhet kø (MSMQ)	*WMI*	Vi har brukt WMI til å overvåke to ytelsestellere som forteller hvor mange meldinger som finnes i køen og hvor mange bytes disse meldinger utgjør.
IIS og ASP.NET	*WMI*	Ettersom News Input Service er en Web-tjeneste kjører den under ASP.NET og IIS. Vi bruker WMI til å hente informasjon fra ytelse tellere som forteller oss hvor mange forespørsler som er gjort mot tjenesten og hvor høy CPU belastning den har.
Windows XP	*WMI*	Vi bruker WMI til å overvåke total prosessor last for alle prosesser som kjører på maskinen. Dette gjøres på begge maskinene.
Biblioteks filer (News Library)	*EIF*	I News Library har vi brukt EIF til å instrumentere en del av kallene som gjøres mot klassene i biblioteket. Blant annet kan det nevnes at det sendes en melding hver gang det blir opprettet et nytt objekt av klassen News.

Tabell 5-1 Elementer i prototypen som blir instrumentert

Skalerbar instrumentasjon

Vi har fått til en skalerbar instrumentasjon ved å kunne legge inn egne meldingsbrønner. Videre har vi skalert Nyhetssystemet over 2 tjenere.

Dynamisk instrumentasjon

Vi har fått til en dynamisk instrumentering gjennom EIF da du kan endre konfigurasjonsfilene til applikasjonen mens den kjører. Dermed kan du filtrere meldingene som flyter gjennom instrumentasjonslaget.

5.3.4 Systematisering av instrumentasjonsdata

I dette kapitlet skal vi se på hvordan vi skal systematiseres instrumenteringsdata og lagre dataene før de brukes senere.

1. *Beregne og prioritere data*
2. *Lagre data*

For å administrere instrumenteringsdata utviklet vi en forvaltningsapplikasjon som heter Nyhet Manager (News Manager). Se figur 5.1.

Beregne og prioritere data

Vi har valgt å plukke ut et utvalg av de dataene som ble overført av EIF-meldingene. Klasse (__CLASS), tidspunkt (EventTime), meldingsnavn (EventSourceName), applikasjonsnavn (AppDomainName), prioritet (Severity), feilkode (ErrorCode) og meldingsfeltet (Message). Vi valgte å beholde verdiene av variablene som rådata. Teksten som står i parentes over i teksten er standardnavn som WMI bruker.

Lagre data

Fra instrumenteringsteorien hadde vi flere valg for å lagre instrumenteringsdata som minne, filer, database, over nettverk og over Internett. Vi valgte å bruke MySql database. Instrumenteringsdata ble lagret i en Tabell som vist i tabell 5.2.

Field	Type	Null	Key	Default	Extra
EventID	int(10) unsigned		PRI	*NULL*	auto_increment
__CLASS	varchar(50)				
EventTime	datetime			0000-00-00 00:00:00	
EventSourceName	varchar(50)				
AppDomainName	varchar(255)				
Severity	double			0	
ErrorCode	varchar(255)				
Message	varchar(255)				

Tabell 5-2 Tabell for instrumentasjonsdata fra MySql

EventID i tabellen over er bare en identifikasjonsnøkkel for tabellen.

5.3.5 Aksjonere på instrumentasjonsdata

Etter instrumenteringsteorien skulle vi utføre følgende punkter under aksjonere på instrumentasjonsdata.

1. *Grenseverdier*
2. *Endre instrumentering*
3. *Sende meldinger*

Vi valgte å ikke innføre noen form for aksjonering på instrumentasjonsdata.

5.3.6 Presentere instrumentasjonsdata

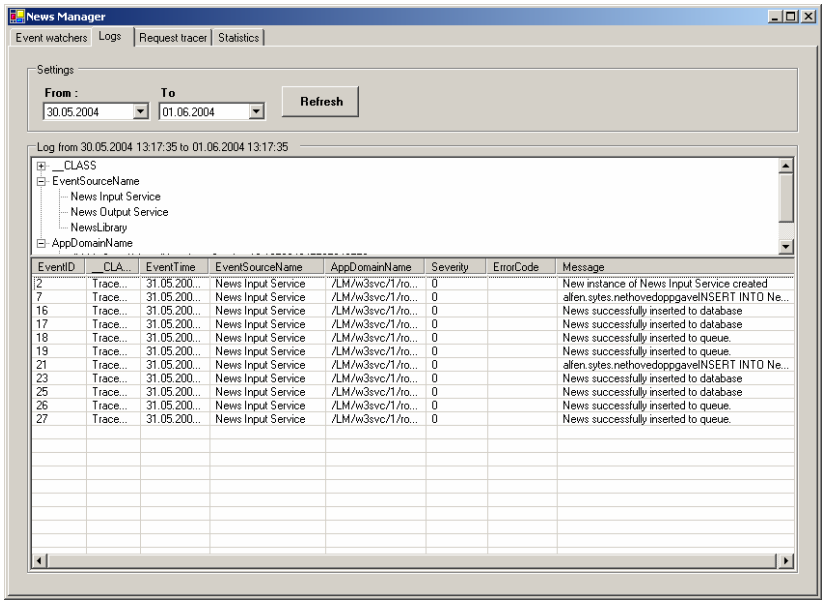
Etter instrumenteringsteorien skulle vi utføre følgende punkter under presentere instrumentasjonsdata.

1. *Målgruppe*
2. *Sikkerhet og informasjonsbehov*
3. *Brukervennlighet og presenteringsform*
4. *Hvilke håndteringer som kan foretas av bruker på prosesser (starte, stoppe osv) og instrumentering.*

Vi har prioritert å fokusere på presentering i punkt 3 og enkle håndteringer i punkt 4. De andre punktene har vi latt være å implementere på grunn av

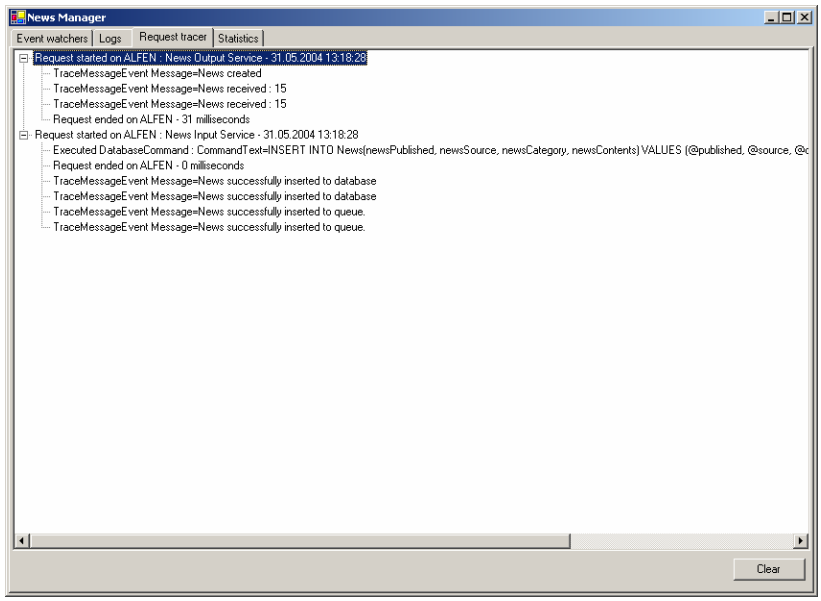
tiden det tar å utvikle programmeringskode i prototypen. De følgende figurer under 5.3.6 er tatt fra "News manager" som er vårt forvaltings system.

Presentering av instrumentasjonsdata



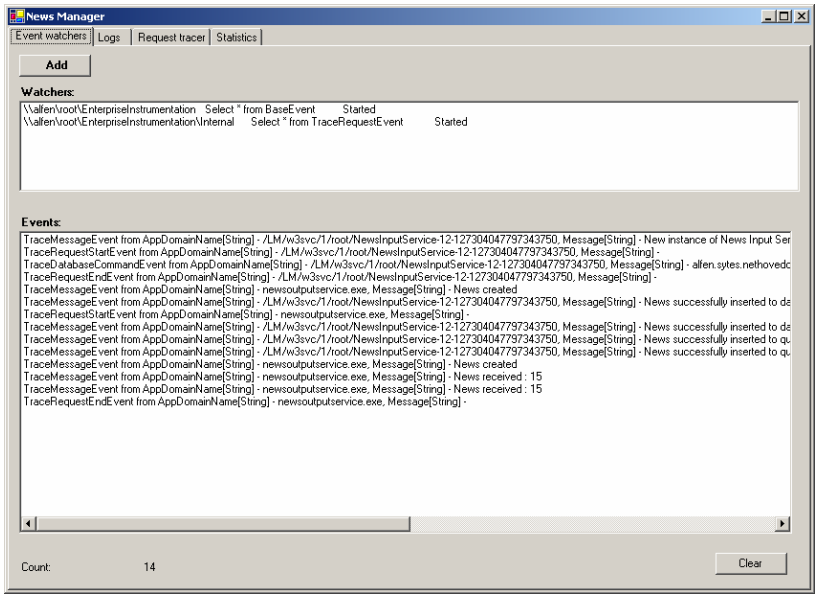
Figur 5-2 Hendelses logg

Alle hendelser sendt fra EIF blir lagret i en database og det er derfor mulig å hente disse historiske dataene ut igjen i hendelses logg, for senere analysering. Her finner vi igjen datafeltene i figur 5.2 vi valgte ut under å systematisere instrumenteringsdata.



Figur 5-3 Sporing ved forespørsel

"News Manger" har en side (Vist i figur 5.3) hvor den viser alle hendelsene sortert etter forespørsel som startet dem. Dette hjelper oss å forstå flyten i systemet og kan brukes til å finne ut hvordan ytelsen er.



Figur 5-4 Hendelses overvåker

For å samle inn hendelsene som blir sendt fra EIF gjennom WMI brønnen er det nødvendig å legge til en WMI abonnement. Dette er gjort mulig i ”News Manager” ved å fortelle den hvilket "namespace" og hva slags hendelse som ønskes. Nyhet systemet krevde fire abonnementer, to på hver server. En som lyttet etter hendelser som ble sendt ved forespørsel og en som lyttet på generelle hendelser. Utskriften er vist i figur 5.4.

Håndteringer

I "News Manager" ble det lagt inn mulighet å starte og stoppe "News Output Service".

6 Resultater

Når bedrifter blir spurt hvorfor de ikke har implementert en instrumentering løsning er det mest vanlige svaret på grunn av belastning. De hevder at instrumentering krever for mye ressurser av maskinen og velger dermed å klare seg uten. Ironien i dette er at de ikke kan vite ytelsen til systemet uten instrumentering.

Vi har laget en prototyp som viser flere av fordelene ved å ta i bruk instrumentering. Blant annet ble det avslørt en svakhet i systemet som førte til at systemet fikk problemer med å holde unna alle de innkommende nyhetene. Flaskehalsen ble funnet ved at det ble lagt inn metoder for å måle hvor lang tid det tar å eksekvere ulike operasjoner innen systemet. Etter en rask analysering kunne vi fort konstantere at årsaken til problemet lå i "News Output Service". En av klientene som var koblet opp mot "News Output Service" hadde en dårlig trådløs forbindelse hvilket gjorde at News Output Service brukte lang tid på å sende nyheten til denne klienten.

Windows har et innebygd instrumentasjon forvaltningsystem WMI, som brukes blant annet av Windows "Performance Monitor". WMI har tilgang til operativsystemets instrumentasjonsdata. EIF er at API som utvider WMI eksisterende hendelse, logging og sporings mekanismer allerede innebygd i Windows. Konfigureringen av EIF er lagret i en XML konfigurasjonsfil. Dersom konfigurasjonsfilen blir endret vil EIF sørge for at endringene gjøres gyldig i instrumenteringslaget, selv om applikasjoner er oppe og kjører.

Dersom meldingsflyten ble satt for høy mot EIF fra WMI, kunne man se at EIF ikke klarte å svelle unna meldingene. Meldingene gikk da tapt. Løsningen er å endre log nivå under drift fra en meldingsbrønn mot en fil. Dermed kunne filen ha ligget og ventet på å bli prosessert til resurser ble gjort tilgjengelig. Eller at kritiske hendelser bør sendes direkte, gjerne over MSMQ.

”Request tracing” eller sporing ved forespørsel er antageligvis en av de beste sidene ved EIF. Sporing ved forespørsel gjør det mulig å spore en forretnings prosess på kryss av applikasjonen sine grenser.

Ved bruk av EIF tilgjengeliggjørs et sett med ytelse tellere uten ekstra arbeid. Gjennom å bruke disse tellerne er det mulig å måle flyten av hendelser fra en hendelses kilde og gjennom en hendelses brønn.

7 Diskusjon

Vi vil i dette kapitlet diskutere resultatene i forrige kapittel. Videre vil vi diskutere noen av valgene vi har tatt i løpet av prosjektet. Først vil vi se på teorien og deretter prototypen.

7.1 Diskusjon av resultater i forrige kapittel

Et problem som kan dukke opp ved implementering av instrumentering er tilgjengeliggjøring av data. Det kan være at sensitive data vil gå ut av sensitive prosesser og inn i instrumentasjonslaget. Dette kan være en avgjørende faktor for å velge å innføre en instrumenteringsløsning.

EIF har sine begrensninger da EIF ikke har eksistert så lenge. EIF er fremdeles i versjon 1. Dermed kan mye ha vært oversett i versjon 1, og nytt materiale i EIF vil komme i neste generasjon. Blant annet har EIF begrensning i enveis kommunikasjon gjennom ”Request tracing”. Meldingsbrønner i EIF er av begrenset størrelse, og dersom disse går fulle kan vi miste viktig instrumenteringsdata. En god løsning vil være å la en applikasjon skrive til en lokal ”Trace log” og la denne loggen deles opp i enten en bestemt størrelse eller over en bestemt tid. Når en ny logg opprettes blir den gamle loggen sendt til en sentral maskin som leser loggen inn i en database for videre prosessering.

7.2 Instrumenteringsteori

Vi skal i dette kapitlet diskutere resultatene vi har fått ut av instrumenteringsteorien.

7.2.1 Identifisering

Gjennom å studere systemlaget har vi funnet ut at operativsystemet Windows er utrolig mye mer en det skrivebordet viser når vi har startet opp

datamaskinen. Studie av teknologi i kapittel 4 viste oss at mange av disse applikasjonene må startes opp fra kommandolinjen. Instrumentasjonslaget tenkte vi etter teorien var et lag mellom operativsystemet og applikasjonene vi skulle instrumentere. Det fant vi ut var en sannhet med modifikasjoner. Fordi teknologistudiet av forvaltningssystemer i kapittel 4 viste oss at grensen mellom applikasjonsdelen til forvaltningssystemet og instrumentasjonslaget gikk inne i applikasjonsdelen til forvaltningssystemet. I "Sun Management Center" og "Patrol" går vår teoretiske grense internt i serveren. For å vite hva en skal hente inn av data ved overvåking av en applikasjon er det viktig å vite hva som er relevant. Noen data vil gi et dårligere innsyn i applikasjonens tilstand en andre type data. Og noe data vil kanskje ikke fortelle noe som helst uten å tilgjengeliggjøre annen data.

7.2.2 Innsamling

Det eksisterer to måter å samle inn data på avhengig om en ønsker kvalitet eller kvantitet. Ved valg av kvalitet ønskes så mye instrumenteringsdata som mulig. Det kan ta lang tid å utvikle instrumenteringsmetoden, samle inn mye data og vil bruke mye prosessorkraft. Løsningen kan være å nyttiggjøre eksisterende instrumenteringsteknikker eller å velge en forhåndsdefinert standard. Dersom det er ønskelig med kvantitet er det ønskelig med få forhåndsdefinerte instrumenteringsdata med stor betydning for instrumenteringen. Utfordringen er lang planlegging og utviklingstid å finne frem til hvilke data en skal benyttes seg av. Fordelen ved å ta i bruk den kvalitative metoden med hjelp av en standard er at du raskt for etablert en instrumentering løsning som kan løse behovet for instrumenterings data. Ulempen er at den vil kreve mye ressurser og du kan risikere å drukne i informasjon. Ved utvikling av nye applikasjoner kan dette være en midlertidig løsning frem til en vet mer hva slags data en er på jakt etter, og dermed innføre en mer avanserte instrumentering etter hvert. I starten av en nylig utviklet applikasjon er ofte mengden data og trafikk også begrenset slik at en slik løsning vil fungere tilfredsstillende.

7.2.3 Instrumentering

Vi sa at den beste instrumenteringsløsningen er en kombinasjon av ekstern og intern instrumentering. Problemet er at vi ikke alltid kan bruke intern instrumentering dersom ikke kildekoden til de prosesser vi skal instrumentere er tilgjengelig. Vi har funnet ut at Windows har en tilleggskomponent WMI som kan installeres i Windows til støtte for ekstern instrumentasjon. WMI er en administrasjonsinfrastruktur med et felles sett med grensesnitt som støtter overvåking, styring av systemressurser, konfigurasjon og statuser. EIF tilbyr en standard metode for å publisere informasjon gjennom WMI. I EIF er det mulig å forandre meldingsbrønnen under drift. Dette kan være nyttig dersom en applikasjon skulle sende meldinger til en meldingslog som blir for full. Da kan vi i instrumentasjonslaget gi melding om at disse meldingene som går til meldingsloggen skal gå til en database istedenfor. Dermed har vi ikke gått inn i selve applikasjonen, men kun gjort disse endringene i instrumenteringslaget. Dette blir på fagspråket kalt for løst koplet.

7.2.4 Systematisere instrumentasjonsdata

Under systematisering av instrumentasjonsdata er det viktig å prioritere instrumentasjonsdata. Data som trengs kort aksjonstid bør prioriteres og lagres strukturert så de er lette å plukke opp igjen senere. Data som ikke trenger umiddelbar reaksjon på kan lagres ustrukturert. Eksempler på lavere prioritet er historiedata som brukes senere til feilanalyse.

7.2.5 Aksjonere på instrumentasjonsdata

Dersom prosessortiden plutselig skyter i været trenger ikke nødvendigvis bety at applikasjon oppfører seg unormalt. Enkelte applikasjoner jobber svært intensivt når den først arbeider og da vil det være naturlig at brukeren av prosessorkraft øker innenfor tidsintervaller.

7.2.6 Presentere instrumentasjonsdata

Kravet til hurtighet er ikke den samme da mye av tiden under presentasjonsdelen går til å vente på tastetrykk hos bruker. Dermed kan heller denne tiden utnyttes til beregning og strukturering av brukerdata. Mens aksjonsdelen av forvaltningssystemet som kjører hele tiden, har et evig behov av oppdatert strukturert instrumentasjonsdata.

7.3 Prototyp

Vi skal i dette kapitlet diskutere valgene vi tok i prototypen.

7.3.1 Identifisering

I vår oppgave har vi bare anvendt instrumentering i praksis på en prototyp som vi selv har laget. Dermed var vi i stand til å identifisere alle prosessene som skulle instrumenteres.

Systemlag

Vi har valgt å bruke Windows XP som operativsystem siden vi hadde bestemt Windows under begrensninger i oppgaven under kapittel 1.5.

Applikasjonslag

Vi valgte de applikasjonene vi selv hadde utviklet i nyhetssystemet.

7.3.2 Innsamling

Vi har valgt å samle inn instrumenteringsdata fra instrumenteringslaget for hente administrasjonsmeldinger, spormeldinger og feilmeldinger.

7.3.3 Instrumentering

Da vi selv hadde utviklet prototypen var vi i stand til å implementere intern instrumentering EIF.

Biblioteks filer (ByteFX) har vi valgt å ikke instrumentere da vi ikke har tilgang til kildekoden. MySQL Database har vi valgt å ikke instrumentere da vi måtte sette en strek for hvor mye vi skulle instrumentere.

7.3.4 Systematisering av instrumentasjonsdata

Beregne og prioritere data

Ved studie av EIF fant vi ut at WMI var veldig avansert og hadde veldig mange variabler. Derfor håndplukket vi noen få variabler. Da målet med prototypen var å vise hvordan vi kan instrumentere valgte vi å beholde verdiene i variablene som rådata, siden kalkulering er en programmering sak.

Lagre data

Siden vi skulle bruke instrumenteringsdata senere i presentasjonsdelen og at vi hadde valgt ut konkrete variabler, hadde vi behov for strukturert lagring. Det var database som hadde best mulighet for strukturert lagring. Dermed falt de andre alternativene bort. Deretter valgte vi MySql database, siden MySql er en gratis programvare og vi hadde kjennskap til MySql fra før. Vi valgte å beholde WMI navnene på variablene i MySql for å lettere få en sammenheng opp mot WMI.

7.3.5 Aksjonere på instrumentasjonsdata

Vi valgte å overføre all aksjonering manuelt i presentasjonsdelen, da dette var for tidkrevende å programmere.

7.3.6 Presentere instrumentasjonsdata

Det ble tatt et begrenset valg i denne delen av prototypen, da dette var for tidkrevende å programmere. Prioriteringen av presentering og håndtering var gjort for å bevise at det var mulig å få presentere instrumentasjonsdata og foreta enkle håndteringer i instrumentasjonslaget via EIF.

8 Konklusjon

Vi har erfart at tilgjengelighet ikke kan måles i oppetid alene. Det er mye viktigere å måle gjennomsnittstiden det tar å få systemet opp igjen etter et avbrudd. Fordi et lengre avbrudd vil føre til større frustrasjon hos bruker enn flere korte avbrudd over tid. Innsamling av instrumenteringsdata er til stor hjelp senere, da man kan bruker logg data til å sammenligne mot hverandre. Dette gjør oss i stand til å krysskontrollere historiedata ved feilsøking. En god instrumenteringsløsning kan forutsi problematiske tilstander i systemet. Dette gjør oss i stand til å planlegge og utføre mottiltak som kan unngå avbrudd. Dette beviser at gjennom instrumentering er det mulig å stabilisere systemer gjennom mottiltak som kan tømme brønner eller styre strømmen av meldinger mot andre brønner eller filer.

Instrumenteringsteorien vi utviklet var et godt utgangspunkt for å evaluere teknologi. Det vi opplevde var at instrumentering også foregår utenfor den teoretiske grensen vi satte i teorien. Vi fant ut at Windows har innebygd instrumentering WMI i operativsystemet for å ivareta styringen av resurser og prosesser. Vi fant ut at vi kunne få tak i WMI data ved bruk av EIF som er et grensesnitt mot applikasjoner. Noen begrensninger var det i EIF, men det vil nok bli forbedret ved neste versjon av EIF.

Under utviklingen av prototypen erfarte vi at instrumentering ikke bare hadde fordeler. Dersom man ikke begrenser seg ved innføring av instrumenteringsteknikker, er det lett å tilføre for mye instrumentasjon. Dette kan lett gå utover resurser som prosessor kraft, og vi opplevde at andre prosesser ble redusert. Som en tommelfinger regel kan man si at instrumentering skal maks bruke 10 % av tilgjengelig prosessorkraft.

Forkortelser

AF	Antall Feil
API	Application Program Interface
CIM	Common Information Model
DMI	Desktop Management Interface
DMTF	Desktop Management Task Force
EIF	Enterprise Instrumentation Framework
GA	Gjennomsnittstid for Avbrudd
GMA	Gjennomsnittstid Mellom Avbrudd
HTML	HypterText Markup Language
HTTP	HyperText Transfer Protocol
LAB	Logging Application Block
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
JMX	Java Management Extensions
MIB	Management Information Base
MOF	Managed Object Format
MSDN	Microsoft Developer Network
MSMQ	Microsoft Message Queuing
OS	Operativ System
SGMP	Simple Gateway Monitoring Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TA	Tid (i et) Avbrudd
TAF	Totalt (sum) Antall Feil
TCP	Transmission Control Protocol
TTNA	Tid Til Neste Avbrudd
UDP	User Datagram Protocol

WBEM	Web-Based Enterprise Management
WITSML	Wellsite Information Transfer Standard Markup Language
WMI	Windows Management Instrumentation
WQL	Windows Management Instrumentation Query Language
WSE	Web Services Enhancements
XML	Extensible Markup Language

Referanser

Alle hyperlenkene er gyldig medio august 2004.

[1] The Internet Engineering Task Force
http://www.ietf.org

[2] Microsoft Developer Network (MSDN)
http://msdn.microsoft.com

[3] Distributed Management Task Force (DMTF)
http://www.dmtf.org

[4] World Wide Web Consortium
http://www.w3.org

[5] A Simple Network Management Protocol (SNMP), RFC1157
http://www.ietf.org/rfc/rfc1157.txt

[6] Simple Gateway Monitoring Protocol (SGMP), RFC1028
http://www.ietf.org/rfc/rfc1028.txt

[7] Enterprise Instrumentation Framework (EIF)
http://msdn.microsoft.com/vstudio/productinfo/enterprise/eif/

[8] Web-Based Enterpise Management (WBEM)
http://www.dmtf.org/standards/wbem

[9] Common Information Model (CIM)
http://www.dmtf.org/standards/cim

[10] CIM Operations over HTTP, v1.0
http://www.dmtf.org/standards/documents/WBEM/CIM_HTTP_Mapping10.html

[11] Representation of CIM in XML (XML Mapping Specification), v2.0.0
http://www.dmtf.org/standards/documents/WBEM/CIM_XML_Mapping20.html

[12] Desktop Management Interface (DMI)
http://www.dmtf.org/standards/dmi

[13] Windows Management Instrumentation (WMI)
http://msdn.microsoft.com/library/en-us/wmisdk/wmi/wmi_start_page.asp

[14] Web Services Enhancements 2.0 (WSE)
http://msdn.microsoft.com/webservices/building/wse/

[15] Gwyn Cole
SNMP vs. WBEM – The Future of Systems Management
http://www.wbem.co.uk/articles/SNMPvsWBEM.pdf

[16] BMC Patrol
http://www.bmc.com/patrol

[17] Sun Management Center
http://wwws.sun.com/software/solaris/sunmanagementcenter/

[18] Microsoft Visual Studio
http://msdn.microsoft.com/vstudio/

[19] MySQL
http://www.mysql.com

[20] R. Kembel
A Comprehensive Introduction,
InternetWeek 4/3/2000 + Fibre Channel p.8.

[21] Jon Fancey
Powerful Instrumentation Options in .NET Let You Build Manageable Apps with Confidence, MSDN Magazine
http://msdn.microsoft.com/msdnmag/issues/04/04/InstrumentationinNET/default.aspx

[22] Evan Marcus, Hal Stern
Blueprints for High Availabilty
Publisert av Wiley
ISBN: 0-471-43026-9

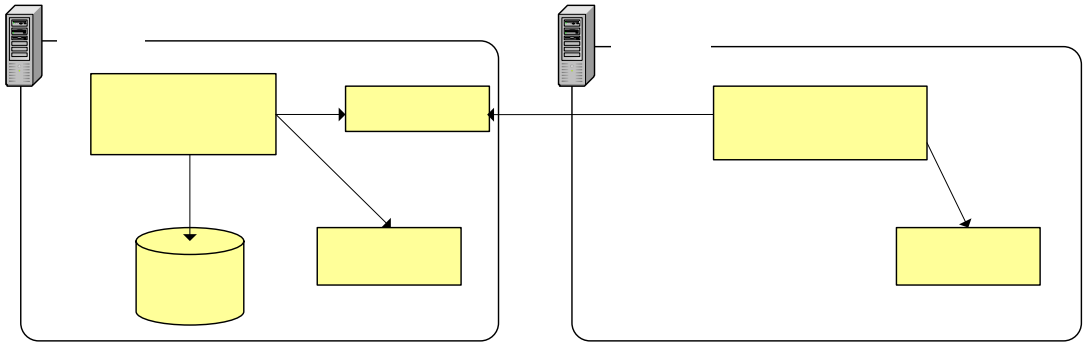
[23] Operating .NET Framework-based Applications
Microsoft Pattern & Practices
ISBN: 0-7356-1840-2

[24] Craig Tunstall, Gwyn Cole
Developing WMI Solutions,
A Guide to Windows Management Instrumentation.
Publisert av Addison Wesley
ISBN: 0-201-61613-0

Appendiks A - Nyhets system

Kildekoden til prototypen finnes på en CD på baksiden av denne rapporten.

A.1 - Arkitektur og design



Figur A-1 Nyhet systemet

Aritekturen involverer bruken av web applikasjoner, Windows Forms, Windows Services, en database og MSMQ.

A.2 - Nyhets In Service

Input Server

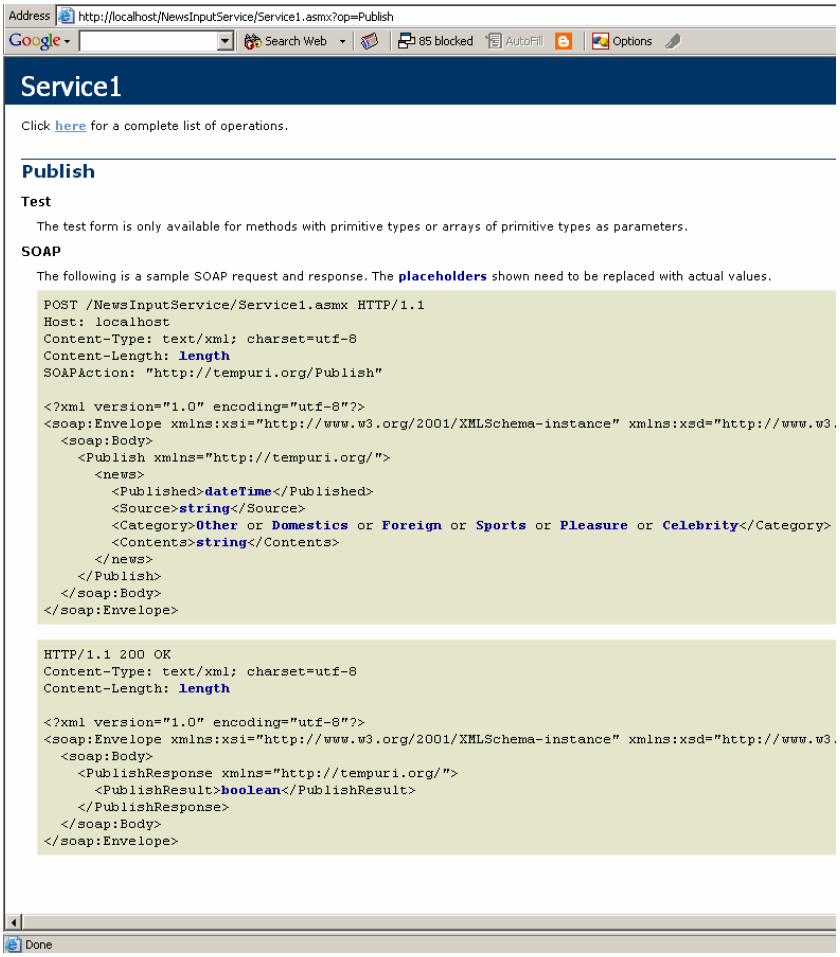
Nyhets In Service (News Input Service) er sammen med News Output Service kjerneprosessen av systemet. Denne Web tjenesten er kjernen for innsammling av nyheter. Web tjenesten samler inn alle nyheter gjennom å dra og skyve metoden, og lagrer nyhetene i en nyhetsdatabase (News Database).

<<Web Service>>
NewsInputService

<<MSMQ>>
NewsQueue

<<Assembly>>
NewsLibrary

<<MySQL>>
News Database



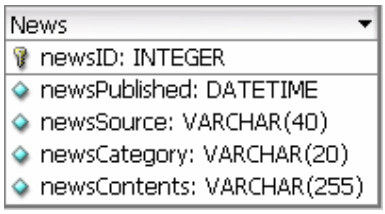
Figur A-2 NewsInputService SOAP forespørsel og svar.

A.3 - Nyhetsbibliotek

Nyhetsbibliotek (News Library) er et bibliotek som innholder noen av klassene som er delt mellom de ulike applikasjonene i systemet. Blant annet inneholder det den generelle nyhetsklassen.

A.4 - Nyhetsbatabasen

Nyhetsdatabasen (News Database) blir vedlikeholdt av MySQL. MySql er en åpen kildekodebasert databasetjener som er gratis til utdanning og personlig bruk. Siste versjon av databasen kan finner du på http://www.mysql.com.



Figur A-3 Nyhetsdatabasen

For å kunne aksessere MySQL har vi benyttet et gratis bibliotek fra ByteFX. Dette biblioteket kan lastes ned fra http://www.bytefx.com.

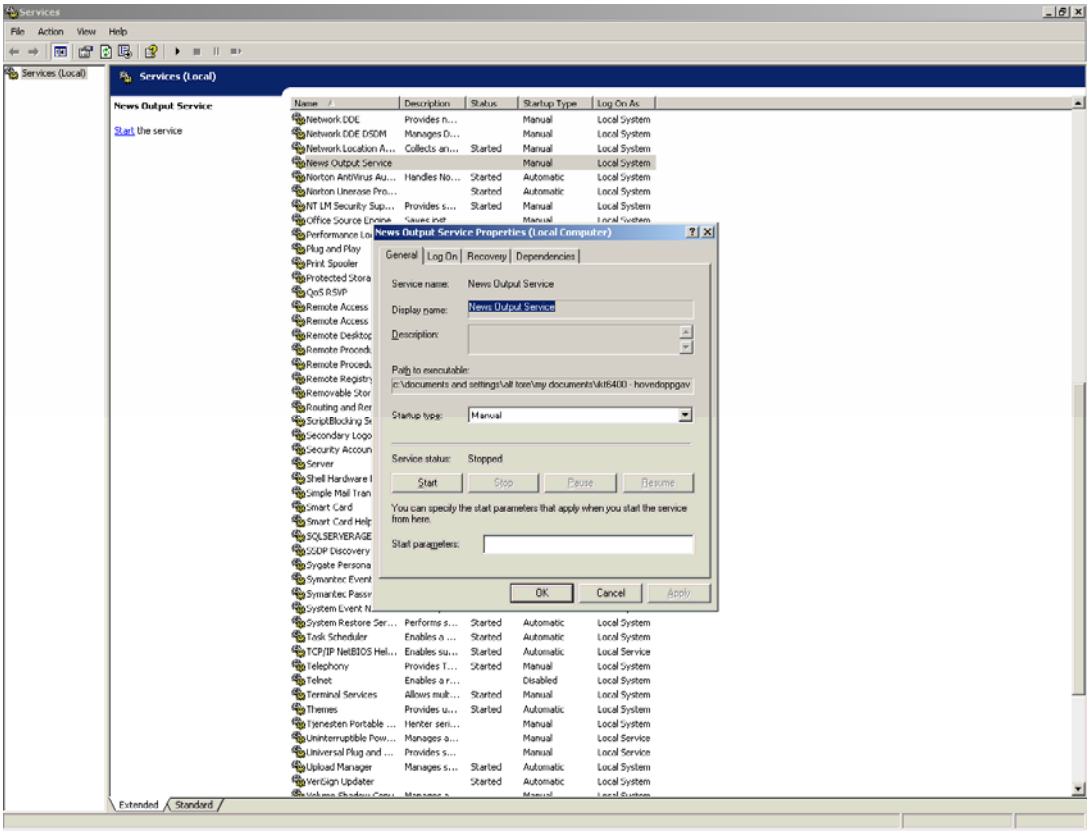
A.5 - Nyhet Ut Service

Nyhet Ut Service (News Output Service) er en Windows tjeneste. En Windows tjeneste er en bakgrunns prosess som er spesialtilpasset for å kunne kjøre stabilt over lengre tid og blir automatisk startet opp ved oppstart av maskinen.

News Output Service mottar nyheter fra News Input Service via en MSMQ.

Tjenesten kontrollerer også flyten ut fra systemet og sørger for at alle interessert og tilkoblede News Viewer klienter får beskjed når en nyhet ankommer systemet.

".NET remoting" er brukt til å sette opp en tjener som nyhetsabonnenter kan koble seg mot.



Figur A-4 Nyhet Ut Service

A.6 - Nyhet Web Input

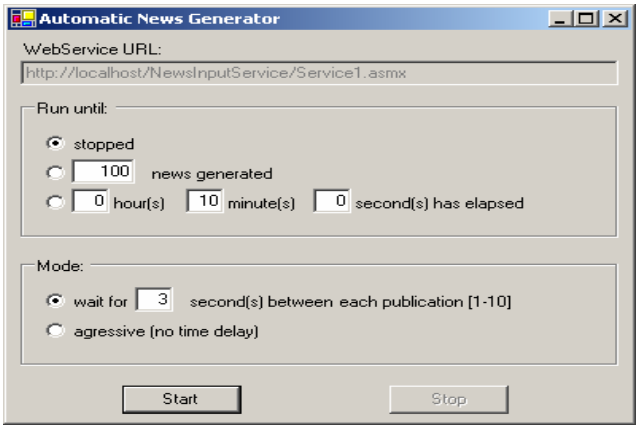
Nyhet Web Input (News Web Input) er en klient i Web format. Det er ingen begrensninger i antall klienter. Den enkle Web formen gjør det mulig å skrive meldinger og publisere meldinger ved å trykke sendeknappen. Deretter vil News Web Input prøve å sende meldingen til News Input Service. Dersom News Input Service prosesserer meldingen vil News Web Input skrive ”News successfylly added!” i Web formen. Dersom prosesseringen ikke går igjennom vil det komme opp en feilmelding i Web formen.

Figur A-5 Nyhet Web Input

Det er mulig å skrive "GENERATEERROR" i "contents" feltet for å tvinge frem en feilmelding til "NewsInputService".

A.7 - Automatisk nyhets generator

Automatisk nyhetsgenerator (News Generator) er en Windows applikasjon som brukes til å generere en mengde med meldinger. Generatoren har 3 forskjellige kjøremetoder, uendelig kjøring inntil den blir stoppet, mengdekjøring og tidskjøring. Alle disse 3 kjøremetodene kan kjøres enten i tidsintervall på et sekund eller i full hastighet.

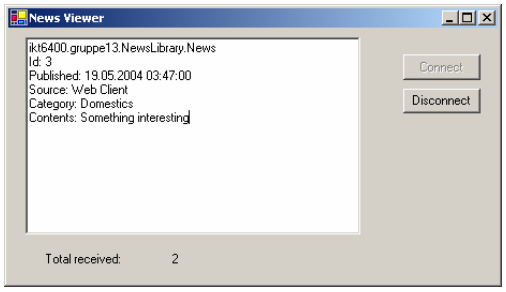


Figur A-6 Automatisk nyhetsgenerator

Generatoren I denne prototypen er bare laget for bruk I testing. Den er konstruert for å lage mange nyheter i en kort tidsperiode for å simulere høy trafikk.

A.8 - Nyhets abbonent

Nyhets abbonent (News Viewer) er en applikasjon som abonnerer på nyheter fra News Outpu Server. Når man trykker på ”Connect” knappen vil den prøve å koble seg til News Outpu Service og legge til et abonnement for nyheter. News Viewer bruker .NET til å sette opp en forbindelse.



Figur A-7 Nyhets abbonent

Appendiks B - Kildekode til prototyp

En kopi av kildekoden finnes på en cd vedlagt rapporten.