# Kriittisen tietojärjestelmän palveluiden valvonta

Pyry Kallio

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Pro gradu -tutkielma
Ohjaaja: Erkki Mäkinen
Marraskuu 2017

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Pyry Kallio: Kriittisen tietojärjestelmän palveluiden valvonta
Pro gradu –tutkielma, 50 sivua
Marraskuu 2017

---

Järjestelmänvalvonta ja sovellusvalvonta ovat tietojärjestelmien palveluntuotantoon liittyviä aktiviteetteja, joilla kohteena olevasta tietojärjestelmästä tai tietokonesovelluksesta kerätään ja analysoidaan sen suoritukseen liittyvää suorituksenaikaista dataa. Tätä dataa voidaan käyttää esimerkiksi järjestelmän ajonaikaisen toiminnan tarkkailuun, kriittisten häiriötilanteiden tunnistamiseen ja ennakointiin ja järjestelmän saatavuuden määrittelyyn ja laskentaan.

Valvonta on tärkeä aktiviteetti erityisesti kriittisissä tietojärjestelmissä, joiden on oltava toiminnassa jatkuvasti. Näissä järjestelmissä häiriötilanteet on havaittava nopeasti ja luotettavasti, jotta niiden vaikutus järjestelmän toiminnalle jää mahdollisimman pieneksi ja korjaavat toimenpiteet voidaan suorittaa mahdollisimman nopeasti.

Tässä tutkielmassa käsitellään valvonnan roolia erityisesti kriittisissä tietojärjestelmissä sekä esitellään valvontaan vaikuttavia keskeisiä suunnitteluperiaatteita ja rajoituksia liittyen kriittisiin tietojärjestelmiin. Näiden suunnitteluperiaatteiden taustalla on sekä aihetta käsittelevää kirjallisuutta että kokemuksia käytännön toteutustyöstä. Toteutustyö tehtiin hätä- ja hälytyskeskuskäyttöön suunniteltuun Insta Response -ohjelmistotuotteeseen.

Avainsanat: valvonta, monitorointi, palveluntuotanto, SLA, JMX, SNMP, saatavuus

**ALKUSANAT**

Haluan kiittää Insta DefSec Oy:tä ja esimiehiäni Teemu Ekolaa, Juha Nurmea ja Jukka Saarta mielenkiintoisesta tutkielman aiheesta ja mahdollisuudesta tehdä opinnäytetyö työtehtävien ohessa. Kiitokset myös Jari Liimataiselle työn sisällöllisestä ohjauksesta ja rakentavasta ja asiantuntevasta palautteesta sekä professori Erkki Mäkiselle työn ohjauksesta ja tarkastuksesta sekä hyvistä neuvoista työn kirjoittamiseen.

Lisäksi haluan kiittää ystäviäni ja perhettäni tuesta ja kannustuksesta työni eri vaiheissa. Erityiskiitoksen ansaitsee avopuolisoni Sofia, jolta olen saanut korvaamatonta tukea ja rohkaisua työtäni kirjoittaessa.

Tampereella, 17.11.2017

Pyry Kallio

Sisällysluettelo

# 1. Johdanto

Tietoyhteiskunta on kehittynyt voimakkaasti viime vuosikymmeninä ja tietoteknisiä ratkaisuja käytetään kaikilla elämän osa-alueilla. Niin julkishallinto, yritykset kuin teollisuuslaitoksetkin hyödyntävät yhä monimutkaisempia tietojärjestelmiä toiminnassaan ja yhteiskunta toimii pitkälti niiden varassa. Tietojärjestelmien ja ohjelmistojen laajamittainen käyttö tuo kuitenkin myös täysin uudenlaisia riskejä, kun vakava virhe järjestelmässä voi pahimmillaan johtaa mittaviin taloudellisiin tai inhimillisiin menetyksiin. Siksi on tärkeää, että tietojärjestelmän toimintaan liittyvät ongelmat ja häiriöt voidaan havaita mahdollisimman varhain ja järjestelmä kykenee toipumaan takaisin toimintakykyiseksi joko automaattisesti tai järjestelmänvalvojan tekemien manuaalisten toimenpiteiden kautta.

Jotta kriittisiin virhetilanteisiin voidaan reagoida mahdollisimman nopeasti ja tapauksen edellyttämällä tavalla, pitää järjestelmästä voida kerätä sellaista dataa, joka liittyy järjestelmän toimintaan ja voi paljastaa suorituksenaikaisia ongelmatilanteita. Tällaisen datan keräämistä tietojärjestelmästä ja sen analysointia kutsutaan järjestelmänvalvonnaksi, ja se on osa suurempaa tietojärjestelmien tuotantoon liittyvää kokonaisuutta, jota kutsutaan järjestelmänhallinnaksi. Toimiva järjestelmänvalvonta on erityisen kriittistä silloin, kun järjestelmävirheellä voi olla kohtalokkaita vaikutuksia joko ihmisille tai liiketoiminnalle.

Yksi kriittinen toimiala, joka nojaa vahvasti tietojärjestelmien varaan, on hälytys- ja hätäkeskustoiminta. Hätäilmoitusten vastaanotto, käsittely ja oikeiden viranomaisten hälyttäminen ei onnistu tehokkaasti ja luotettavasti ilman toimivaa tietojärjestelmää. Tietojärjestelmän kriittinen virhe hätäkeskuksessa voi johtaa myös ihmishenkien menetykseen, jos tarvittavaa apua ei saada hälytettyä ajoissa tai hälyttäminen epäonnistuu. Erityisesti hätäkeskustoiminnassa, joka on julkisesti rahoitettua ja kuuluu yhteiskunnan perusinfrastruktuuriin, on lisäksi paljon lainsäädännöllisiä vaatimuksia liittyen tietojen käsittelyyn ja palveluiden saatavuuteen, sillä hätäkeskusten on kyettävä vastaanottamaan hätäpuheluita kaiken aikaa ympäri vuorokauden ja käsiteltävä huolellisesti tietojärjestelmässä liikkuvaa tietoa.

Tämä tutkielma on tehty Insta DefSec Oy:n Insta Response -tuotekehitysprojektissa liittyen Insta Response -ohjelmistossa käytettyihin valvontaratkaisuihin. Insta Response on hälytys- ja hätäkeskustoimintaan

suunniteltu ohjelmistoratkaisu, joka toimii myös Suomen tulevan hätäkeskustietojärjestelmän ERICA:n perustana. ERICA on valtakunnallinen moniviranomaistietojärjestelmä, jota käyttävät kaikki Suomen hätäkeskukset Ahvenanmaata lukuun ottamatta sekä kaikki keskeiset viranomaiset: poliisi, pelastustoimi, ensihoito, Rajavartiolaitos ja sosiaalitoimi. ERICA-tietojärjestelmä on tarkoitus ottaa käyttöön vuoden 2018 aikana.

Tutkielmassani käsittelen valvontaa erityisesti kriittisen tietojärjestelmän osana ja siihen vaikuttavia tekijöitä, sekä teknologisia ratkaisuja ja käytäntöjä, jotka liittyvät kriittisen tietojärjestelmän palveluille toteutettavaan valvontaan. Käytännön kontekstina on toiminut Insta Response -ohjelmisto, johon on toteutettu tämän työn puitteissa sekä uusia valvontaominaisuuksia että jo aiemmin olemassa ollutta valvontaa täydentäviä ratkaisuja.

Luvussa 2 esittelen sovellus- ja järjestelmänvalvontaa yleisellä tasolla ja erityisesti hajautettujen tietojärjestelmien näkökulmasta. Lisäksi luvun viimeisessä kohdassa 2.5. esittelen tutkielman teoreettisena taustana käytettyä sovellus- ja järjestelmänvalvonnasta tehtyä tieteellistä tutkimusta sekä kirjallisuutta.

Luku 3 käsittelee nykyään yleisesti käytössä olevia valvontateknologioita sekä sovellusvalvonnan että järjestelmävalvonnan näkökulmasta. Tämä luku on jaettu kahteen kohtaan: ensimmäinen kohta 3.1. käsittelee sovellusvalvontateknologioita, joista tärkeimmät esitellään omissa alakohdissaan. Toinen kohta 3.2. esittelee järjestelmänvalvontaan käytettyjä tunnettuja teknologioita lähinnä siltä osin kuin lähdemateriaalissa on niitä käsitelty.

Luku 4 käsittelee valvonnan roolia kriittisissä tietojärjestelmissä sekä valvonnan vaatimuksia ja potentiaalisia ongelmia erityisesti kriittisten tietojärjestelmien yhteydessä. Kohdissa 4.1-4.3 esittelen keskeisiä käsitteitä liittyen kriittisiin tietojärjestelmiin ja niiden valvontaan. Kohta 4.4. esittelee kriittisten järjestelmien valvonnan suunnitteluun vaikuttavia tekijöitä ja kohta 4.5. valvontaan liittyviä potentiaalisia ongelmia kriittisissä tietojärjestelmissä.

Luvussa 5 esittelen hälytys- ja hätäkeskustoimintaa sekä yleisellä tasolla että Insta Response -ohjelmiston ja ERICA-tietojärjestelmän osalta ja tarkastelen myös valvontaa osana hätäkeskustietojärjestelmää. Esittely on melko pintapuolinen, koska tutkielman aihe ei liity suoraan näiden järjestelmien tavalliseen toimintalogiikkaan. Tarkoitus on lähinnä antaa lukijalle yleisen tason ymmärrys lukijalle hälytys- ja hätäkeskustoiminnasta ja tietojärjestelmän roolista ja käyttötarkoituksista näillä toimialoilla.

Luku 6 käsittelee Insta Response -ohjelmistoon toteutettuja valvontaratkaisuja. Luvussa esitellään yleisellä tasolla ja yleistasoisilla esimerkeillä, kuinka Insta Response -ohjelmistotuotteessa on toteutettu valvontarajapintoja ja

-toteutuksia. Lisäksi mukana on pohdintaa mahdollisista jatkokehitysideoista tuotteeseen. Viimeinen luku 7 on yhteenveto tutkielman käsittelemästä aiheesta.

## 2. Sovellusvalvonnan ja siihen liittyvien keskeisten käsitteiden määrittely

Valvonta on aktiviteetti, jonka määrittelyyn on kontekstista riippuen käytetty hieman erilaisia käsitteitä, kuten esimerkiksi *ajonaikainen verifiointi* (runtime verification), *ohjelmiston valvonta* (software monitoring), *järjestelmänvalvonta* (systems monitoring) ja *sovelluksen valvonta* (application monitoring). Näillä käsitteillä myös yleisesti ottaen tarkoitetaan hieman eri asioita. Pääsääntöisesti valvonta kuitenkin tarkoittaa kaikissa tapauksissa suorituksenaikaisen datan keräämistä järjestelmästä, tämän datan analysointia mahdollisten vikatilanteiden löytämiseksi, virhetilanteiden tunnistamista ja virhetilanteiden indikoimista asianosaisille henkilöille tai muille ulkopuolisille järjestelmille.

Valvonnan keskeisin tarkoitus on löytää järjestelmästä tai sovelluksista vikoja, jotka voivat olla kohtalokkaita järjestelmälle. Vika voi ilmentyä äkillisesti, mutta se voi ilmentyä myös pidempiaikaisen virheellisen toiminnan seurauksena. Goodloe ja Pike [2010] esittelevät kolmiosaisen luokittelun järjestelmävirheille: *vika* (fault), *virhe* (error) ja *häiriö* (failure). Häiriö tapahtuu, kun järjestelmä tai sovellus ei kykene tarjoamaan siltä vaadittuja toimintoja. Virhe on oletettavasti häiriöön johtava järjestelmän tila, kun taas vika on virheen oletettu tai todistettu aiheuttaja. Hyvän valvontatoteutuksen pitäisi aina havaita järjestelmän häiriöt, mutta parhaassa tapauksessa myös vikoja ja virheitä voidaan havaita valvonnan avulla.

Varhaisimmat lähdemateriaalit käsittelevät valvontaa lähinnä *ohjelmiston valvontana* (software monitoring) tai *ajonaikaisena verifiointina* (runtime verification), joka yleisemmin liittyy *ohjelmiston verifiointiin* (software verification). Delgadon ja muiden [2004] mukaan ohjelmiston verifiointimenetelmien, joihin kuuluu myös mallintarkistus (model checking) ja ohjelmistotestaus (software testing), tarkoituksena oli todentaa, että ohjelmiston *toteutus* vastaa sille tehtyä *määrittelyä.* Mallintarkistus ja ohjelmistotestaus nähtiin lähinnä tapoina, joilla pystytään verifioimaan ohjelmistoa kehitysvaiheessa, mutta ajonaikainen verifiointi tarjosi tietoa, miten ohjelmiston todellinen, ajonaikainen suoritus vastaa annettua määrittelyä.

Nämä varhaisimmat ratkaisut ohjelmistojen valvontaan painottuivat ns. *ei-ajonaikaiseen valvontaan* (offline monitoring), jossa kohdejärjestelmästä kerättiin valvontadataa sen suorituksen aikana, mutta tulokset analysoitiin vasta jälkikäteen. Vastaavaan käytäntöön viittaavat myös Limoncelli ja muut [2007]

termillä *historiallinen valvonta* (historical monitoring). Myöhemmin painopiste siirtyi *ajonaikaiseen valvontaan* (online monitoring), tai *reaaliaikaiseen valvontaan* (real-time monitoring), jossa määrittelyn ja ohjelman suorituksesta kerätyn valvontadatan vertaaminen keskenään tapahtui ajonaikaisesti. Toinen varhainen luokittelu valvontaratkaisuille on *järjestelmän sisäinen valvonta* (inline monitoring), jossa ohjelmaa tarkkaileva monitori on osa ohjelmakoodia ja *ulkoinen valvonta* (outline monitoring), jossa se on muusta järjestelmästä erillinen prosessi. [Goodloe and Pike, 2010]

Myöhemmin, kun *hajautetut tietojärjestelmät* (distributed systems) ovat kehittyneet, on myös hajautettujen tietojärjestelmien valvonnan tutkimus lisääntynyt. Erityisesti hajautetuissa tietojärjestelmissä valvonnasta on tullut keskeinen osa toteutusta ja tuotantokäyttöä, sillä nämä järjestelmät ovat usein rakenteeltaan niin monimutkaisia, että kaikkia mahdollisia ongelmatilanteita ei pystytä määrittelyllä tai testauksella tunnistamaan, ja vasta pitkäaikainen jatkuva tuotantokäyttö paljastaa, miten järjestelmä todella käyttäytyy.

### 2.1. Hajautetut tietojärjestelmät

Hajautettu tietojärjestelmä on tietojärjestelmä, joka koostuu fyysisesti ja loogisesti hajautetuista itsenäisistä osista, kuten käyttöliittymistä, tietokone- ja ohjelmistokomponenteista ja tietokannoista, jotka kommunikoivat keskenään ns. *väliohjelmiston* (middleware) avulla, yleensä tietoverkkojen kautta. Hajautettu järjestelmä pystyy suorittamaan hyvinkin raskasta rinnakkaista laskentaa, koska resursseja ja kuormitusta voidaan jakaa järjestelmän osien kesken, lisäksi järjestelmän osat voidaan hajauttaa myös maantieteellisesti ja järjestelmällä on tyypillisesti parempi *vikasietoisuus* (fault tolerance) kuin hajauttamattomalla järjestelmällä. [Techopedia, 2017a]

Tyypillinen esimerkki hajautetusta tietojärjestelmästä on suuri web-sovellus, jonka toimintalogiikan toteuttavat sovellukset toimivat palvelimella tai joka koostuu useista eri palvelimilla toimivista keskenään kommunikoivista sovelluksista, ja loppukäyttäjät käyttävät sovellusta web-käyttöliittymän kautta Internet-selaimella. Loppukäyttäjälle järjestelmä näyttää yhtenäiseltä kokonaisuudelta, vaikka se voikin olla hyvin moneen tasoon hajautettu.

Hajautetut tietojärjestelmät on usein myös ohjelmistotasolla hajautettu useiksi toisistaan erillisiksi *palveluiksi* (service), joista kukin vastaa jostakin kokonaisuudesta liittyen sovelluksen *toimintalogiikkaan* (business logic). Tämänkaltaista erillisiin palveluihin perustuvaa arkkitehtuuria kutsutaan *palvelukeskeiseksi arkkitehtuuriksi* (service-oriented architecture). Palvelukeskeinen arkkitehtuuri perustuu usein palveluiden tarjoamiin standardisoituihin avoimiin

rajapintoihin, mikä mahdollistaa kolmannen osapuolen komponenttien helpon käytön ja toisaalta myös vähentää riippuvuutta sovellusalustasta, koska järjestelmässä voidaan käyttää ristiin useamman eri osapuolen komponentteja.

Limoncelli ja muut [2007] määrittelevät palvelun olevan ”toiminto, jonka palvelin tarjoaa”. Edelleen Limoncellin ja muiden [2007] mukaan palveluiden on täytettävä asiakasvaatimukset ja oltava luotettavia ja ylläpidettäviä, joten palvelu ei ole ainoastaan laitteiston ja ohjelmiston yhteensovittamista, vaan ollakseen palvelu ohjelmistosta pitää tehdä luotettava sekä sitä pitää valvoa, hallita ja tukea, jotta ylläpidettävyys toteutuu.

### 2.2. Miksi tietojärjestelmää valvotaan?

Kuten aiemmin mainittiin, tietokonesovelluksien valvontaa toteutettiin alun perin varmentamaan, että sovelluksen ajonaikainen toiminta vastaa sovelluksen määrittelyä. Määrittelyn toteutumista on vaikea enää todentaa nykyaikaisilla hajautetuilla järjestelmillä, koska järjestelmät ovat niin monimutkaisia, että yhtenäistä määrittelyä on vaikea luoda, mutta ajonaikaisen toiminnan varmentaminen on silti edelleenkin tärkeä osa valvontaa. Sovellusvalvonta tarjoaa ulospäin näkymän järjestelmän sisälle ja muuttujiin, jotka eivät näy loppukäyttäjälle, joten valvonta voi tarjota todenmukaisempaa tietoa siitä, miten järjestelmä oikeasti käyttäytyy, sen sijaan, miten se näyttää käyttäytyvän, tai miten sen oletetaan käyttäytyvän. Tällaista valvontaa, joka perustuu järjestelmän tarjoamaan metriikkaan, kutsutaan ns. mustalaatikkovalvonnaksi, kun taas ulkoisesti näkyvän toiminnan valvonta on ns. valkolaatikkovalvontaa [Ewaschuck, 2016].

Sovellusvalvonta toimii kahteen eri suuntaan: sen avulla voidaan toisaalta todentaa, että sovelluksessa on vika tai häiriö, mutta myöskin, että sovellus toimii oikein. Valvontadata voi todistaa asiakasorganisaatiolle, että järjestelmä toimii oikein ja täyttää sille asetetut palvelutaso- ja saatavuusvaatimukset. Tämä on usein myös tärkeää, jos ohjelmistoa tarjotaan asiakkaalle palveluna: Limoncelli ja muut [2007] toteavat, että palvelu ei ole täydellinen, eikä sitä voi kutsua palveluksi, ellei saatavuuden, suorituskyvyn ja sovelluksen ”terveyden” valvonta ole toteutettu osana palvelua. Valvonta takaa parhaimmillaan sen, että ylläpitäjät havaitsevat ongelmat ajoissa ja voivat tehdä korjaavat toimenpiteet ennen kuin se vaikuttaa useamman käyttäjän toimintaan [Limoncelli et al., 2007]

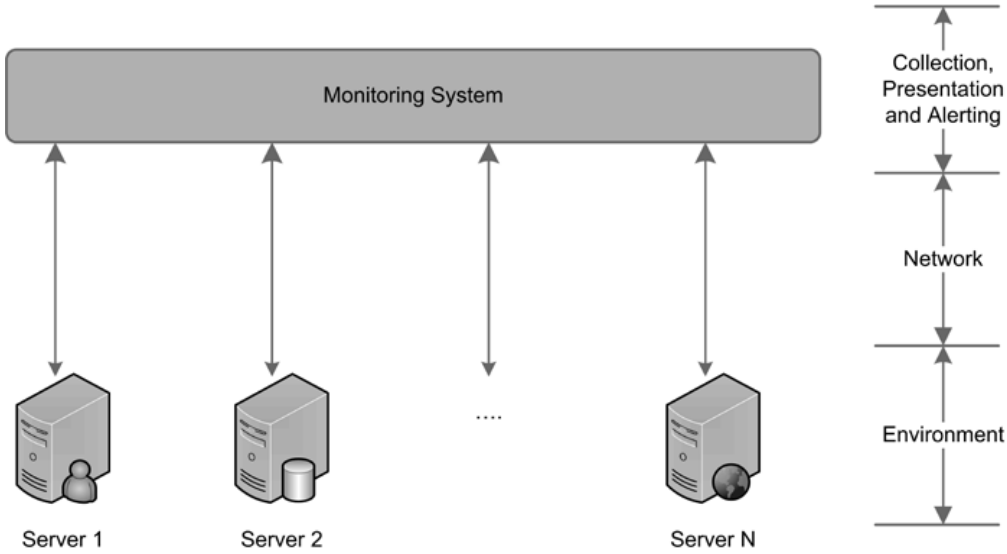
Valvontadatan kerääminen sekä vikatilanteiden että normaalin käytön osalta on tärkeää myös ns. *historiallisten trendien* (historical trends) tunnistamiseksi. Ohjelmistoon voi tulla vika, joka ilmenee välittömästi, mutta monet

viat nousevat esiin hiljalleen, ja niistä voi olla merkkejä nähtävissä jo aiemmin. Sovelluksen muistinkäyttö voi esimerkiksi kasvaa hitaasti, mutta jos sovelluksessa on muistivuoto, on todennäköistä, että muisti loppuu jossain vaiheessa, joten historiallisia trendejä seuraamalla ongelma voidaan havaita jo varhain ja korjata ennen kuin siitä tulee kriittinen. Muita keskeisiä tietojärjestelmästä valvonnan avulla seurattavia muuttujia ovat esimerkiksi tallennustilan ja tallennusmedioiden käyttö- ja täyttöaste, tietoverkkoliikenne järjestelmän osien välillä sekä muistin ja konekohtaisten järjestelmäresurssien käyttö.

### 2.3. Hajautetun järjestelmän valvonta

Hajautetuissa järjestelmissä *valvonta* (monitoring) voidaan määritellä siten, että se on informaation välittämistä käytetyistä resursseista ja järjestelmän osien ”terveydestä”. Edmiston [2007] erottelee valvonnan kahteen osakokonaisuuteen: *järjestelmänvalvontaan* (systems monitoring) ja *sovellusvalvontaan* (application monitoring). Edmistonin [2007] mukaan järjestelmänhallintaan käytetyn tiedon, kuten koko järjestelmän vasteaikojen ja resurssien käytön välittäminen, on järjestelmänvalvontaa, kun taas saman informaation välittäminen sovelluksista on sovellusvalvontaa. Keskeinen ero näillä on se, että järjestelmänvalvonta valvoo koko tietojärjestelmän infrastruktuuria, kun taas sovellusvalvonta valvoo yksittäisiä palveluita tai muita sovelluksia, jotka toimivat tämän infrastruktuurin päällä.

Sovellusvalvonnasta käytetään myös erilaisia nimityksiä, kuten *sovelluksen valvonta* (application monitoring), *sovelluksen suorituskyvyn valvonta* (application performance monitoring), *järjestelmän tietoverkon valvonta* (network monitoring), *saatavuuden valvonta* (availability monitoring) tai *sovelluksen suorituksen valvonta* (application execution monitoring). Vaikka nämä kaikki liittyvät toisiinsa, ovat ne kuitenkin erillisiä osakokonaisuuksia.




*Kuva 1: Hajautetun tietojärjestelmän valvonta-arkkitehtuuri [Kufel, 2016]*

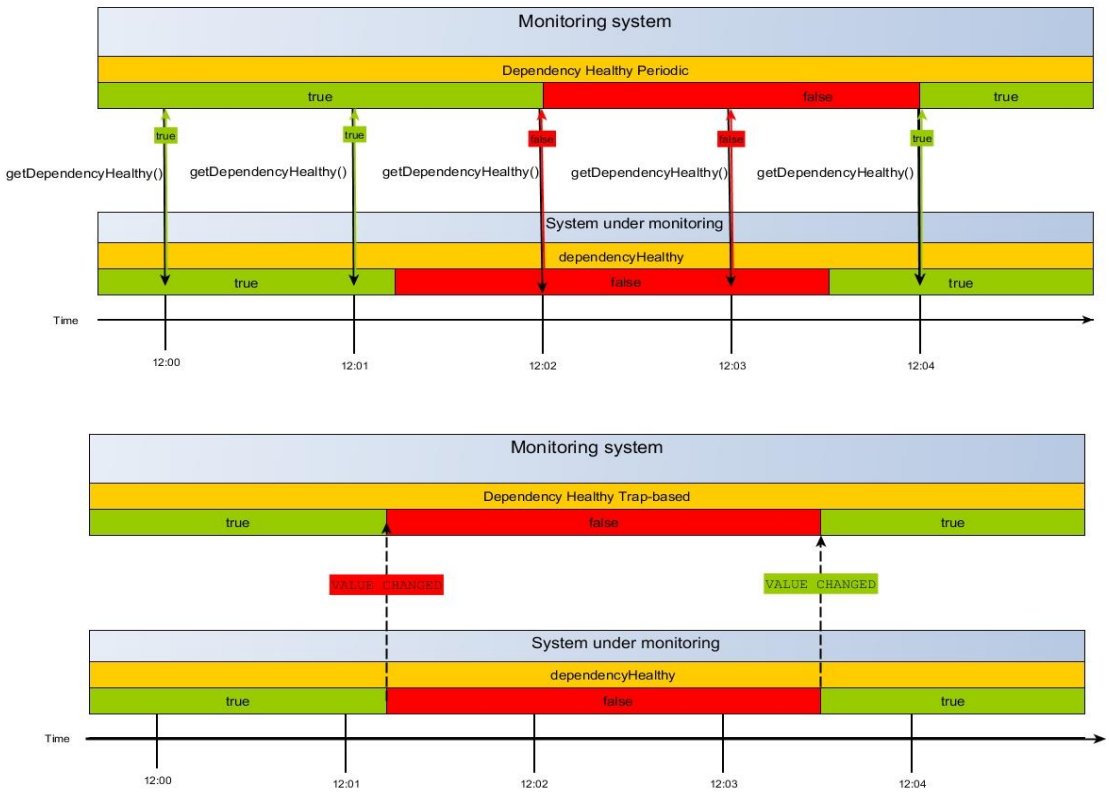
Hajautetussa järjestelmässä myös valvonta on hajautettu useampaan osaan. Hajautetun järjestelmän valvonta voidaan jakaa kolmeen kerrokseen: ympäristöön, järjestelmän tietoverkkoon ja valvontaan [Kufel, 2016]. Näiden osien keskinäinen toiminta on esitetty kuvassa 1. Ympäristö käsittää varsinaisen valvottavan järjestelmän, eli laitteisto- ja ohjelmistoresurssit sekä tallennusmediat. Tietoverkko välittää kerättyä valvontadataa kohdejärjestelmästä valvomolle, jossa data analysoidaan ja analyysin tulosten pohjalta tehdään tarvittavat toimenpiteet. Valvomolla on tyypillisesti käytössään jonkinlainen *valvontajärjestelmä* (monitoring system), joka kerää kohdejärjestelmältä dataa, analysoi sitä ja muodostaa hälytyksiä, jos esimerkiksi jonkin valvottavan muuttujan kriittinen arvo ylittyy. Valvontajärjestelmä pystyy usein myös tekemään jonkinlaisia automaattisia toimenpiteitä valvottavalle järjestelmälle häiriöiden korjaamiseksi.

Sekä järjestelmänvalvonta että sovellusvalvonta ovat osa suurempaa kokonaisuutta, jota kutsutaan nimellä *järjestelmänhallinta* (systems management). Järjestelmänhallinta kokonaisuutena on joukko aktiviteetteja, joilla järjestelmä pidetään toiminnassa tuotantokäytössä. Valvonta auttaa järjestelmänvalvojia tunnistamaan mahdollisia vikatilanteita järjestelmästä, ja parhaassa tapauksessa valvonta tarjoaa myös ennakoivaa tietoa vikatilanteista, jolloin järjestelmänvalvoja voi tehdä korjaavia toimenpiteitä jo ennen kuin vikatilanteita ilmenee järjestelmässä. Muita järjestelmänhallintaan liittyviä aktiviteetteja ovat esimerkiksi järjestelmäpäivitysten asentaminen, järjestelmän osien sammuttaminen tai uudelleenkäynnistäminen ja järjestelmäkonfiguraatioiden muokkaaminen.

## 2.4. Yleiset valvonta-arkkitehtuurit

Järjestelmänvalvonta koostuu yleensä kahdesta osasta: valvottavasta järjestelmästä ja valvontajärjestelmästä. Valvottavassa järjestelmässä tai sen osissa on tyypillisesti joko sisäinen tai siihen integroitu valvontatoteutus, joka tarjoaa valvontarajapintojen kautta valvontadataa valvontajärjestelmälle. Tällaista toteutusta kutsutaan *monitoriksi*. Valvontadatan lähteenä käytetään varsinaisten monitorien lisäksi usein myös sovelluksen *käyttölokia* (log) eli sovelluksen suoritukseen liittyviä viestejä, jotka kuvaavat sovelluksen toimintaa kronologisessa järjestyksessä. Tässä tutkielmassa käsittelen kuitenkin lähinnä varsinaisia monitoreita valvontadatan lähteenä.

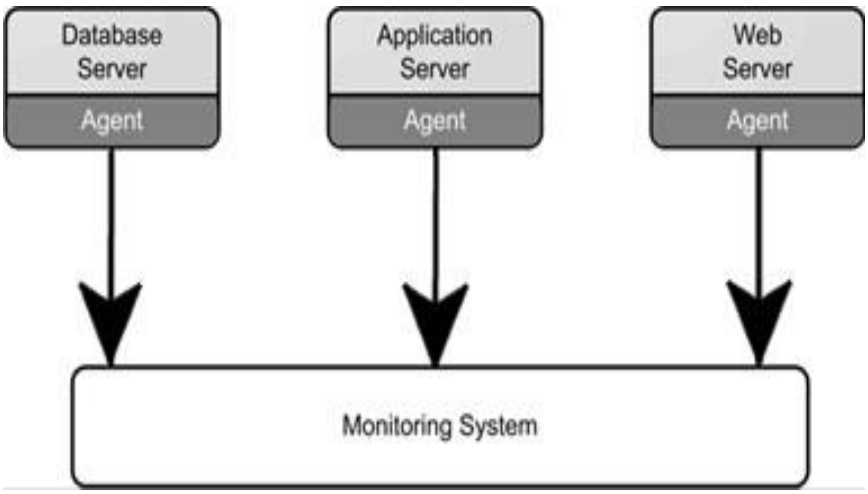
Valvontajärjestelmä kerää näiden rajapintojen kautta valvottavalta järjestelmältä tai sovellukselta valvontadataa, pitää sitä tallessa, analysoi sitä ja muodostaa siitä järjestelmänvalvojien ja järjestelmän ylläpitäjien käyttöön kuvaajia tai hälytyksiä. Valvontajärjestelmää voidaan joissakin tapauksissa kutsua myös hallintajärjestelmäksi, jos siihen on toteutettu myös muita järjestelmänhallintaominaisuuksia.




*Kuva 2: Periodinen ja paikallisperustainen valvonta*

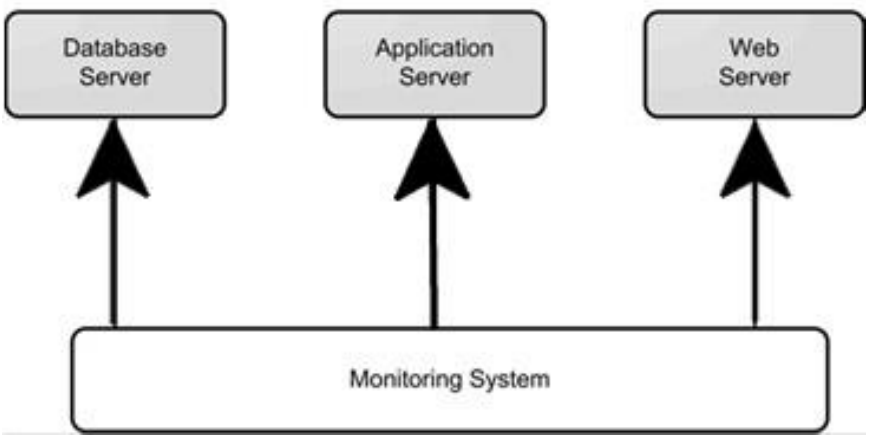
Valvontadatan kerääminen voi tapahtua joko *periodisesti* (polling-based) tai *paikallistamisperustaisesti* (trap-based) [Limoncelli et al., 2007]. Periodisesti

tapahtuvassa kyselyssä kunkin muuttujan arvo kysytään valvottavalta järjestelmältä tietyn aikaintervallin välein, esimerkiksi kerran viidessä minuutissa. Paikallistamisperustainen valvonta taas tarkoittaa, että valvottava järjestelmä tai sen osa lähettää itsenäisesti jonkinlaisen viestin tai hälytyksen valvontajärjestelmälle, jos jonkin valvottavan muuttujan arvo järjestelmässä muuttuu, tai muuttujan kriittinen raja-arvo ylitetään. Näitä kahta eri tapaa kerätä valvontadataa on havainnollistettu kuvassa 2.




*Kuva 3: Agenttipohjainen valvonta-arkkitehtuuri [Kufel, 2016, mukaillen].*

Monet hajautetuissa järjestelmissä nykyään käytettävät valvontaratkaisut perustuvat ns. manageri-agentti-arkkitehtuuriin, jossa valvontadataa keräävät järjestelmään integroidut *ohjelmistoagentit* (software agent), ja hallintajärjestelmään on toteutettu ns. manageri, jonka kanssa agentit kommunikoivat ja jolle ne välittävät keräämäänsä dataa ja tapahtumia. Kuva 3 havainnollistaa, kuinka agentit ovat integroituna valvottaviin järjestelmiin ja pystyvät lähettämään itsenäisiä viestejä managerikomponentille.




*Kuva 4: Agentiton valvonta-arkkitehtuuri [Kufel, 2016, mukaillen]*

On olemassa myös agentittomia valvontaratkaisuja, jotka käyttävät järjestelmän sisäänrakennettuja valvontateknologioita. Tällainen arkkitehtuuri on esitetty kuvassa 4. Nämä ratkaisut eivät aina tarjoa yhtä yksityiskohtaista tietoa kuin agentteja käyttävät ratkaisut, mutta niiden käyttöönotto on helpompaa, koska ne ovat kevyitä ratkaisuja, jotka eivät vaadi lisäohjelmistojen asentamista ja käyttöönottoa kohdeympäristöön. [Kufel, 2016].

### 2.5. Valvontaa käsittelevä kirjallisuus ja tutkimus

Valvontaa käsittelevää tutkimusta ja kirjallisuutta on olemassa, mutta niiden kokonaisuus on kuitenkin enemmän tai vähemmän hajanainen. Yksi ensimmäisiä koostavia tutkimuksia aihealueeseen liittyen on Plattnerin ja Nievergeltin [1981] tutkimus. Heidän mukaansa aiemmat tutkimukset aiheeseen liittyen olivat lähinnä tapaustutkimuksia ilman, että niissä varsinaisesti yritettiin muodostaa perustavanlaatuisia käsitteitä liittyen aihealueeseen. Tämän takia he pyrkivät koostamaan aiempia tutkimuksia ja esittämään niiden pohjalta muodostettuja yleisiä käsitteitä.

Myös hajautettujen tietojärjestelmien valvontaa on käsitelty useissa tutkimuksissa jo 80- ja 90-luvulla. Näistä mainittakoon esimerkiksi Joycen ja muiden [1987] tekemä tutkimus. Nämä tutkimukset eivät tarjoa välttämättä nykyaikana kaikilta osin kovin relevanttia tietoa, koska teknologiat ja tietojärjestelmät ovat kehittyneet paljon, mutta toisaalta niistä on löydettävissä myös edelleen tärkeitä perusperiaatteita valvontaan liittyen.

Muuten useat valvontaan liittyvät tutkimukset ovat melko yksittäisiä tapauksia. Monissa artikkeleissa on toteutettu jonkinlaiseen tarkoitukseen valvontaratkaisu ja analysoitu sen tuloksia, tällainen on esimerkiksi van Hoornin ja muiden [2012] tutkimus. Lisäksi joissakin tutkimuksissa on tutkittu tarkemmin jotain erityistä valvontaan liittyvää tekijää, kuten esimerkiksi valvontaan liittyvää suorituskyky- ja resurssikuormitusta Lahmadin ja muiden [2009] tutkimuksessa.

Sovellusvalvontaa ja järjestelmänvalvontaa erityisesti yritystason järjestelmissä on esitelty kattavammin esimerkiksi Changin ja Minkinin [2008] sekä Edmistonin [2007] tutkimuksissa. Myös Kufelin [2016] ja Hernantesin ja muiden [2015] artikkelit käsittelevät yritystason valvontajärjestelmien keskeisiä käsitteitä, vaatimuksia ja valintakriteereitä. Näissä tutkimuksissa on koostetusti myös tietoa kriittisen tietojärjestelmän valvontaan keskeisesti liittyvistä tekijöistä, kuten palvelutasovaatimuksista, valvontaintervalleista ja kerätyn valvontadatan säilytysajoista.

Kriittisiin tietojärjestelmiin on viitattu myös Yhdysvaltain ilmailu- ja avaruushallinto NASA:n teknisessä raportissa [Goodloe and Pike, 2010], jossa on esitelty erityisesti reaaliaikaisten hajautettujen järjestelmien valvontaa. Raportissa on esitelty myös todellisia järjestelmähäiriöitä lentokoneiden ja avaruussukkuloiden ohjausjärjestelmissä, jotka olisivat mahdollisesti olleet valvonnalla havaittavissa.

Järjestelmänvalvontaan ja sovellusvalvontaan liittyen on kirjoitettu myös paljon kirjoja, sekä yksittäisistä valvontateknologioista ja -järjestelmistä että yleisemminkin järjestelmänhallinnasta, hajautettujen järjestelmien rakentamisesta, palveluntuotannosta ja muista teemoista, joissa valvonta on keskeinen osa. Teknologiakohtaista kirjallisuutta edustaa esimerkiksi Kregerin ja muiden [2002] kirja JMX-valvontateknologiasta, joka on tosin jo melko vanha kirja ja siksi lähteenä kriittisesti tutkittava, mutta kuitenkin hyvä perusteos JMX-teknologiasta. Limoncelli ja muut [2007] puolestaan käsittelevät kirjassaan enemmänkin järjestelmänhallintaa ja verkonhallintaa, mutta myös valvontaa käsitellään useammassa kirjan luvussa. Ewaschuck [2016] puolestaan esittelee kirjan luvussa valvontaan liittyviä periaatteita ja käytäntöjä erityisesti teknologiayhtiö Googlen ohjelmistopalveluiden palveluntuotannosta vastaavan Site Reliability Engineering -tiimin (SRE) näkökulmasta.

# 3. Tunnettuja valvontateknologioita

Sovellusvalvontaan ja järjestelmänvalvontaan on kehitetty useita erilaisia teknologioita ja työkaluja; tässä luvussa esittelen muutamia tunnetuimpia. Esittely keskittyy pääasiassa avoimesti saatavilla oleviin ratkaisuihin, koska kaupallisista teknologioista ei ole kovin paljon maksutonta tietoa saatavilla. Niitä on kuitenkin jonkin verran esitelty lähdemateriaaleissa, joten tässä luvussa kerrotaan myös tunnetuimmista kaupallisista teknologioista yleisellä tasolla.

## 3.1. Sovellusvalvonta

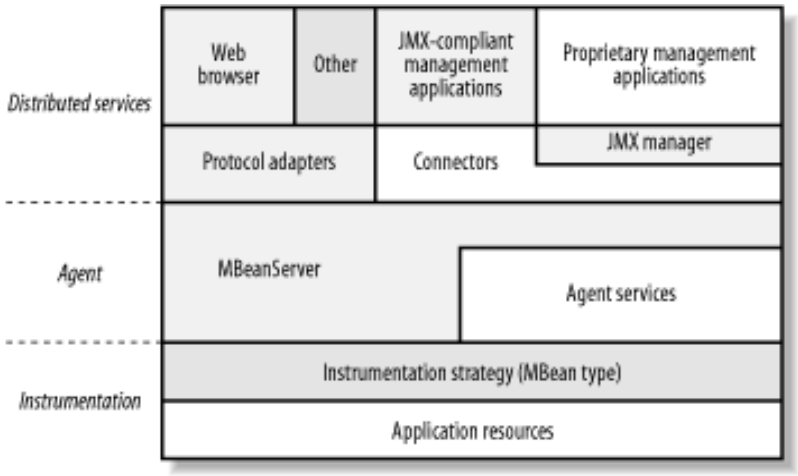
Sovellusvalvontaan kuuluu teknologioita, joiden avulla valvonta voidaan integroida osaksi valvottavaa järjestelmää. Osa näistä teknologioista on ympäristöriippuvaisia, esimerkiksi JMX on Java-spesifinen teknologia, mutta myös alustariippumattomasti erilaisiin ympäristöihin soveltuvia ratkaisuja, kuten esimerkiksi SNMP, on toteutettu. On myös teknologioita, jotka on toteutettu tietyllä teknologialla, mutta jotka tarjoavat ohjelmointirajapinnan (API) myös muille teknologioille.

Sovellusvalvonnassa ei ole kuitenkaan olemassa kovin montaa standarditeknologiaa, vaan usein sovellusvalvonta on toteutettu olemassa olevilla ohjelmointiteknologioilla, tai niiden pohjalta on kehitetty pienempiä sovelluskehyksiä, jotka eivät ole välttämättä saavuttaneet laajaa suosiota. Tällaisia itse tehtyjä sovelluskehyksiä sovellusvalvontaan ovat esimerkiksi Kieker Framework [van Hoorn et al., 2012] ja Yammer Metrics [Yammer Metrics, 2017].

### 3.1.1 JMX

Java Management Extensions (JMX) on Javan tarjoama hallinta-arkkitehtuuri ja sovellusohjelmointirajapinta (API) Java-sovellusten hallintaan. Se otettiin jo varhaisessa vaiheessa osaksi Javan J2EE-ohjelmistokehitysalustaa (Java 2 Platform Enterprise Edition), joka on suunniteltu erityisesti hajautettujen yritysohjelmistojen ja -tietojärjestelmien kehitykseen. Koska J2EE on laajalti käytetty ohjelmointiteknologia hajautetuissa järjestelmissä, on myös JMX:stä tullut tietynlainen de facto -standardi järjestelmänhallinnassa ja -valvonnassa erityisesti hajautettujen järjestelmien osalta. JMX:n arkkitehtuuri koostuu kolmesta kerroksesta: mittauskerroksesta, agenttikerroksesta ja hallintakerroksesta. Kor-

keammalla tasolla myös JMX:n voidaan nähdä toteuttavan manageri-agentti-tyyppisen arkkitehtuurin. Kuvassa 5 on esitelty JMX:n ohjelmistoarkkitehtuuri.




*Kuva 5. JMX:n arkkitehtuuri [Kreger et al., 2002]*

Mittauskerros (instrumentation layer) mahdollistaa sovelluksen resurssien valvonnan ja hallinnan sovellukselle määriteltyjen hallintarajapintojen (Managed Beans, myöhemmin MBeans) kautta. MBean-rajapinta on Java-kielellä kirjoitettava rajapintaluokka, joka usein nimetään MBean-päätteiseksi, mutta se ei ole välttämättä pakollista. Rajapinnan metodeita kutsutaan myös JMX-attribuuteiksi, ja ne määritellään Javan nimeämiskäytäntöjen (naming conventions) mukaan.

Sovelluksen valvontaan käytetään yleensä get- tai is-tyyppisiä attribuutteja, jotka palauttavat sovelluksen resursseihin sidottujen muuttujien arvot. Get-tyyppiset attribuutit kuvaavat minkä tahansa tyyppisen muuttujan, esimerkiksi *getResponseTimeMilliseconds()* palauttaa vasteajan millisekunteina. Is-tyyppiset attribuutit puolestaan kertovat boolean-tyyppisen muuttujan, esimerkiksi *isDatabaseConnectionHealthy()* kertoo, onko sovelluksen tietokantayhteys kunnossa. Lisäksi on olemassa set-tyyppisiä JMX-operaatioita, joilla voidaan käskeä JMX-asiakassovelluksesta hallittavaa sovellusta suorittamaan operaatioita. Esimerkiksi *setEnabled()* voisi asettaa jonkin sovelluksen osan käytettäväksi. Set-attribuutit liittyvät kuitenkin muihin sovellushallinnan osiin kuin valvontaan, koska valvonnan tarkoitus ei ole manipuloida valvottavaa järjestelmää.

Agenttikerros muodostuu MBean-palvelimesta (MBean Server), johon kaikki käytettävät hallintarajapinnat rekisteröidään. Rekisteröintiin käytetään rajapinnan instanssin yksilöivää ns. objektinimeä (objectName), joka voi olla esimerkiksi tyyppiä *com.example:type=ExampleMonitor*. Kun valvontarajapinnan

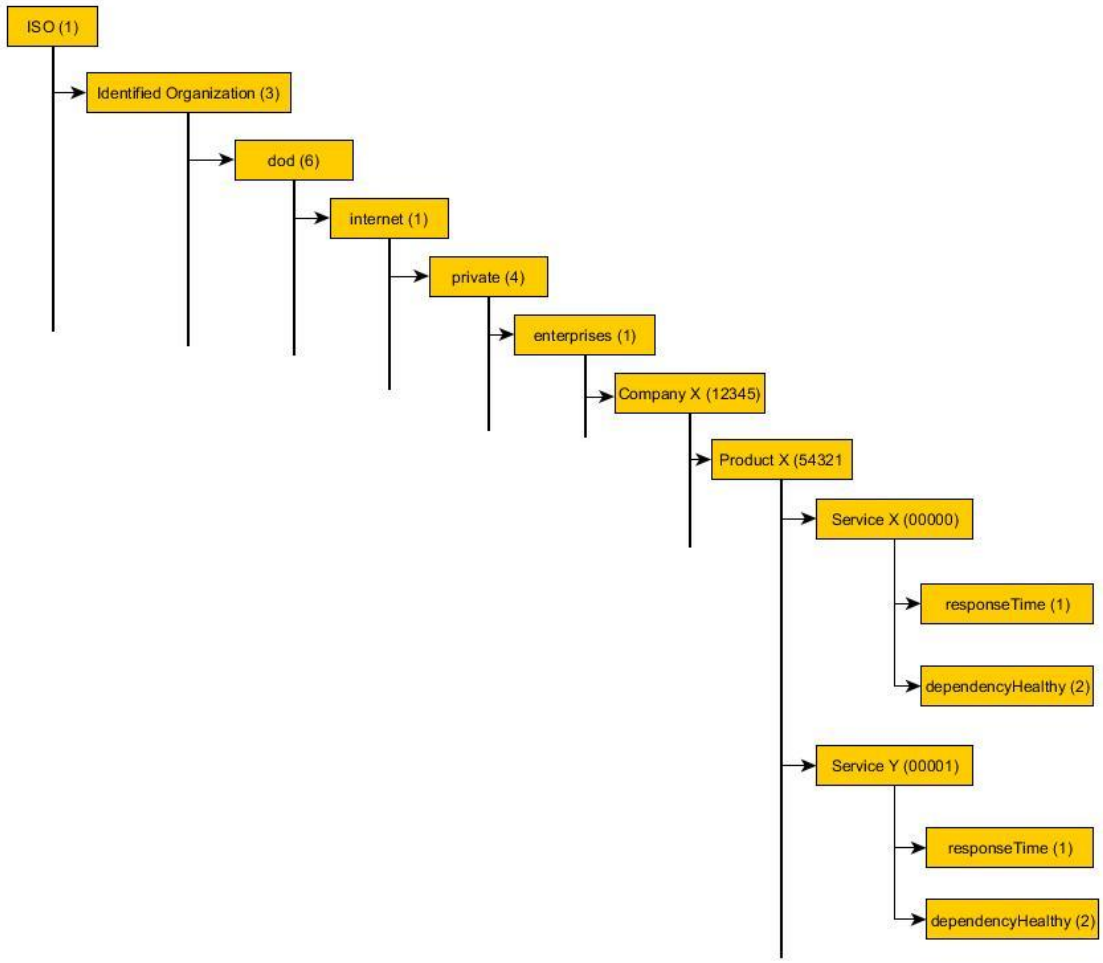
toteuttava instanssi on rekisteröity MBean-palvelimelle, siitä tulee aktiivinen ohjelmistoagentti, johon JMX-etäasiakassovellukset voivat luoda yhteyden ja tehdä interaktioita sen kanssa.

Kolmanteen kerrokseen viitataan sekä hallintakerroksena (management layer) että hajautettujen palvelujen kerroksena (distributed services level). Hallintakerros ei itse varsinaisesti toteuta manageria, vaan tarjoaa työkalut, joiden avulla voidaan luoda JMX-etäasiakassovelluksia ja kerätä valvottavasta sovelluksesta dataa JMX-rajapintojen kautta. Tähän kuuluvat niin sanotut konnektorit (connectors), joilla voidaan luoda yhteys MBean-palvelimelle, ja adapterit, joiden avulla JMX-hallinta onnistuu myös muilla kuin Java-teknologioilla toteutetuilla sovelluksilla: JMX-dataa voidaan näyttää esimerkiksi HTML-adapterilla Internet-selaimessa.

### 3.1.2. SNMP

Simple Network Management Protocol (myöhemmin SNMP) on erityisesti tietoverkkojen ja laitteiden hallintaan (device and network management) kehitetty hallintateknologia, joka perustuu manageri-agentti-arkkitehtuuriin. Kufel [2016] tosin mainitsee SNMP:n agentittomana valvontaratkaisuna, mutta se liittyy siihen, että SNMP-tuki on olemassa useimmissa SNMP:llä valvottavissa laitteissa. Sovellushallinnassa sitä ei ole kuitenkaan kovinkaan laajalti käytetty. Kreger ja muut [2002] mainitsevat syiksi tähän erityisesti sen, että SNMP:n turvallisuus ei ole riittävä konfiguraatiopäivityksiin ja arkaluontoisen informaation välittämiseen. Edelleen heidän mukaansa SNMP:tä käytetään lähinnä vain luku-tyyppisen ei-arkaluontoisen tiedon esittämiseen, koska se ei tue kovin hyvin hallittujen resurssien päivitysoperaatioita. Valvonnassa tämä ei kuitenkaan ole välttämättä merkitsevää, koska valvonta on nimenomaan lukuoperaatioita.

SNMP:llä hallittu tietoverkko koostuu niin sanotuista hallituista laitteista (managed device), agenteista, jotka toimivat hallittujen laitteiden sisällä, sekä hallinta-asemasta (Network Management Station, NMS), joka koostuu useista eri osista. Hallitut laitteet ovat SNMP:n tapauksessa usein fyysisiä laitteita, mutta sovellusvalvonnassa voidaan myös SNMP-agentin toteuttava sovellus mieltää tällaiseksi “hallituksi laitteeksi”.

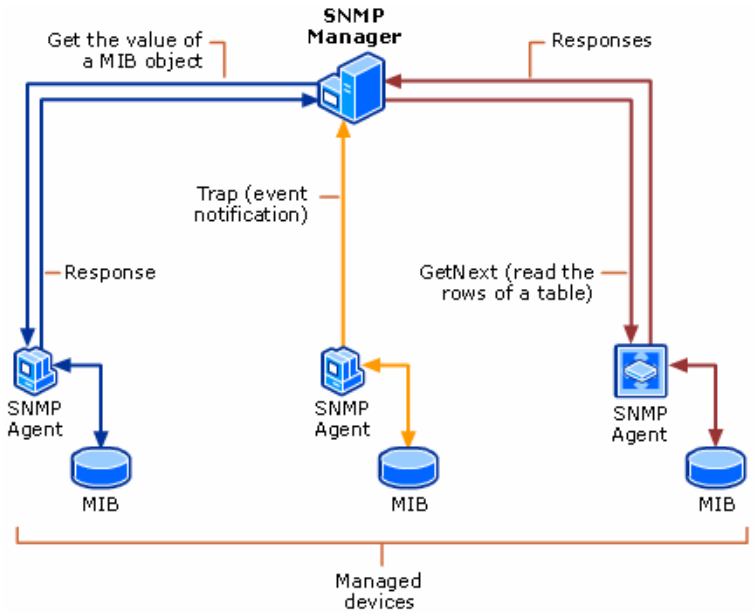



*Kuva 6: esimerkki MIB-rakenteesta*

"Hallittavan laitteen" kuvaus tapahtuu MIB:llä (Management Information Base), joka on SNMP-oliot kuvaava "tietokanta". Kuvassa 6 on annettu graafisessa muodossa esimerkki MIB-rakenteesta. MIB-puun rakenteen määrittelee SMI (Structure of Management Information). MIB-hierarkia muodostaa puun juuresta lehtiä kohti kuljettaessa OID:n (Object Identifier), johon jokainen hierarkiataso lisää aina yksilöivän numeron. Palvelun "Service X" SNMP-attribuutin "dependencyHealthy" avaimena toimiva OID-tunniste olisi kuvan esimerkin mukaan *1.3.6.1.4.1.12345.54321.00000.2.* Muuttujan arvon kertova SNMP-viesti voisi olla esimerkiksi muotoa *1.3.6.1.4.1.12345.54321.00000.2:1,* jossa kaksoispisteen jälkeinen ykkönen on totuusarvon "tosi" arvo numeerisessa muodossa (epätosi olisi 0).

SNMP-agentti on ohjelmistokomponentti, joka toimii hallittavan laitteen tai sovelluksen sisällä. Se kykenee vastaanottamaan kutsuja managerilta ja lähettämään myös itse sanomia valvomansa kokonaisuuden tilaan nähden. SNMP-datatyyppejä (PDU, protocol data unit) on määritelty seitsemän erilaista: *GetRequest*, *SetRequest*, *GetNextRequest*, *GetBulkRequest*, *Response*, *Trap* ja

*Inform*. Näistä neljä ensimmäistä on managerin agentille lähettämiä sanomia ja kolme viimeistä puolestaan agentin managerille lähettämiä sanomia. SNMP-viestien kulkua managerin ja agentin välillä on havainnollistettu kuvassa 7.




*Kuva 7: SNMP-viestien kulku [Microsoft, 2017]*

GetRequest-tyyppiä käytetään, kun halutaan kysyä SNMP-agentilta jonkin MIB-muuttujan arvo. GetNextRequest-pyyntöä käytetään, kun halutaan saada puuhierarkiassa parametrina annettavasta muuttujasta seuraavan alemman tason muuttujan arvo. Sen erikoistapaus on GetBulkRequest, jolla yleensä pyydetään SNMP-agentilta suuria määriä dataa, sillä sille voidaan määritellä useita OID-muuttujia, joiden arvoa kysytään. Esimerkiksi kuvan 6 tapauksessa pyyntö *GetNextRequest*(*1.3.6.1.4.1.12345.54321.00000*) palauttaisi Service X:n muuttujien ResponseTime ja DependencyHealthy arvot. Tätä käytetään tavallisesti, kun halutaan MIB-hierarkian lapsimuuttujat. SetRequest-tyyppi asettaa jonkin tietyn arvon MIB-muuttujalle.

GetResponse-operaatio on SNMP-agentin tekemä, ja agentti suorittaa sen vastauksena saamaansa pyyntöön, joita edellä mainitut operaatiot ovat. TRAP ja INFORM ovat puolestaan agentin itsenäisesti lähettämiä sanomia. TRAP-viestillä indikoidaan SNMP-managerille, että jotain kriittistä on tapahtunut hallittavassa laitteessa. TRAP-viestin heikkous on kuitenkin se, että manageri ei vastaa agentille millään tavalla että on vastaanottanut viestin. Tämän vuoksi SNMPV2-versioon kehitettiin myös INFORM-viestityyppi, joka on muuten samankaltainen kuin TRAP, mutta manageri lähettää myös kuittauksen agentille takaisin, että on vastaanottanut viestin.

TRAP- ja INFORM-tyyppiset muuttujat ovat erityisesti järjestelmänvalvontakäytössä käytettyjä viestejä, koska ne antavat mahdollisuuden välittää hälytykset nopeasti valvontajärjestelmälle, eikä tarvitse odottaa, että valvontasovellus kysyy tätä tietoa periodisesti. Tässä on kyseessä aiemmin mainittu ns. paikallistamisperustainen valvonta.

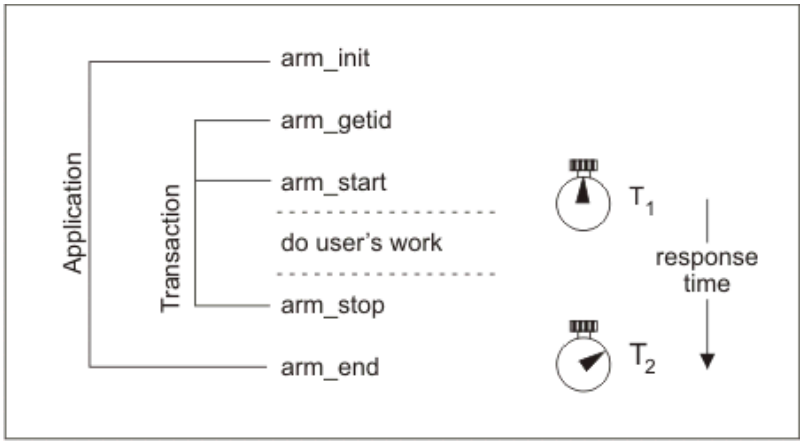
### 3.1.3. Muita sovellusvalvontateknologioita

Common Information Model (CIM) on Distributed Management Task Forcen (DMTF) määrittelemä yleinen, standardisoitu tietomalli laitteiden, tietoverkkojen ja sovellusten hallintaan. DMTF on teollisuusstandardiorganisaatio, joka pyrkii yksinkertaistamaan järjestelmänhallintaa luomalla avoimia hallintastandardeja yhdessä suurten teknologiayritysten kanssa. Oikeastaan CIM on standardina enemmänkin toimintafilosofia kuin toteutettu teknologia, koska se lähinnä määrittelee yleiset standardit, joiden mukaan laitteiden ja ohjelmistojen hallinta tulisi määritellä, mutta sen pohjalta on tehty myös toteutuksia.

CIM:in liittyy keskeisesti myös Web Based Enterprise Management (WBEM) [Kreger et al., 2002], joka on CIM:n pohjalta luotu malli hajautettujen tietojärjestelmien hallintaan. WBEM:n toteuttavia teknologioita käytetään pääasiassa web-järjestelmien hallintaan, mutta se ei ole kuitenkaan riippuvainen mistään tietystä käyttöliittymätyypistä [Wikipedia, 2017] Erityisesti WBEM tarjoaa korkeamman tason mallina mahdollisuuden toteuttaa standardin mukaisia ratkaisuja useimmilla käytettävissä olevilla teknologioilla.

Application Information and Control (AIC) on The Open Groupin Enterprise Management -ohjelman kehittämä C-kielellä toteutettu ja Java-sovellusohjelmointirajapinnan (API) sisältävä teknologia sovelluksen mittarien ja poikkeamien valvontaan. The Open Group on DMTF:n kaltainen konsortio, joka kehittää myös IT-standardeja. Lukuun ottamatta Kregerin ja muiden [2002] esittelyä, teknologiasta ei löytynyt kovin ajantasaista tietoa, joten se ei välttämättä ole ollut kovin pitkäikäinen.

Application Response Measurement (ARM) on myös The Open Groupin kehittämä avoin standardi erityisesti suorituskykyyn liittyvien ongelmakohtien valvontaan ja diagnosointiin. ARM:n toteutus on kirjoitettu C-kielellä, mutta se tarjoaa lisäksi sovellusohjelmointirajapinnan (API) myös Javalle ja C++-kielelle. [Kreger et al., 2002]. ARM API:a käytetään pääasiassa vasteaikojen tallentamiseen, mutta sitä voidaan käyttää myös sovelluksen saatavuuden mittaamiseen.




*Kuva 8: ARM API:n kutsut [IBM, 2017]*

Kuvassa 8 on esitetty ARM API:n toiminta ja metodit, joita kutsutaan. ARM API:lla voidaan mitata sovelluksen vasteaikoja kutsumalla sopivia API:n metodeita transaktioiden eli sovelluksen tapahtumasarjojen alku- ja loppuvaiheessa, jolloin ARM käsittelee saamansa tiedon, ja sovellusvalvonta puolestaan tapahtuu keräämällä ARM-agentilta tietoa.

### 3.2. Järjestelmänvalvonta

Järjestelmänvalvontaan käytetään tyypillisesti valvontajärjestelmiä tai hallintajärjestelmiä, jotka ovat useimmiten kohdejärjestelmästä erillisiä tietojärjestelmiä tai ohjelmistoja, joilla valvottavasta järjestelmästä voidaan hankkia valvontadataa, näyttää kerättyjä valvontamittarien arvoja ja muodostaa kuvaajia niistä. Valvontajärjestelmien tehtävä on myös muodostaa poikkeavista valvontamittareiden arvoista hälytyksiä, jotka indikoidaan järjestelmänvalvojille. Hallintajärjestelmä-termi liittyy siihen, että järjestelmällä voi usein tehdä myös muita järjestelmänhallintaan liittyviä aktiviteetteja kuin valvontaa. Valvontajärjestelmää tai sen käyttöliittymää joka näyttää koostetun kokonaiskuvan valvottavasta järjestelmästä, voidaan kutsua myös *kojetauluksi* (dashboard) [Ewaschuck, 2016].

Valvontajärjestelmä voi myös olla osa kohdejärjestelmän järjestelmäkokonaisuutta, mutta kuitenkin toteutettu varsinaisesta toimintalogiikasta erilliseksi osaksi. Useimmiten valvontajärjestelmät ovat kuitenkin kolmannen osapuolen järjestelmiä, koska valvontajärjestelmät eivät ole useimpien yritysten erikoisosaamista ja varsinkin pienempien sovellusten ollessa kyseessä valvontajärjestelmä voi hyvin olla suurempi ja monimutkaisempi kuin valvottava kohdejärjestelmä.

Valvontajärjestelmiä on yleisesti käytössä sekä kaupallisina että avoimen lähdekoodin ratkaisuina. Kaupalliset järjestelmät sisältävät usein enemmän ominaisuuksia ja niissä on parempi käytettävyys ja konfiguroitavuus kuin avoimen lähdekoodin järjestelmissä, mutta toisaalta avoimen lähdekoodin järjestelmillä on usein myös aktiivinen kehittäjäyhteisö ja tukiverkosto, varsinkin silloin, kun on kyse paljon käytetystä järjestelmästä. Avoimen lähdekoodin järjestelmien laajentaminen on myös yleensä vapaammin toteutettavissa kuin kaupallisissa järjestelmissä, koska muutkin kuin järjestelmän toimittaja voivat kehittää järjestelmälle lisäosia.

Jakelulisenssin (avoin lähdekoodi tai kaupallinen) lisäksi muita ominaispiirteitä ja luokittelutapoja valvontajärjestelmien luokitteluun ovat esimerkiksi järjestelmän kohdemarkkinat tai käyttökohteet, järjestelmän tukemat valvonta-arkkitehtuurit ja -teknologiat, tai järjestelmän tukemat hälytysmuodot [Kufel, 2016]. Käyttökohteita eli valvottavia järjestelmiä voivat olla esimerkiksi pienet, keskisuuret, suuret tai yritystason järjestelmät. Valvonta-arkkitehtuurit ja -teknologiat kertovat, tukeeko järjestelmä esimerkiksi vain agenttipohjaisia valvontateknologioita, vai myös agentittomia, tai vain agentittomia valvonta-ratkaisuja. Hälytysmuodot viittaavat järjestelmän tukemiin viestintäkeinoihin, joilla hälytykset voidaan välittää järjestelmänvalvojan tietoon, tällaisia tapoja voivat olla esimerkiksi tekstiviesti (SMS) tai sähköposti.

### 3.2.1. Avoimen lähdekoodin valvontajärjestelmät

Avoimen lähdekoodin valvontajärjestelmiä on esitelty esimerkiksi Kufelin [2016] artikkelissa sekä Hernantesin ja muiden [2015] artikkelissa. Kummassakin tutkimuksessa on taulukoitu näiden järjestelmien keskeisiä ominaisuuksia, kuten tuettuja valvonta-arkkitehtuureja, hälytystapoja, kohdemarkkinoita ja ominaispiirteitä. Tässä alakohdassa esitellään yleisellä tasolla muutamia näistä esitellyistä avoimen lähdekoodin järjestelmistä, lähinnä kuitenkin sellaisia järjestelmiä, joiden kohdemarkkinoina ja suunniteltuina käyttökohteina ovat erityisesti suuret ja yritystason järjestelmät.

Zabbix on avoimen lähdekoodin valvontaohjelmisto, joka on suunniteltu erityisesti yrityskäyttöön tietoverkkojen ja sovellusten hallintaan. Zabbix tukee sekä agentteihin perustuvia että agentittomia valvontaratkaisuja, ja siinä on tuki useille eri valvontateknologioille. Avoimen lähdekoodin järjestelmänä Zabbix on paljon käytetty järjestelmä niin yrityksissä kuin yksityisilläkin käyttäjillä. Sen heikkoutena pidetään vaikeaa konfiguroitavuutta, mutta muuten se tarjoaa kattavasti erilaisia valvontamekanismeja ja myös aktiivisen kehittäjäyhteisön ja tukiverkoston. [Kufel, 2016]

Nagios on myös yleisesti käytetty avoimen lähdekoodin järjestelmä sovellusvalvontaan. Se ei kuitenkin ole täysin puhtaasti avoimen lähdekoodin järjestelmä, vaan sen perustoiminnallisuudet on toteutettu ilmaiseksi saatavassa Nagios Coressa ja lisäksi siitä on olemassa kaupallisella lisenssillä jaettava versio Nagios XI. Nagiosilla on myös melko vaikea konfiguroitavuus, vaikka itse asennus on helppoa. Lisäksi avoimen lähdekoodin perusversiossa kojetaulun käyttöliittymä on huono, mutta maksullisessa XI-versiossa se on hyvin interaktiivinen ja kehittynyt. Zabbixin tavoin myös Nagiosilla on aktiivinen kehittäjäyhteisö ja kolmansien osapuolien kehittämiä lisäosia on saatavilla hyvin. [Kufel, 2016]

Muista avoimen lähdekoodin suuren mittakaavan valvontajärjestelmistä mainittakoon esimerkiksi Ganglia, joka toteutettiin akateemisena projektina alun perin Berkeleyn yliopistossa [Kufel, 2016]. Se on suunniteltu suurten hajautettujen järjestelmien valvontaan. Vaikka järjestelmä on alun perin syntynyt yliopiston tutkimustyönä, sitä käytetään myös yksityisten yritysten ja julkisen sektorin toimijoiden käytössä, kuten esimerkiksi Internet-sanakirja Wikipedian järjestelmänvalvontaan [Kufel, 2016].

### 3.2.2. Kaupalliset valvontajärjestelmät

Kaupallisia valvontajärjestelmiä on myös esitelty Kufelin [2016] ja Hernantesin ja muiden [2015] artikkeleissa. Tässä alakohdassa esittelen yleisellä tasolla muutamia tunnetuimpia kaupallisia valvontajärjestelmiä, kuitenkin edellisen kappaleen tavoin lähinnä sellaisia, joiden kohdemarkkinoina ja suunniteltuina käyttökohteina ovat erityisesti suuret ja yritystason järjestelmät.

IBM SmartCloud Monitoring yhdistää kaksi teknologiayritys IBM:n valvontatyökalua, nämä ovat Tivoli Monitoring ja Tivoli Monitoring for Virtual Environment [Kufel, 2016]. Sen asennus on helppoa, mutta konfigurointi ja ylläpito vaatii enemmän IT-asiantuntemusta. Sen etuihin kuuluvat kehittyneet raportointi- ja analysointityökalut erityisesti ennustavan analyysin osalta.

HP Operations Manager on teknologiayritys HP:n kehittämä valvontajärjestelmä, jonka vahvuutena on erityisesti hyvä käyttöliittymä. Sen avulla voidaan analysoida valvontadataa ennakoivasti, automatisoida hälytyksiä ja valvoa proaktiivisesti: järjestelmä pystyy ehdottamaan käyttäjälle suositeltuja toimenpiteitä häiriötilanteisiin ja määrittelemään myös automatisoituja korjaustoimenpiteitä. [Kufel, 2016]

AppDynamics on tietovirtojen käyttöön perustuva valvontajärjestelmä, jonka avulla valvonta voidaan toteuttaa joustavasti asiakasorganisaation ympäristössä ns. ohjelmistojen palvelumallin (Software as a Service) avulla. Se käyttää keräämänsä tiedon analysointiin koneoppimisalgoritmeja. Suoritus-

kyvyn valvonnan lisäksi se kykenee valvomaan web-asiakassovelluksien vasteaikoja loppukäyttäjän näkökulmasta. [Kufel, 2016]

# 4. Valvonta osana kriittistä tietojärjestelmää

Sovellusvalvonta ja järjestelmänvalvonta ovat suurten tietojärjestelmien keskeisiä osia. Niiden merkitys korostuu tietojärjestelmän ollessa kriittinen, koska järjestelmän alasajo korjauksia varten ei ole välttämättä mahdollinen. Tämän takia valvonnassa korostuu myös sen ennaltaehkäisevä vaikutus: kriittiset ongelmat on, jos mahdollista, löydettävä ennen kuin niistä seuraa kriittisiä virheitä, jotta palvelut pysyvät saatavana mahdollisimman suuren osan toiminta-ajastaan.

## 4.1. Kriittisen tietojärjestelmän määrittely

Kriittiselle tietojärjestelmälle on olemassa useita erilaisia määrittelyjä, koska kriittisyys voi liittyä moneen eri järjestelmän osatekijään. Tehtäväkriittisellä järjestelmällä (*mission-critical system*) tarkoitetaan järjestelmää, jolla sen toimintalogiikkaan liittyvät tehtävät ovat kriittisiä liiketoiminnan tai organisaation tavoitteille [Techopedia, 2017b]. Tehtäväkriittisen järjestelmän järjestelmävirhe voi aiheuttaa liiketoiminnalle tai organisaatiolle huomattavia taloudellisia tai inhimillisiä menetyksiä. Turvallisuuskriittinen järjestelmä (*safety-critical system*) taas voi olla joko kriittistä infrastruktuuria käyttävä tai ohjaava järjestelmä, jonka vääränlainen toiminta saattaa vaarantaa tämän infrastruktuurin toiminnan, tai tietoturvakriittinen järjestelmä (*security-critical system*), jolloin järjestelmä saattaa käsitellä arkaluonteista tietoa, jonka katoaminen tai päätyminen vääriin käsiin voi olla kohtalokasta.

Kriittisen tietojärjestelmän eri tasot eivät ole toisiaan poissulkevia, vaan sama järjestelmä voi olla usealla eri tavalla kriittinen. Esimerkiksi hätäkeskustietojärjestelmä on tehtäväkriittinen, koska järjestelmävirhe voi estää sen toimintalogiikkaan liittyvien operaatioiden suorittamista ja siten estää järjestelmän käytön hätäkeskustoimintaan. Toisaalta hätäkeskustietojärjestelmä voidaan nähdä myös turvallisuuskriittisenä, koska sen toimimattomuus voi vaarantaa ihmisten turvallisuuden, ja myös tietoturvakriittisenä, koska järjestelmä käsittelee myös esim. henkilö- ja viranomaistietoja, jotka eivät saa joutua ulkopuolisten käsiin.

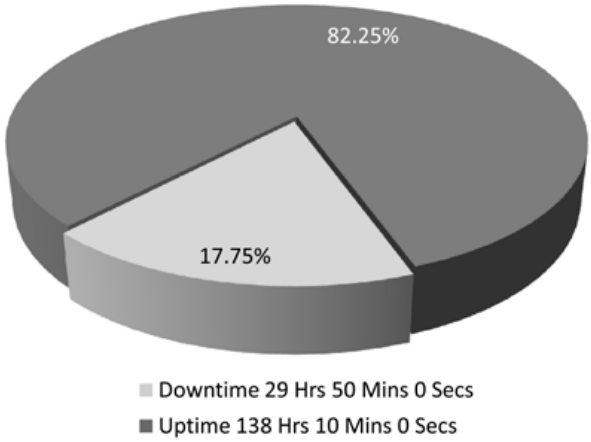
### 4.2. Tietojärjestelmän saatavuus ja vikasietoisuus

Kriittisistä tietojärjestelmistä puhutaan usein myös korkean saatavuuden järjestelminä (*high availability systems*). Saatavuus on järjestelmän ei-toiminnallinen vaatimus, joka voidaan määritellä järjestelmävirheiden välisen ajan ja järjestelmän korjaamiseen käytettävän ajan välisenä funktiona [Chang and Minkin, 2008]. Saatavuus lasketaan kaikista järjestelmään liittyvistä saatavuustekijöistä, ja sen pohjana käytetään historiallisen valvonnan tuottamaa dataa [Limoncelli et al., 2007]. Korkean saatavuuden järjestelmissä saatavuus voi olla joskus vaikeasti määriteltävissä, koska saatavuus voi tarkoittaa eri käyttäjille eri asioita. Näitä saatavuustekijöitä voivat olla kuitenkin esimerkiksi käytössä olevien järjestelmäresurssien määrä suhteessa kaikkiin järjestelmäresursseihin.

Toimiva järjestelmänvalvonta on yksi osatekijä, jota käytetään korkean saatavuuden saavuttamiseksi. Muita tällaisia tekijöitä ovat esimerkiksi järjestelmän loogisten ja fyysisten komponenttien, kuten palvelusovellusten, palvelimien, virtalähteiden tai tietokantojen kahdentaminen tai monistaminen, erilaisten varamekanismien määrittely tai *yksittäisten virhepisteiden* (SPOF, single point of failure) välttäminen järjestelmän arkkitehtuurin suunnittelussa.

Järjestelmänvalvonnan keskeinen tehtävä korkean saatavuuden järjestelmissä on osoittaa, että järjestelmä täyttää jatkuvasti sille määritellyt suorituskyky- ja saatavuusvaatimukset. Nämä vaatimukset on määritelty järjestelmän *palvelutasosopimuksessa* eli myöhemmin *SLA:s*sa (Service Level Agreement). Järjestelmän palvelutason valvonta tai yleisemminkin laskeminen nähdään yleensä ongelmallisena ja vaikeana toimenpiteenä, koska se vaatii suuren datamäärän keräystä ja analysointia [Edmiston, 2007].

SLA määritellään teollisuudessa tyypillisesti ns. 9-luvuilla tai *yhdeksiköillä* (nines), joilla indikoidaan, kuinka suuren osan prosentuaalisesti toiminta-ajastaan järjestelmä on saatavilla operatiiviseen käyttöön [Chang and Minkin, 2008]. Kevyemmän saatavuuden järjestelmän SLA voi olla esimerkiksi 95 %, jolloin järjestelmän on oltava toiminnassa 347 vuorokautta vuodessa (0,95 * 365 = 346,75). Korkean saatavuuden järjestelmän SLA voi puolestaan olla esimerkiksi 99,99 %, jolloin se saa olla vain alle tunnin vuodessa ei-saatavana. (Vuodessa on 526 000 minuuttia, joten 0,9999 * 526 000 = 525 947.44. Kun vähennetään saatavuusaika vuoden minuuttien määrästä, saadaan 526 000 – 525 547 = 53). SLA:n mukaista saatavuutta on havainnollistettu kuvassa 9.




*Kuva 9: Esimerkki palvelun SLA-saatavuudesta [Kufel, 2016]*

Erittäin korkean saatavuuden järjestelmistä käytetään myös nimitystä *vikasietoinen järjestelmä* (fault-tolerant system), joka tarkoittaa, että järjestelmän toiminta ei keskeydy mahdolliseen vikatilanteeseen ja järjestelmä pystyy useimmiten myös toipumaan vikatilanteesta automaattisesti. Valvonta on kuitenkin tärkeässä osassa myös vikasietoisessa järjestelmässä, jotta järjestelmän toipuminen vikatilanteesta ja toiminta vikatilanteen jälkeen voidaan varmistaa.

### 4.3. Kriittisen tietojärjestelmän valvonta-aktiviteetit

Periodinen valvonta on tyypillinen valvonta-aktiviteetti myös kriittisissä tietojärjestelmissä. Koska saatavuusvaatimus on yleensä korkea ja vikatilanteet on huomattava nopeasti, suositus periodiksi valvontadatan keräämiseksi on tyypillisesti esimerkiksi yksi minuutti, kuten Kufel [2016] on esitellyt tutkimuksessaan. Joissakin tapauksissa tämä periodi voi olla pidempikin, jos mittarit ovat luonteeltaan sellaisia, että niiden arvot eivät voi kasvaa nopeasti. Myös paikantamisperustainen valvonta, jossa valvottavaan järjestelmään integroitu valvonta-agentti tai jokin muu komponentti lähettää valvontajärjestelmälle tiedon muuttuneesta järjestelmän tilasta, on kriittisissä tietojärjestelmissä usein käytössä kaikkein kriittisimpien ominaisuuksien valvontaan.

Kriittisen tietojärjestelmän valvonta tapahtuu valvomosta, jossa järjestelmänvalvojat käyttävät järjestelmänhallintasovellusta ja näkevät järjestelmäkomponenttien toiminnallisen tilan valvontasovelluksen kojetaulusta. Valvontajärjestelmän tehtävä on analysoida kerätty valvontadata ja tehdä tulkinta, milloin kyseessä on kriittinen tilanne. Tämä tapahtuu säännöillä, joita annetaan valvontamittarien palauttamille arvoille ja joilla määritellään arvot kullekin hälytystilanteelle. Näihin sääntöihin sisältyy yleensä myös sääntö siitä, milloin

mittarin arvot ovat normaaleja, jotta mahdollinen hälytys otetaan pois käytöstä, jos tilanne korjaantuu. Hälytyksille on tavallisesti määritelty toimintaohjeet, joilla häiriön saa käsiteltyä.

### 4.4. Kriittisen tietojärjestelmän valvonnan suunnittelun periaatteita

Kriittisessä tietojärjestelmässä myös valvonta on järjestelmän kriittinen osa, joten se on suunniteltava ja toteutettava yhtä huolellisesti kuin itse valvottava järjestelmä. Jos valvonta ei ole saatavilla tai toimii epäluotettavasti, voi olla, että todelliset häiriötilanteet jäävät huomiotta, koska vikaa ei havaita tai ajatellaan sen olevan vain valvonnan vika.

#### 4.4.1. Yksinkertaisuus

Kriittisen järjestelmän palveluiden tarjoaman valvontadatan tulisi olla mahdollisimman yksiselitteistä ja yksinkertaista. Tämä tarkoittaa sitä, että datan tuottaminen ja analysointi pidetään vahvasti erillään: sovellukseen toteutetut valvonta-agentit tarjoavat valvontajärjestelmälle vain dataa, eivätkä itse päättele, onko sovelluksessa häiriö vai ei. Valvontasovellukselle on konfiguroitu muuttujien raja-arvot, jolloin valvontasovellus päättää, muodostetaanko datan perusteella hälytys.

Yksinkertaisten ja yksiselitteisten valvontamittarien toteuttaminen ja määritteleminen voi kuitenkin olla ristiriitaista, jos valvottavien mittarien määrää halutaan rajoittaa. Koska mittarien määrä voi valvonnan aiheuttaman suorituskykyhaitan minimoimiseksi olla hyvinkin rajattu erittäin suorituskykykriittisissä järjestelmissä, voisivat jonkinlaisen aggregointilogiikan kautta tarjottavat mittarit antaa enemmän tietoa valvontajärjestelmälle kuin pelkkä mittarien määrän vähentäminen.

Toisaalta myös valvontajärjestelmän on tehtävä oikeelliset tulkinnat keräämästään valvontadatasta: kun hälytys aktivoidaan, hälytyksen on informoitava, mitä on oikeasti tapahtunut, eikä välittää informaatiota tapahtumasta, jota ei tiedetä tapahtuneeksi. [Limoncelli et al., 2007]. Edelleen Limoncellin ja muiden [2007] mukaan esimerkiksi virheviesti ”palvelin ei vastaa” on parempi kuin ”palvelin ei ole käynnissä”, jos hälytys on muodostettu alun perin informaatiosta, että palvelin ei vastaa. Tämä siksi, että jälkimmäinen vaihtoehto on enemmänkin tulkinta häiriötilanteesta.

#### 4.4.2. Johdonmukaiset tilat

Erityisesti totuusarvotyyppisillä muuttujilla ja varsinkin, kun käsitellään palvelun riippuvuuksien tilaa, muuttujien tilan pitää indikoida johdonmukaisesti,

onko valvottavan attribuutin arvo tosi vai epätosi. Jos esimerkiksi on kyse jonkin yhteyden valvonnasta, tilan on oltava tosi, ellei ole oikeasti tapahtunut virhettä. Tämä toimii kuitenkin myös toisinpäin: yhteyden menettämisen jälkeen tilan ei pitäisi muuttua takaisin todeksi, ellei olla varmistettu, että yhteys on palautunut.

Tilojen johdonmukaisuus on sidoksissa myös valvonnan luotettavuuteen, sillä jos valvonta näyttää järjestelmän tilan epäjohdonmukaisena, sitä aletaan helposti pitää epäluotettavana ja mahdolliset hälytykset ohitetaan. Tällöin on vaarana, että todellinen ongelmatilanne peittyy ja jätetään huomioimatta. Tätä käsittelevät myös Limoncelli ja muut [2007] edellisessä alakohdassa mainitussa esimerkissään, jossa virheviesti on ”palvelin ei ole käynnissä” eikä ”palvelin ei vastaa”. Heidän mukaansa ensimmäisessä tapauksessa järjestelmänvalvoja voisi tehdä tulkinnan, että hälytys on väärä, koska tässä tapauksessa palvelin on käynnissä mutta ei vastaa. Jos taas jälkimmäinen virheviesti olisi käytössä, järjestelmänvalvoja tietäisi, ettei kaikki ole kunnossa ja tarkistaisi palvelimen toiminnan.

### 4.4.3. Säännöllinen valvonta

Suurissa tietojärjestelmissä on usein luonteeltaan kriittisiä palveluita, joiden on oltava toiminnassa jatkuvasti, mutta on myös palveluita, jotka ovat vähemmän kriittisiä ja joita ei edes käytetä jatkuvasti. Näillä palveluilla, kuten myös kriittisillä palveluilla, voi olla yksittäisiä sisäisiä tai ulkoisia riippuvuuksia, joita ei käytetä jatkuvasti. Nämä riippuvuudet, kuten esimerkiksi liittymät kolmannen osapuolen web-palveluihin, tai yhteydet niihin, voivat kuitenkin vikaantua myös silloin, kun liittymää ei aktiivisesti käytetä.

Tämän takia on olemassa suuria riskejä, jos liittymän tai yhteyden tilaa päivitetään vain silloin, kun yhteyttä käytetään, sillä silloin häiriö havaitaan vasta, kun liittymää tarvitaan aktiivisesti. Parempi ratkaisu on esimerkiksi kysyä ajastetusti yhteyden tai riippuvuuden tilaa, jos vastinpään rajapinta toteuttaa jonkinlaisen metodin tilan kysymiseksi ja mahdollistaa täten periodisen valvonnan.

### 4.4.4. Standardinmukaiset valvontatoteutukset

Kriittisillä tietojärjestelmillä on usein pitkä elinkaari, minkä takia myös käytettyjen valvontateknologioiden olisi oltava sellaisia, joita tuetaan pitkään. Tällöin järjestelmän elinkaaren aikana ei välttämättä tarvitse toteuttaa valvontatoteutuksia uudestaan toisella teknologialla, mikä saattaisi aiheuttaa merkittävän regressioriskin järjestelmään. Limoncelli ja muut [2007] mainitsevat, että

sekä historiallisen että reaaliaikaisen valvonnan osalta tuki standarditeknologioille on tärkeää.

Standardinmukaiset valvontatoteutukset tyypillisesti myös mahdollistavat sen, että hallittavan järjestelmän tarjoama valvontarajapinta on mahdollisimman yhteensopiva erilaisten kolmannen osapuolen valvontajärjestelmien kanssa. Yleisesti käytetyt standarditeknologiat ovat useimpien tunnettujen valvontajärjestelmien tukemia. Tämän merkitys korostuu erityisesti silloin, kun tietojärjestelmä on kaupallinen tuote tai pohjautuu kaupalliseen ohjelmistoon. Tämä selittyy sillä, että palveluntuotantoa ja järjestelmänvalvontaa saattaa eri asiakasjakeluissa hoitaa eri toimijat, jotka käyttävät eri valvontajärjestelmiä.

### 4.4.5. Valvontajärjestelmän ja valvottavan järjestelmän yhteensopivuus

Vaikka valvottava järjestelmä ei saa olla liian riippuvainen yksittäisestä valvontajärjestelmästä, on valvontajärjestelmien asettamat rajoitteet ja vaatimukset valvottavan järjestelmän valvontarajapinnoille kuitenkin tunnistettava ja otettava huomioon toteutuksessa. Valvontajärjestelmät tukevat kyllä tyypillisesti tunnetuimpia sovellusvalvontateknologioita, mutta tuki ja yhteensopivuus voi olla kuitenkin rajoitetumpi kuin mitä käytetty teknologia sallii.

Hyvä esimerkki tästä on edellisessä luvussa esitelty JMX. Kyseinen teknologia tarjoaa laajat mahdollisuudet toteuttaa valvontarajapintoja ja -toteutuksia ja sallii esimerkiksi valvontadatan tarjoamisen melkein minkä tahansa Java-olion muodossa, tai SNMP TRAP -tyyppisiä viestejä vastaavat automaattinotifikaatiot valvontajärjestelmälle. Nämä erityyppiset JMX-attribuutit ja notifikaatiot ovat myös yhteensopivia Javan omien JMX-asiakassovellusten, kuten Java Mission Control -sovelluksen kanssa, mutta esimerkiksi valvontajärjestelmä Zabbix ei tue JMX-notifikaatioita tai monimutkaisempia attribuuttityyppejä.

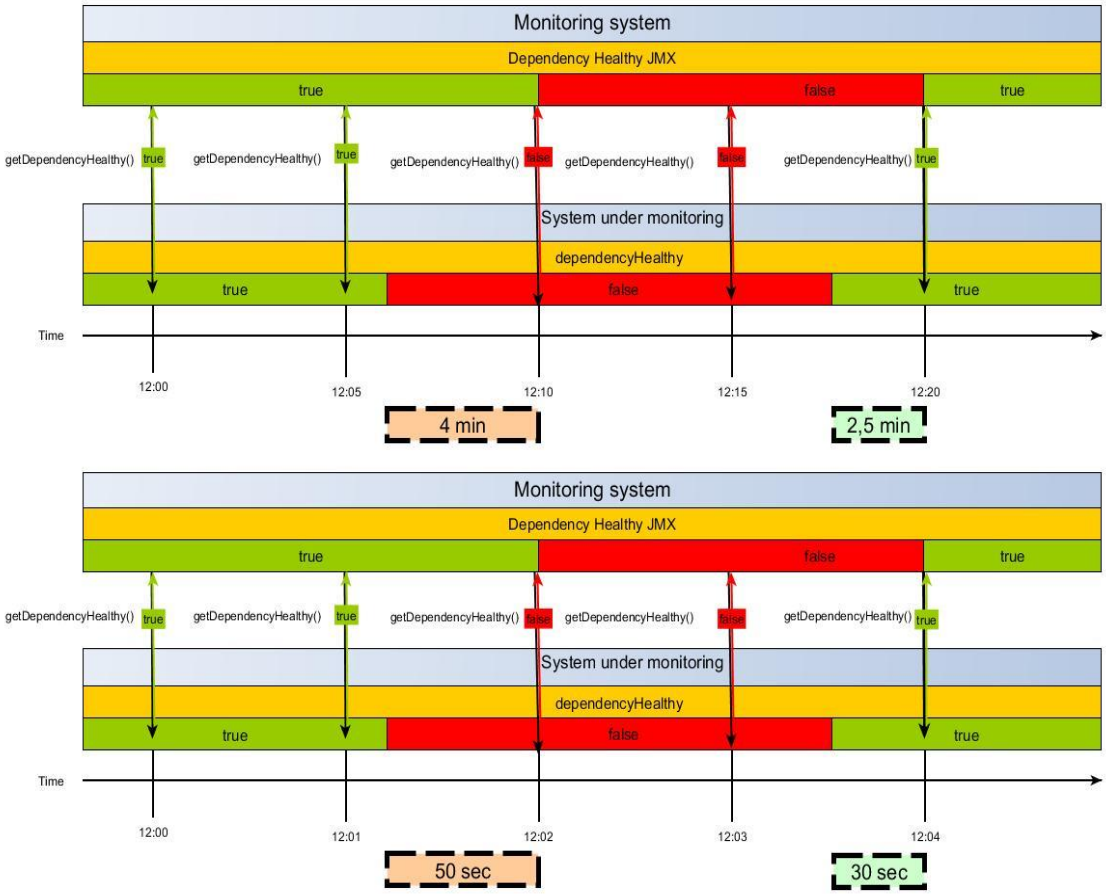
## 4.5. Valvontaan liittyviä keskeisiä ongelmia kriittisissä tietojärjestelmissä

Valvonta on välttämätön osa kriittistä tietojärjestelmää, mutta valvonta ei automaattisesti paranna järjestelmän laatua, eikä se ole resurssien käytön kannalta ”ilmaista”. Kun resurssien riittävyys operatiiviseen käyttöön vaatii huolellista suunnittelua ja optimointia, voidaan myös valvonta-aktiviteetit ja kehitysresurssien käyttö niiden toteutukseen nähdä ylimääräisenä järjestelmän osana, josta voi tarvittaessa joustaa myös operatiivisen toiminnan kustannuksella. Näin ei pitäisi ajatella, koska vaikka valvonta on tärkeä osa valvottavaa järjestelmää, niin valvontaan liittyvät ongelmat on tunnistettava ja otettava huomioon järjestelmää suunniteltaessa, jotta valvonta kykenee täyttämään tehtävänsä tehokkaasti ja parantamaan järjestelmän yleistä laatua.

### 4.5.1. Valvontajärjestelmän aiheuttama kuormitus valvottavalle järjestelmälle

Sovelluksen valvonnan keskeisenä ongelmana on sen mahdollisesti aiheuttama ylimääräinen suorituskykykuormitus valvottavalle järjestelmälle. Jos valvontatoteutus on osa sovelluksen ohjelmakoodia, aiheuttaa sen suorittaminen samanaikaisesti kohdejärjestelmän toiminnallisen koodin kanssa ylimääräistä laskentaa, joka voi vaikuttaa sovelluksen vasteaikoihin tai muistin ja laskentatehon käyttöön. Tämä voi pahimmillaan myös vaikuttaa muun ohjelman toimintaan, varsinkin jos kyseessä on rinnakkainen järjestelmä. Limoncelli ja muut [2007] mainitsevat valvonnan lisäävän kuormaa esimerkiksi laskentatehon, muistin ja verkkoliikenteen käytössä.

Yksi keskeinen valvonnan toteutukselle asetettava vaatimus on, että järjestelmän pitää täyttää sille asetetut suorituskykyvaatimukset, vaikka valvonta on käytössä. Tässä joudutaan usein etsimään jonkinlaista kompromissia suorituskyvyn ja luotettavuuden välillä, koska tiheästi tapahtuva valvontadatan keruu aiheuttaa väistämättä kuormitusta järjestelmälle, mutta samalla kun viive uuden datan saamiseksi kasvaa, kasvaa myös todennäköisyys, että vikatilanne huomataan vasta monen minuutin kuluttua, mikä voi kriittisimmissä järjestelmissä olla liian pitkä aika.




*Kuva 10: Valvontaperiodin vaikutus häiriön havaitsemiseen*

Kuva 10 havainnollistaa valvontadatan keräykseen käytetyn intervallin vaikutusta häiriötilanteen havaitsemiseen. Ylemmässä esimerkissä dataa kerätään viiden minuutin välein, jolloin klo 12:06 tapahtuva vikatilanne huomataan vasta neljän minuutin kuluttua klo 12:10. Alemmassa esimerkissä valvontadataa kerätään minuutin välein, jolloin häiriö havaitaan jo 50 sekunnin kuluttua vikatilanteesta. Intervalli vaikuttaa myös hälytyksen poistumiseen: viiden minuutin intervallilla järjestelmä on esimerkissä tämän mittarin osalta toimintakuntoinen 2,5 minuuttia ennen kuin tieto päivittyy valvomoon, kun taas jälkimmäisessä esimerkin tilanteessa hälytys poistuu jo 30 sekunnin kuluttua järjestelmän palautumisesta.

Myös esimerkiksi JMX-valvonnalla on havaittu olevan vaikutuksia valvottavien sovellusten suorituskykyyn. Vaikka JMX-yhteys etähallintasovelluksesta kohdejärjestelmään ei itsessään lisää sovelluksen resurssien käyttöä tai verkkoliikenteen kuormitusta merkittävästi, valvontadatan keräys kohdejärjestelmästä tuottaa kuitenkin kuormitusta erityisesti silloin, kun sitä tehdään tiheästi. Toisaalta kuormituksen määrä riippuu myös valvottavan sovelluksen käyttämien mittarien määrästä, koska mitä enemmän sovelluksessa on aktiivisia mittareita,

joiden arvoja haetaan, sitä korkeampi on kuormitus, mutta yleensä datan kerääminen itsessään aiheuttaa jo pientä kuormitusta. Valvonnan aiheuttamaa kuormitusta ja sen vaikutusta valvottavan sovelluksen suorituskykyyn ovat tutkineet esimerkiksi Lahmadi ja muut [2009], jotka toteuttivat tapaustutkimuksen JMX-pohjaisella valvonnalla ja määrittelivät mittarin, jolla valvonnan kuormitusta kohdejärjestelmälle voisi mitata.

Toisaalta hajautetuissa tietojärjestelmissä, joissa valvontajärjestelmä on eri palvelimella ja dataa kerätään kohdejärjestelmästä tietoverkon kautta, myös tietoverkon kapasiteetti voi olla ratkaisevassa asemassa Joissakin tapauksissa alhaisella kaistanleveydellä, erityisesti pitkän etäisyyden tiedonsiirto, voi ruuhkautua valvontadatasta ja siten jumittaa valvottavan järjestelmän [Limoncelli et al., 2007]. Vaikka valvontadataa voisikin kerätä kohdesovellukselta ilman suurempaa kuormitusta, voi verkko olla kuormitettuna, jos järjestelmä on suuri ja dataa välitetään tietoverkon kautta paljon. Tämän takia olisi myös hyvä, jos sovelluksen tietoliikenne kulkisi eri väylien kautta kuin valvonnan ja hallinnan vaatima tietoliikenne.

Yksi ratkaisumalli, jolla valvonnan kuormituksen ongelmaa on yritetty ratkaista, on ns. dynaaminen tai älykäs valvonta, joka tunnistaa ongelmalliset sovelluksen osat ja valvoo niitä aktiivisesti ja valvoo passiivisemmin sellaisia sovelluksen osia, joissa suoritus on normaalia. Dynaaminen valvonta voidaan toteuttaa joko automaattisesti siten, että valvontatoteutus itse pystyy tekemään jonkinlaista data-analyysiä siitä, mitä pitäisi valvoa, tai manuaalisesti, kuten usein suurilla hajautetuissa järjestelmissä, joissa järjestelmänvalvoja kytkee päälle tai ottaa pois päältä suorituskyvyn kannalta tärkeitä mittareita.

Dynaamisen valvonnan toteutus on helpointa silloin, jos sekä sovelluksen valvontatoteutus että etähallintasovellus kuuluvat samaan tuotteeseen. Näin ei kuitenkaan useimmiten ole, koska valvontaohjelmistot eivät ole useimpien yritysten ydinliiketoimintaa ja näin ollen järjestelmänhallinnassa käytetään kolmannen osapuolen ohjelmistoja. Toisaalta toiset teknologiat myös mahdollistavat paremmin dynaamisen valvonnan. Zabbix-valvontasovellus esimerkiksi pystyy keräämään JMX-dataa ainoastaan staattisesti sille määritellyllä intervallilla, mutta kohdesovellukseen toteutettu SNMP-agentti kykenee lähettämään managerille viestejä myös ilman, että Zabbix kutsuu ko. agenttia.

Erityisesti kriittisissä järjestelmissä keskeinen suunnitteluperiaate on, että valvontajärjestelmän luotettavuuden pitäisi olla korkeampi kuin valvottavan sovelluksen [Goodloe and Pike, 2010]. Valvontadataa pitäisi siis kyetä keräämään valvottavalta sovellukselta myös silloin, kun jokin kohdesovelluksen osio

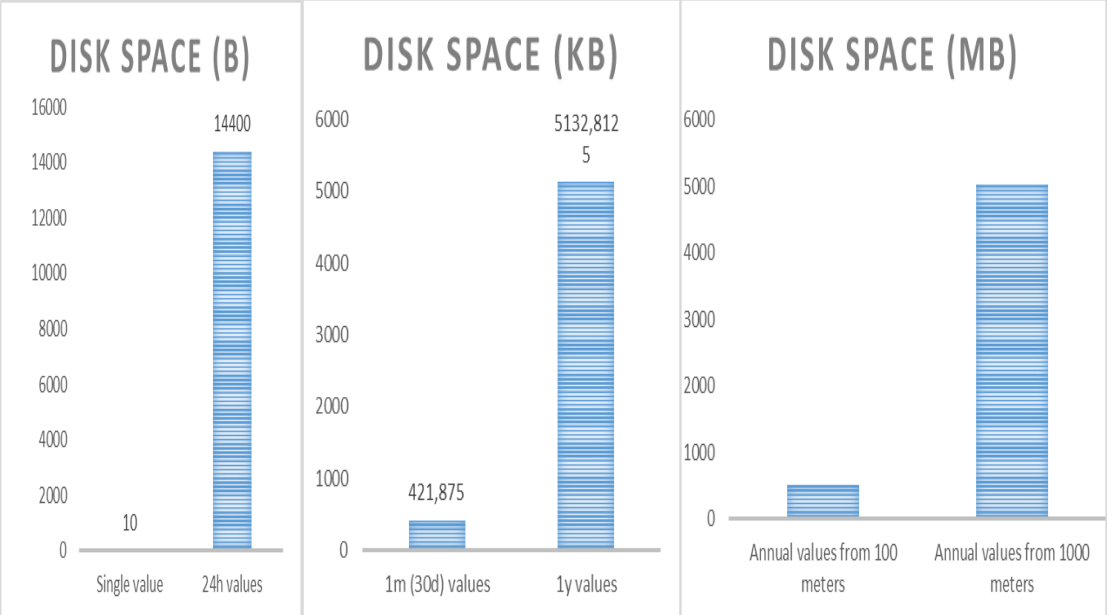
ei ole toimintakykyinen, jotta esimerkiksi vikatilanteen korjaantuminen ja palautuminen toimintakykyiseen tilaan voitaisiin tunnistaa.

Valvontajärjestelmän korkeampi saatavuus on yleensä helpompi toteuttaa silloin, kun valvontatoteutus on kohdejärjestelmästä erillinen prosessi, koska tällöin valvontatoteutus ei vikaannu, vaikka kohdesovelluksen prosessi vikaantuisikin. Toisaalta, vaikka valvontaratkaisu olisikin kohdejärjestelmän lähdekoodin sisällä, se voi silti pysyä toimintakykyisenä, vaikka jokin toinen osio sovelluksesta lakkaisikin toimimasta. Myös se, että valvottavalta sovellukselta ei saada valvontadataa, voi indikoida riittävästi sitä, että valvottava sovellus ei toimi.

### 4.5.2. Valvontadatan säilytys ja arkistointi

Valvontajärjestelmien keskeinen tarkoitus on paitsi kerätä valvontadataa, myös säilyttää sitä riittävän pitkältä aikaväliltä. Tämä on tärkeää, jotta valvontadatasta voidaan muodostaa kuvaajia ja niiden perusteella tunnistaa trendejä ja saada jonkinlainen käsitys, miten valvottava järjestelmä toimii tuotantokäytössä pitkän ajan kuluessa. Tämän historiallisen datan perusteella voidaan mahdollisesti myöhemmin tunnistaa samankaltaista käyttäytymistä kohdejärjestelmässä ja ennakoida mahdollisia ongelmatilanteita.

Ongelmaksi voi kuitenkin muodostua valvontadatan suuri määrä. Yksittäinen monitorin arvo joltakin tietyltä ajanhetkeltä ei itsessään vie muistia kovinkaan paljon, mutta jos kohdejärjestelmässä on valvottavia mittareita satoja tai tuhansia, uniikkeja arvoja on niin monta, että tallennustilaakin vaaditaan huomattavan paljon. Jos esimerkiksi yksittäisen mittarin arvo kerätään minuutin välein, tulee vuorokaudessa tallennettavaksi 60 * 24 = 1440 uniikkia arvoa. Tämä pitää vielä kertoa mittarien määrällä, jotta saadaan vuorokautisten uniikkien arvojen määrä, esim. 100 mittaria vuorokaudessa tuottaa 144 000 uniikkia arvoa. Kun tämä vielä kerrotaan vuorokausien määrällä, esimerkiksi kymmeneltä vuorokaudelta tulee noin puolitoista miljoonaa (1 440 000) uniikkia arvoa tallennettavaksi.




*Kuva 11: Esimerkki valvontadatan tarvitsemasta levytilasta*

Valvontadatan vaatimaa levytilan käyttöä on havainnollistettu kuvassa 11. Tässä esimerkissä mittarin yksittäinen arvo vaatii 10 tavua tallennustilaa, jolloin vuorokaudessa mittarin arvojen tallennus tarvitsee noin 14 kilotavua. Kuukaudessa valvontadataa kertyy kyseisen mittarin osalta 420 kilotavua ja vuositasolla noin 5 megatavua. Tämä on vielä melko vähän, mutta jos vastaavanlaisia mittareita on käytössä 100 kappaletta, niiden vaatima tallennustila vuodessa on noin 500 megatavua ja jos taas mittareita on 1000, joka on vielä melko kohtuullinen määrä yksittäiselle tietojärjestelmälle, tallennustilaa tarvitaan noin 5 gigatavua.

Valvontadatan säilytykseen liittyvät ongelmat voidaan jossain määrin ratkaista suunnittelemalla tuotettava valvontadata muistinkäytön kannalta tehokkailla rakenteilla. Tavallisesti numeraaliset ja totuusarvotyyppiset arvot vaativat vähemmän tallennustilaa kuin merkkijonotyyppinen data, joten valvottavat mittarit kannattaa toteuttaa siten, että esimerkiksi niiden nimi tai kuvaus kertoo käyttötarkoituksen niin hyvin, että mittari voidaan toteuttaa muistinkäytön kannalta tehokkailla tietotyypeillä. Arkistoitavaa valvontadataa voisi yrittää myös suodattaa siten, että ainoastaan muutokset tallennettaisiin, koska monet mittarit (esimerkiksi käyttäjän istunnon tila) ovat sellaisia, jotka pysyvät pitkän aikaa samana. Tällöin kaiken datan tallentamisesta syntyy myös merkittävää redundanssia, koska tallennetaan myös pitkiä sekvenssejä, joiden sisältö on sama.

Useimmat valvontajärjestelmät kykenevät myös prosessoimaan keräämäänsä valvontadataa siten, että esimerkiksi pidemmän aikaa säilytetty data

aggregoidaan ja keskiarvoistetaan sitä pienemmälle tarkkuudelle mitä pidempään sitä on säilytetty. Tätä käsittelytapaa voidaan kutsua myös datan tiivistämiseksi [Limoncelli et al., 2017]. Esimerkiksi viikon vanha valvontadata voitaisiin keskiarvoistamalla pitää tallessa sellaisena kuin se on, kun taas esimerkiksi yli viikon mutta alle kuukauden vanha data esitettäisiin vain kunkin tunnin keskiarvona. Vanhempi data voitaisiin puolestaan esittää esimerkiksi vuorokautisena keskiarvona.

Toisaalta joissakin tilanteissa toimintaan voi myös vaikuttaa lainsäädännöllisiä tekijöitä, jotka velvoittavat arkistoimaan kaiken valvonnasta syntyvän datan. Muutenkin erityisesti kriittisissä järjestelmissä voi säilytettävän datan määrä olla suurempi kuin ei-kriittisissä, esimerkiksi Kufelin [2016] mukaan kriittisen järjestelmän tyypillinen datan säilytysaikasuositus on 3-12 kuukautta. Riittävän pitkä säilytysaika on perusteltu myös siksi, että valvontadataa voidaan tarvita mahdollisten järjestelmävirheiden selvityksessä ja korjauksessa. Tällöin datan säilytyksen kannalta optimoitu mittarien suunnittelu on kuitenkin keino, jota voidaan edelleen käyttää.

### 4.5.3. Oikean ja relevantin valvontadatan tunnistaminen

Hajautetut järjestelmät ovat yleensä rakenteeltaan hyvin monimutkaisia, eikä järjestelmän käyttäytymistä välttämättä pystytä täysin ennustamaan. Siksi yksittäinen virhe voi laukaista useampia erilaisia hälytyksiä valvomoon, ja useat samanaikaiset hälytykset voivatkin aiheuttaa sekaannusta järjestelmänvalvojille. Monitorien tulisikin olla sellaisia, että niistä voidaan tunnistaa ongelman juurisyy mahdollisimman hyvin. Toisaalta useat hälytykset samasta pisteestä myös varmentavat toisiaan; kun vaikka yksittäinen palvelu kaatuu ja useampi siitä riippuva palvelu indikoi, että riippuvuus ei ole kunnossa, tiedetään, että valvonta toimii oikein, kun kaikki kaatuneesta palvelusta riippuvat palvelut hälyttävät siitä.

Useat valvontajärjestelmät kykenevät myös näyttämään kojetaulullaan koostetusti tämänkaltaiset häiriötilanteet, mikä auttaa järjestelmänvalvojaa keskittymään olennaisiin hälytyksiin. Tällaista ominaisuutta kutsutaan myös kerrokselliseksi valvonnaksi, ja se perustuu siihen, että matalalla tasolla kerätään valvontadataa ja ylemmällä tasolla analysoidaan ja yhdistetään raakadatasta tunnistettu informaatio siten, että monesta lähteestä tulevat samanaikaiset hälytykset näytetään vain yhtenä.

Ongelmana voi olla myös liian tiuhat hälytykset, jos hälytysten lähetystä ja vastaanottoa ei pystytä rajoittamaan. Tämä on riskinä esimerkiksi sellaisissa toteutuksissa, joissa agentti kykenee itsenäisesti lähettämään sanomia valvontajärjestelmään. Jos taas mittarien arvoja kysytään järjestelmältä periodisesti

esimerkiksi minuutin välein, tämänkaltaista tilannetta ei periaatteessa pääse syntymään, koska valvottavan mittarin tila voi vaihtua vain kerran käytettävän valvontaperiodin aikana. Toisaalta silloin häiriötä ei myöskään havaita yhtä nopeasti kuin valvonta-agentin itsenäisesti indikoidessa muutosta, joten kaikkein kriittisimpiä toiminnallisuuksia valvottaessa periodinen valvonta ei yksin ole riittävä ratkaisu. Myös Limoncelli ja muut [2007] mainitsevat, että sekä periodiselle että paikallistamisperustaiselle valvonnalle on hyvä olla tuki niin historiallisen kuin reaaliaikaisenkin valvontadatan keruussa.

Hälytysten määrä voi olla ongelmallinen myös silloin, jos niitä on paljon ja hälytyksiä tulee jatkuvasti, koska silloin järjestelmänvalvojalta saattaa hämärtyä kokonaiskuva siitä, mitä on vialla. Liian usein toistuvat hälytykset voivat myös heikentää luottamusta järjestelmänvalvontaan, erityisesti silloin, jos osa hälytyksistä on turhia, ja tällöin todellinen häiriötilanne saatetaan jättää huomiotta, kun sitä ei uskota enää todelliseksi tilanteeksi. Tätä myös Limoncelli ja muut [2007] käsittelevät kirjassaan.

### 4.5.4. Järjestelmänvalvonnan puutteellinen dokumentointi

Järjestelmänvalvontaa dokumentoidessa ei aina ajatella, että monesti järjestelmänvalvojana saattaa toimia henkilö, joka ei tunne järjestelmää yhtä hyvin kuin sen kehittäjät tai loppukäyttäjät, etenkään järjestelmän liiketoimintakontekstia. Tämä on relevantti tapaus erityisesti silloin, kun järjestelmän palveluntuotanto on ulkoistettu toiselle yritykselle, joka on eri kuin järjestelmän toimittava yritys.

Tämän takia mahdollisimman yksiselitteiset toimintaohjeet erilaisiin vikatilanteisiin ja valvottavien mittarien ja niiden tulkinnan riittävän tarkka kuvaus on tärkeätä. Tähän voi vaikuttaa jo sovellusvalvonnan suunnittelussa toteuttamalla mahdollisimman intuitiivisesti ja yksiselitteisesti nimettyjä valvontarajapintoja ja -mittareita, mutta kaikkea ei pysty välttämättä suunnittelemaan täysin erillisesti järjestelmän toimintaympäristöstä.

Kattavan dokumentaation toteuttamista voi helpottaa, jos dokumentaation pystyy tekemään ohjelmakoodin kirjoittamisen yhteydessä. Tällöin yksittäisten valvontarajapintojen dokumentointi on osa kehitysprosessia ja varsinaisen valvontadokumentaation luonti voidaan automatisoida. Tämä helpottaa myös järjestelmän ylläpitoa, kun dokumentaatio voidaan päivittää rajapintaa päivitettäessä.

# 5. Hälytys- ja hätäkeskustoiminnan sekä Insta Response -ohjelmiston ja ERICA-tietojärjestelmän esittely

Hälytyskeskustoiminta liittyy turvallisuusalan vartiointiliikkeiden toimintaan ja tarkoittaa teknisten ilmaisimien antamien hälytysten vastaanottamista ja välittämistä eteenpäin [Turvallisuusala, 2017]. Tällaisia järjestelmiä ovat esimerkiksi rikosturvallisuuteen liittyvät ryöstö- ja murtoilmaisimet sekä erilaiset kiinteistöautomaatiojärjestelmät valvottavissa kohteissa. Hälytyskeskustoimintaan kuuluu usein myös erilaisia päivystyspalveluita, esimerkiksi vartiointi- tai kiinteistönhuoltopäivystystä.

Hätäkeskustoiminta tarkoittaa valtioiden julkisesti rahoittamaa hätäilmoitus- ja hälytystoimintaa. Suomessa on tällä hetkellä kuusi hätäkeskusta, joiden tehtävänä on ottaa vastaan ensihoito-, poliisi- ja pelastustoimialan sekä sosiaalitoimen hätäilmoituksia ja muodostaa niistä tarvittaessa hälytyksiä viranomaisille [Sisäministeriö, 2017]. Manner-Suomen hätäkeskusten toiminnasta vastaa Hätäkeskuslaitos, joka toimii sisäministeriön alaisuudessa. Lainsäädännön tasolla hätäkeskustoimintaa ohjaa laki hätäkeskustoiminnasta [Finlex, 2010a] ja valtioneuvoston asetus hätäkeskustoiminnasta [Finlex, 2010b].

Hätäkeskuksessa on tyypillisesti muutama kymmenen hätäkeskuspäivystäjää, jotka ottavat hätäilmoituksia vastaan ja hälyttävät toimialojen kenttäyksiköitä ilmoituksista luoduille tehtäville. Toiminnan keskeisin osa on ns. hälytysketju, joka koostuu karkeasti ottaen hätäilmoituksen vastaanottamisesta ja käsittelystä sekä hälyttämisestä. Yleisimmät hätäilmoitukset ovat yleiseen hätänumeroon (Euroopassa 112) soitetut hätäpuhelut, mutta myös esimerkiksi kiinteistöjen palo- ja murtoilmaisimien hälytykset ohjataan hätäkeskukseen ja hälytettäväksi, jos ne on hyväksytty Hätäkeskuslaitoksen valvonnan piiriin.

Lisäksi monet uudet teknologiat mahdollistavat myös täysin uudenlaiset keinot hätäilmoitusten välittämiseen, kuten esimerkiksi hätätekstiviestit, jotka on suunnattu erityisesti kuulovammaisille, autoissa käytettävä eCall-järjestelmä, joka soittaa hätäkeskukseen automaattisesti kolarin sattuessa, videopuhelut hälytyskeinona tai Suomessa käytössä oleva 112 Suomi -mobiilisovellus, jolla soitettaessa soittaja voidaan paikantaa puhelimen omaa GPS-vastaanotinta hyödyntäen jopa kymmenen metrin tarkkuudella. Tavallisesti hätäpaikantamiseen käytettävä operaattorin solupaikannus voi olla tarkkuudeltaan taajamien ulkopuolella jopa useita kilometrejä [Hätäkeskuslaitos, 2017].

Vastaanottaessaan hätäilmoituksen hätäkeskuspäivystäjä yrittää muodostaa tilannekuvan hätäilmoitukseen liittyvien tietojen pohjalta. Tällaisia tietoja ovat esimerkiksi hätäilmoituksen tekijältä kysyttävät tarkentavat tiedot tapahtuneesta sekä erilaisten paikannuslähteiden tuottama tieto ilmoittajan sijainnista. Näistä tiedoista järjestelmälle annettujen syötteiden perusteella järjestelmä tyypillisesti laskee jonkinlaisen *vasteen* (response) hätäilmoitusta mallintavalle tapahtumalle, eli viranomaisyksiköistä ja muista tekijöistä muodostuvan kokonaisuuden, joka on järjestelmän käsittelemän datan perusteella optimaalinen tilanteen hoitamiseksi. Tällaista aktiviteettia, joka laskee suuria datamääriä ja tekee ehdotuksen tilanteeseen vaadittavista kenttäyksiköistä, kutsutaan vasteenlaskennaksi. Vasteenlaskentaan vaikuttavia tekijöitä ovat esimerkiksi yksikön (esimerkiksi poliisipartio tai paloauto varusteineen ja miehistöineen) sijainti, tehtävän kohdetoimialat, yksikön käytettävissä olevat resurssit (esimerkiksi henkilöt ja varusteet), odottavien tehtävien määrä ja tehtävän kiireellisyys. Järjestelmän laskeman vasteen perusteella päivystäjä voi joko hyväksyä hälytettävät yksiköt tai muokata vastetta.

Kun päivystäjän mielestä tarkoituksenmukaiset yksiköt on valittu tehtävälle, päivystäjä hälyttää tehtävän. Tehtävän hälyttäminen tapahtuu jonkin kommunikaatiokanavan kautta, joita ovat esimerkiksi matkapuhelinverkot, sähköposti, viranomaiskäyttöön suunnitellut radioverkot ja kenttäjohtojärjestelmät. Tyypillisesti hälytyskanavia on samaan aikaan useita käytössä siltä varalta, että jokin niistä ei toimi, tai jos esimerkiksi yksittäinen verkko ei ole toiminnassa. Jos yksikkö hyväksyy hälytyksen, yksikkö lähtee suorittamaan saamaansa tehtävää ja hätäkeskus saa myös tästä tiedon.

Hätäpuheluiden vastaanottamisen ja käsittelyn ja hälyttämisen lisäksi hätäkeskuksilla on myös muita kriittisiä tehtäviä, jotka liittyvät usein esimerkiksi lainsäädännöllisiin vaatimuksiin. Hätäkeskusten pitää esimerkiksi tallentaa toimintaansa liittyviä asiakirjoja ja muuta dataa, kuten puhelutallenteita, useiden vuosien ajan, jotta toiminta on läpinäkyvää ja tapahtumien kulku on epäselvissä tilanteissa mahdollisimman tarkasti selvitettävissä. Järjestelmässä olevaa tietoa on myös päivitettävä aina kun muutoksia tulee. Nämä muutokset saattavat liittyä yksittäiseen toimialaan, mutta voivat olla myös suurempia, kaikkia koskevia muutoksia.

### 5.1. Insta Response -ohjelmisto ja ERICA-tietojärjestelmä

Insta DefSec Oy (jatkossa Insta DefSec) [Insta DefSec, 2017b] on suomalainen turvallisuusteknologia-alan perheyritys, joka tuottaa verkkokeskeisiä puolustus- ja turvallisuusteknologiapalveluita ja -ratkaisuja sekä kotimaisille että kan-

sainvälisille asiakkaille. Eräs Insta DefSecin kehittämistä tuotteista on Insta Response -tuoteperhe [Insta DefSec, 2017a], joka on hälytys- ja hätäkeskustoimintaan suunniteltu ohjelmistotuotekokonaisuus. Insta Response -ohjelmistotuoteperhe koostuu yhteisestä ohjelmistoalustasta ja kolmesta sitä käyttävästä ohjelmistotuotteesta, jotka ovat Insta Response CC, Insta Response Portal ja Insta Response Field.

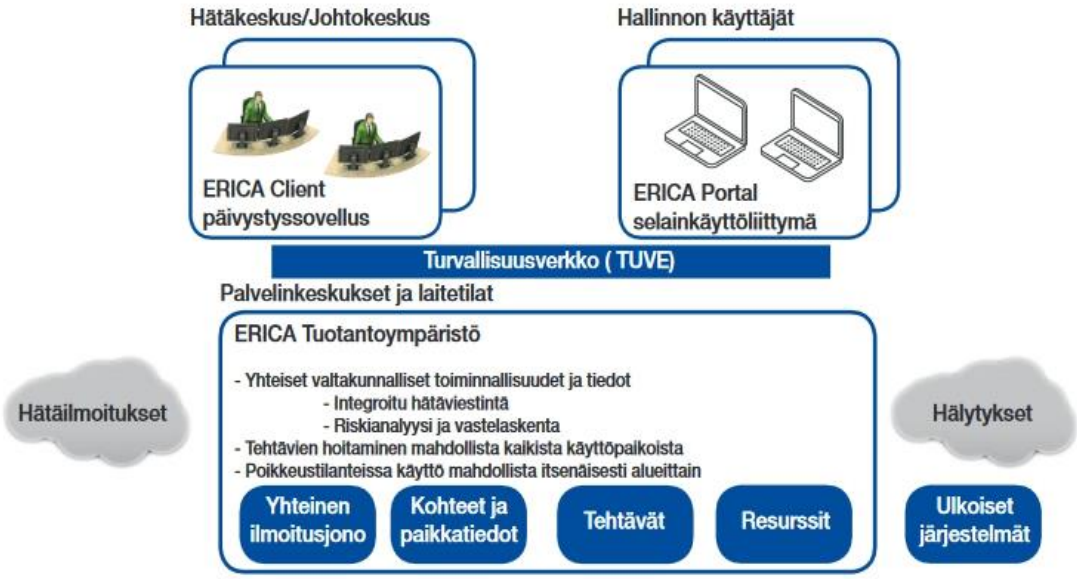
Insta Response CC, tai Response Client, on päivystyskäyttöön suunniteltu työasemasovellus, joka tarjoaa hätä- tai hälytyskeskuksen päivystäjille työkalut hätäilmoitusten vastaanottamiseen, käsittelemiseen, hälyttämiseen ja tilannekuvan seuraamiseen. Insta Response Portal puolestaan on Internet–selaimella käytettävä web-sovellus, jolla hallinnolliset käyttäjät voivat konfiguroida järjestelmädataa. Kolmas ohjelmistotuote, Insta Response Field, on suunniteltu mobiililaitteilla käytettäväksi kenttäsovellukseksi, jolla esimerkiksi kenttäyksikkö voi välittää tietoa operatiivisesta toiminnasta hätäkeskukseen.

Insta Response -ohjelmiston pohjalle rakentuu myös ERICA-tietojärjestelmä, Suomen tuleva hätäkeskustietojärjestelmä, joka otetaan käyttöön kaikissa Suomen hätäkeskuksissa vuoden 2018 aikana. ERICA on moniviranomaistietojärjestelmä, jota tulevat käyttämään kaikki Suomen keskeiset turvallisuusviranomaiset: poliisi, pelastustoimi, ensihoito, sosiaalitoimi ja Rajavartiolaitos. Viranomaisten lisäksi ERICA-hankkeessa on mukana myös useita kolmannen osapuolen toimijoita, jotka toimittavat järjestelmän osia ja jotka vastaavat järjestelmään liittyvästä infrastruktuurista.

ERICA on hajautettu tietojärjestelmä, joka koostuu useasta erillisestä palvelusta, jotka kukin vastaavat jostain järjestelmän osa-alueesta. Suurin osa näistä palveluista toimii palvelinkeskuksessa, johon kaikki hätäkeskukset ovat yhteydessä kansallisen turvallisuusverkon (TUVE) kautta, mutta osa palveluista asennetaan paikallisesti myös hätäkeskuksiin. Normaalitilanteessa hätäkeskus käyttää palvelinkeskuksen palveluita, mutta poikkeustilanteessa, jos esimerkiksi verkkoyhteys palvelinkeskukseen katkeaa, on hätäkeskus edelleen rajoitetusti toimintakykyinen omien palvelujensa avulla. [Hätäkeskuslaitos, 2015].

Palvelinkeskuksessa on järjestelmän valtakunnallinen data, jota kaikki hätäkeskukset käyttävät. Normaalitilanteessa hätäpuhelu tai muu hätäilmoitus ohjataan ilmoittajan sijainnin mukaan paikallisen hätäkeskusalueen hätäkeskukseen. Poikkeustilanteissa, kuten esimerkiksi yksittäisen hätäkeskuksen ruuhkautuessa tai vikaantuessa, hätäilmoitukset voidaan ohjata myös toiseen hätäkeskukseen. Koska hätäkeskusten käyttämä data on valtakunnallinen, hä-

täkeskukset voivat käsitellä myös muiden kuin oman hätäkeskusalueen hätäilmoituksia. [Hätäkeskuslaitos, 2015].




*Kuva 12. ERICA-järjestelmän arkkitehtuuri [Hätäkeskuslaitos, 2015]*

Kuvassa 12 on kuvattu ERICA-järjestelmän arkkitehtuuria. Hätäkeskuspäivystäjät ja johtokeskusten työntekijät käyttävät hätä- ja johtokeskuksissa päivystyssovellusta (ERICA Client, vastaava kuin Insta Response CC), jonka avulla he vastaavat hätäilmoituksiin ja luovat niistä toimialoille tehtäviä. Tehtävät hälytetään ulkoisten järjestelmien, kuten esimerkiksi viranomaisradioverkon (VIRVE) kautta asianosaisille viranomaisille. Itse datan käsittely ja tallennus tapahtuu palvelinkeskuksissa ja laitetiloissa. Lisäksi järjestelmään kuuluu ERICA Portal -selainkäyttöliittymä (vastaava kuin Insta Response Portal), jonka kautta eri toimialojen hallinnon käyttäjät voivat lisätä ja päivittää järjestelmään oman toimialansa tarvitsemia ja käyttämiä perustietoja. [Hätäkeskuslaitos, 2015].

### 5.2. Hätäkeskustietojärjestelmän valvonta

Toimiva järjestelmänvalvonta on oleellinen osa hätäkeskustietojärjestelmää, sillä hätäkeskustietojärjestelmälle on usein määritelty hyvin korkea saatavuusvaatimus, eikä näin ollen koko järjestelmää voida ajaa alas huoltokatkon ajaksi, vaan sen on oltava jatkuvasti toiminnassa. Tämän takia myös mahdolliset vikakorjaukset on kyettävä tekemään nopeasti, joten on oleellista, että vikatilanteet huomataan mahdollisimman varhain.

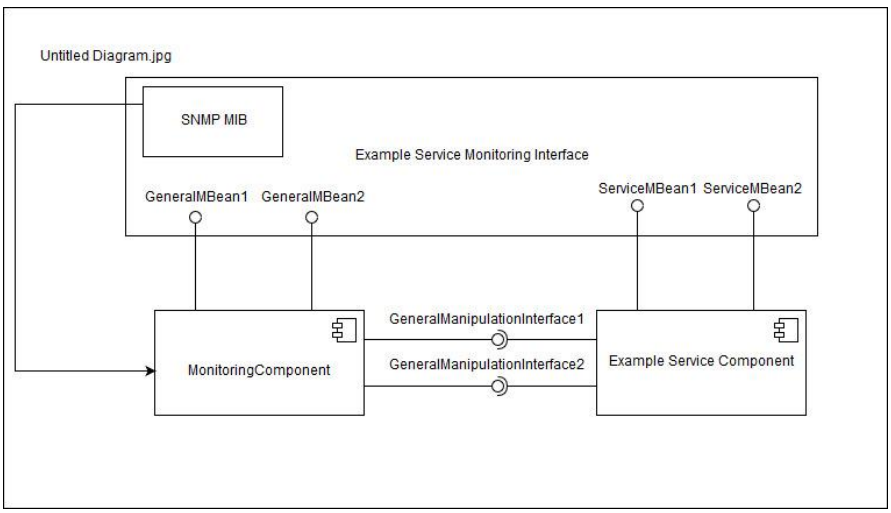
Valvonnan kautta kerättävä data voi paitsi paljastaa vikoja, myös todentaa sen, että järjestelmä toimii oikealla tavalla ja näin ollen vastaa asiakkaan vaatimuksia liittyen järjestelmän toimintaan. Tämä on erityisesti palvelun-tuotantona tuotettavien järjestelmien osalta tärkeää, jotta palvelutasovelvoitteet ja korkeat saatavuusvaatimukset voidaan todentaa asiakkaalle. Esimerkiksi ERICA-tietojärjestelmälle on määritelty palvelutasovelvoitteeksi 99,996 % [Insta DefSec, 2011].

Hätäkeskustietojärjestelmässä on usein käytössä myös kolmannen osa-puolen palveluita. Tällöin on tärkeää, että järjestelmä pystyy myös tunnis-tamaan vikatilanteita, jotka ovat näissä kolmannen osapuolen palveluissa, kos-ka niiden korjaaminen on tällöin toisen palveluntarjoajan vastuulla ja he eivät välttämättä ole tietoisia ongelmasta. Pitkäaikainen häiriö kolmannen osapuolen palvelussa rajoittaa myös koko hätäkeskustietojärjestelmän toimintakykyä, jo-ten mahdolliset viat on korjattava nopeasti järjestelmän saatavuuden varmis-tamiseksi.

# 6. Insta Response -ohjelmiston palveluiden valvonta

Koska Insta Response -ohjelmisto on toteutettu Java-teknologioilla, ohjelmistoon kuuluvien palveluiden valvonta on toteutettu pitkälti JMX-teknologialla. Lisäksi osa valvonnasta on toteutettu SNMP:llä, jotta kaikkein kriittisimmät tilamuutokset voidaan välittää valvomoon välittömästi. Apuna tähän on Java-kirjasto, jonka avulla Java-sovelluksiin voi integroida SNMP-agentteja ja sovellus voi näin lähettää ja vastaanottaa SNMP-viestejä.

Insta Response -ohjelmiston kokonaisuus on hyvin laaja ja siksi myös valvontakokonaisuus on laaja. Järjestelmä koostuu useasta eri palvelusta, joista kullakin on tietyt samat valvontaominaisuudet ja lisäksi monta erilaista palvelukohtaista toteutusta. Tämänkaltaisessa kriittisessä ohjelmistossa, jota käytetään kriittiseen käyttötarkoitukseen, on kuitenkin perusteltuakin olla laaja valvontatoteutus, joka takaa hyvän näkyvyyden sen osien toimintaan. Vaikka kaikkia mittareita ei välttämättä oteta varsinaiseen järjestelmänvalvontaan asiakasversioissa mukaan, on helpompi aktivoida tarpeen tullen olemassa oleva mutta ei-valvottava mittari valvottavaksi kuin toteuttaa kokonaan uusi mittari ohjelmakoodiin.




*Kuva 13: Esimerkki Response-palvelun käyttämistä valvontarajapinnoista*

Kuvassa 13 on esitelty yleisellä tasolla yksittäisen palvelun tarjoamat valvontarajapinnat. Valvontakomponentti tarjoaa palvelun muille komponenteille manipulointirajapinnat, joiden avulla valvottavien attribuuttien arvoja voidaan päivittää, ja varsinaiset valvontarajapinnat ulkopuolisten ohjelmistojen ja järjestelmien käyttöön. Palvelukomponentti voi myös tarjota valvontaraja-

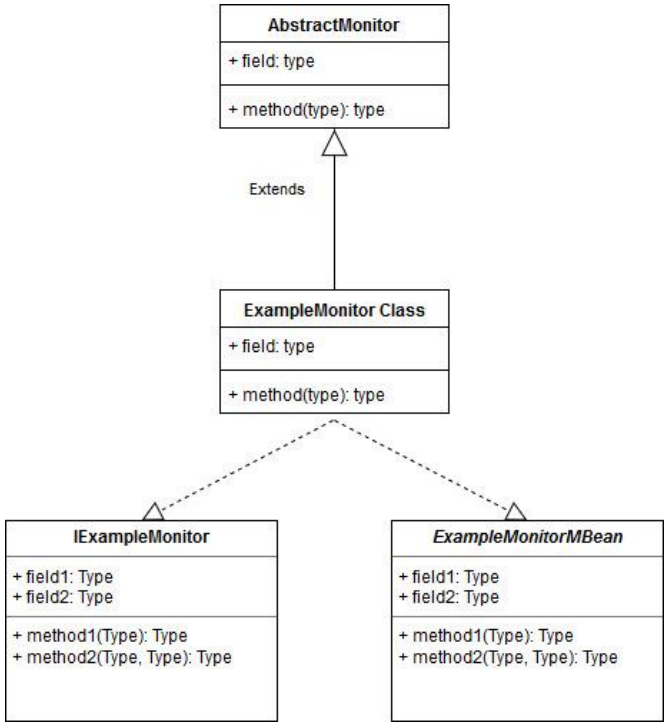
pintoja ulospäin. Palvelukomponentin ja valvontakomponentin JMX-rajapinnat sekä valvontakomponentin tarjoama SNMP-toiminnallisuus muodostavat yhdessä koko palvelun valvontarajapinnan.

### 6.1. JMX-valvontarajapinnat

Insta Response -ohjelmistoon on toteutettu kaikille palveluille yhteisiä JMX-valvontarajapintoja ja -toteutuksia, jotka ovat kaikkien palveluiden käytössä olevassa valvontakomponentissa. Tämä mahdollistaa sen, että kaikkien palveluiden käyttämään valvontatoiminnallisuuteen voidaan tehdä tarvittaessa muutoksia yhteen paikkaan ja muutokset tulevat siitä huolimatta voimaan kaikkiin niitä käyttäviin palveluihin. Lisäksi tämä osittain takaa sen, että kaikilla palveluilla on tietynlainen perustason valvontatoiminnallisuus, joka toimii johdonmukaisesti samalla tavalla kaikilla palveluilla.

Lisäksi palveluilla on palveluiden toimintalogiikkaan ja siihen liittyviin muuttujiin liittyviä valvontarajapintoja ja -toteutuksia. Nämä on pääosin toteutettu palveluiden komponenteissa eikä varsinaisesti valvontakomponentissa, mutta valvontakomponentissa on kuitenkin apuluokkia, joiden avulla näitä palvelukohtaisia monitoreita voidaan toteuttaa. Tämäkin on perusteltua siltä kannalta, että yhteisiä kantaluokkia käytettäessä voidaan tehdä isommat muutokset yhteen paikkaan ja peruslogiikka on johdonmukaisempaa kaikissa toteuttavissa konteksteissa.

Kuvassa 14 on esimerkki tavallisesta Insta Responsessa käytettävän valvontatoteutuksen luokkarakenteesta. Valvontatoteutus toteuttaa pääsääntöisesti kaksi erillistä Java-rajapintaa. Toinen on varsinainen valvontarajapinta (MBean) ja toinen on muokkausrajapinta, jonka avulla valvottavien JMX-attribuuttien arvoja voidaan päivittää. Osassa MBean-rajapinnoista on mukana myös JMX-manipulointioperaatioita, mutta vain silloin, jos rajapinnalla halutaan mahdollistaa järjestelmän manipulointi JMX:n avulla. Pääsääntöisesti manipuloinnille ei kuitenkaan ole tarvetta, ja koska järjestelmää käytetään kriittisessä toimintaympäristössä, on myös tarkkaan harkittava, millainen manipulointi on välttämättä tarpeen JMX:n avulla.




*Kuva 14: Esimerkki yksittäisen Responsen valvontatoteutuksen luokkarakenteesta*

Päivitysoperaatiot tehdään pääsääntöisesti silloin, kun palvelussa tapahtuu jokin muutos liittyen JMX-rajapinnan julkaisemien attribuuttien arvoihin. Esimerkiksi ulkoisten rajapintojen yhteyksien valvonnassa on kuitenkin toteutettu mekanismi, jolla yhteys tarkistetaan tietyn aikaintervallin välein. Tämä on tarpeen erityisesti kolmannen osapuolen tarjoamien ulkoisten liityntöjen kanssa, koska joissakin tapauksissa liityntää käytetään toiminnalliseen tarkoitukseensa hyvin harvoin ja liityntä ei välttämättä itse viestitä aktiivisesti valvontaan mahdollisesta vikatilanteesta. Myös järjestelmän sisäisissä osissa, kuten tietokantayhteyksissä, on tärkeää välittää tieto yhteyskatkosta valvomoon nopeasti.

### 6.2. SNMP

Kriittisimmät valvontamittarit, jotka liittyvät palveluiden saatavuuteen tai riippuvuuksien tiloihin, ovat JMX:n lisäksi toteutettu myös SNMP-teknologialla. Tähän on syynä erityisesti se, että SNMP-agentti voi lähettää itsenäisesti viestejä hallintajärjestelmälle, eikä se ole riippuvainen arvojen periodisesta kyselystä. Periaatteessa myös JMX:lle voitaisiin toteuttaa SNMP-viestien kaltaisia notifikaatioita, mutta useimmat kolmannen osapuolen valvontasovellukset ja -järjestelmät eivät tue tällaisia JMX-notifikaatioita, vaan ainoastaan periodista valvontaa JMX-attribuuteista.

Pääsääntöisesti SNMP-toteutus on tehty JMX-toteutuksen rinnalle, jolloin sama tieto välitetään periaatteessa kahteen kertaan valvontajärjestelmälle.

Tämä kuitenkin toimii myös varmentavana mekanismina, koska todennäköisesti ainakin jompikumpi mekanismi toimii. Koska hälytys saapuu SNMP:n kautta melkein välittömästi valvontajärjestelmään, voi järjestelmänvalvoja aloittaa korjaavat toimenpiteet parhaassa tapauksessa ennen kuin SLA:n mukainen palvelun saatavuusprosentti laskee. Tämä on kriittistä erityisesti silloin, kun lyhytkin periodinen valvontaväli on huomattava osuus ajasta, jonka järjestelmä voi olla ei-saatavana.

Näihin kriittisimpiin tapahtumiin kuuluvat esimerkiksi palveluiden väliset ulkoiset ja sisäiset riippuvuudet (toiset palvelut, ulkoisten liittymien rajapinnat tai tietokannat), palvelun sisäiseen tilaan vaikuttavat tekijät kuten esimerkiksi säikeiden tilat, mahdolliset deadlock-tilanteet sekä yksittäisten palveluinstanssien roolit palveluklusterissa.

### 6.3. Palveluiden valvonnan dokumentointi

Insta Responseen on myös toteutettu työkalu, jolla voidaan generoida lähdekoodin perusteella automaattisesti dokumentaatio kunkin komponentin käyttämistä valvontarajapinnoista. Dokumentaation pohjana on MBean-valvontarajapinnoille määritellyt omat kuvausannotaatiot ja erilliset kuvausoliot, jotka kukin esittävät tietyn valvontarajapinnan instanssin. Kuvausoliot kootaan kaikki yhteen kooditiedostoon, josta ne voidaan hakea reflektion avulla.

Uudenlainen tapa dokumentoida valvonta ei ole kuitenkaan vielä järjestelmänlaajuisesti käytössä. Tulevaisuudessa on tarkoitus dokumentoida kaikki olemassa olevat valvontarajapinnat uudella mekanismilla, jolloin manuaalisen dokumentaation määrä vähenee. Lisäksi dokumentaatiota pitäisi vielä laajentaa siten, että se kykenisi luomaan myös SNMP-dokumentaation, sillä tällä hetkellä dokumentaatio toimii vain JMX-rajapintojen kanssa.

### 6.4. Jatkokehitysideoita Insta Response -tuotteen valvontaan

Tällä hetkellä Insta Response -tuotteen valvontakokonaisuus on toimiva ja tarjoaa hyvän näkyvyyden valvomolle ohjelmiston sisäiseen toimintaan. Sovellusvalvontaa toteutettaessa ei kuitenkaan aina pystytä täysin aukottomasti tunnistamaan kaikkea, mitä pitäisi valvoa, joten uusien mittarien toteuttaminen on jatkuva prosessi. Lisäksi uusia ominaisuuksia toteutetaan tuotteeseen koko tuotteen elinkaaren ajan, joten myös niiden osalta on tärkeää tunnistaa valvonnan tarpeet mahdollisimman kattavasti.

Yksi potentiaalinen jatkokehitysidea valvonnan osalta olisi jonkinlainen mekanismi, joka mahdollistaisi ohjelmakoodin perusteella valvontajärjestelmälle sopivien valvontamäärityksien määrittelyn. Tähän liittyen on myös

tehty alustavaa selvitystyötä valvontamääritysten automaattisesta generoinnista käytössä olevalle valvontajärjestelmälle. Toisaalta mekanismin olisi kuitenkin oltava sellainen, että se tukee useampien eri valvontajärjestelmien tukemaa syntaksia tai esitystapaa, jolloin olisi helposti mahdollista käyttää eri asiakasversioissa eri valvontajärjestelmiä.

Jatkossa olisi mielenkiintoista toteuttaa myös työasemasovellusten ja ylipäätään työasemien valvontaan oma palvelu, joka keräisi toimipisteen asiakassovelluksista ja niitä pyörittävistä työasemista dataa, analysoisi sitä ja välittäisi havaitsemiansa tietoja valvomoon valvontasovellukselle. Tämänkaltaiseen ratkaisuun voisi mahdollisesti toteuttaa myös jonkinlaisen käyttöliittymän, jolloin se kykenisi näyttämään myös valvonnan tilannekuvaa.

# 7. Yhteenveto

Järjestelmänvalvonta ja sovellusvalvonta ovat nykyaikaisten tietojärjestelmien ja tietokonesovellusten keskeisiä aktiviteetteja, jotka mahdollistavat ajonaikaisen datan keruun järjestelmästä tai sovelluksesta ja sen analysoinnin. Valvonta antaa järjestelmän ylläpitäjille näkyvyyden järjestelmän todelliseen ajonaikaiseen toimintaan ja voi sekä paljastaa järjestelmävirheitä että myös ennakoida mahdollisia häiriötilanteita. Käänteisesti valvonta voi myöskin osoittaa, että järjestelmä toimii oikein ja siten täyttää sille määritellyt toiminnalliset ja ei-toiminnalliset vaatimukset, ja siksi valvontadatan tuottama informaatio järjestelmän toiminnasta toimii myös palveluntuotantoon liittyvän SLA-laskennan pohjana.

Kriittisten tietojärjestelmien ollessa kyseessä myös valvonta on kriittinen aktiviteetti, jonka on oltava vähintään yhtä luotettava ja toimiva kuin valvottava järjestelmä. Valvonta on välttämätön osa kriittisten järjestelmien palveluntuotantoa, jotta vikatilanteet voidaan havaita mahdollisimman nopeasti ja mahdollisesti myös korjata ennen kuin ne näkyvät järjestelmän loppukäyttäjille. Viime kädessä valvonta tarjoaa tietojärjestelmän tuotannon eri sidosryhmille tietoa järjestelmän todellisesta laadusta ja toiminnasta.

Valvonta on myös suunniteltava huolellisesti kriittisissä tietojärjestelmissä, koska järjestelmäresurssit ovat rajalliset ja valvonta ja sen käyttö vaatii myös oman osansa niistä. Samaan aikaan, kun häiriötilanteet on havaittava nopeasti ja valvontadataa on kerättävä riittävän säännöllisesti ja riittävän paljon, on myös pidettävä huoli, että järjestelmän tavallinen toiminta ei häiriinny tai hidastu merkittävästi valvonnan takia. Toisinaan on kuitenkin tehtävä kompromissiratkaisuja, jotta sekä reaaliaikainen näkyvyys järjestelmän toiminnalliseen tilaan että järjestelmän oikeellinen toiminta voidaan taata. Valvonta on kuitenkin tunnistettava jo suunnittelun varhaisessa vaiheessa tärkeäksi osaksi järjestelmää ja ohjata myös järjestelmän kehitystä siten, että valvonnan vaatimukset otetaan huomioon.

Tässä työssä olen käsitellyt valvontaa myös osana todellista kontekstia, hätä- ja hälytyskeskustoiminnassa käytettäviä tietojärjestelmiä ja ohjelmistoja sekä niiden kehitystyötä ja palveluntuotantoa. Hätä- ja hälytyskeskustoiminta on välttämätöntä nykyaikaisen yhteiskunnan päivittäisen toiminnan sekä kansalaisten turvallisuuden kannalta, eikä niiden tehtäviä kyetä tekemään tehokkaasti ja luotettavasti ilman tietojärjestelmien apua. Siksi on tärkeää, että järjestelmänvalvonta on mahdollisimman toimiva myös näissä toimintaympäristöissä ja kykenee takaamaan näiden järjestelmien toimivuuden ja

mahdollisten häiriötilanteiden nopean havaitsemisen. Toimiva järjestelmänvalvonta hätä- ja hälytyskeskusten tietojärjestelmissä on myös tärkeää mahdollisten integroitavien kolmannen osapuolen järjestelmien valvomiseksi ja niiden ongelmien havaitsemiseksi.

## Viiteluettelo

[Chang and Minkin, 2008] Chang, Shuchih E. and Boris Minkin. 2008. Monitoring enterprise applications and the future of self-healing applications. *International Journal of Enterprise Information Systems*, 4(2), 54-66.

[Delgado et al., 2004] Nelly Delgado, Ann Quiroz Gates, and Steve Roach. 2004. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on Software Engineering*, 30, 12, 859-872.

[Edmiston, 2007] A H Edmiston. 2007. The role of systems and applications monitoring in operational risk management. *BT Technology Journal*, January 2007, 25, 1, 68–78.

[Ewaschuck, 2016] Rob Ewaschuk. Monitoring distributed systems. In: Betsy Beyer, Chris Jones, Jennifer Petoff, And Niall Richard Murphy, *Site Reliability Engineering*. O'Reilly Media.

[Finlex, 2010a] Laki hätäkeskustoiminnasta 692/2010. Saatavissa http://www.finlex.fi/fi/laki/alkup/2010/20100692. Tarkastettu 17.10.2017.

[Finlex, 2010b] Valtioneuvoston asetus hätäkeskustoiminnasta 877/2010. Saatavissa http://www.finlex.fi/fi/laki/ajantasa/2010/20100877. Tarkastettu 17.10.2017.

[Goodloe and Pike, 2010] Alwyn E. Goodloe and Lee Pike. 2010. Monitoring Distributed Real-Time Systems: A Survey and Future Directions. NASA/CR-2010-216724. NASA Langley Research Center.

[Hernantes et al., 2015] Josune Hernantes, Gorka Gallardo, and Nicolás Serrano. 2015. IT Infrastructure-Monitoring Tools. *IEEE Software*, 32, 4, 88-93.

[van Hoorn et al., 2012] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. 2012. Kieker: A Framework for Application Performance

Monitoring and Dynamic Software Analysis. In: *ICPE '12 Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. 247-248

[Hätäkeskuslaitos, 2015] Hätäkeskuslaitos, Uusi hätäkeskustietojärjestelmä ERICA, esite. Saatavissa https://www.112.fi/download/63468_ERICA_esite_high_res.pdf?6400473c0ee5d288. Tarkastettu 17.10.2017

[Hätäkeskuslaitos, 2017] Hätäkeskuslaitos, Matkapuhelinpaikannus. Saatavissa http://www.112.fi/hatatilanne/matkapuhelinpaikannus. Tarkastettu 17.10.2017

[IBM, 2017] IBM Knowledge Center - Instrumenting ARM. Saatavissa https://www.ibm.com/support/knowledgecenter/en/SS5MD2_7.4.0.1/com.ibm.itcamt.doc/arm/arm_intro.html. Checked 13.11.2017

[Insta DefSec, 2011] Insta DefSec, Instan-tietojarjestelmaliiketoiminta-laajentunut-merkittavasti. Saatavissa http://www.instagroup.fi/defsec/fi/blog/instan-tietojarjestelmaliiketoiminta-laajentunut-merkittavasti/. Tarkastettu 17.10.2017

[Insta DefSec, 2017a] Insta DefSec, Hätä-ja hälytyskeskusratkaisut. Saatavissa https://www.instadefsec.fi/fi/ratkaisut/hata-ja-halytyskeskusratkaisut.html. Tarkastettu 17.10.2017

[Insta DefSec, 2017b] Insta DefSec. Saatavissa http://www.instadefsec.fi. Tarkastettu 17.10.2017

[Joyce et al., 1987] Jeffrey Joyce, Greg Lomow, Konrad Slind, and Brian Unger. 1987. Monitoring distributed systems. *ACM Transactions on Computer Systems*, 5, 2, May 1987. 121-150.

[Kreger et al., 2002] Heather Kreger, Ward Harold, and Leigh Williamson. 2002. *Java and JMX: Building Manageable Systems.* Addison-Wesley.

[Kufel, 2016] Łukasz Kufel. Tools for distributed systems monitoring. 2016. *Foundations of Computing and Decision Sciences*, 41(4). 237-260.

[Lahmadi et al., 2009] Abdelkader Lahmadi, Laurent Andrey, and Olivier Festor. 2009. Performance of network and service monitoring frameworks. In *Proc. of the 2009 IFIP/IEEE International Symposium on Integrated Network Management*, 815-820.

[Limoncelli et al., 2007] Thomas A. Limoncelli, Christina J. Hogan, and Strata R. Chalup. 2007. *The Practice of System and Network Administration.* Addison-Wesley

[Microsoft, 2017] How SNMP Works. Saatavissa https://technet.microsoft.com/en-us/library/cc783142(v=ws.10).aspx. Checked 13.11.2017

[Plattner and Nievergelt, 1981] Bernhard Plattner and Jurg Nievergelt. 1981. Special Feature: Monitoring Program Execution: A Survey. *Computer* 14, 11, Nov. 1981. 76-93.

[Sisäministeriö, 2017] Sisäministeriö, Hätäkeskustoiminta. Saatavissa http://intermin.fi/fi/hatakeskustoiminta. Tarkastettu 17.10.2017

[Techopedia, 2017a] Techopedia: Distributed System. Available at https://www.techopedia.com/definition/18909/distributed-system. Checked 17.10.2017

[Techopedia, 2017b] Techopedia: Mission-critical-system. Available at https://www.techopedia.com/definition/23583/mission-critical-system. Checked 17.10.2017.

[Turvallisuusala, 2017] Turvallisuusala. Saatavissa http://www.mol.fi/avo/alat/90.htm. Tarkastettu 17.10.2017.

[Wikipedia, 2017] Web-Based Enterprise Management - Wikipedia. Saatavissa https://en.wikipedia.org/wiki/Web-Based_Enterprise_Management. Tarkastettu 17.11.2017.

[Yammer Metrics, 2017] Yammer Metrics. Available at http://metrics.dropwizard.io/3.2.3/. Checked 17.10.2017.