Tampereen yliopisto

Mikael Penttinen

SUUNNITTELUMALLIEN HYÖDYNTÄMINEN TURVALLISUUSKRIITTISIEN JÄRJESTELMIEN KEHITYKSESSÄ

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Joulukuu 2021

TIIVISTELMÄ

Mikael Penttinen: Suunnittelumallien hyödyntäminen turvallisuuskriittisien järjestelmien kehityksessä
Kandidaattitutkielma
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Joulukuu 2021

Turvallisuuskriittiset järjestelmät ovat järjestelmiä, joiden vikaantuminen voi johtaa hengenvaaraan, ympäristövahinkoihin tai muuhun vaaratilanteeseen. Näiden järjestelmien tärkeimpiä ominaisuuksia ovat luotettavuus, ennalta-arvattavuus ja vikasietoisuus. Useat turvallisuuskriittiset järjestelmät ovat sulautettuja ohjausjärjestelmiä eri teollisuuden aloilla. Esimerkkejä sulautetuista turvallisuuskriittisistä järjestelmistä ovat autojen ABS-jarrujärjestelmät, raideliikenteen raidekytkimet ja voimalaitosten ohjausjärjestelmät. Keskeinen osa sulautettujen turvallisuuskriittisten järjestelmän laitteiston ja ohjelmiston kehitystä, on hyvien suunnittelumallien hyödyntäminen. Suunnittelumalli on yleinen järjestelmän suunnittelun ratkaisu johonkin tiettyyn ongelmaan. Sen tarkoituksena on kuvata järjestelmän rakennetta ja arkkitehtuuria yleismaailmallisesti. Suunnittelumalli ei ole esimerkiksi valmis ohjelmistoarkkitehtuuri tai laitteiston kytkentäkaavio. Turvakriittisissä järjestelmissä hyödynnettävillä suunnittelumalleilla pyritään parantamaan järjestelmän luotettavuutta, saatavuutta ja vikasietoisuutta.

Tämän työn tavoitteena oli kartoittaa turvallisuuskriittisille järjestelmille tutkittuja suunnittelumalleja ja niiden suhdetta turvallisuuskriittisille järjestelmille asetettuihin standardeihin. Työssä esitellään muutamia keskeisiä suunnittelumalleja. Käsitellään sekä niiden hyötyjä ja haittoja että soveltuvuutta eri tasoisille turvallisuuskriittisille järjestelmille.

IEC 61508 -standardi on keskeisin tässä työssä hyödynnetty turvallisuuskriittisten järjestelmien standardi. Standardissa määritetään neljä turvallisuuden eheyden tasoa, joiden kasvavia vaatimuksia voidaan täyttää tietyillä standardissa määritetyillä suunnittelutekniikoilla. IEC 61508 -standardissa määritettyjä suunnittelutekniikoita hyödynnetään tässä työssä käsiteltävissä suunnittelumalleissa.

Turvallisuuskriittisille järjestelmille tutkituissa suunnittelumalleissa yhteistä on järjestelmän kriittisien ominaisuuksien hajauttaminen tai monentaminen. Lisäksi turvallisuutta parannetaan erottamalla turvallisuudesta vastaavat funktiot omaan järjestelmän osaan. Järjestelmän tilan ja ohjelman sekvenssin valvonta ulkoisella valvonnalla on tehokas ja yksinkertainen tapa varmistaa, että järjestelmä toimii oikein eikä esimerkiksi kaadu tai pysähdy. Pelkkä valvonta ei kuitenkaan varmista järjestelmän toiminnan eheyttä. N-versio ohjelmoinnilla tarkoitetaan ohjelmiston kehittämistä itsenäisesti N-kertaa. Tällä pyritään pienentämään suunnitteluvirheestä aiheutuvaa virheellistä toimintaa. N-versioiden suorituksen tuloksia voidaan verrata äänestäjällä tai hyväksyttävyystestillä. Lisäksi N-versiot voidaan ajaa yhdellä tai useammalla eri laitteisolla. Samalla laitteistolla N-versioita suorittaessa voidaan hyödyntää järjestelmän tilan palauttamista järjestelmäresurssien säästämiseksi. Lisäksi suunnittelumalleilla voidaan parantaa järjestelmän vikasietoisuutta hyödyntäen sulavaa alasajoa, hiljaista vikaantumista tai dynaamista uudelleenkonfigurointia.

Avainsanat: Sulautetut järjestelmät, turvallisuuskriittiset järjestelmät, suunnittelumallit, ohjelmistoarkkitehtuuri, laitteistokehitys

Sisällysluettelo

1 Johdanto

Turvallisuuskriittisiä järjestelmiä on käytössä usealla eri teollisuuden alalla. Yleinen hyväksytty määritelmä turvallisuuskriittiselle järjestelmälle on tämä: ”Järjestelmä, jonka vikaantuminen voi johtaa hengenvaaraan tai vahinkoon käyttäjälle tai ympäristölle” [1]([2], s. 344). Tämä tarkoittaa, että näiden järjestelmien toimintavarmuus on erittäin tärkeää ja että niiden toiminta vikatilanteessa on oltava ennalta tiedossa. Turvallisuuskriittisien järjestelmien suunnitteluun on asetettu alariippuvaisesti useita standardeja. Näissä standardeissa asetetaan suunnittelumallien kannalta tärkeitä ohjelmisto- ja laitteistokehitystekniikoita [3] [4]. Suunnittelutekniikoiden vaatimukset ja suositukset vaihtelevat näissä standardeissa asetettujen turvallisuuden eheyden tasojen mukaisesti.

Turvakriittisille järjestelmille oleellista on järjestelmän luotettavuus ja saatavuus ([5], s. 90). Saatavuudella tarkoitetaan, että osa tai kaikki järjestelmän toiminnallisuudesta on saatavilla myös toimintavirheen jälkeen. Järjestelmän rakenteen suunnittelun apuna voidaan käyttää suunnittelumalleja. Turvallisuuskriittisessä kontekstissa suunnittelumalleilla pyritään parantamaan nimenomaan turvakriittisille järjestelmille oleellisia ominaisuuksia. Niitä sovelletaan sekä ohjelmiston että laitteiston kehityksessä.

Tässä työssä kartoitetaan turvallisuuskriittisille järjestelmille tutkittuja suunnittelumalleja sekä tutkitaan, miten niitä hyödyntämällä voidaan saavuttaa turvallisuuskriittisille järjestelmille asetettuja vaatimuksia ja suosituksia sekä parantaa järjestelmien luotettavuutta ja saatavuutta. Luvussa 2 käsitellään turvallisuuskriittisien järjestelmien ja suunnittelumallien ymmärtämiseen tarvittavat taustatiedot. Luvussa 3 käsitellään tutkimusmenetelmät. Luvussa 4 käsitellään kirjallisuuskatsauksen keskeisimmät tulokset. Luvussa 5 käydään läpi turvallisuuskriittisille järjestelmille tämän työn kannalta keskeisiä standardeja. Luvuissa 6–8 esitellään tässä työssä tutkitut suunnittelumallit tarkemmin ja luvussa 9 yhteenveto tästä työstä.

2 Tausta

2.1 Sulautetut turvallisuuskriittiset järjestelmät

Turvallisuuskriittiset järjestelmät ovat tietokonejärjestelmiä, joiden toimintahäiriöillä voi olla vakavia seuraamuksia. Seuraamus voi olla esimerkiksi käyttäjän asettaminen hengenvaaraan tai vahingon aiheutuminen ympäristölle [1]. Turvallisuuskriittisiä järjestelmiä on käytössä useilla aloilla, esimerkkejä ovat liikenteen sovelluksissa käytettävät laiteohjaimet ([2], s. 341), lääkintäteollisuudessa käytettävät tietokoneisiin perustuvat laitteet, ydinvoimaloiden ohjausjärjestelmät sekä monet muut teollisuuden laitteet [6].

Turvallisuuskriittisille järjestelmille oleellista on toimintavarmuus. Järjestelmät on suunniteltava siten, että niiden vikaantuminen on erittäin harvinaista ja niiden saattaminen vikaantuneeseen tilaan on vaikeaa. Lisäksi järjestelmien käyttäytyminen vikatilanteessa on oltava ennalta tiedossa. Turvallisuuskriittisten laitteiden vaatimuksista on säädetty useassa standardissa. Yleinen standardi turvallisuuskriittisille laitteille IEC-61508 määrittelee turvallisuus vaatimuksia ja suosituksia sähköisille ja ohjelmoitaville laitteille [7]. IEC-61508 on käytetty pohjana muille alakohtaisille standardeille.

Monet turvallisuuskriittiset järjestelmät ovat ohjelmoitavia sulautettuja ohjausjärjestelmiä. Tämä tarkoittaa, että niiden turvallisuus toteutetaan lähinnä ohjelmistossa. Ohjelmistojen kehittäminen siten, että niiden toimintaa voidaan pitää erittäin turvallisena, on erittäin haastavaa, erityisesti ohjelmiston koon ja toiminnallisuuden kasvaessa.

2.2 Suunnittelumallit

Suunnittelumalli (eng. design pattern) on yleinen ratkaisu toistuvasti esiintyvään ongelmaan ([8], kappale 1). Sen tarkoituksena on yksinkertaistaa ohjelmiston kehitystä ja helpottaa projektin ylläpitoa. Jokaisella suunnittelumallilla on neljä keskeistä elementtiä: nimi, ongelma, sen ratkaisu ja seuraamukset. Seuraamuksilla tarkoitetaan mahdollisia suunnittelumallin haittoja tai kompromisseja. Tietyn suunnittelumallin hyvien ja huonojen puolien ymmärtäminen on tärkeää malleja vertailtaessa. Koska yksi suunnittelumalli vastaa tasan yhtä ratkaisua tiettyyn ongelmaan, voi jonkun ongelman ratkaisemiseksi olla monia erilaisia suunnittelumalleja. Suunnittelumallin tarkoituksena ei ole olla täydellinen kuvaus esimerkiksi ohjelmistoarkkitehtuurista tai järjestelmän toteutuksesta, vaan toimia pintapuolisena mallina, jonka päälle järjestelmä toteutetaan.

Suunnittelumalleja voidaan hyödyntää myös laitteiston kehityksessä [9]. Vaikka suunnittelumalleja tutkittiin aluksi olio-ohjelmoinnin laadun parantamiseksi, voidaan suunnitte-

lumalleja hyödyntää myös laitteistokehityksessä. Uudelleenkäytettäviä laitteistokomponentteja kuten muisteja, suorittimia, digitaalisia signaaliprosessoreja ja muita laitteistokomponentteja kehitetään erillä toisistaan. Laitteistotason suunnittelumalleja voidaan hyödyntää järjestelmä kokonaisuuden kehittämisessä näistä erillisistä komponenteista. Suunnittelumallien tavoitteet laitteistokehityksessä ovat laajalti samat kuin ohjelmistokehityksessä: laadun parantaminen ja kehitystyön helpottaminen. Laitteistokehityksessä on kuitenkin huomioitava, että suunnittelumallit eivät takaa erillisten laitteistokomponenttien laatua, vaan siitä vastaa jokainen valmistaja itse. Kuten monissa sulautetuissa järjestelmissä, turvallisuuskriittisissä järjestelmissä voidaan hyödyntää samanaikaisesti eri laitteiston ja ohjelmiston suunnittelumalleja [10].

3 Tutkimusmenetelmä

Tämä tutkielma on toteutettu kirjallisuuskatsauksena. Sopivia lähteitä on etsitty pääasiallisesti IEEE/IET Electronic Librarystä, ScienceDirectistä ja SpringerLinkistä. Lisäksi lähteitä on etsitty helmenkasvatustekniikalla aineistoja lukemalla sekä Google Scholarista.

Käytettyjä hakusanoja: ”safety-critical”, ”design patterns”, ”embedded systems”, “software developent”

Tietoa on etsitty lähinnä englannin kielellä. Haetun materiaalin julkaisuvuosien rajauksessa on otettu huomioon turvallisuuskriittisten järjestelmien pitkä kehitysaika ja käyttöikä, joten hyödyllisiä lähteitä on voitu julkaista jo 90-luvulla.

Tutkielma keskittyy turvallisuuskriittisten järjestelmien ohjelmistojen ja laitteistojen suunnittelumallien etsimiseen ja vertailemiseen. Lisäksi suunnittelumalleista tehtyä riskianalyysiä ja muuta relevanttia tutkimusta on voitu hyödyntää. Etsityssä materiaalissa on jonkin verran päällekkäisyyksiä suunnittelumallien ja ohjelmistoarkkitehtuurin välillä. Lisäksi sellaiset tulokset, joissa käsitellään turvallisuuskriittistä ohjelmistosuunnittelua jollain tietyllä ohjelmointikielellä, on jätetty pois, koska ohjelmointikielikohtaiset suunnittelurajoitteet eivät välttämättä päde kaikissa tapauksissa.

4 Kirjallisuuskatsauksen tulokset

Suunnittelumalleja on tutkittu turvallisuuskriittiseen käyttöön 90-luvulta alkaen. Erityisesti N-versio-ohjelmointia on käsitelty laajasti. Lisäksi erilaiset valvojat ja palautuslohkot ovat yleisiä aiheita. Tässä työssä on otettu mukaan myös uudempia ajoneuvoteollisuuden sovelluksiin tutkittuja suunnittelumalleja, joilla pyritään parantamaan järjestelmän toimintaa vikatilanteessa. Suunnittelumalleja käsittelevän lähdemateriaalin keskeisimmät tulokset on esitetty taulukossa 1. Taulukko on jaettu yleisimipien teemojen mukaisesti. Näiden lisäksi myös muita suunnittelumalleja on mainittu lähdemateriaalissa. Löydettyjä suunnittelumalleja käydään läpi tämän työn luvuissa 6–8.

Näiden lähteiden lisäksi tässä työssä on käytetty lähdemateriaalia myös tutkimaan turvallisuuskriittisien järjestelmien ja suunnittelumallien perusteita. Lisäksi työssä viitataan IEC 61508 ja ISO 26262 standardeihin, koska ne ovat keskeisiä suunnittelumalleja kehitettäessä ja arvioidessa.

Taulukko 1. Lähdemateriaalissa käsitellyt suunnittelumallit

Lähde	**Valvojat**	**N-versio ohjelmointi**	**Palautus-lohkot**	**Toimintakyvyn säilytys vikatilanteessa**
Armoush (2008)		x	x	
Armoush (2010)	x	x	x	x
Avizienis (1985)		x		
Douglass (2003)	x	x		
Hobbs (2017)	x	x	x	
Hudson et al. (2018)		x		
Idirin et al. (2011)		x		
Latif-Shabgahi & Bass (2004)		x		
Oszwald et al. (2018)				x
Penha et al. (2015)		x		x
Sinha (2011)		x		
Wu & Kelly (2004)	x	x		

5 Standardit ja suositukset

5.1 IEC 61508

IEC 61508 standardi määrittelee suosituksia ja vaatimuksia yleisille turvallisuuskriittisille järjestelmille ja sitä käytetään pohjana muille standardeille eri aloilla [11]. Tässä standardissa määritellään suositeltuja tekniikoita turvallisuuskriittisien järjestelmien laitteiston ja ohjelmiston suunnitteluun.

IEC 61508 määrittelee neljä turvallisuuden eheyden tasoa (eng. Safety Integrity Level, SIL), joista alin taso on SIL1 ja korkein SIL4 [7]. SIL-tasolla tarkoitetaan turvallisuusfunktion tai turvallisuustoteutuksen suorituskykyä ja eri SIL-tasoille on asetettu vaatimuksia luotettavuudesta, kehityksen laadun vakuuttamisesta sekä testauksesta. Luotettavuutta mitataan vaarantavan vikatilanteen aiheutumisen todennäköisyytenä. Tämä todennäköisyys mitataan eri tavoin jatkuvasti käytössä oleville ja tarvittaessa käytössä oleville laitteille. Taulukossa 2 on esitetty vaatimukset luotettavuudesta laitteille, joita käytetään pyydettäessä. Taulukossa 3 on esitetty luottavuusvaatimukset jatkuvatoimiselle laitteelle.

Taulukko 2. IEC 61508 määritykset SIL tasojen luotettavuudesta pyydettäessä toimille laitteille

SIL	Vaarallisen vikatilanteen todennäköisyys laitteen toimintaa pyydettäessä	Riskin pienennys kerroin
1	0,1–0,01	10–100
2	0,01–0,001	100–1000
3	0,001–0,0001	1000–10000
4	0,0001–0,00001	10000–100000

Taulukko 3. IEC 61508 määritykset SIL tasojen luotettavuudesta jatkuvatoimisille laitteille

SIL	Vaarallisen vikatilanteen todennäköisyys jatkuvatoimiselle laitteelle käyttötuntia kohden	Riskin pienennys kerroin
1	0,00001–0,000001	100000–1000000
2	0,000001–0,0000001	1000000–10000000
3	0,0000001–0,00000001	10000000–100000000
4	0,00000001–0,000000001	100000000–1000000000

IEC 61508 suosittelee erilaisia ohjelmointi- ja laitteistokehitystekniikoita käytettäväksi turvallisuuskriittisissä laitteissa. Suositelluilla kehitystekniikoilla pyritään parantamaan laitteiden luotettavuutta ja vikasietoisuutta. Vertailukomponenttia voidaan käyttää tunnistamaan virheitä aikaisessa vaiheessa. Vertailun periaatteena on, että kahden itsenäisen suorittimen signaaleita vertaillaan laitteistossa toteutetulla vertailijalla ([12] , A.1.3). Äänestäjäkomponenttia suositellaan tunnistamaan virheitä useasta laitteistolla toteutetusta signaalikanavasta. Äänestäjää voidaan valvoa ja testata ulkoisesti tai se voi valvoa itse itseään ([12], A.1.4). Osa 7 määrittelee myös suunnittelutekniikoita muistivirheiden välttämiseksi ja tiedon eheyden säilyttämiseksi [12].

IEC 61508-3 käsittelee turvallisuuskriittisien järjestelmien ohjelmistoja ja niissä käytettäviä tekniikoita [3]. Liite A käsittelee eri SIL-tasoille suositeltuja ohjelmointitekniikoita ja järjestelmän toiminnan malleja. Tässä työssä merkitykselliset tekniikat on esitetty taulukossa 4.

Taulukko 4. Eri SIL-tasoille suositeltuja ohjelmistoarkkitehtuurin tekniikoita ([3] taulukko A.2), jossa R: suositeltu, HR: vahvasti suositeltu, NR: ei koskaan suositeltu ja -: suositusta ei ole määritelty.

Tekniikka	**SIL1**	**SIL2**	**SIL3**	**SIL4**
Vikojen havaitseminen	-	R	HR	HR
Virheen havaitsevat koodit	R	R	R	HR
Erilaisuuden käyttö valvontatekniikoissa (riippumattomuus valvovan ja valvottavan toiminnon välillä samassa tietokoneessa)	-	R	R	-
Erilaisuuden käyttö valvontatekniikoissa (missä valvova tietokone ja valvottava tietokone ovat erillään	-	R	R	HR
Virheestä toipuminen palaamalla aikaisempaan tilaan	R	R	-	NR
Virheestä toipumisen mekanismi yrittämällä uudelleen	R	R	-	-
Tekoälyyn perustuva vikojen korjaus	-	NR	NR	NR
Modulaarinen lähestymistapa	HR	HR	HR	HR
Hallittu toimintojen heikentäminen / Sulava alasajo	R	R	HR	HR
Dynaaminen uudelleenkonfigurointi	-	NR	NR	NR

5.2 ISO 26262

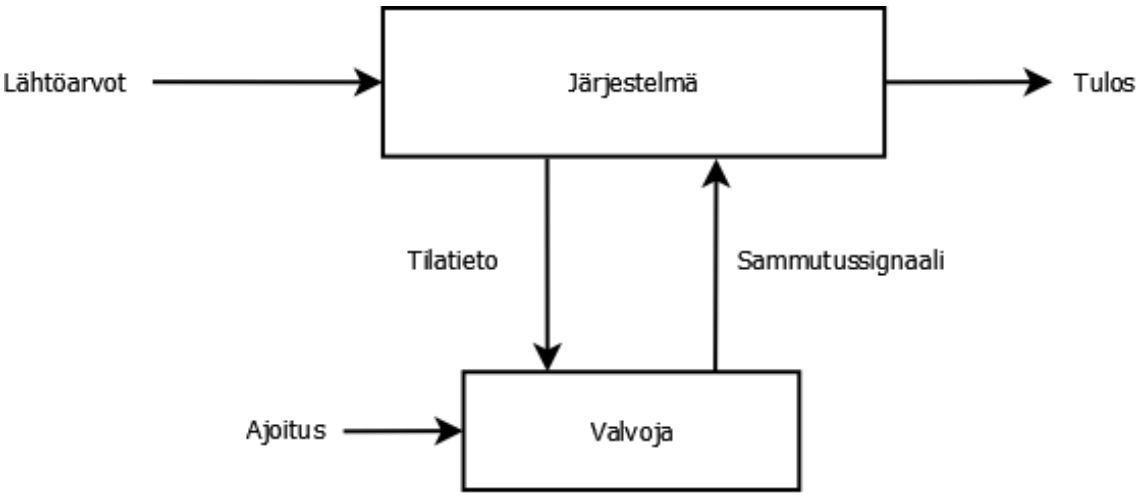
ISO 26262 -standardi soveltaa IEC 61508 standardia ajoneuvoteollisuuden sovelluksiin. Siinä määritellään SIL turvallisuustasoihin perustuvat ASIL-tasot (Automotive Safety Integrity Level). ASIL tasoja on neljä, tasot A-D, joista A on matalin ja D korkein [13] [14].

Standardissa suositellaan tekniikoita laitteiston ja ohjelmiston kehitykseen. Osa 5 käsittelee järjestelmäkehitystä laitteistotasolla. Esimerkkinä järjestelmän tilan valvonnasta suositellaan ulkoista vahtikoiraa ([4], Liite H). Toisena esimerkkinä laitteistovikaantumisen ehkäisystä on esitetty ohjausjärjestelmän ulostulojen suodatus, jolla pyritään poistamaan ulostulosignaalista häiriöitä.

6 Järjestelmän tilan valvonta

6.1 Vahtikoira

Järjestelmän turvallisuutta voidaan parantaa hyödyntämällä järjestelmän tilaa valvovia lohkoja, joista voidaan käyttää myös termiä vahtikoira (eng. watchdog) ([5], s. 149). Yksinkertaisen valvontalohkon suhde pääjärjestelmään on esitetty kuvassa 1. Pääjärjestelmälohkosta irrallaan oleva valvoja vastaanottaa säännöllisesti signaaleita pääjärjestelmältä. Jos pääjärjestelmän toiminnasta kertovaa signaalia ei vastaanoteta ennalta määrätyssä ajassa, ajetaan pääjärjestelmä turvalliseen tilaan. Valvojalohko voi normaalitoiminnan signaalin lisäksi valvoa pääjärjestelmälohkon sisäistä tilaa, sisään meneviä lähtöarvoja ja tuloksia. Valvontalohkon tarkoituksena on varmistaa järjestelmän turvallinen alasajo toimintavirheen sattuessa. Ohjelmisto sekvenssin valvonnan suositukset eri SIL-tasoille on esitetty taulukossa 5.



Kuva 1. Yksinkertainen valvoja

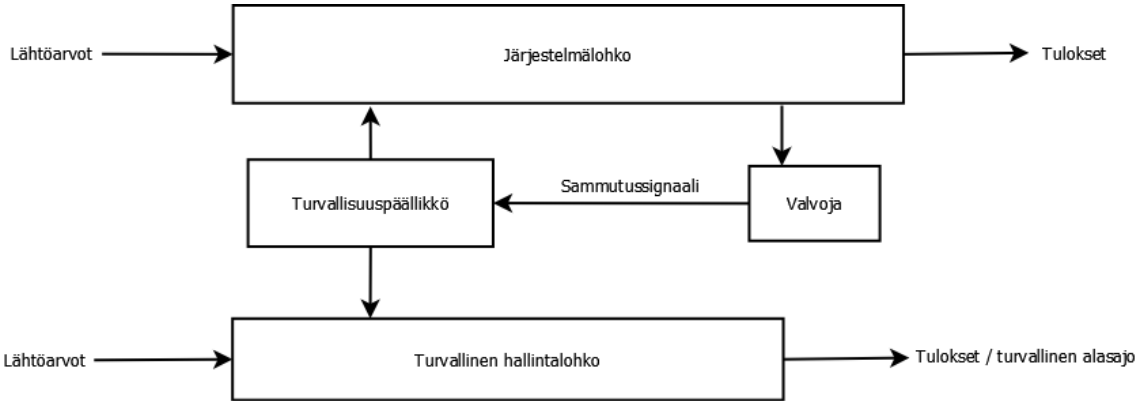
Valvontalohko on suunniteltava mahdollisimman yksinkertaiseksi, eikä sen tarkoituksena ole kahdentaa pääjärjestelmän toiminnallisuutta. Jos valvontalohko valvoo pääjärjestelmän arvoja, varmistaa se niiden pysymisen ennalta määrätyissä raja-arvoissa. Valvontalohko voidaan toteuttaa esimerkiksi ohjelmoitavalla porttiverkolla. Pääjärjestelmälohkon antava säännöllinen signaali on toteutettava siten, että se lakkaa järjestelmän vikaantuessa. Signaalin antaminen erillisessä ohjelmistokeskeytyksessä on huono idea, sillä keskeytyksiä saatetaan ajaa silloinkin, kun järjestelmän toiminta on virheellistä tai järjestelmä on kohdannut pysäyttävän vikatilan. Pääjärjestelmä voi antaa signaaleja myös useammassa vaiheessa. Esimerkiksi kerran sisääntulodatan käsittelyn aikana ja toisen kerran ulostulojen antamisen aikana ([10], s. 98). Koska valvontalohko ja pääjärjestelmälohko ovat osa samaa järjestelmää, saattaa niiden kehitys olla jossain määrin toisiinsa sidottua.

Jos valvojan toiminnallisuus on riittävän kattava, voidaan valvojan ja pääjärjestelmän tuloksia verrata toisiinsa ennen oikean tuloksen antamista [15]. Mikäli valvoja ja pääjärjestelmä eivät päädy samaan tulokseen, erillinen ulostulolohko varmistaa, että järjestelmä ajetaan turvalliseen vikatilaan ajoissa.

Valvontalohkon käyttöön liittyvä haaste on varmistaa, että pääjärjestelmälohkon toiminta on haluttua, vaikka valvontalohko ei tiedä paljoa sen toiminnasta. Niin kutsuttu ”live lock” ([16], s 310) on ongelmatilanne, jossa pääjärjestelmä on tilanteessa, jossa se lähettää olemassaolon signaalin oikein, vaikka olisi vikaantunut tai toimintakelvoton. Yksi tapa parantaa toimintavirheen tunnistettavuutta on vaatia pääjärjestelmälohkolta jotain sen toimintaan liittyvää kriittistä dataa olemassaolon signaalin lisäksi. Tällaisen datan tulisi olla dynaamisesti normaalitoiminnan aikana laskettua, jotta toistuva väärä data voidaan tunnistaa. Toinen parannus pääjärjestelmän oikean toiminnan varmistamiseen, on ajaa sisäänrakennettu testi olemassaolon signaalin antamisen jälkeen. Jos pääjärjestelmän sisäinen testi epäonnistuu, ajetaan järjestelmä alas.

6.2 Turvallisuuspäällikkö

Yksinkertaisen valvontalohkon ongelmana on, että järjestelmän alas ajaminen vikatilanteessa saattaa itsessään olla vaarallista ([10], s. 103). Vaara voi aiheutua esimerkiksi turvallisuuskriittisen järjestelmän hallitessa ympäristöä, jossa on korkeita nopeuksia tai jännitteitä. Ratkaisuna tähän ongelmaan on suunnittelumalli, jossa käytetään valvontalohkon lisäksi turvallisuuspäällikköä (eng. Safety executive) ([10], s. 103). Turvallisuuspäällikkömalli on esitetty kuvassa 2. Turvallisuuspäällikkö on erillinen lohko, jonka tarkoituksena on hallita järjestelmän toiminta virhetilanteen sattuessa. Se sisältää järjestelmän turvallisuuden valvontaan ja turvallisuuden hallintaan liittyvät osat. Turvallisuuspäällikköä käyttävässä järjestelmässä pääjärjestelmälohkoa valvoo valvontalohko, joka vikatilanteen huomatessaan, käskee turvallisuuspäällikkölohkon käyttöön. Turvallisuuspäällikkö ottaa sitten käyttöön toisen järjestelmälohkon, jonka tarkoituksena on ajaa järjestelmä turvallisesti alas. Turvallisuuspäällikkömalli on kallis suunnittelumalli, koska se vaatii useita järjestelmälohkoja ja paljon erilaista toiminnallisuutta. Sitä käytetään järjestelmissä, jotka ovat erittäin kriittisiä ja joiden toiminnan äkillinen keskeytyminen voi aiheuttaa vaaratilanteen. Tätä suunnittelumallia on käytetty mm. suurnopeusjunien ohjausjärjestelmissä ([16], s 314). Turvallisuuspäällikön suositukset eri SIL-tasoille on esitetty taulukossa 5.



Kuva 2. Turvallisuuspäällikkömalli (mukaillen [10], kuva 7.10)

Taulukko 5. Valvojan ja turvallisuuspäällikön soveltuvuus eri SIL tasoille ([10], Taulukko 7.16). WR: Heikosti suositeltu.

Suunnittelumalli	**SIL1**	**SIL2**	**SIL3**	**SIL4**
Ohjelmasekvenssin valvonta	HR	HR	HR	HR
Turvallisuuspäällikkömalli	WR	R	R	R

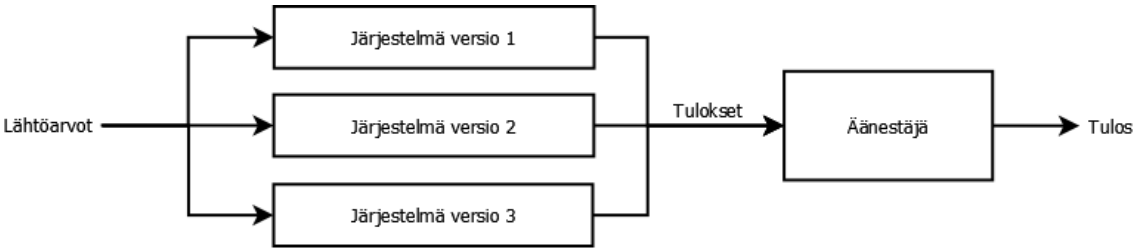
7 N-versio-ohjelmointi

7.1 Äänestäjät

N-versio-ohjelmointi (NVP) on suunnittelumalli, jossa halutusta järjestelmästä kehitetään N määrä erilaisia versioita ([10], s. 111). Kaikki järjestelmä versiot ovat käytössä samanaikaisesti ja käyttävät samaa sisääntulo dataa. Kaikki järjestelmät tuottavat oman tuloksen. Näin saadaan N määrä vertailtavia ulostuloja yhtä sisääntuloarvoa kohden. Näitä ulostuloja vertaa jokin toteutus äänestäjästä. Yksinkertaisessa suunnittelumallissa on yksi äänestäjälohko, joka valitsee N ulostuloista enemmistöarvon. N-versio ohjelmointia voidaan käyttää reaaliaikajärjestelmissä, joissa luotettavuus on kriittistä ja joissa ohjelmiston ja laitteiston kustannukset voidaan kattaa.

N-versio Ohjelmoinnin luotettavuus perustuu useiden itsenäisten tulosten tuottamiseen. Usean keskenään erilaisten ohjelmistojen tuottamisen tavoitteena on pienentää ohjelmointivirheen aiheuttamaa toimintahäiriötä. Jokainen ohjelmistoversio on ajossa omalla suorittimella. Parhaassa tapauksessa jokainen suoritin on arkkitehtuuriltaan erilainen, mutta kustannussyistä usein käytetään moniydinsuorittimia, joissa kukin suoritinydin ajaa yhtä ohjelmistoversiota. Ohjelmistoversiot voidaan ajaa myös peräkkäin yhdellä suorittimella, mutta tämä lisää suoritusaikaa N kertaiseksi ja on täten vähemmän mieleinen lähestymistapa. Ohjelmaversioita kehitettäessä erityisen tärkeää on eri kehitystiimien itsenäisyys, jotta minimoidaan samantyyppiset virheet useassa ohjelmaversiossa [17].

Suosittu ja pitkään käytetty versio NVP-mallista on kolmimodulaarinen redundanssi -malli (triple modular redundance, TMR). TMR-mallissa järjestelmästä toteutetaan kolme versiota, joiden tuloksia vertaillaan äänestäjällä [18]. Yksinkertainen versio TMR-mallista on kuvattu kuvassa 3.



Kuva 3. TMR-äänestäjä

Yksinkertainen NVP-malli käyttää yhtä äänestyslohkoa. Äänestäjän on täten oltava mahdollisimman yksinkertainen, hyvin suunniteltu ja laajasti testattu virheellisen toiminnan estämiseksi.

Äänestäjäkomponentti voidaan toteuttaa ohjelmallisesti tai laitteistolla. Äänestäjä valitsee oikean ulostuloarvon äänestysalgoritmilla, jonka voi suunnitella eri tavoin käyttösovellukseen sopivasti. Äänestäjän toteuttaminen laitteistolla on kannattavaa silloin, kun käsitellään matalan tason korkeataajuista tarkkaa äänestämistä. Kun järjestelmän ulostulodata on monimutkaisen laskennan tulos, ohjelmistossa toteutettu äänestäjä voi olla parempi. Ohjelmallisesti toteutettu äänestäjä voi käyttää monimutkaisempia äänestysalgoritmeja. Lisäksi ohjelmalliseen äänestäjään voidaan toteuttaa monimutkaisempia turvatoimintoja [19].

Sovellusalueesta riippuen äänestäjä voi olla tarkka tai epätarkka. Tarkka äänestäjä vatii, että kaikki järjestelmäversiot tuottavat tarkan tuloksen. Useat liukulukuja käsittelevät järjestelmät saattavat vaatia epätarkan äänestäjän. Esimerkiksi sensoriarvoja lukevat järjestelmäkomponentit saattavat tuottaa toisistaan hieman poikkeavia arvoja myös häiriövapaassa ympäristössä [19]. Epätarkka äänestäjä tuottaa tuloksena aproksimaation sille annetuista lähtöarvoista. Tuloksien sykronointi on erittäin tärkeää epätarkkaa äänestäjää käytettäessä. Erityisesti järjestelmälohkojen toteutuksen poiketessä toisistaan, saattaa myös niiden suoritusajat poiketa toisistaa. Jos äänestäjälle annettavat arvot on tuotettu eri ajassa ja eri lähtöarvoilla, voi tulos olla virheellinen, vaikka järjestelmien toiminta olisikin virheetöntä.

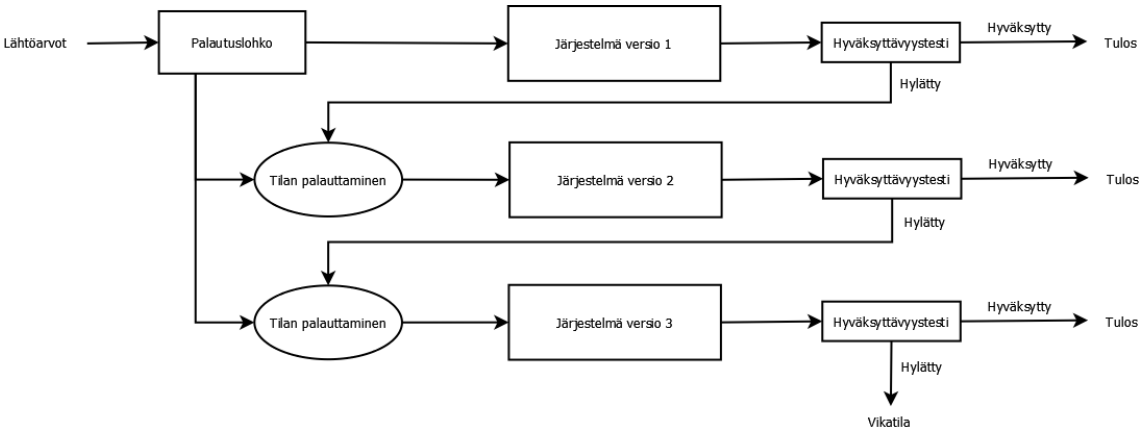
Myös äänestäjäkomponentteja voidaan monentaa, jolloin toimintavarmuutta voidaan nostaa. Suunnittelumalli, jossa äänestäjiä on useita, on aiempaa luotettavampi ja monimutkaisempi ja näin kalliimpi toteuttaa.

SIL4-tasolle voidaan käyttää suunnittelumallia, jossa on kaksi järjestelmälohkoa, joilla toteutetaan kaksi kahdesta (2 out of 2) äänestäjä osaksi laajempaa järjestelmää [20].

Molemmat järjestelmälohkot toimivat omalla suorittimella ja ne niille on osoitettu master-slave-kytkentä. Järjestelmälohkot vaihtavat tietoa keskenään kolmessa vaiheessa. Molemmat järjestelmät lähettävät kukin omat lähtöarvot toisilleen, jonka jälkeen ne äänestävät yhteisen identtisen lähtöarvon molemmille järjestelmille. Master-järjestelmä lähettää slave-järjestelmälle ulostulorajapinnan tiedot, jotta slave-järjestelmä osaa tuottaa oikean tulosviestin. Tämä on tarpeellista, koska tässä suunnittelumallissa vain master-järjestelmä äänestää ulostulon ja kommunikoi muun järjestelmän kanssa. Kun slave-järjestelmä on tuottanut tuloksen, se lähetetään master-järjestelmälle. Tämän jälkeen master vertaa omaa ja slave-järjestelmän tulosta ja äänestää oikean tuloksen. Jos tulokset eroavat toisistaan järjestelmä ei tuota mitään tulosta vaan antaa vikaviestin ylemmälle hallintajärjestelmälle.

7.2 Palautuslohko

Palautuslohko-malli on variaatio NVP-mallista, jossa kukin ohjelmistoversio ajetaan kerrallaan ([5], s. 117). Jokainen versio on varustettu omalla hyväksymistestilohkolla, joka arvioi ohjelmistoversion tuloksen. Jos tulos hylätään, järjestelmä palaa aikaisempaan tilaan ja vaihtaa seuraavaan ohjelmistototeutukseen. Mikäli kaikki peräkkäin ajetut ohjelmistoversiot hylkäävät kukin oman tuloksensa, järjestelmä ajetaan virheturvalliseen tilaan. Kolme järjestelmäversiota sisältävä palautuslohkomalli on kuvattu kuvassa 4.



Kuva 4. kolme järjestelmää sisältävä palautuslohkomalli. Mukaillen ([10], kuva 8.2).

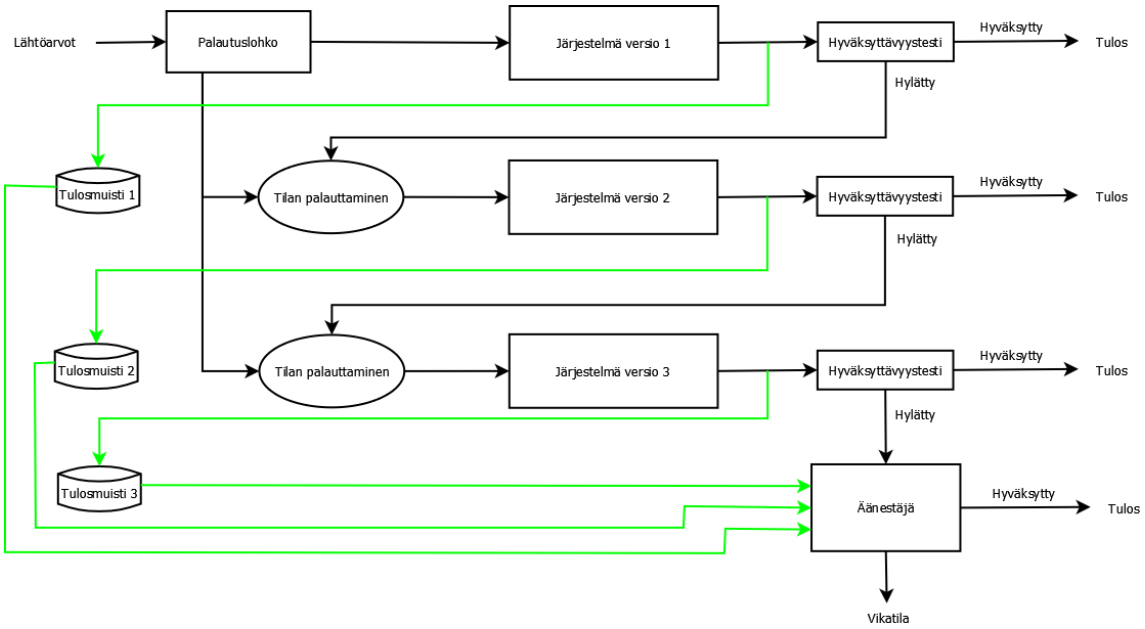
Palautuslohko-malli toteutetaan järjestelmän tilan tallentavalla lohkolla, jonka avulla järjestelmä voidaan palauttaa tiettyyn lähtötilaan kutakin ohjelmistoversiota varten. Palautuslohko tallentaa järjestelmän lähtöarvot ja tilan siten, että lähtötilanne on identtinen kaikille ohjelmistoversioille. Hyväksymistestilohko täytyy suunnitella laadukkaasti. Sen tulisi olla yksinkertainen, tehokas ja luotettava ([10], s. 119).

Koska ohjelmistoversiot suoritetaan peräkkäin, suoritusaika voi pidentyä. Ensimmäisen hyväksymistestin onnistuessa järjestelmä tuottaa ulostulon heti sen valmistuttua. Muussa tapauksessa järjestelmä ei tuota ulostuloa ennen, kuin se pääsee ohjelmistoversioon, jonka

hyväksymistesti onnistuu. Vaihtoehtoisesti järjestelmä ei tuota mitään ulostuloa, jos kaikki hyväksymistestit epäonnistuvat. Suoritusaika on siis riippuvainen toimintavirheiden määrästä. Tästä syystä palautuslohko-malli ei sovellu tilanteisiin, jossa suoritusaika täytyy olla ennalta tiedossa.

7.3 Äänestämisen lisääminen palautuslohkoon

Liian suppea, huonosti suunniteltu tai virheellinen hyväksyttävyystesti voi johtaa tilanteisiin, jossa järjestelmälohkon oikeaa tulosta ei jostain syystä hyväksytä. Tällaiset väärät negatiiviset tulokset voivat johtaa huonossa tapauksessa jopa koko järjestelmän vikatilaan, vaikka osa tai kaikki järjestelmäversioista toimisikin oikein. Tässä tapauksessa palautuslohkomalliin voidaan liittää varalle äänestäjälohko, joka valitsee järjestelmälohkojen hylätyistä tuloksista enemmistön mukaisen tuloksen. Jos enemmistöä ei voida päättää, ajetaan järjestelmä vikatilaan. Tulosten tallentaminen vaatii hieman lisää resursseja mutta on todennäköisesti toteutettavissa samalla laitteistolla, kuin normaali palautuslohkoon perustuva järjestelmä [21]. Tällainen suunnittelumalli on esitetty kuvassa 5. Äänestäjän lisääminen palautuslohkoon ei paranna järjestelmän luotettavuutta väärän negatiivisen tuloksen osalta. Jos jokin hyväksymistesti päästää virheellisen tuloksen läpi, ei varaäänestäjää koskaan käytetä.



Kuva 5. Palautuslohko varalla olevalla äänestäjällä. Mukaillen ([21], kuva 1). Äänestäjän vaatimat osat lisätty vihreällä.

8 Toimintakyvyn säilyttäminen vikatilanteessa

8.1 Sulava alasajo

Monien turvallisuuskriittisten järjestelmien vaatimuksena on, että järjestelmän kohdatessa vikatila, se pysyy jossain määrin saatavilla. Tällaisista järjestelmistä käytetään termiä vikaantuessa toimintakykyinen. Toimintakyvyn säilyttäminen vikatilanteessa vaatii suunnittelumallin, jossa toiminnallisuutta on replikoitu useaan järjestelmälohkoon. Suunnittelumallit voidaan jakaa niin kutsuttuihin metasuunnittelumalleihin, jotka ottavat huomioon toimintakyvyn säilymisen vikatilanteen sattuessa [22]. Toimintakyvyn voi säilyttää kokonaan järjestelmien redundanssilla tai osittain sulavalla alasajolla (eng. graceful degradation), jossa järjestelmän vikaantunut osa ajetaan turvallisesti alas ja toimintaa jatketaan jäljelle jääneellä kapasiteetilla.

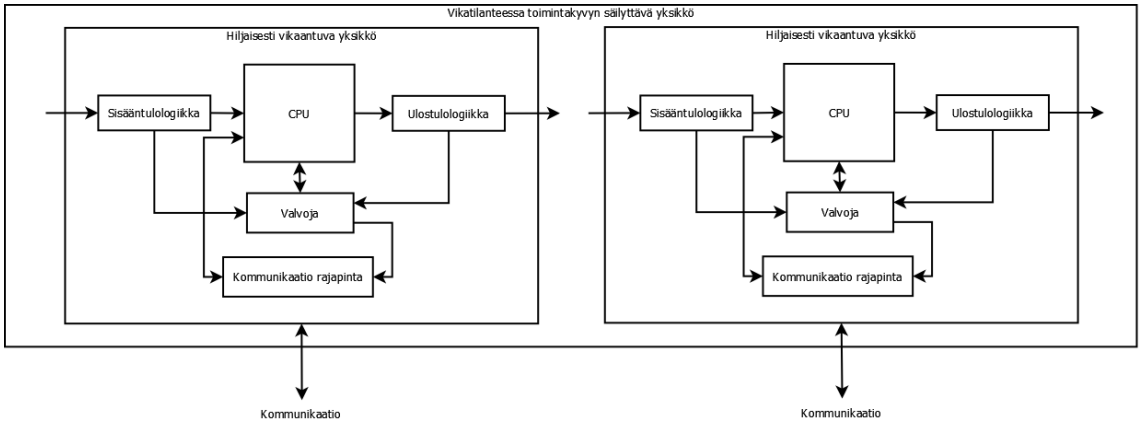
Adaptiivinen toimintakyvyn säilyttävä redundanssimalli (eng. Adaptive Fail-Operational Redundancy pattern, AFOR) on NVP-malliin perustuva metamalli, jossa järjestelmästä on toteutettu N-versiota heterogeenisiä tai homogeenisiä järjestelmäversioita ja äänestäjä. Järjestelmäversiolle on annettu lepotilaluokittelu, jonka perusteella niitä otetaan käyttöön vikatilanteessa. AFOR mallia voidaan hyödyntää esimerkiksi autojen lukkiutumattomien jarrujen ohjausjärjestelmissä [22].

Sulavasti alas ajettava toimintakyvyn säilyttävä suunnittelumalli (eng. Fail-operational Graceful Degradation pattern, FOGD) on suunnittelumalli, jossa järjestelmän toiminta on jaettu usealle tasolle. Tasojen kasvaessa toiminnallisuus ja ohjelmiston monimutkaisuus kasvaa. Jos jollain tietyllä tasolla tulee toimintavirhe, järjestelmä asetetaan alemmalle toiminnan tasolle tietyssä ajassa. Toinen käytännön sovellus sulavalle alasajolle voidaan toteuttaa nykyaikaisten autojen aktiivisissa vakionopeudensäätimissä [22]. Aktiivinen vakionopeudensäädin säätää auton nopeutta siten, että se pitää tietyn kuljettajan asettaman tavoitenopeuden ja säilyttää turvallisen matkan edellä ajavaan autoon. Välimatka mitataan auton etupuolelle sijoitetulla tutkalla. Jos tutkaan tulee toimintahäiriö, aktiivinen vakionopeudensäädin ajetaan sulavasti alas, kuljettajaa huomautetaan toimintahäiriöstä ja vakionopeuden säätämistä jatketaan perinteisellä vakionopeudensäätimellä. Sulavaa alasajoa suositellaan SIL1 ja SIL2 turvallisuustasoilla ja suositellaan vahvasti SIL3 ja SIL4 tasoilla (Taulukko 4).

8.2 Hiljainen vikaantuminen

Turvallisuuskriittisen järjestelmän on tietyissä sovelluksissa hallittava vikatilanne tai järjestelmän toiminnan keskeytyminen ilman järjestelmän toiminnan ja saatavuuden alene-

mista [23]. Tällaisissa tilanteissa voidaan hyödyntää hiljaisesti vikaantuvia (eng. fail-silent) järjestelmälohkoja. Esimerkki kahdesta hiljaisesti vikaantuvasta järjestelmälohkosta on esitetty kuvassa 6. Hiljaisesti vikaantuva järjestelmä tuottaa oikeita tulosarvoja, ei tulosarvoja lainkaan tai selvästi virheellisiä tulosarvoja. Tällainen järjestelmälohko ei yksinään voi kuitenkaan tuottaa vikatilanteessa haluttua saatavuutta vaan niitä on oltava replikoituna kaksi tai useampi. Luotettavuuden lisäämiseksi on tärkeää, että kukin hiljaisesti vikaantuva lohko on kytketty omaan itsenäiseen virtalähteeseen ja niiden välinen kommunikointikanava on kahdennettu. Ajatuksena hiljaisesti vikaantuvassa toimintakyvyn säilyttävässä suunnittelumallissa on, että turvallisuuskriittinen järjestelmä on replikoitu useaan hiljaisesti vikaantuvaan lohkoon, jotka voivat kompensoida toistensa saatavuutta. Hiljaisesti vikaantuvaa toimintakyvyn säilyttävää järjestelmä voidaan käyttää esimerkiksi autoteollisuuden tietokoneohjatussa jarrutusjärjestelmässä [23]. Tässä järjestelmässä jarrupoljin data kerätään TMR-mallisella kolmen äänestäjän logiikalla ja tuloksen äänestyksen jälkeen, se syötetään kahdesta hiljaisesti vikaantuvasta järjestelmälohkosta koostuvaan toimintakyvyn säilyttävään lohkoon. Näin saavutetaan järjestelmä, jonka saatavuus ei ole kiinni mistään yhdestä vikaantumisesta.



Kuva 6. Toimintakyvyn säilyttävä hiljaisesti vikaantuva yksikkö. Mukaillen ([23], kuva 2).

8.3 Dynaaminen uudelleenkonfigurointi

Toimintakyvyn voi säilyttää vikatilanteessa myös hyödyntämällä uudentyyppistä mallia, jossa yhden järjestelmän toiminnallisuus voidaan vikatilanteessa korvata täysin [24]. Tässä arkkitehtuurissa hyödynnetään uudelleen ohjelmoitavaa varajärjestelmää, jolla voidaan korvata hajonneen järjestelmän toiminta. Esityksessä on käytetty esimerkkiä, jossa auton jarruja ja ohjausta ohjataan molempia omalla ohjausjärjestelmällä. Lisäksi autossa on kolmas vastaava järjestelmä, joka voidaan tarvittaessa ohjelmoida uudelleen, täyttämään kumman tahansa pääjärjestelmän toiminnallisuus. Varajärjestelmä on kytketty sekä ohjauksen, että jarrujen kommunikointikanaviin katkaisijoilla, jotka varmistavat, että varajärjestelmä ei ole turhaan kytkettynä tai, se ei ole kytkettynä molempiin järjestelmiin

yhtä aikaa. kumman tahansa pääjärjestelmän vikatilanteen sattuessa, varajärjestelmä ohjelmoidaan ja kytketään korvaamaan vikaantunut järjestelmä ISO 26262 ASIL D määräämässä ajassa. Tässä suunnittelumallissa rajoitteena on, että yksi varajärjestelmä ei kykene kattamaan tilannetta, jossa molemmat pääjärjestelmät vikaantuvat. Lisäksi ei ole määritetty, miten järjestelmän vikaantuminen havaitaan eikä, miten varmistetaan, että molemmat järjestelmät eivät vikaannu yhtä aikaa samasta syystä esimerkiksi virransyötön lakatessa [24]. Dynaamista uudelleenkonfigurointia ei kuitenkaan suositella IEC 61508 standardissa SIL2-4 tasoille (Taulukko 4).

9 Yhteenveto

Turvallisuuskriittistä järjestelmää suunniteltaessa voidaan hyödyntää tunnettuja suunnittelumalleja vikaantumisen riksin pienentämiseksi. Nykyiset standardit vaativat tiettyjä laitteiston ja ohjelmiston ominaisuuksia, jotka voidaan myös täyttää näillä tunnetuilla suunnittelumalleilla. Näin vähennetään järjestelmän suunnittelun monimutkaisuutta ja kustannusta. Lisäksi voidaan välttää ylimääräisiä ongelmatilanteita ja haasteita, koska suunnittelumallit on kehitetty nimenomaan olemassa olleiden yleisien ongelmien ratkaisuiksi.

Pääteemana kaikissa turvallisuuskriittisien järjestelmien suunnittelumalleissa on turvallisuuden vastuun jakamien erilliseen järjestelmälohkoon. Tällä pyritään välttämään yhden järjestelmän suunnitteluvirheestä aiheutuvaa vikaantumista ja vaaraa. Turvallisuutta voidaan toteuttaa järjestelmää valvomalla ja vikatilanteessa turvafunktioita suorittaen. Pelkkä järjestelmän tilan valvominen ei kuitenkaan paranna järjestelmän saatavuutta. Vikatilanteessa, jossa järjestelmän toiminta täytyy lopettaa, ei mitään toiminnallisuutta jää jäljelle. Turvallisuuskriittisten järjestelmien standardi IEC 61508 suosittelee valvojia kaikille kriittisyystasoille (Taulukko 5).

Järjestelmästä voidaan toteuttaa monta itsenäistä versiota ja niiden tuloksia voidaan tutkia äänestämällä, hyväksyttävyystestillä tai näiden yhdistelmällä. Äänestämisen etuna on se, että järjestelmän luotettavuutta ja saatavuutta voidaan parantaa yhtä aikaa ilman, että vaikutetaan negatiivisesti järjestelmän suorituskykyyn. Vaikka yksi versio vikaantuisikin, ei järjestelmän toiminta keskeydy. Järjestelmää voidaan myös palauttaa lähtötilaan ja ajaa eri järjestelmäversioita peräkkäin, jolloin laitteistovaatimukset eivät kasva mutta suorituskyky heikkenee. N-versio-ohjelmoinnilla voidaan myös parantaa vikatilanteiden tunnistettavuutta ja järjestelmän diagnosoinnin helppoutta [3].

Suunnittelumalleja voidaan hyödyntää myös vikatilan hallinnassa jakamalla järjestelmän toimintaa siten, että kaikkea toiminnallisuutta ei ole toteutettu samassa lohkossa. Vikaantuneita osia voidaan poistaa käytöstä siten, että loput järjestelmästä pysyy saatavilla. Järjestelmää voidaan myös kahdentaa siten, että toisen järjestelmän vikaantuessa, toinen voi korvata sen toiminnan. Järjestelmän toiminnan voi myös korvata passiivisena olevalla varajärjestelmällä dynaamisesti vikatilanteen sattuessa.

10 Lähdeluettelo

[1] I. Sommerville. [Online]. Available: http://iansommerville.com/software-engineering-book/web/critical-systems/. [Haettu 29 11 2021].

[2] I. Sommerville, Software Engineering, Boston : Pearson, 2016.

[3] *SFS-EN 61508-3:2010 Sähköisten/elektronisten/ohjelmoitavien elektronisten turvallisuuteen liittyvien järjestelmien toiminnallinen turvallisuus. Osa 3: Ohjelmistovaatimukset.*

[4] *SFS-ISO 26262-5:2019:en Road vehicles — Functional safety — Part 5: Product development at the hardware level.*

[5] C. Hobbs, Embedded Software Development for Safety-Critical Systems. First edition, Boca Raton: FL: CRC Press, 2017.

[6] D. Cotroneo, Innovative Technologies for Dependable OTS-Based Critical Systems, Springer, 2013.

[7] *SFS-EN 61508-1:2010 Sähköisten/elektronisten/ohjelmoitavien elektronisten turvallisuuteen liittyvien järjestelmien toiminnallinen turvallisuus. Osa 1: Yleiset vaatimukset.*

[8] J. Vlissides, E. Gamma, R. Helm ja R. Johnson, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[9] R. Damaševičius, G. Majauskas ja V. Štuikys, ”Application of design patterns for hardware design,” tekijä: *Proceedings of the 40th annual Design Automation Conference*, 2003.

[10] A. Armoush, Design Patterns for Safety-Critical Embedded Systems, Aachen, Germany: RWTH Aachen University, 2010.

[11] D. J. Smith ja K. G. Simpson, The Safety Critical Systems Handbook : a Straightforward Guide to Functional Safety: IEC 61508 (2010 Edition), IEC 61511 (2016 Edition) & Related Guidance, Including Machinery and Other Industrial Sectors . Fourth edition., Amsterdam: Netherlands: Butterworth-Heinemann, 2016.

[12] *SFS-EN 61508-7:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 7: Overview of techniques and measures.*

[13] *SFS-ISO 26262-1:2019:en Road vehicles — Functional safety — Part 1: Vocabulary.*

[14] *SFS-ISO 26262-2:2019:en Road vehicles — Functional safety — Part 2: Management of functional safety.*

[15] W. Wu ja T. Kelly, ”Safety Tactics for Software Architecture Design,” tekijä: *28th Annual International Computer Software and Applications Conference*, 2004.

[16] B. P. Douglass, Real-Time Design Patterns : Robust Scalable Architecture for Real-Time Systems, Boston: Addison-Wesley, 2003.

[17] A. Avizienis, ”The N-Version Approach to Fault-Tolerant Software,” tekijä: *IEEE Transactions on software engineering, VOL. SE-I 1, NO. 12,*, 1985.

[18] S. Hudson, R. S. S. Sundar ja S. Koppu, ”Fault Control Using Triple Modular Redundancy (TMR),” tekijä: *Progress in Computing, Analytics and Networking*, 2018, pp. 471 - 480.

[19] G. Latif-Shabgahi ja J. M. Bass, ”A Taxonomy for Software Voting Algorithms Used in Safety-Critical Systems.,” tekijä: *IEEE transactions on reliability*, 2004.

[20] M. Idirin, X. Aizpurua, A. Villaro, J. Legarda ja J. Melendez, ”Implementation Details and Safety Analysis of a Microcontroller-Based SIL-4 Software Voter,” tekijä: *IEEE transactions on industrial electronics 58.3 (2011)*, 2011.

[21] A. Armoush, F. Salewski ja S. Kowalewski, ”Recovery Block with Backup Voting: A New Pattern with Extended Representation for Safety Critical Embedded Systems,” tekijä: *2008 International Conference on Information Technology*, 2008.

[22] D. Penha, G. Weiss ja A. Stante, ”Pattern-Based Approach for Designing Fail-operational Safety-Critical Embedded Systems,” tekijä: *13th International Conference on Embedded and Ubiquitous Computing*, Munich, 2015.

[23] P. Sinha, ”Architectural Design and Reliability Analysis of a Fail-Operational Brake-by-Wire System from ISO 26262 Perspectives.,” tekijä: *Reliability engineering & system safety 96.10*, 2011.

[24] F. Oszwald, J. Becker, P. Obergfell ja M. Traub, ”Dynamic Reconfiguration for Real-Time Automotive Embedded Systems in Fail-Operational Context.,” tekijä: *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2018.