IKER GAZTE
NAZIOARTEKO IKERKETA EUSKARAZ

# III. IKERGAZTE
NAZIOARTEKO IKERKETA EUSKARAZ

2019ko maiatzaren 27, 28 eta 29
Baiona, Euskal Herria

ANTOLATZAILEA:
Udako Euskal Unibertsitatea (UEU)

**INGENIARITZA ETA ARKITEKTURA**

## Exekuzio Denboran barne egoera ikusi eta aldatzea ahalbideratzen duten UML Egoera Makinak: Models@run.time

*Miren Illarramendi, Leire Etxeberria, Xabier Elkorobarrutia*
eta *Goiuria Sagardui*

34-41 or.
https://dx.doi.org/10.26876/ikergazte.iii.03.04

# Exekuzio Denboran barne egoera ikusi eta aldatzea ahalbideratzen duten UML Egoera Makinak: Models@run.time

Illarramendi, Miren, Etxeberria, Leire; Elkorobarrutia, Xabier eta Sagardui Goiuria

*Mondragon Goi Eskola Politeknikoa S.Coop. Software eta Sistemen Ingenieritza Arrasate*

*millarramendi@mondragon.edu*

***Laburpena***

Egungo ingurune industrialetan, gailu desberdinak kontrolatzen dituzten sistema txertatuen softwareak geroz eta konplexuago, sendoago eta fidagarriago izan beharra dauka. Egoera honi aurre egiteko moduetako bat kontroleko software horren portaera deskribatzen duen eredua, exekuzio denboran eskuragarri egotea da (models@run.time). Artikulu honetan, kontrol sistemak eredu bidez deskribatu ondoren, eredu horietan oinarrituaz software osagaiak modu automatiko batean sortuko dituen plataforma bat aurkeztuko da. Bere balio erantsi nagusia, software osagai hauek exekuzio denboran euren barne informazioa eskaintzeko gaitasuna da, berauek deskribatzeko erabili den eredu lenguaia berbera erabiliaz gainera. Horretaz gain, aurreikusi gabea den edo errore egoera baten aurrean, software osagai hauek eraldatzeko gaitasuna ere izango dute. Eraldatze hau, aurrez definitua izan den portaera seguru batera izango da.

Hitz gakoak: Models@runtime, Exekuzio denborako eraldatzea, Sistema Txertatuak , UML Egoera Makinak.

***Abstract***

*In current industrial environments, the software of embedded systems have to cope with the increasing complexity and robustness requirements at runtime. One way to manage these requirements is having the software component's behaviour model available at runtime (models@run.time). In this paper, we present a model-driven approach to generate software components which are able both to provide their internal information in model terms at runtime and adapt their behaviour automatically when an error or an unexpected situation is detected. Thanks to this introspection ability at runtime, the software components are able to adapt automatically from their normal-mode behaviour to a safe-mode behaviour which was defined to be used in erroneous or unexpected situations at runtime.*

*Keywords: Models@runtime, Runtime Adaptation, Embedded Systems, UML State Machines.*

## 1. Sarrera eta motibazioa

Sistema Ziberfisikoak, prozesu fisikoak eta konputazio, kontrol zein komunikazio digitalak bateratzen dituzten sistemak dira. Sistema hauek, Derler *et al.* (2012)-ek azaltzen duen moduan, sistema txertatuez eta sareez osatuta daude eta sentsore eta aktuadore bidez prozesu fisikoak monitorizatu eta kontrolatzen dituzte.

Sistema Ziberfisikoen konplexutasunak eta presentziak ikagaragarrizko gorakada izan du azken urteetan. Egunero gure eskura ditugun gailu gehienetan aurkitu genitzake. Beraz, eurek izan ditzaketen akatsek ondorio benetan larriak ekarri ahal ditzakete. Sistema Ziberfisikoen garapenek, maiz, ebentuetan oinarritutako arkitektura patroia jarraitzen dute. Sistema hauen logika modelatzeko berriz, askotan, Unified Modeling Language (UML)-an oinarritutako Egoera Makinen formalismoa erabiltzen da. Gainera, Model Driven Engineering (MDE) paradigma jarraituaz, software konponenteen azken kodea modu automatiko baten sortzeko gaitasuna ere badaukagu horretarako ditugun erremintak erabiliaz.

Hala ere, azken kode hori automatikoki sortzeak ez dizkigu arazo guztiak kenduko. Exekuzio denboran ere, software osagaien portaera egokia dela ziurtatu beharra dugu. Models@run.time planteamenduak, MDE bidez sorturiko modeloak exekuzio denboraren ingurunetan ere erabiltzea bilatzen du. Exekuzio denboran software osagaien eredu edo modeloa eskuragarri izatea exekuzio denboran egiaztapenak egin ahal izateko lehenengo pausoa

da; eta hurrengo pausua, behin errore edo aurreikusi gabeko egoera bat antzematean, software osagai horren modeloaren eguneraketa bat egitea. Honela, sistemaren fidagarritasun maila handitu egiten da.

Sistemen fidagarritasuna handitzea helburu hartuta, lan honetan, UML- Egoera Makinetan oinarrituaz eta models@run.time planteamentua jarraituaz, sistema ziberfisikoak kontrolatuko dituzten software osagaien kodea automatikoki sortzeko gai den RESCO (REflective State-Machines based observable software COmponents) izeneko plataforma bat aurkeztuko dugu. Osagai hauek, introspekzio eta eraldatze gaitasuna izango dute exekuzio denboran. Honela, egoera desegoki batez jabetzean edo aurreikusi gabeko egoera baten aurrean, sistema eraldatu egingo da egoera seguru batera. Hona hemen, lanaren ekarpen nagusiak:

1. Exekuzio denboran behatzeko, egiaztatzeko eta eraldatzeko gaitasuna duten software osagaiak sortzeko plataforma eta berauek egiaztatzeko eta eraldatzeko gai den kanpo egiaztatzaile sistema,
2. Software garatzaileari eskainitako laguntza: berak ez dauka azken kodea eskuz ikutu beharrik, portaeraren modeloa diseinatu besterik ez du egin behar UML Egoera Makinak erabiliaz. Exekuzio denboran introspekzio eta eraldatze gaitasuna izateko beharrezko azpiegiturak, automatikoki sortzen dira.

## 2. Arloko egoera eta ikerketaren helburuak

Exekuzio denboran egiaztapenak egiteko modu bat, software kontroladoreak berak bidaltzen dituen informazio trazak behatzea da. Traza zuzenen kopuruak finitua izan behar du eta exekuzio denboran egiaztapenak egiten ari den sistemak ezagutu behar ditu. Sistema honek, traza zuzen bezala definiturik ez dagoen bat jasotzean, traza urratze egoeran sartzen da eta eraldatze prozesua hasieratzen du.

Kanpo egiaztapen sistemak software osagaia monitorizatu ahal izateko exekuzio denborako trazak behar dituenez, software osagaien kodea instrumentatu beharra daukagu. Horretarako, aukera desberdinak ditugu:

- Iturburu Kodea instrumentatu: teknika hau erabiliaz, iturburu kodea bera aldatzen da behar den lekuan kodea erantsiaz (konpilazio garaian aktibatu edo desaktibatzeko aukera gehituaz).
- Modelo edo Eredua instrumentatu: teknika hau erabiliaz, exekuzio garaian behatzea nahi ditugun elementuak espezifikatzen dira modeloan. Ondoren, kodea automatikoki sortzean, trazak sortzeko beharrezko kodea ere automatikoki sortuko da.

Gure soluzioan, bigarren aukera hau erabili dugu.

Lan honen bidez detektatu ahal izango ditugun akats eta erroreak honako hauek dira:

- ausazko hardware akatsak: bit inbertsioak, errore aldakorrak,...
- ausazko software akatsak: heisenbug-ak,...
- gainerako software akatsak: garapen fasean, balidazio eta egiaztapen prozesuak pasatu ondoren, ezkutuan geratzen diren erroreak,
- aurreikusi gabeko ingurumen akatsak: diseinu eta garapen prozesuan aintzat hartu ez direnak.

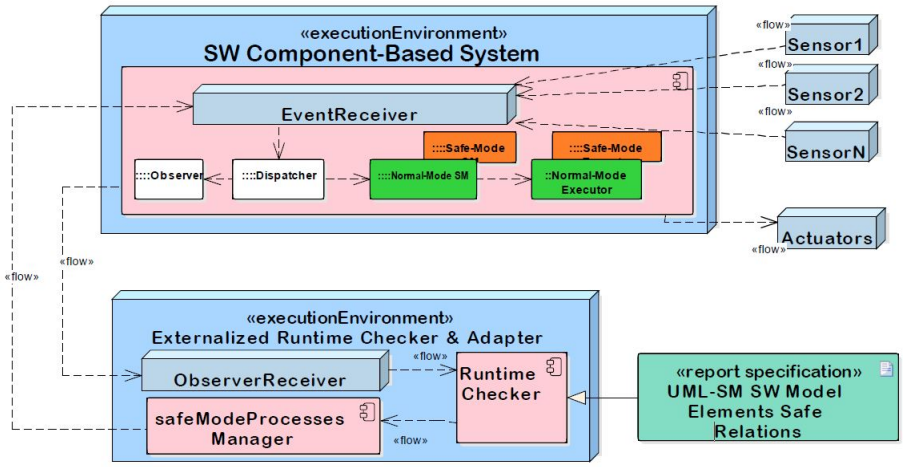
Exekuzio denboran egiaztapenak egiteko modu desberdinak daude. Hauetako bat, software kontrolaren eredu edo modeloko elementuak erabiliaz da (momentuko egoera, ebentua, hurrengo egoera,...). Honen abantaila, diseinu eta garapen faseetan eta exekuzio denboran lenguaia berbera erabiltzea da.

Modelua exekuzio denboran mantendu ahal izateko, software osagaiak behatua izateko gaitasuna eskeini behar du. Honetarako, software osagaiak introspekzio ezaugarria izan behar du. Introspekzioak, exekutatzen ari den programaren monitorizazioa egitea ahalbideratzen du eta honen helburua exekuzio denboran erroreak identifikatu, lokalizatu eta analizatzea izango da James *et al.* (2010)-ek dioen moduan.

Eraldatzeari dagokionez, Gomaa *et al.* (2017)-an esaten duten bezala, bi modu nagusi daude softwarea dinamikoki eraldatu ahal izateko: planifikatua eta ez planifikatua. Gure soluzioa, planifikatu gabekoen artean kokatzen da. Eraldatze mota hau, espero ez den ebentu edo errore egoera batetik abiarazten da. Ez da aurrez momentu jakin batean egitea pentsatu den eraldaketa bat.

Garlan eta Schmerl (2002)-en lanean bezala, gure soluzioan ere egiaztapen eta eraldatze sistema kanpoan dago. Hori dela eta, behatutako software osagaiak informazio trazak bidali behar ditu bertara. Kanpo sistema hau izango

**1. irudia. Soluzioaren Arkitektura: Kontrolatzailea eta Kanpo Egiaztatzaile/Eraldatzailea**




da akats, errore edo aurreikusi gabeko egoerak detektatuko dituena eta dagokion kasuan eraldatze prozesua martxan jarriko duena. 1. irudiak, soluzioaren arkitektura erakusten digu.

## 3. RESCO: Introspekzio eta Eraldatze gaitasuna duten Software osagaiak sortzea

Atal honetan, introspekzio eta eraldatze gaitasuna duten software osagaiak sortzeko planteamentua aurkeztuko dugu. Horretarako, UML Egoera Makinak diseinatzeko lengoaian oinarritzen da gure soluzioa eta models@runtime lan ildoa jarraitzen du. Behin metodologia aurkeztuta, egindako lanaren balidazio experimentala azalduko da.

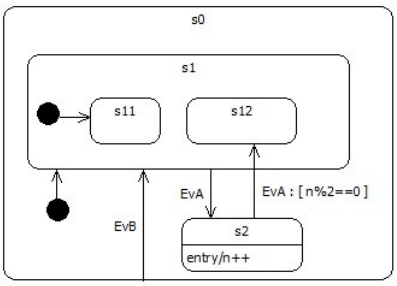
### 3.1. RESCO software osagaiak: sortze prozesua eta ezaugarriak

Prozesua modu errazago batean ulertu asmoz, 2. irudian agertzen den adibidea hartuko dugu. Egoera honen irudikapena, RESCO-ren baitan, 3. irudian agertzen da. Egoerak, zuhaitz formako objektu egitura baten bidez adierazten dira eta egitura honek, software kontrolaren egoerak isladatzen ditu. Egoera bakoitzak, bere portaeraren espezifikazio darama atxekiturik. Beraz, objektu egitura hau aldatzen badugu, benetan software konponente edo kontrolaren modeloa bera aldatzen gabiltza.

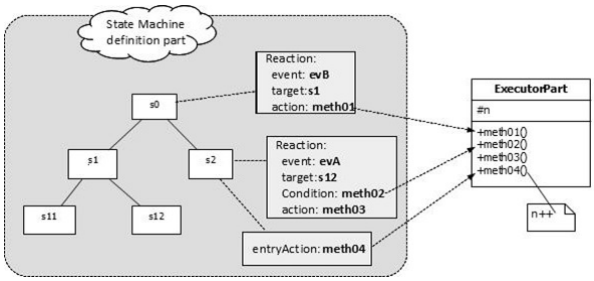
Egoera makina horretatik abiatuta, ondorengo puntuetan software osagaiak automatikoki sortzeko metodoa eta pausuak azalduko dira. Laburbilduz, 4. irudian agertzen da prozesu hau.

Lehenengo, kontrolaren portaera diseinatzen da UML Egoera Makinak erabiliaz. Honetarako, Papyrus erreminta erabili da eta behatuak izango diren egoerak anotatu beharko dira. Eraldaketa posible izan dadin, portaera normala eta larrialdietakoa (safe-mode) diseinatuko dira. Erroreren bat edo uste gabeko egoeraren bat detektatzean, larrialdietako egoerara pasatuko da sistema automatikoki.

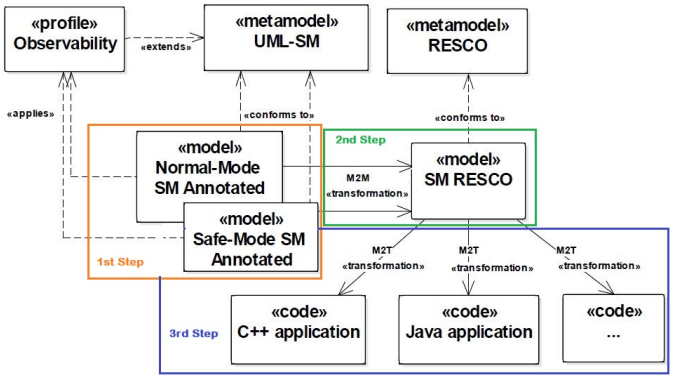
**2. irudia. Egoera makina, adibidea**



**3. irudia. Egoera makinaren irudikapena RESCO plataforman**




**4. irudia. Eredu/Modelo-etan oinarritutako workflow-a**



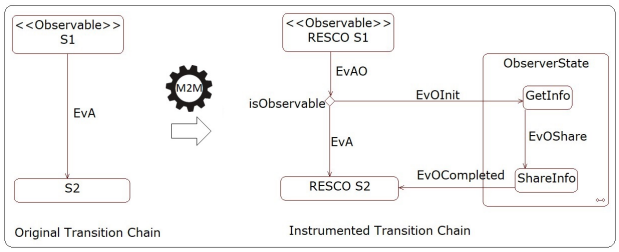

Bigarren pausuan, RESCO-ri dagokion modeloa sortuko da automatikoki. Modelo hau, instrumentatua izango da eta ATL erreminta erabiliaz model-to-model (M2M) transformazioa egingo da. Azken pausuan, RESCO-ri dagokion modeloa iturburu kodera transformatua izango da Acceleo erreminta erabiliaz. Azken honek, model-to-text (M2T) transformazioa egiten du.

Gure soluzioa, instrumentazioa modelo mailan egitea denez, plataformarekiko erabat independientea da. Bagherzadeh *et al.* (2017)-ek aurkezturiko lanean bezala, guk ere M2M transformazioa erabiltzen dugu introspekzioa eta eraldatzea posible izan dadin modeloa instrumentatuaz. Gure planteamentua formalizatu ahal izateko, trantsizioetan gertatzen diren akzio eta baldintzetako konputazioak bakarrik izan ditugu kontutan.

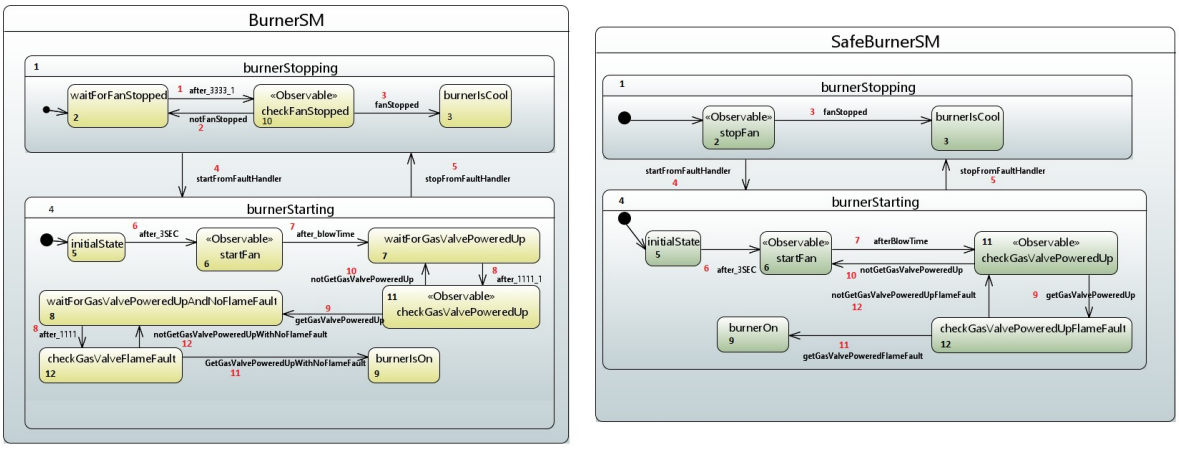
5. irudiak trantsizioak nola gauzatzen diren adierazten du. Ezker aldean, software osagaia *S1* egoeran egonik, *EvA* ebentua gertatzen deneko jatorrizko trantsizioa agertzen da. Eskuinaldean aldiz, modeloa instrumentatu ondorengo baliokidea den trantsizioa agertzen da. Bigarrenean, aukera puntu bat gehitu da eta exekuzio garaian jasotako trazaren arabera hartzen den erabakiaren ondoren aukera horretako bide bat edo beste bat jarraituko du.

Kanpo egiaztatzaile eta eraldatze sistemak, errore edo aurreikusi gabeko egoera baten aurrean, eraldatze sistema martxan jartzen du. Horretarako, safeAdapt ebentua bidaliko dio kaltetutako software osagaiari. Behin software osagaiak ebentu hori jasotzean, larrialdi egoerarako diseinatua duen egoera makinara eraldatzeko prozesua martxan jarriko du eta bere portaera aldatua izango da.

**5. irudia. Modeloaren Instrumentazioa: Transformazio Erregela.**



**6. irudia. Erregailuaren egoera normala eta larrialdiko egoera adierazteko UML Egoera Makinak**




## 3.2. Ebaluazio Enpirikoa

Lana ebaluatzeko erabili den erabilpen kasua, Whispergen (Pradip (2016)) izeneko gailu komertzialaren kontroleko softwarean oinarritzen da. Bertatik, erregailuaren software osagaia hartu dugu eta beronekin egin ditugu aprobak.

Beraz, 6. irudian agertzen den Erregailuaren kontrola gauzatu dugu RESCO plataforma erabiliaz, eta konparazio batzuk egin ahal izateko SinelaboreRT 3.7.2.2 (Mueller (2018)) erreminta (denbora errealeko sistemak garatzeko erreminta), eta Sparx Systems Enterprise Architecture (EA) 11 (Systems (2015)) erreminta generikoa ere erabili dugu. Aukeratutako Erregailu kontrolaren portaera normala modelatu ahal izateko, 13 egoera sinple, 2 egoera konposatu, 13 trantsizio eta 13 ebentu dituen UML Egoera Makina diseinatu dugu. Larrialdi kasuetarako berriz, 7 egoera sinple, 2 egoera konposatu, 9 trantsizio eta 9 ebentu dituena.

Egindako ebaluazioaren helburu nagusiak ondorengoak izan dira: (1) exekuzio denboran software kontrolaren barne egoera ikusi ahal izatea egoera makinaren elementu bidez, (2) RESCO plataformarekin sortutako software konponenteek exekuzio denboran duten introspekzio eta eraldatze gaitasunak egiaztatzea eta (3) soluzioaren gainkarga neurtzea. Honela, sistema ziberfisikoen kontrolen fidagarritasun maila gehituko duten software konponenteak sortzeko gai garela egiaztatuko dugu. Hemen horretarako definitu ditugun Ikerketa Galderak (IG):

**IG1.** Posible al da software osagaiaren modeloa ateratzea exekuzio denboran jasotako trazak analizatuaz?

**IG2.** Erabili al genezake exekuzio denborako informazio hau egiaztapenak eta eraldaketak egiteko?

**IG3.** RESCO plataformarekin sortutako software osagaien errendimendua beste erreminta komertzial batzuekin sortutakoen adinako ona al da (EA v11 eta SinelaboreRT v3.7.2.2)?

Hiru galdera hauei erantzuna emateko, 10 experimentu definitu ditugu. Emaitza fidagarriak izateko, experimentu bakoitza 1000 aldiz errepikatu dugu. 1. taulak, experimentu bakoitzaren ezaugarriak erakusten dizkigu. SM1 egoera makina jatorrizko Erregailu kontrola da. SM2-SM7 egoera makinak artifizialki sortu ditugu errendimendua hobeto neurtu asmoz. Honela, tamaina eta komplexutasun desberdineko egoera makinak izan ditugu experimentazioan.

Beldjehem (2013) eta Genero *et al.* (2003) lanak kontutan hartuaz, egoera makinen tamaina eta komplexutasuna neurtzeko ondorengo metrikak erabili ditugu: Egoera Sinpleen Kopurua (ESK, tamaina neurtzeko), Egoera Konposatu Kopurua (EKK, tamaina neurtzeko) eta McCaberen Zenbaki Ziklomatikoa (Egituraren Konplexutasuna neurtzeko). Metrika hauek, egoera makinen kasura egokitu dira.

### 3.2.1. Emaitzak

Lehenengo galdera erantzuteko, SM1 egoera makina hasieratu eta 10.000 ausazko ebentu bidali dizkiogu. Kanpo monitorizazio eta eraldatze sistemak, Erregailu kontrolaren trazak (egoera makinaren modelatze elementu bidez osatuak) jaso ditu exekuzio denboran eta informazio guzti hau gorde dugu. Informazio hori aztertuaz, Erregailu kontrolaren portaera atera dugu. Experimentu hau gauzatzeko orduan, egoera guztiak behatuak izan dira exekuzio denboran. Emaitza gisara, lista 1-an exekuzio denboran jasotako traza batzuk ikusi genitzake. Traza hauek

**1. taula. Experimentuak**

| Egoera Makina | Ikerketa Galdera | Behaketa % | ESK | EKK | McCabe | Akats injekzioa |
|---|---|---|---|---|---|---|
| SM1 | IG1, IG3 | 100 | 13 | 2 | 13 | EZ |
| SM1 | IG2 | 100 | 13 | 2 | 13 | BAI |
| SM1 | IG3 | 50 | 13 | 2 | 13 | EZ |
| SM1 | IG3 | 0 | 13 | 2 | 13 | EZ |
| SM2 | IG3 | 0 | 25 | 4 | 26 | EZ |
| SM3 | IG3 | 0 | 49 | 4 | 52 | EZ |
| SM4 | IG3 | 0 | 113 | 9 | 117 | EZ |
| SM5 | IG3 | 0 | 25 | 5 | 26 | EZ |
| SM6 | IG3 | 0 | 49 | 11 | 52 | EZ |
| SM7 | IG3 | 0 | 113 | 26 | 117 | EZ |

interpretatuaz (egoera, ebentu zenbakiak), 6. irudian dagoen jatorrizko egoera makina atera genezake.

```
EvId 4; CurrentState 2;NextState 4; FatherState 1;
EvId 6; CurrentState 5;NextState 6; FatherState 4;
EvId 7; CurrentState 6;NextState 7; FatherState 4;
EvId 8; CurrentState 7;NextState 11;FatherState 4;
```

Listing 1: Exekuzio denboran jasotako traza batzuk

Bigarren galderari dagokionez, kanpo egiaztatze eta eraldatze sistema erabili dugu. Sistema hau, Arcaini *et al.* (2014)-en lanean oinarrituta diseinatu dugu eta honek, momentu horretan exekutatzen dabilen kontrolaren trazak konparatzen ditu aurrez egiaztatze sisteman konfiguratuta daukagun informazioarekin.

Hutsegiteen aurkako detekzio ahalmena neurtzeko, akats injekzio kanpainak egin ditugu iturburu kodea aldatuaz Libfiu (Libfiu) erreminta bidez. Lehenengo experimentazio fasean, ez dugu eraldatze sistema aktibatu, akatsak detektatzeko ahalmena bakarrik neurtu bait dugu. 6 akats injektatu ditugu ausazko hardware, software eta aurreikusi gabeko ingurumen akatsak emulatu ahal izateko. Libfiu liburutegia erabiliaz, akats mota desberdinak emulatu ditugu (ausazko akatsak, probabilitate desberdina dutenak, beti gertatzen diren akatsak, ...). Ondorioz, kanpo egiaztatze sistemak akats guztiak detektatu ditu soluzioaren aplikagarritasuna egiaztatuaz. Lista 2-an, exekuzio denboran detektatutako bi akats agertzen dira:

```
EvId 8; CurrentState 7;NextState 8; FatherState 4;
EvId 8; CurrentState 8;NextState 7; FatherState 4;
```

Listing 2: Exekuzio denborako akatsdun trazak

Experimentuaren bigarren atalean, eraldatze prozesua aktibatu dugu eta sistema birabiarazi. Ondoren, aurreko experimentuko akats berdinak injektatu ditugu baina oraingoan, lehenengo akatsa detektatzean, jatorrizko egoera makina desaktibatu eta larrialditakoa aktibatu da. Beraz, kontrolaren portaera automatikoki aldatu da exekuzio denboran eta kanpo egiaztatze sistemak trazak jasotzen jarraitu du. Behin experimentazioa amaitzean, traza hauek aztertu ditugu eta ez dugu akatsdun trazarik topatu. Aldiz, behin akatsa detektatu ondorengo trazak larrialdiko egoera makinari dagozkionak direla ikusi dugu. Lista 3-ak larrialditako egoera makinaren trazak erakusten dizkigu.

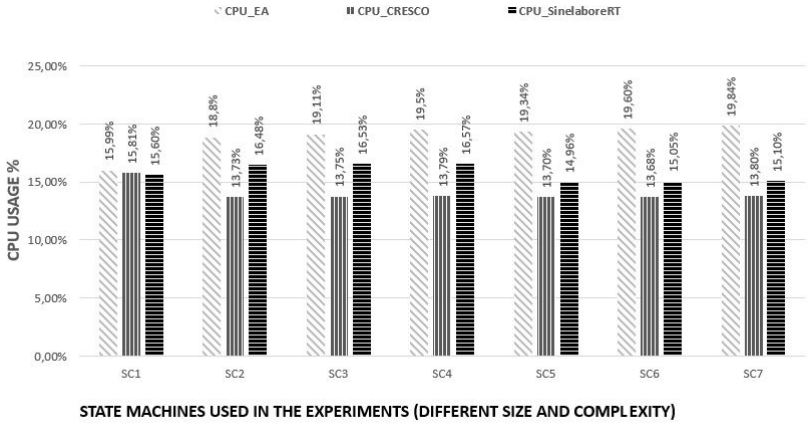
```
EvId 3; CurrentState 2;NextState 3; FatherState 1;
EvId 7; CurrentState 6;NextState 11; FatherState 4;
```

Listing 3: Exekuzio denborako larrialdi egoera makinaren trazak

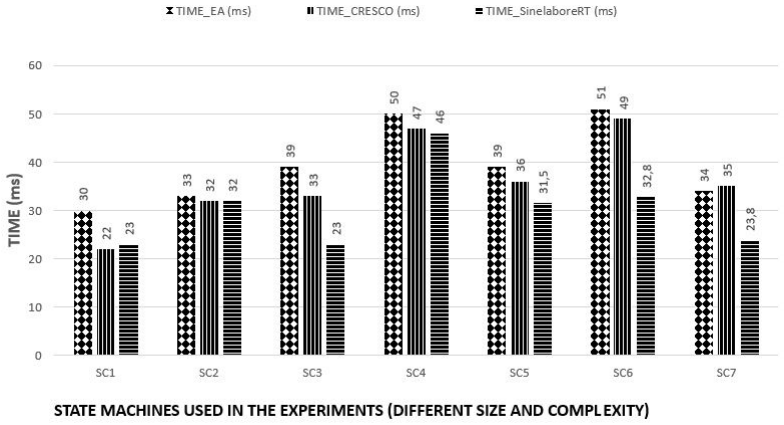
Azkenik, erreminta desberdinekin sortutako software osagaien errendimendua neurtu dugu bai denboran (milisegunduak) eta baita CPU-aren erabilera portzentaian ere.

Lehenengo, RESCO beraren gainkarga neurtu dugu. Horretarako, SM1 egoera makina bera % 0, % 50 eta % 100-eko behaketa portzentaiarekin exekutatu dugu. Behaketa maila % 0 denean, 22 milisegundukoa izan da exekuzio denbora, % 50-ekin 272, eta egoera guztiak behatu ditugunean aldiz 411. Experimentu guztietan 1.000 ebentu erabili ditugu. Beraz, behaketa maila handitzean, errendimenduak nabarmen behera egiten du RESCO-rekin sortutako software osagaietan. Fidagarritasun maila ordea, igo egiten da.

**7. irudia. CPU erabilera SinelaboreRT, EA eta RESCO erremintentzat (behaketa portzentaia % 0).**




**8. irudia. Denborak SinelaboreRT, EA eta RESCO erremintentzat (behaketa portzentaia % 0).**




7 eta 8 irudiek azken IG-ren emaitzak erakusten dizkigute. Experimentu hauetan 1.000 ebentu erabili ditugu eta RESCO-ren kasuan, behaketa portzentaia % 0-ra jarri dugu. Denbora erantzunari dagokionez, SinelaboreRT erreminta izan da erantzunik hoberena izan duena. Hala ere, RESCO-ren emaitzak nahiko antzekoak izan dira, eta pixkat hobeak egoera makinen konplexutasun maila bajua denean. CPU-aren erabilpenari dagokionez, emaitza guztiak nahiko antzekoak dira. Honen arrazoia experimentazio guztiak antzeko egoeran egin izana izan daiteke.

Egoera makina artifizialak kontutan hartuaz (SM2-SM7), RESCO eta EA erreminten kasuan, tamaina eta konplexutasuna handitzean errendimenduak txarrera egiten du eta txikitu egiten da: exekuzio denborak gora egiten du nahiz eta CPU-aren erabilpen portzentaia asko ez aldatu. SinelaboreRT-ren kasuan ere, egoera makinen tamaina eta konplexutasuna handitzean, exekuzio denborak ere gora egiten du, baina kasu honetan aldaketa txikiagoa da. Aipatu beharra dago, azken erreminta hau denbora errealeko sistementzako egina dagoela.

## 4. Ondorioak

Lan honetan, lehenengo, software osagaiak automatikoki sortzeko ereduetan oinarritutako planteamentu bat aurkeztu da. Soluzioa UML Egoera Makinetan oinarritzen da eta sortutako osagaiek euren barne egoera exekuzio denboran emateko ahalmena dute. Gainera, informazio hori, beraiek modelatzeko erabili diren termino berberetan adierazten dute. Guzti honez gain, egoera makina hauek exekuzio denboran eraldatu ere egin daitezke.

Guzti hau egiaztatu ahal izateko, ebaluazio enpiriko bat gauzatu da. Honela, RESCO bidez sortutako software osagaiek introspekzio gaitasuna dutela frogatu da lehenengo. Ondoren, akatsak topatzeko gai den kanpo egiaztatzaile bat ere frogatu dugu. Akatsak topatzeaz gain, hori gertatzean, software osagaia eraldatzeko prozesua martxan jartzeko gai dela ere ikusi dugu. Emaitzek erakutsi digute guzti hau posible dela.

Bukatzeko, sortutako plataformaren errendimendua ere neurtu dugu, baita erreminta komertzial batzurekin (EAv11 eta SinelaboreRT) konparatu ere. Izan kontutan, RESCO bidez sortutako software osagaiak direla exekuzio denboran informazio trazak emateko gaitasuna duten bakarrak, besteek ez diote introspekzio ahalmenik gehitzen sortutako software osagaiei.

## 5. Etorkizunerako planteatzen den norabidea

Etorkizunean, egindako ebaluazioa beste kasu erreal batzuetan egitea ere nahiko genuke. Beste erreminta komertzial batzuekin konparaketak egiten jarraitzea ere bada egin beharreko beste lan bat.

Soluzioak exekuzio denboran eraldaketa egitea ahalbideratzen du. Etorkizunean ordea, eraldaketa hori bera aurreikusi gabe eta dinamikoki exekuzio denboran sortutako egoera makinetara egiteko ahalmena gehituko genioke. Guzti hau kontutan hartuta eta Mazak *et al.* (2016) lanean inspiratuaz, etorkizuneko lan ildo interesgarri bat Process Mining (PM) teknikak integratzea litzateke.

## Erreferentziak


Arcaini, Paolo, Angelo Gargantini, eta Elvinia Riccobene. 2014. Offline model-based testing and runtime monitoring of the sensor voting module. *Communications in Computer and Information Science* .

Bagherzadeh, Mojtaba, Nicolas Hili, eta Juergen Dingel. 2017. Model-level, platform-independent debugging in the context of the model-driven development of real-time systems. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 419–430. ACM.

Beldjehem, Mokhtar. 2013. A granular hierarchical multiview metrics suite for statecharts quality. *Advances in Software Engineering* Volume 2013.13.

Derler, Patricia, Edward A. Lee, eta Alberto Sangiovanni Vincentelli. 2012. Modeling cyber-physical systems. volume Special issue on CPS, 13–28. IEEE.

Garlan, David, eta Bradley Schmerl. 2002. Model-based adaptation for self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, 27–32. ACM.

Genero, Marcela, David Miranda, eta Mario Piattini. 2003. Defining metrics for uml statechart diagrams in a methodological way. In *International Conference on Conceptual Modeling*, 118–128. Springer.

Gomaa, Hassan, Emad Albassam, eta Daniel A Menascé. 2017. Run-time software architectural models for adaptation, recovery and evolution. In *MODELS (Satellite Events)*, 193–200.

James, Mark, Paul Springer, eta Hans Zima. 2010. Adaptive fault tolerance for many-core based space-borne computing. In *European Conference on Parallel Processing*, 260–274. Springer.

Libfiu. Libfiu: faultinjection in userspace.

Mazak, Alexandra, Manuel Wimmer, eta Polina Patsuk-Bösch. 2016. Execution-based model profiling. In *International Symposium on Data-Driven Process Discovery and Analysis*, 37–52. Springer.

Mueller, Peter. 2018. sinelaborert. Technical report.

Pradip, Prashant Kaliram. 2016. Commissioning and performance analysis of whispergen stirling engine. Technical report, University of Windsor.

Systems, Sparx. 2015. Enterprise architecture v11. Technical report, Sparx Systems (http://sparxsystems.com/products/ea/).


## 6. Eskerrak eta oharrak

Proiektu hau MGEP-eko Sistema Txertatuen taldeak garatu du eta Eusko Jaurlaritzaren Hezkuntza, Unibertsitate eta Ikerketa sailak lagundua izan da Ikerketa Taldeak (Grupo de Sistemas Embebidos) eta TEKINTZE (Elkartek 2018) dirulaguntzen bidez. Europako H2020 ikerketa eta innobazio programaren barruan aurkitzen den ECSEL Joint Undertaking, eta Nazio mailako dirulaguntza iturriak ere baliatu dira Productive 4.0 proeiktuan(grant agreement no. GAP-737459 - 999978918) lana eginez.