LABURPENA

Memoria honetan lan-fluxu sistemen inguruko proiektu bat aurkezten da. Lan-fluxuak lan prozesuen automatizazioak dira, erakunde edo enpresa baten egunero egin ohi den lana era automatikoan kudeatzen dutenak.

Nazioarteko Workflow Management Coalition erakundeak, lan-fluxu horiek estandarizatzeko pausoak eman ditu eta estandar horietako bat, XPDL lengoaia erabili dut proiektu honetan. Zope plataforman lan-fluxuak inplementatzeko OpenFlow produkturako bi modulu garatu ditut. Lehenengoan OpenFlow-ren lan-fluxuak eta XPDL dokumentuen arteko bihurketak egiten dira. Bigarren moduluak, aldiz, OpenFlow produkturako lan-fluxuen diseinatzaileek egin ohi dituzten akatsak kontrolatzen ditu.

Lan hau burutzeko, OpenFlow eta XPDLz gain, hainbat tresna eta teknologia aztertu dira, hala nola, Zope web aplikazioen zerbitzaria eta Python programazio lengoaia OpenFlowren erabilerarako edo SAX eta DOM ereduak XPDL dokumentuak tratatzeko. Bestalde lan-fluxuen alderdi teorikoa ere aztertu da, eskaintzen dituzten aukerak zein diren ezagutzeko.

Keywords: Workflow, XPDL, OpenFlow, Zope, Python

Edukien aurkibidea

Irudien indizea

Taulen indizea

1. SARRERA

Bi arrazoi nagusik eraman naute proiektu hau aukeratzera, alde batetik Zope [Zope] plataforma hurbilagotik ezagutu ahal izateak eta beste alde batetik enpresa batean lan egiteak. Proiektu hau egiteko workflow edo lan-fluxuen mundua aukeratzea enpresaren ideia izan zen hein handi batean, bere bezero askok eskatu baitiete lan-fluxuen inguruko soluzioren bat jadanik eskaintzen dizkieten zerbitzuetatik gain.

Proiektua Eibarko CodeSyntax [CS] enpresan eramango da aurrera Garikoitz Araolazaren gidaritzapean, Zope web aplikazioen zerbitzarian oinarritutako webgune dinamikoak egiten ditu enpresa honek eta azkenaldian Debagoienako hainbat udaletako webguneak garatu ditu. Beste alde batetik Lanbide Heziketako zentruen elkargoaren intranet eta web ataria ere garatzen ari da, eta bi bezero horiek eskatuta sartu da lan-fluxuen inplementazioa eta Zope plataformarekiko integrazioa nola egin daitekeen aztertzera.

Lan-fluxu sistemen garrantzia handitzen joan da urtetik urtera, batez ere enpresa eta beste hainbat erakunde ISO kalitate ziurtagiriak lortu nahian dabilelako. Lan-fluxuen merkatua oso zabalik dago eta sistema jabedunak dira merkatu horretan nagusi. Software askearen filosofia ere iritsi da merkatu horretara eta horren erakusgarri memoria honetan garatu den proiektuan erabili den OpenFlow Zope plataformarako sortutako produktu italiarra. Lan-fluxu sistema batek bete behar dituen ezaugarri nagusiak betetzen ditu, eta gainera Zope plataformako gainontzeko modulu eta produktuekin harremana izan dezake, tresna ahaltsu bat lortuz.

Zope plataforma ere indarra hartzen ari da munduan zehar, eta baita gure ingurunean ere. Web aplikazioak garatzeko plataforma honek web bidezko kudeaketa eskaintzen du normalean ohituta gauden HTML editoreak eta FTP zerbitzarira fitxategiak jaso beharretik alde eginez.

Ikasturte honetan garatu dudan proiektuaren hasierako helburua enpresaren beharretara eta erabiltzen dituzten teknologia eta tresnetara egokitzen zen lan-fluxu sistema bat garatzea zen. Azkenean behar horiek hasetzen dituen produktu bat aurkitu

dugu, OpenFlow, eta ondorioz sistema berri bat eraiki ordez, produktu hori osatzeko bi garapen egin dira.

Helburuak lan-fluxu sistemen mundua bera ezagutu eta aztertzea, sistema horiek CodeSyntax enpresak darabilen Zope plataforman nola integratzen diren aztertzea, eta aipatutako plataformarako garapen bat egitea izan dira.

Lehenengo helburua betetetzeko, lan-fluxuen teoria ikasi behar izan dut eta baita XPDL lengoaia ere, lan-fluxuen adierazpena egiten duen XML lengoaiaren dialektoa.

Bigarren helbururako, OpenFlow tresna bera ere aztertu behar izan dut lan-fluxu sistema baten inplementazio bat ezagutu eta proiektuan zehar erabili dudan tresna ezagutzeko. Horretaz gain, Zope web aplikazioen zerbitzaria eta Python programazio lengoaia erabiltzen ikasi behar izan dut, OpenFlow plataforma horretarako delako eta aipatutako lengoaian programatuta dagoelako.

OpenFlow produktuaren modulu gehigarriak garatzeko, XML dokumentuak aztertzeko eredu nagusiak, DOM eta SAX aztertu, eta Zope eta Pythonen horiek nola erabili aztertu behar izan dut. Horretaz gain, Python programazio lengoaiarentzako beste modulu bat ere erabili behar izan dut proiektuko bigarren modulua garatzeko. Gainera Zope plataformako Zope Page Templates tresna ere erabiltzen ikasi behar izan dut bigarren modulu hau garatzeko orduan.

Memoria honen egitura 10 atal eta 5 eranskinetan antolatuta dago. Lehenengo, aurkezten den proiektuaren helburuen dokumentua dator, bertan jasotzen da proiektuaren atazen banaketa eta bete beharreko lan orduen baliospena, plangintza eta proiektuaren garapenean zehar egin behar izan zen plangintza berria. Ondoren lan-fluxu sistemen inguruko kapitulu bat dator eta 4. atalean, proiektuan erabili diren teknologia eta tresna nagusien azalpena. 5. eta 6. ataletan, OpenFlow produktua eta XPDL formatua azaltzen dira, hurrenez hurren. 7. kapituluan, hurrengo kapituluan OpenFlow eta XPDL arteko bihurketa egiten duen moduluaren diseinua eta inplementazioa azaltzen dira. 8. kapituluan, proiektuan garatutako bigarren moduluaren ezaugarriak datoz, konpiladorearenak hain zuzen ere. Azkenik proiektuaren ondorioak eta

bibliografia eta erreferentzia interesgarriak azaltzen dira.

Memoriaren eranskin gisa gehitu ditut, XPDL formatua definitzen duen XML-Schema fitxategiaren edukia, garatutako moduluen eskuliburuak, OpenFlow produktuaren APIa eta azkenik aztertu ditudan lan-fluxu sistemen inguruko txostena.

Horretaz gain, memoriarekin batera CD bat ere banatu da eta hau da bere barneko karpeten edukia:

- *soft* izeneko karpetan, modulu bakoitzaren eskuliburuetan aipatutako softwarea dago, hala nola, Zope web aplikazioen zerbitzaria, OpenFlow produktua eta proiektu honetan garatutako moduluak funtzionatzeko pakete gehigarriak.
- *OpenFlow-XPDL* izeneko karpetan, memoria honen 7. atalean azaltzen den OpenFlow eta XPDL lengoaiaren arteko esportatzaile eta inportatzailea daude.
- *Konpiladorea* izeneko karpetan, memoriaren 8. atalean azaltzen den konpiladoreari dagozkion fitxategiak daude.

Banatutako CDaren edukiak erabiltzeko, memoria honen B eta C eranskinetako argibideak irakurtzea gomendatzen da.

2. PROIEKTUAREN HELBURUEN DOKUMENTUA

Helburuen dokumentu honetan proiektuaren helburuak aurkezteaz gain, berau garatzeko identifikatutako ataza ezberdinak eta estimatutako iraupena azaltzen dira. Era berean, erabiliko den lan-metodologia zein den azaltzen da eta gertatu diren plangintza aldaketen inguruko informazioa ere ematen da. Azkenik, proiektuaren kudeaketaren inguruko ondorioak eta laburpen bat ematen dira.

2.1. Helburuak

Proiektu honen helburuak hiru dira:

1. Lan-fluxu edo workflowen mundua ezagutu eta existitzen diren softwarezko inplementazioak zein diren eta zein aukera eskaintzen dituzten ezagutu.
2. Lan-fluxuen mundua Zope plataforman nola integratzen den ikasi eta bereziki OpenFlow produktuaren erabileran sakondu.
3. Zope plataformarako produktu eta moduluen garapena nolakoa den ikasi eta OpenFlow produktuarentzako modulu bi garatu.

2.2. Azalpena eta motibazioa

Hasiera baten workflowen mundua ezagutu eta Zope plataformarako garapen bat egitea zen helburuetako bat, hala ere, lan-fluxu sistema ezberdinen arteko analisi eta konparaketa bat egin eta enpresaren beharrak eta eskakizunak aztertu ostean, ezaugarri egoki horietara egokitzen den produktu bat ezagutu genuen: OpenFlow [Icube]. Bere garatzaileekin kontaktuan jarri ostean, produktuak beharko lituzkeen hobekuntza eta modulu gehigarri batzuk zein ziren esan ziguten eta horien artetik modulu biren garapena egitea proposatu genien.

Garatu beharreko moduluak bi dira: alde batetik XPDL [WfMC-XPDL, 2002] fitxategi bat inportatu eta esportatuko dituena eta bestetik lan-fluxuen konpilazio antzeko lan bat egingo duena.

Hasiera batean salbuespenen kudeaketa egingo zuen modulu bat garatzea proposatu genien, aurrerago gauzak zehazteko utziz. Hala ere, zehaztapen hori egin behar zen garaia iristean gaia birplanteatu egin behar izan genuen, ez baikenuen argi zer ulertzen zuten euren salbuespenen kudeaketarekin: salbuespenak hasieratik bukaerara automatikoki kudeatuko zituen tresna bat ala OpenFlowk integratzen dituen salbuespenak kudeatzeko deiak beste produktu batera atera.

XPDL lengoaia Workflow Management Coalition [WfMC] delakoak estandarizatua da eta XMLren dialektoa izaki berau tratazeko erabiltzen diren teknikak erabil daitezke.

Proiektua enpresa batean aurrera eramango denez, lan munduarekin izaten den harremana ezagutzeko balioko dit, benetan lan mundura jauzia eman aurretik. Beste alde batetik Zope [Zope] plataforma azken urteetan nabarmen ari da sartzen Interneteko munduan, azpian duen Python [Van Rossum] programazio lengoaiarekin batera. Plataforma honen ezagutza txiki bat dudan arren, bertan sakontzea oso motibagarria dela uste dut lan munduari begira.

Beste alde batetik enpresa batean lan egiteak bai niretzat eta bai eurentzat esperientzia berri bat izango da eta oso erakargarria iruditzen zait nik eurentzat egin dezakedan lan bat baliagarria izatea.

2.3. Proiektuaren faseak eta lanorduen balioespena

Proiektuaen plangintza eta kudeaketan esperientzia handirik ez dudanez, laugarren ikasturteko irakasgaietan ikusitakoa soilik, ikasgai horietan ikusitakoaren arabera eta inpresio pertsonalen arabera egin ditut estimazioak.

Aurrekarien azterketa (10 ordu)

Enpresak dituen beharrak zein diren eta jadanik zer eginda duten aztertuko da.

Emangarria: beharren dokumentua.

Dagoen softwarearen azterketa (30 ordu)

Internet erabiliz lan-fluxuen kudeaketarako zein sistema dagoen eta eskaintzen dituzten ezaugarriak aztertuko dira. Enpresak Zope plataformaren gainean lan egiten duenez, bertarako den softwareari lehentasuna emango zaio, hori posible ez bada, ahal bada software librea den produkturen bat bilatuko da.

Emangarria: lan-fluxu sistemen txostena

OpenFlowren ezagutza (52 ordu)

Eskeintzen dituen aukerak aztertu eta dokumentazioa irakurriko da, ezagutza sakonagoa lortu asmoz. Era berean, lan-fluxu sinple eta zailago batzuk inplementatuko dira produktuarekin erraztasuna lortu eta berau hobeto ezagutzeko.

Emangarria: OpenFlowri buruzko txostena eta aplikazioak programatzeko gida.

XPDL import/export

XPDL formatuaren ezagutza (40 ordu)

Formatu honi buruzko informazioa eta berau erabiltzen duten tresnen azterketa egingo da, garatu behar den inportatzaile eta esportatzailearen oinarri izateko.

Emangarria: XPDL erabiltzean jakin beharreko oinarrizko kontzeptuak

Zope eta Pythonek XML tratatzeko dutenaren azterketa (36 ordu)

XML dokumentuak tratatzeko orduan DOM eta SAX ereduak jarraitu ohi dira. Lehenengoak XMLren dokumentua zuhaitz egiturako datu-mota batean biltzen du gero errazago tratatzeko. Bigarrenak XML dokumentuak interpretatzerakoan egiten du lana, irakurritako elementu bakoitzeko metodo bati deituz. Zope eta Pythonekin bi eredu horiek nola erabili, aztertu eta eskaintzen dituzten aukeren azterketa egingo da, bietako zein, nola eta zein kasutan erabiltzeko. Eskeintzen dituzten baliabideen azterketa egingo da, eta baita frogak ere, egokiena zein izan daitekeen jakiteko.

Emangarria: Inportatzailea eta esportatzailea egiteko zein eredu jarraituko den

XPDL esportatzailearen diseinua, inplementazioa eta probak (64 ordu)

OpenFlowtik XPDL estandarrerako itzulpena egingo duen moduluaren analisia, diseinua eta inplementazioa egingo dira. Esportatzailea probatzeko proba kasuak diseinatuko dira eta erantzun egokiak ematen dituzten egiaztatuko da. Sortutako XPDL fitxategiak existitzen den beste tresna batekin probatuko dira [Enhydra Jawe]. Bukaera inguruan OpenFlowren egileei bidaliko zaie modulua euren probak egin ditzaten.

Emangarria: modulua eta bere dokumentazioa.

XPDL inportatzailearen diseinua, inplementazioa eta probak (75 ordu)

XPDL estandarretik OpenFlowrako itzulpena egingo duen moduluaren analisia, diseinua eta inplementazioa egingo dira. Proba kasuak diseinatuko dira eta XPDL fitxategi ezberdinekin OpenFlowren instantzia egokiak sortzen direnentz egiaztatuko da. Bukaera inguruan OpenFlowren egileei bidaliko zaie modulua euren probak egin ditzaten.

Emangarria: modulua eta bere dokumentazioa.

Exception Handler

Modulu honen zeregina OpenFlow-n errepresentatuko diren prozesuen salbuespen egoerak kudeatzea izango da.

Beharren analisia (27 ordu)

Lan-fluxu sistema baten salbuespenak noiz eta nola ematen diren ezagutu behar da eta baita kasu errealetan zer egiten den horrelako kasuetan. Beste lan-fluxu kudeaketa sistema batzuk horrelako kasuetan nola funtzionatzen duten aztertuko da. Bezero potentzialekin eta OpenFlowren egileekin komentatuko dira behar hauek

Emangarria: beharren analisiaren dokumentua

Inplementazioa eta probak (100 ordu)

Salbuespenak kudeatuko dituen modulua garatuko da eta behar diren probak egin. OpenFlowren egileekin harremana beste faseetan baino handixeagoa izango da.

Emangarriak: moduluaren inplementazioa eta dokumentazioa

Dokumentazioa

Dokumentazio orokorra

Proiektuaren inguruko dokumentu guztiak bildu eta era egokian gordeko dira gero Memoria zein aurkezpenari begira.

Memoria (20 ordu)

Egindako emangarriak, bildu, dokumentazio gehigarria idatzi eta txostena prestatu.

Emangarria: proiektuaren memoria.

Aurkezpena (20 ordu)

Proiektuaren defentsarako egunari begira aurkezpenean erabili beharreko materiala prestatuko da.

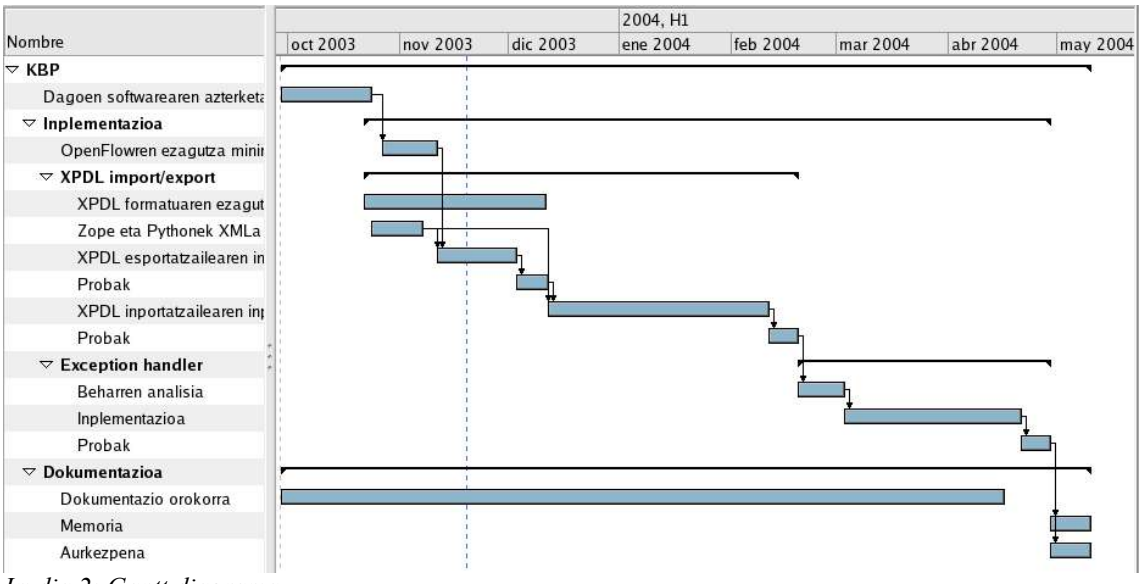
Emangarria: aurkezpeneko materiala: aurkezpena eta adibideak

Azalpena

Hasiera batean egunean 4 ordu lan egitea planifikatu dut guztira 438 ordu balioetsiz. Irudia 1 eta Irudia 2n atazen banaketa eta dagokien Gantt diagrama ikusi daitezke.

Nombre	Inicio	Fin	Obra
KBP	**sep 29**	**may 10**	**109d**
Dagoen softwarearen azterketa	sep 29	oct 24	15d
Inplementazioa	**oct 22**	**abr 29**	**92d 3h**
OpenFlowren ezagutza minimo bat lortu	oct 26	nov 11	10d
XPDL import/export	**oct 22**	**feb 19**	**54d 4h**
XPDL formatuaren ezagutza eta azterketa	oct 22	dic 11	30d
Zope eta Pythonek XMLa tratatzeko dutenaren azterketa	oct 24	nov 7	9d 1h
XPDL esportatzailearen inplementazioa	nov 11	dic 3	12d 1h
Probak	dic 3	dic 12	5d 2h
XPDL inportatzailearen inplementazioa	dic 12	feb 11	20d
Probak	feb 11	feb 19	5d
Exception handler	**feb 19**	**abr 29**	**40d**
Beharren analisia	feb 19	mar 3	6d 3h
Inplementazioa	mar 3	abr 21	23d 4h
Probak	abr 21	abr 29	5d
Dokumentazioa	**sep 29**	**may 10**	**109d**
Dokumentazio orokorra	sep 29	abr 16	97d
Memoria	abr 29	may 10	5d
Aurkezpena	abr 29	may 10	5d

Irudia 1: Hasierako atazen banaketa



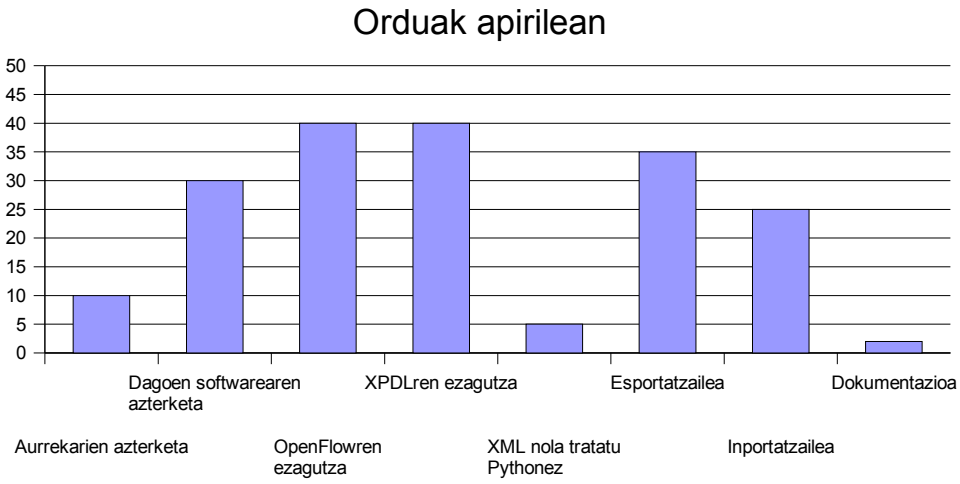
Irudia 2: Gantt diagrama

2.4. Plangintza aldaketak

Hasierako plangintza guztiz osatuta egon gabe hasi nintzen lanean eta ondorioz aldaketak jasan ditu. Aurreko atalean aurkeztutakoa hasiera bateko plangintza zen arren, apirilean plagintza berregin egin behar izan zen bi arrazoirengatik: alde batetik lehenengo moduluaren garapenaren ostean ikasleak OpenFlow produktuarekin lan-fluxu zehatz batzuen inplementazioa egin zuen CodeSyntax enpresaren proiektu baterako, eta bestetik OpenFlowren garatzaileek ez zuten erantzunik eman salbuespenen kudeaketari buruz luzatutako galderei. Ondorioz, Irudia 3n ikus daitekeena zen proiektuaren egoera apirilean.

Horrek eragin zuen salbuespen kudeatzaileari zegokion zatiaren plangintza berregitea eta OpenFlow konpiladore bategatik ordeztea, proiektuaren amaiera ekainaren bukaeraraino luzatuz.

Plangintza horren ondorio gisa, astean lanordu gehiago sartu behar dudala garbi ikusi dur. Horretarako, behin klaseak amaituta, goizez lan egingo dut proiektuan azterketak iritsi arte, eta ondoren egun guztian. Asteburuetan ere dokumentazio lanak aurreratzea egokia dela dirudi, proiektua, nahi bezala, ekainaren bukaerarako amaituta egon dadin.



Irudia 3: Lan egindako ordu kopurua apirilean

Ondorioz bigarren zatiaren plangintza eta helburuak aldatu egin dira, eta bide batez, orain arteko dokumentazio lan eskasa ikusita, fase bakoitzaren dokumentazioari dagokion lanordu kopurua handitzea erabaki dut.

Dagoen softwarearen azterketa (35 ordu)

Hasieran 30 ordu planifikatu nituen ataza honetarako, baina dokumentazio lana dela eta beste 5 ordu gehitzea erabaki dut.

OpenFlowren ezagutza (60 ordu)

Dokumentazio lanetarako beste 8 ordu gehitu ditut.

XPDL import/export

Atal honetan ere dokumentazio lanari dagokion ordu kopurua inkrementatu dut 37 ordutan, fasez fase, hau izango da ordu banaketa berria.

- XPDL formatuaren ezagutza (50 ordu)
- Zope eta Pythonek XML tratatzeko dutenaren azterketa (41 ordu)
- OpenFlow - XPDL esportatzailea (71 ordu)
- XPDL – OpenFlow inportatzailea (80 ordu)

OpenFlow konpiladorea

Modulu honek OpenFlown errepresentatuko diren lan-fluxu ezberdinen zuzentasuna aztertzea du helburu.

Kasuen azterketa (5 ordu)

Arazoak ematen dituzten lan-fluxuak zein ezaugarri betetzen dituzten aztertuko da.

Emangarria: kasuen azterketa eta bakoitzean egin beharrekoa adierazten duen txostena

Diseinua eta inplementazioa (40 ordu)

Moduluaren diseinua eta inplementazioa egingo dira produktua OpenFlown bertan integratuz, Zopek horretarako eskaintzen dituen tresnak erabiliz.

Emangarria: modulua

Proba kasuen diseinua eta egikaritzea (5 ordu)

Modulua probatzeko lan-fluxuak inplementatuko dira (grafoak bakarrik) eta beraiekin probak egin

Emangarria: proba kasuen emaitzak

Moduluaren dokumentazioa (15 ordu)

Modulua dokumentatuko da.

Emangarria: erabiltzailearen eskuliburua

Dokumentazioa

Dokumentazio orokorra

Proiektuaren inguruko dokumentu guztiak bildu eta era egokian gordeko dira gero memoria zein aurkezpenari begira.

Memoria (16 ordu)

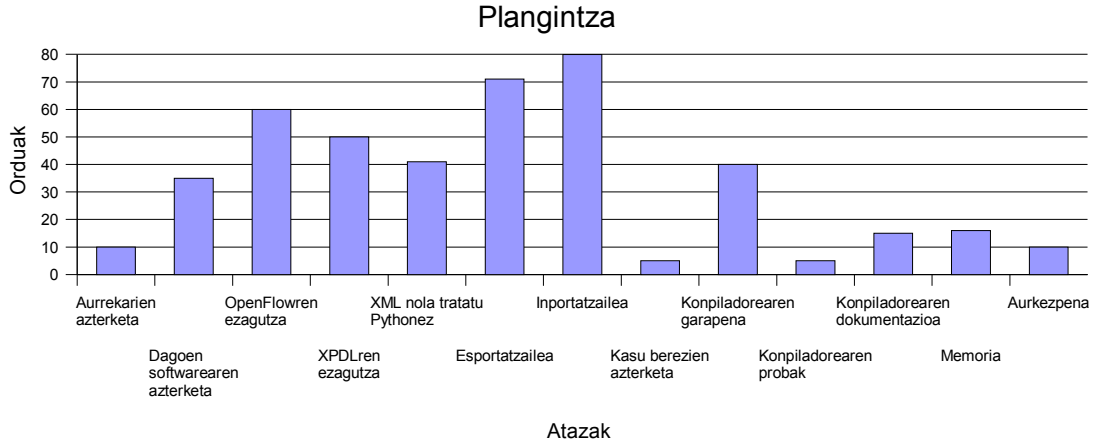
Idatzitako dokumentazioa bildu eta azken txostena osatzeko egin behar diren txukuntze eta osatze lanak egingo dira.

Emangarria: proiektuaren memoria.

Aurkezpena (10 ordu)

Proiektuaren defentsarako egunari begira aurkezpenean erabili beharreko materiala prestatuko da.

Emangarria: aurkezpeneko materiala



Irudia 4: Plangintzako ordu banaketa

2.5. Lan metodologia

Proiektua enpresako tutorearen gidaritzapean egingo da, enpresa beraren egoitzan, Eibarko Azitaingo industrialdean. Bertako materialarekin egingo da eta ikasleak ordenagailu bat eta enpresako beste baliabideen erabilpena egingo du.

Internet izango da ikaslearen lan tresna garrantzitsuenetako bat, bai informazioa aurkitzeko zein jadanik dagoena probatzeko. Horretarako enpresaren lanorduetara mugatuko da ikaslea berak Unibertsitatean dituen ordutegien arabera: lehen lauhilabetean goizez eta bigarrenean arratsaldez. Plangintza egitean kontuan hartu da jaiegun, gabon, azterketa garai eta aste santuan zehar ez dela lanik egingo.

Irakaslearekin jarraipen bilerak egingo ditu biek horretarako beharra dagoela ikusten dutenean. Enpresan bertan, eta bertako metodologiari jarraiki, bertako proiektuen jarraipenerako bileretan parte hartuko du ikasleak hauek egiten direnean.

Enpresaren bezero potentzialekin ere bilerak izango ditu, lan-fluxuen mundua hobeto ezagutu asmoz eta bezero horien beharretara egokitzen den produktu baten bila.

Softwarearen aldetik, Zope 2.6.1 plataforma eta bere azpian dagoen Python 2.1.3 programazio lengoaia erabiliko dira Linux sistema eragilea duen makina batean. Dokumentaziorako OpenOffice.org ofimatika paketea, Dia diagramak egiteko softwarea eta GIMP irudigintzarako softwarea erabiliko dira.

2.6. Arrisku faktoreak

Faktorea	*Arrisku maila*	*Soluzioa*
Arazo teknikoak	Baxua - erdia	Proiektuaren epeen atzerapena eragingo du
Gaixotasunak	Baxua - erdia	Proiektuaren epeen atzerapena eragingo du
Lana eginda egotea	Erdia	Proiektuaren helburuen aldaketa
Garatzaileekin kontaktatzeko arazoak	Erdia	Proiektuaren epe eta helburuen aldaketa
Esperientziarik eza proiektuen kudeaketan	Erdia	Beste proiektu batzuen estimazioetatik ikasi

Taula 1: Identifikatutako arrisku faktoreak

3. WORKFLOW EDO LAN-FLUXU SISTEMAK

Atal honetan lan-fluxu sistemen inguruko informazioa ematen da, zer diren eta zertarako balio duten, zein ezaugarri dituzten, zein motatakoak dauden eta berauen estandarizazioaren bidetik zein lan egin den azalduz.

3.1. Sarrera

Lan-fluxuak lan prozesu osoen eta berauen zatien automatizazioak dira, non dokumentuak, informazioa edo lanak partehartzaile batengandik beste batengana doazen berauek burutzeko, aurretik ezarritako prozeduren arauak jarraituz [WfMC].

Lan-fluxu bat burutzeak pauso edo ekintza batzuk aurrera eramatea dakar, ekintzen artean trantsizio batzuetan zehar mugituz. Ekintza bat aurrera eramateak erabiltzailearekin elkarrekintza egotea ekar dezake (datu sarrera bat, eskuz egin beharreko datuen gainbegirada bat, ...), baina beste era bateko ekintzak ere egin daitezke, hala nola, nagusiari abisu bat bidaltzea, gutun bat bidaltzea, kontabilitate kontuetarako ekintza konkretu bat burutu dela erregistratzea, ...

Ekintzen arteko trantsizioak automatikoki egiteak, paperak eskuan dituzula bulegotik bulegora ibili beharrean, lan horiek automatizatzeak azken finean, lan-fluxuaren eraginkortasuna eta lanorduen erabilpen hobea dakartza [Prior, 2003] eta [Allen, 2001]k diotenez.

Lan-fluxu baten adibidea enpresa baten erreklamazioak kudeatzeko sistema izan daiteke. Erreklamazioak daudenean begirale batek erabakiko du onargarriak diren ala ez eta hala bada kudeatzaileak erabakiko du nola aurre egin erreklamazioari kexa jarri duenari erantzun bat emanez eta kexaren inguruko datuak erregistratuz.

Ondorengo puntuetan zehar lan-fluxuen kudeaketa sistemek aurkezten dituzten ezaugarriak azalduko dira eta azkenik erabilitako OpenFlow tresnak inplementatzen duen ekintzetan oinarritutako lan-fluxu sistemaren azalpen zabal bat ematen da, egoeretan oinarritutakora sarrera bat ere eskainiz.

3.2. Lan-fluxuen kudeaketa sistemak

Kudeaketa sistema hauek lan-fluxuen sorrera eta kudeaketaz arduratzen diren softwarezko aplikazioak dira. Alde batetik lan-fluxu jakin baten errepresentazioa izaten dute, grafo baten bidez adierazten dena, eta fluxu horretako instantzia jakin baten informazioan oinarriturik prozesuaren jarraipena egiten dute, bitartean dauden ekintzak burutuz (automatikoak baldin badira) edo erabiltzailearekiko elkarrekintza eskatuz.

Automatizazioari esker lan-fluxuaren instantzia jakin baten egoera zein den erraz jakin daiteke. Adibidez, udaletxe baten espedienteen kudeaketarako lan-fluxuan, uneoro jakin behar da espedientea tramitazioaren zein unetan dagoen, inork informazio hori eskatuko balu, berehala emateko. Lan-fluxua automatizatuta egotean instantziari buruzko datu guztiak era erraz batean lor daitezke, kudeaketa sistema batekin instantzia guztien inguruko informazioa eta sistemaren kontabilitatea kudeaketa sisteman bertan zentralizatuta baitago.

Gainera, erabiltzen den kudeaketa sistemaren arabera, enpresaren gainontzeko informazioaren konexioak egin daitezke, datu-baseekin adibidez, une bakoitzean egin beharreko lana aurrera eramateko.

3.3. Lan-fluxu motak

Bi lan-fluxu mota nagusi bereizten dira: ekintzetan oinarritutakoak eta egoeretan oinarritutakoak. Lehenengoetan garrantzitsuena ekintzak dira. Lan-fluxuan egin beharreko ekintzak bukatzeak ahalbidetzen du lan-fluxuak aurrera jarraitzea. Egoeretan oinarritutakoak, aldiz, entitate baten inguruan jarduten dute. Entitate horren egoera aldaketetan datza lan-fluxua.

3.3.1. Ekintzetan oinarritutakoak

Lan-fluxu mota hauetan ekintzak dira garrantzitsuena, lan-fluxu bat aurrera eramateko ekintza batzuk gauzatu behar dira. Mota honetako kudeaketa sistema batek egin behar duen zeregin nagusia esaldi baten bil daiteke, [WfMC]ren esanetan: *“nork*

egin behar du zer, noiz eta nola". Prozesuaren definizioak mugatzen du *zer* hori *nork* eta *noiz* egin behar duen. Ekintza bakoitzak aplikazio bat edo gehiago eramango du lotuta eta horrek mugatuko du lan-fluxuaren *nola* atala.

Proiektu honen garapenean erabili dudan OpenFlow produktua, mota honetakoa da, eta era garbi batean inplementatzen ditu jarraian azalduko ditudan ekintzetan oinarritutako lan-fluxuen ezaugarriak.

Ekintzetan oinarritutako lan-fluxuetan, lau kontzeptu nagusitzen dira: ekintzak, trantsizioak, aplikazioak eta rolak.

3.3.1.1. Ekintzak

Ekintza batek egin beharreko lana adierazten du. Lan hori aplikazio bat exekutatzea, beste azpifluxu bati deitzea edo bideratze lan arrunta (ezer egin gabe hurrengo ekintzara pasatzea) izan daiteke.

Adibide bat hartuz, udaletxe bateko espedienteen kudeaketa daraman lan-fluxuan, interesatuak paper guztiak bete eta udaletxeko idazkaritzan aurkeztea izan daiteke ekintza bat.

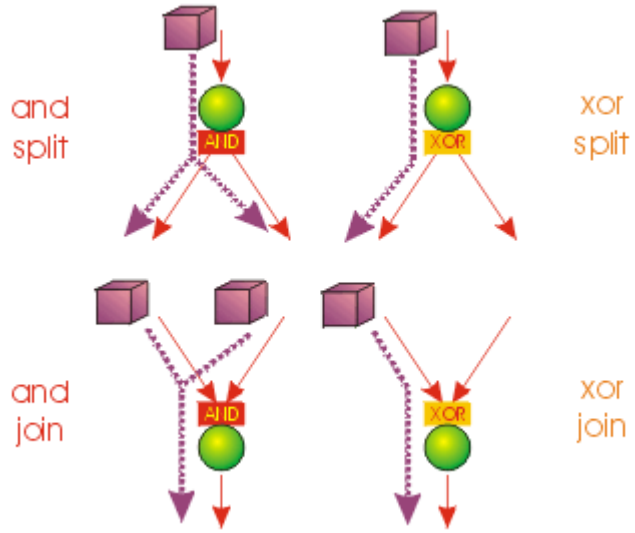
Egin beharrekoaz gain, ekintza batek sarrera eta irteera babes batzuk ditu, bertara sartzean lana noiz abiatu jakiteko eta bertatik irtetean lana nondik bideratu jakiteko. Babes horiek *and* edo *xor* modukoak izaten dira eta esanahi hau daukate:

Sarrera babesa (ikusi Irudia 5):

- *and*: uneko ekintzara aurreko ekintza guztietatik igaro ondoren pasatuko da instantzia. Adibidez uneko ekintzara heltzeko bi bide baldin badaude, aurreko bi bideetatik etorri behar da instantzia uneko egoerako lana abiatu aurretik.
- *xor*: nahikoa da aurreko ekintzetako bakar batetik instantzia etortzea uneko ekintza aurrera eramaten hasteko. Uneko ekintzara etortzeko bi bide baldin badaude, instantzia bietako batetik etortzea nahiko da uneko ekintza martxan jartzeko. Ondoren etorriko diren ekintzei ez zaie kasurik egiten.

Irteera babesak (ikusi Irudia 5):

- and: uneko ekintza bukatzen denean ondorengo ekintza guztietara bideratuko da instantzia.
- *xor*: uneko ekintza bukatzean ondorengo ekintzetako bakar batera bideratuko da instantzia, aurrez ezarritako baldintza bat betetzen duen trantsizio bat erabiliz hain zuzen ere.



Irudia 5: and eta xor babesak

Sarrera eta irteera babesak ulertzeko adibide bat erabiliko dut. Demagun erreklamazio berri bat iristen denean, hau bi arduraduni bidaltzen zaiela. Kasu horretan *and* erako irteera babes bat edukiko genuke, sarrerakoa bi helburutara bideratu behar delako. Ondoren bi arduradunetako batek bere iritzia ematea nahikoa da erreklamazioak aurrera jarraitzeko. Kasu horretan *xor* erako sarrera babes bat izango genuke bietako batetik instantzia etortzea nahikoa delako aurrera jarraitzeko.

Era berean ekintza batek hasiera eta bukaera ditu, eskuzkoak ala automatikoak izan daitezkeenak. Hasiera automatikodun ekintzak berari lotutako aplikazioa era automatikoan abiaraziko du eta bukaera automatikoak lan-fluxuaren instantzia hurrengo ekintzara bidaliko du berari lotutako aplikazioa bukatzeaz bat.

3.3.1.2. Trantsizioak

Trantsizioak ekintzak elkarrekin lotzeko tresnak dira. Gainera baldintza bat eraman dezakete atxikituta ekintza trantsizio horretatik pasatu behar den ala ez erabakitzeko, ekintza baten irteera babesa *xor* motakoa bada automatikoki ebaluatuko den baldintza, hain zuzen ere. Lehen azaldutako moduan *xor* erako irteera babes batek, ekintza baten ondoren lana banatzeko balio du, bertako irteeretako batetik bideratuz. Erabaki hori hartzeko erabiltzen da trantsizioari ezarritako baldintza.

Espedienteen kudeaketako adibidearekin jarraituz, behin espedientea aurkeztuta, lehenengo errebisio bat egin beharko zaio paper guztiak entregatu dituen ala ez ikusteko eta horrek instantziari propietate berri bat gehituko dio, adibidez *"Paper guztiak ondo"* izenekoa.

Ekintza horretatik bi trantsizio irtengo dira *xor* babes batekin, trantsizio batek *"Paper guztiak ondo = True"* baldintza izango du eta besteak *"Paper guztiak ondo = False"*. Lehenengo errebisioaren ondoren propietateari ezarri zaion balioaren arabera trantsizio batetik edo bestetik zehar joango da aurrera instantzia.

3.3.1.3. Aplikazioak

Lan-fluxuen ekintzetan exekutatu behar dira aplikazioak. Aplikazio hau formulario sinple bat betetzetik, datu-base batean SQL aginduak exekutatzerainokoa izan daiteke, posta elektronikoko mezuak bidali, kudeaketa sistematik kanpoko aplikazio bat martxan jarri edo gure sistemak ahalbidetzen duen beste edozein gauza egitetik pasatuz.

Aplikazio hauei, gehienetan, uneko lan-fluxuaren instantziaren ezaugarriak pasatu beharko zaizkie parametro gisa euren lana ongi egiteko. Adibidez, espedienteen kudeaketarako lan-fluxuan, paper guztiak ondo daudela egiaztatu ostean, teknikarien batzorde batek bere iritzia eman behar du aztertu behar den gaiaren inguruan. Kasu honetan, demagun, kalean tabernetako mahaiak jartzeko baimena eskatzen dela, bada, udaltzainak joan beharko dira eta txosten bat osatu kalean zehar jendea pasatzeko lekurik uzten duten, tabernatik urrutiegi jartzen duten edo alboko dendaren bati traba

egiten dion esanez. Lan-fluxuaren ekintzak, txosten hori instantziari gehitzen dio eta aplikazioa informazio horretaz baliatuko da estatistikak jasoko dituen datu-basea eguneratzeko.

3.3.1.4. Rolak

Erabiltzaileak ezaugarri batzuk betetzen dituzten rol edo taldeetan banatzen dira, erabiltzaile bakoitza rol bat baino gehiagotan egon daitekeelarik. Ekintza bakoitza rol bati edo gehiagori esleitzen zaie eta rol hori duten erabiltzaileek egin dezakete ekintza horri esleitutako lana.

Horretaz gain, lan-fluxu kudeaketa sistema gehienetan, kudeatzaileak erabiltzaile konkretu bati lan zehatz bat esleitzeko aukera ere egon ohi da.

Espedienteen kudeaketako adibidearekin jarraituz, udaltzainen txostena osatzeko ekintza *Udaltzaina* rola duen pertsona batek egin ahal izango du edo espedientearen tramitazioaren hasiera, berau lan-fluxuan erregistratzen duen *Erregistratzailea* rola duen langileak.

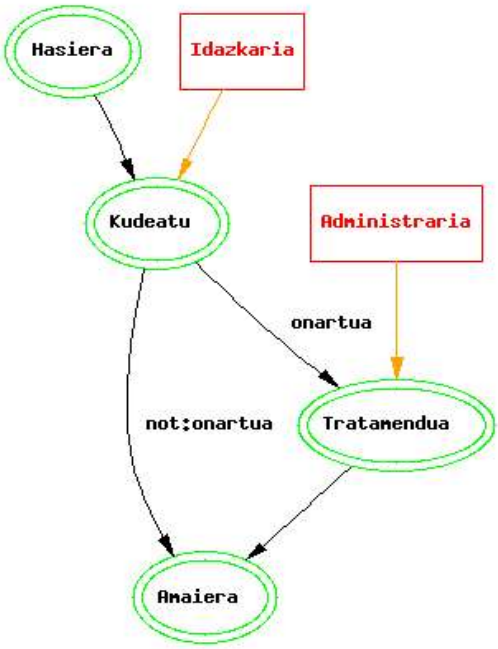
3.3.1.5. Adibidea

Azaldutako lau kontzeptu hauek adibide integratu baten ikus daitezke. Erreklamazioak kudeatzeko lan-fluxu sinple bat erabiliko dut adibidea azaltzeko. Lan-fluxu hori Irudia 6ko grafoarekin adieraz daiteke.

Hasiera, *Kudeatu*, *Tratamendua* eta *Amaiera* ekintzak izango dira, fluxua *Hasiera* izenekoan hasiko da eta *Amaiera* izenekoan bukatu.

Ekintzen arteko geziek trantsizioak adierazten dituzte, horietako bitan gainera, baldintzak ikus ditzakegu (*onartua* eta *not:onartua* diotenak hain zuzen ere).

Idazkaria eta *Administraria* rolak izango dira, *Kudeatu* eta *Tratamendua* ekintzak aurrera eraman ditzaketen rolak hain zuzen ere.



Irudia 6: Erreklamazioen lan-fluxua

Lan-fluxua abiatzeko erabiltzaile batek automatikoki *Hasiera* egoeran kokatuko den grafoaren instantzia bat sortu behar du. Honek grafoa jarraituko duen elementu bat sortzea esan nahi du. Elementu horren sorrera automatikoa izango da erabiltzaileak lan-fluxu horretan lan egiteko asmoa duela adierazten duenean, adibidez formulario baten bidez erreklamazioa erregistratzen duen pertsonaren datuak, arazoa gertatu den lekua, eguna eta ordua eta erreklamazioaren azalpen bat ematen dituenean.

Behin erreklamazioa erregistratzean, *Kudeatu* egoerara pasatuko da. Egoera honetan *Idazkaria* rola duen erabiltzaile batek har dezake, lana aurrera eramateko. Instantzia aukeratzen duenean, erabiltzaileak erreklamazioa erregistratzean eman dituen datuak agertuko zaizkio eta horren arabera erabakiko du erreklamazioa onartu ala ez. Erabakiaren arabera instantziaren datuei erreklamazioa onartua dagoen ala ez dioen propietate bat gehituko zaio. Propietate hori erabiliko da irteera aukeratzeko, horretarako *Kudeatu* ekintzaren irteera babesa *xor* moduan konfiguratuta egongo da irteera bietako batetik bakarrik joateko.

Tratamendua ekintza, aurrekoaren antzekoa izango da. Kasu honetan *Administraria*

rola duten erabiltzaileek bakarrik eraman ahal izango dute aurrera ekintza. Erreklamazioarekin datozen datuak ikusi ondoren erabiltzaileak erabakiko du zer egin eta formulario bat beteko du egindakoa zer izan den esanez. Bukatzean instantzia *Amaiera* egoerara pasatuko da.

Amaiera egoerak ez du izango erabiltzailearen partehartzea eskatuko duen aplikaziorik lotuta, horren ordez datu-base batean sartuko ditu erreklamazioaren datuak erabiltzailearekiko era garden batean.

Beste alde batetik sarrera babesa *xor* moduan zehaztuta izango du, horrela bietako sarrera batetik instantzia etorrita nahikoa izango zaiolako aplikazio automatiko hori abiatu eta lana bukatzeko. Izan ere, adibide honetan, instantzia sarrera bietako batetik bakarrik etorriko da, kudeatu adarrean egin den *xor* erako banaketa dela eta.

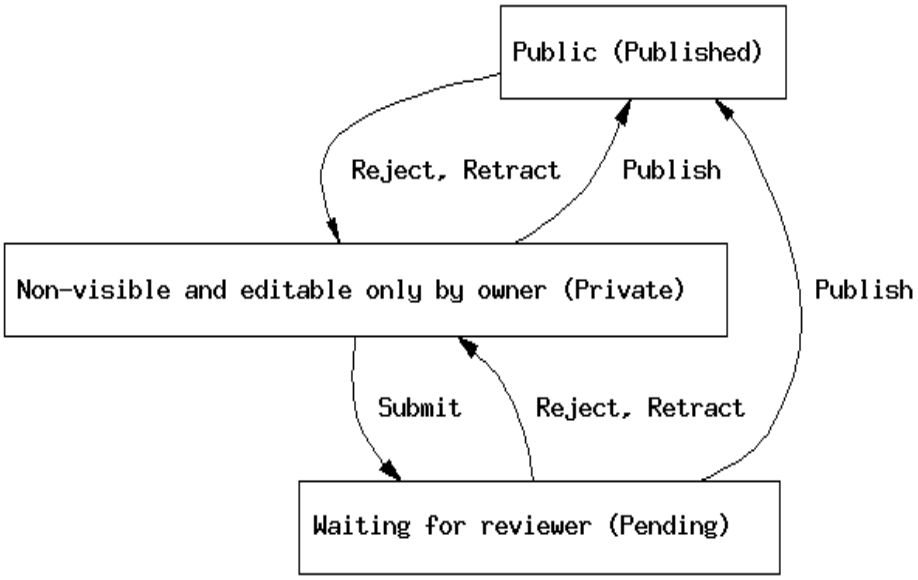
3.3.2. Egoeretan oinarritutakoak

Lan-fluxu mota hauetan, guztiak entitate baten inguruan lan egiten du. Entitate hori dokumentu bat izan daiteke adibidez eta egoera bat izango du atxekiturik. Egoera hori izango da aldatuko dena, baldintza batzuk betetzen direnean. Lan-fluxuaren exekuzioa, entitatearen egoera aldaketetan oinarritu ohi da mota honetako sistemetan.

Egoeretan oinarritutako lan-fluxu hauek gauza sinpleetarako erabiltzen dira, entitateak une bakoitzean egoera bakar batean egon daitezkeelako (gogoratu ekintzetan oinarritutakoetan *and* erako irteera babesak daudela instantzia bi ekintzatara bideratzeko).

Era honetako lan-fluxuak kolaborazio lanetan erabili ohi dira, publikazio sistema baten oinarri gisa adibidez. Adibidez Zope plataformako *CMF* delakoak (*Content Management Framework*), horrelako lan-fluxu bat inplementatzen duen *DCWorkflow* izeneko tresna bat integratzen du. *CMF* produktuak, elkarlanean aritzeko webguneak egitea ahalbidetzen du, eta elkarlan horretarako erabiltzen da egoeratan oinarritutako lan-fluxu sistema inplementatzen duen produktua.

Adibide gisa, publikazio sistema sinple baten lan-fluxuaren grafoa ikus daiteke Irudia 7n.



Irudia 7: Publikazio lan-fluxua

Erabiltzaileak dokumentu bat sortzen duenean *Private* egoeran sortuko du eta bertan egin ditzake aldaketak dokumentua errebisatzera bidali aurretik. Ondoren dokumentua errebisatzera bidali dezake (*Submit* trantsizioaren bidez, *Pending* egoerara) edo horretarako baimena baldin badu zuzenean publiko egin dezake (*Publish* trantsizioaren bidez *Published* egoerara bidaliz).

Errebisoreak dokumentua atzera bota dezake (*Reject*, *Retract*) edo oniritzia eman eta publiko egin (*Publish*). Behin dokumentua publikoa denean horretarako baimena duen erabiltzaileak dokumentua atzera bota dezake (*Reject*, *Retract*) eta berriz egoera pribatura pasatzen da.

Egoeren arteko trantsizioak kontrolatzeko, lan-fluxuari atxikitutako aldagai batzuen balioak erabiltzen dira, aurreko atalean azaldutako propietateen antzera. Era berean, entitatea egoera batetara iristean edo bertatik irtetean, aplikazio batzuk exekutatu daitezke, ekintzetan oinarritutako lan-fluxuetako aplikazioen antzeko zerbait.

Normalean mota honetako lan-fluxuek ez dute bukaera jakin bat izaten, ez baitaude ekintza segida bati lotuta, ekintzetan oinarritutakoak ez bezala. Dokumentua, edo era

orokorrago batean esateko, entitatea, lan-fluxuan zehar egoeraz egoera mugitzen ibil daiteke gelditu gabe.

4. ERABILITAKO TEKNOLOGIA ETA TRESNAK

Atal honetan proiektua garatzean erabili ditudan tresna eta teknologien inguruko azalpen batzuk emango ditut, erabilitako tresnetako batzuk oraindik oso ezagunak ez direnez, zer diren, zertarako balio duten eta zein ahalmen duten erakusteko.

4.1. Python programazio lengoaia

Guido Van Rossum holandarrak idatzitako objektuetara zuzendutako programazio lengoaia interpretatua da Python. Sintaxi argia eta erraza darabiltzanez oso erraza da beste lengoaiatan baino lan gutxiagorekin programak idaztea.

Python programazio lengoaia XPDL eta OpenFlowren arteko bihurketak egiteko modulua eta konpilazioa egiten duen modulua idazteko erabili dut.

Hemengo azalpenak [Pilgrim, 2004], [Marzal & Gracia, 2003] eta [Downey, Elkner and Meyers, 2003]-tik egokitu ditut. Baita [Van Rossum]-en dagoen informaziotik ere.

4.1.1. Sarrera

Instalazioarekin batera datozen liburutegi ezberdin kopuruari eta eskaintzen dituzten funtzionalitateei esker, Python “pilak kargatuta” datorrela esaten da, defektuzko banaketak hainbat gauza ezberdinetarako moduluak eskaintzen baititu: posta elektronikoak bidali eta jasotzeko moduluak, adierazpen erregularrak tratatzekoa, testuzko interfazeak egiteko modulua, sistema eragilearekin komunikatzekoa, ...

Helburu orokorreko programazio lengoaia bezala edo scripting lengoaia bezala erabil daiteke, aplikazioekin batera interpretea bera banatzeko aukera baitago. Gainera, konpilazio beharrik ez daukanez idatzitako programak berehala probatu eta erabil daitezke.

Interpreteari guk idatzitako programak pasatzean, Javaren *byte-code* antzeko zerbait sortzen du eta berehala hori exekutatu, erabiltzailea programa alde batetik konpilatu eta bestetik exekutatzera behartu gabe.

Kontrol egituren blokeak definitzeko ez du giltzik edo hitz berezirik erabiltzen C,

Java edo Adaz egiten den bezala. Hona hemen nola idatziko litzatekeen Java eta Pythonez begizta baten barnean if egitura bat daukan programa zatia:

Javaz:

```
class Proba{
      public static void main (String args[]){
       int[] test = new int[3];
       test[0]=-1; test[1]= 2; test[2] = 0;
       int i;
       for (i=0;i<3;i++){
        if (test[i]==0){System.out.println(i+".posizioan 0koa dago");
      } else {System.out.println(test[i] + "zenbakia ez da 0koa");}
       }
      }
}
```

Pythonez:

```
test = [-1, 2, 0]
for elementua in test:
      if elementua == 0:
                  print test.index(elementua) + " posizioan dago 0koa"
      else:
                  print elementua + " zenbakia ez da 0koa"
```

Javaz giltzekin mugatzen duguna, Pythonez indentazioarekin konpontzen da, bloke baten barruan dagoen guztia indentazio maila berean joan behar delako.

4.1.2. Datu-motak

Pythonek edozein programazio lengoaiatan aurki ditzakegun datu-motak dauzka: osokoak, errealak, karaktere-kateak, azken hauek komatxo sinple (*'karaktere katea'*) edo komatxo bikoitzen artean (*"hau ere karaktere katea da"*) joan daitezke. Gainera beste hiru datu-mota berezi ere definitzen ditu: zerrendak, tuplak eta hiztegiak, eta horien azalpena emango dut hurrengo parrafoetan zehar, beren erabilgarritasuna dela eta, asko erabili baititut moduluak garatzerakoan.

Zerrendak, beste programazio lengoaietan inplementatu daitezkeen zerrenda dinamikoak bezalakoak dira, exekuzio garaian tamainaz handitu eta txikitu

daitezkeenak. Oso erabilgarriak dira array antzeko egiturak tratatzeko array-en estatikotasunetik ihes eginez. Gainera zerrendaren elementuek ez dute zertan mota berekoak izan. Zerrenda bat makoen bidez adierazten da era honetan:

```
>> zerrenda = [1, “Mikel”, 9]
```

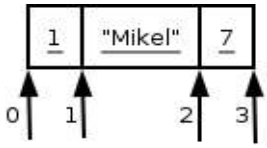
Zerrenda motako objektuak dinamikoki hedatu daitezke *append* metodoarekin eta elementuak ezaba daitezke *pop* metodoarekin indizea emanez, era honetan:

```
>> zerrenda.append(7)
>> zerrenda
[1, “Mikel”, 9, 7]
>> zerrenda.pop(2)
9
>> zerrenda
[1, “Mikel”,7]
```

Zerrendak, arrayak balira lez ere atzitu daitezke:

```
>> zerrenda[1]
“Mikel”
```

Gainera *slicing* eran ere atzitu daitezke, zerrenda zatiak lortzeko. Horretarako lortu nahi den lehenengo elementuaren indizea eta lortu nahi den azken elementuaren hurrengo indizea eman behar dira : ikurraz banatuta. Nolabait azaltzeko, indizeak elementuaren aurreaaldea seinalatzen duten erakusleak direla eta *slicing*-ak, hasiera eta bukaerako indizeek esandako posizioen arteko zatia itzultzen duela esan daiteke.



Irudia 8: Slicing-a

```
>> zerrenda[0:2]
[1, “Mikel”]
```

Zerrendak begizten bidez korritzea askotan egin behar izaten den lana da eta oso

sinplea da. Ez dago zertan array-ak bezala indize batekin egin, lengoaia batzutan agertzen den *for each* antzeko begizta batekin baizik:

```
for elementua in zerrenda:
      print elementua
```

Tupla, zerrendaren antzeko datu-mota da baina ez dira aldagarriak. Tupla bat sortu ostean, ez dago elementu berririk sartzerik edo kentzerik. Eraginkortasunaren aldetik, tuplak zerrendak baino azkarragoak dira korritzerakoan.

```
>> tupla = ('bat', 2, 3)
# objektuaren metodoak erakutsi
>> dir(tupla)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__',
'__setattr__', '__str__']
```

Elementuak gehitzeko daukan metodo bakarra *__add__* da (metodo berezi bat da, bi azpimarrarekin hasi eta bukatzen delako), baina metodo horri deitzea parametro bezala beste tupla bat pasatuz, + eragilearekin bi tupla “elkartzea” bezalakoa da. Eragile honek, metodo berezia kapsulatu egiten du. Ikus dezagun adibide batekin:

```
>> tupla1 = (1,2,3)
>> tupla2 = (4,5,6)
>> tupla1 + tupla2
(1,2,3,4,5,6)
>> tupla1.__add__(tupla2)
(1,2,3,4,5,6)
# tuplak ez dira aldatzen
>> tupla1
(1,2,3)
>> tupla2
(4,5,6)
```

Hiztegiak aldiz, beste lengoaia batzutan existitzen diren array elkarkorren antzekoak dira, indize bat erabili beharrean, edukiak, gako baten bidez atzitzen direlako. Beraien elementuak *(gakoa, balioa)* erako bikoteak dira: gakoak zenbakiak, stringak edo tuplak

izan daitezke, balioak edozein motatako objektuak izanik. Hiztegiak giltzen bidez mugatzen dira eta balio eta gakoak bi puntu ikurraren bidez. Hiztegietan, elementuen kokapena ez da esanguratsua, ordena batean sartutako elementuak beste ordena batean ager daitezke hiztegia korritu edo pantailaratzean.

```
>> birenPotentzia = { 1:2, 2:4, 3:8, 4:16 }
>> birenPotentzia[2]
4
>> itzulpena = { “etxea”: [“casa”, “hogar”], “kalea”: “calle”,
“zuhaitza”: “árbol”}
>> itzulpena[“etxea”]
[“casa”, “hogar”]
```

Azken kasu honetan, hiztegiaren lehenengo elementuan zerrenda bat gordetzen da *“etxea”* gakoaren atzean.

Hiztegi batean elementuak dinamikoki gehitu daitezke nahi den unean eta baita aldatu eta ezabatu ere:

```
>> hiztegia[“berebila”] = “coche”
>> hiztegia
 {“etxea”: [“casa”, “hogar”], “kalea”: “calle”, “zuhaitza”: “árbol”,
“berebila” : “coche”}
>> hiztegia.pop(“etxea”)
[“casa”, “hogar”]
>> hiztegia
{“kalea”: “calle”, “zuhaitza”: “árbol”, “berebila” : “coche”}
>> hiztegia['kalea'] = 'street'
>> hiztegia['zuhaitza'] = 'tree'
>> hiztegia['berebila'] = 'car'
>> hiztegia
{'zuhaitza': 'tree', 'berebila': 'car', 'kalea': 'street'}
```

Hiztegiak korritzeko aurreko atalean bezalako *for* egitura bat erabil daiteke hiztegiaren gako edo balioak erabiliz. Adibidez gakoak bakarrik inprimatzeko:

```
for gakoa in itzulpena.keys():
      print gakoa
```

Gako-balio bikoteak inprimatzeko *for* egituran binaka eslei ditzakegu hiztegiko elementuak:

```
for gakoa,balioa in itzulpena.items():
      print gakoa +” gakoari dagokion balioa “ + balioa + “ da”.
```

4.1.3. Funtzioak

Funtzioak beste edozein programazio lengoaiatan bezala definitzen dira, *def* hitz gakoaren ondoren izena jarriz eta azkenik parentesi artean parametroak jarriz. Parametroetan ez da zein motatakoak diren jarri behar eta funtzioak itzultzen duenaren mota ere ez da jarri behar. Gainera, parametro batzuk hautazko bezala markatu daitezke bere defektuzko balioa zein den esanez:

```
def probaFuntzioa (a, b=”Bilbo”, c=5):
      pass # ezer egiten ez duen sententzia
```

Kasu honetan b eta c izeneko parametroek defektuzko balio bat dute eta funtzioari deitzean baliorik ez bazaie esleitzen defektuzko hori hartuko dute.

```
>> probaFuntzioa(3, “Donostia”)
```

Dei hori eginez, *b* parametroan *“Donostia”* balioa pasatzen dugu eta *c* parametroak defektuzko balioa hartuko luke. Azkenengo parametroari balioa eman nahi badiogu eta bigarrenari ez, zera egingo genuke:

```
>> probaFuntzioa(3, c=17)
```

Horrela egin behar da definizioan dagoen parametroen ordena errespetatzen ari ez garelako.

4.1.4. Klaseak eta herentzia

Objektuei zuzendutako programazioaren barnean, klaseak definitzeko *class* hitz erreserbatua erabiltzen da Pythonez. Gainera herentzia anitza onartzen du eta nahiz eta interfazeak edo klase abstraktuak definitzeko zuzeneko aukerarik izan ez, inplementazio gabeko metodoak jarriz simulatu daitezke.

Atributu publiko eta pribatuak ez dira hitz gakoen bidez bereizten, baizik eta izenaren bidez: pribatuei bi azpimarra jarri behar zaizkie izenaren aurretik.

Metodo eraikitzaileari dagokionez, *__init__* (aurretik eta atzetik bina azpimarra dituela) metodoak egiten du lan hori, klaseko objektu bat sortzean metodo horri deitzen zaiolako automatikoki.

Bestalde, metodo guztien lehenengo parametro gisa, objektua erreferentziatzen duen parametro bat pasatzen dela adierazi behar da. Gero metodo horri deitzean ez da parametro hori erabiltzen, baina definitzerako orduan egin egin behar da. Metodo bati deitzean *objektua.metodoa(parametroak)* nomenklatura erabiltzen denez, *objektua* horrek ordezkatzen du, nolabait esateko, metodo guztietan agertzen den lehenengo parametroa. Hitzarmenez, lehenengo parametro horren izena *self* izan ohi da.

```
class NireProba:
      def __init__(self):
            self.publikoa = 0
            self.__pribatua = 0

      def getPublikoa(self):
                  return self.publikoa

      def setPublikoa(self, balioa):
                  self.publikoa = balioa

      def getPribatua(self):
                  return self.__pribatua

      def setPribatua(self, balioa):
                  self.__pribatua = balioa


      def __metodoPribatua(self):
                  print 'Kaixo, metodo pribatua naiz'

# Objektua sortu
>> nip = NireProba()
# Metodoen bidez lortu
>> print nip.getPublikoa()
0
>> print nip.getPribatua()
0
# Publikoa zuzenean aldatu
```

```
>> nip.publikoa = 17
>> print nip.getPublikoa()
17
# Pribatua ikusten saiatu
>> print nip.__pribatua
<Errorea>
# Pribatua aldatu
>> nip.setPribatua(19)
>> nip.getPribatua()
19
# Metodo pribatuari deitu
>> nip.__metodoPribatua()
<Errorea>
```

Beste klaseetatik herentzia egiteko azpiklasea definitzerakoan parentesi artean agertu behar dira superklase horiek. Klase bat baino gehiagotatik heredatu nahi izanez gero, komarekin banatu behar dira klaseen izenak:

```
class ProbaKlasea(LehenengoAita, BigarrenAita):
      pass # ezer egiten ez duen sententzia
```

4.1.5. Salbuespenak

Salbuespenen kudeaketa egiteko, Pythonek Java programazio lengoaiaren antzeko eredua erabiltzen du, kasu honetan *try*, *except*, eta *raise* hitz erreserbatuak erabiliz.

Programatzaileak nahi dituen salbuespenak inplementatu ditzake *Exception* klasea heredatuz. Kode zati baten salbuespen bat gerta badaiteke *try* batekin mugatuko dugu blokea eta ondoren *except* erabiliko dugu salbuespena harrapatzeko. Beste hitz erreserbatu bat erabili daiteke, *raise*, salbuespenak programatzaileak nahi duenean altxatzeko.

```
# Salbuespen mota berriak sortu
class EzDaZerrenda(Exception):
      pass
class EzDaZenbakia(Exception):
      pass

def probaFuntzioa(zerrenda=[]):
      # parametro gisa datorrena zerrenda bat den egiaztatu
      # mota begiratuz
```

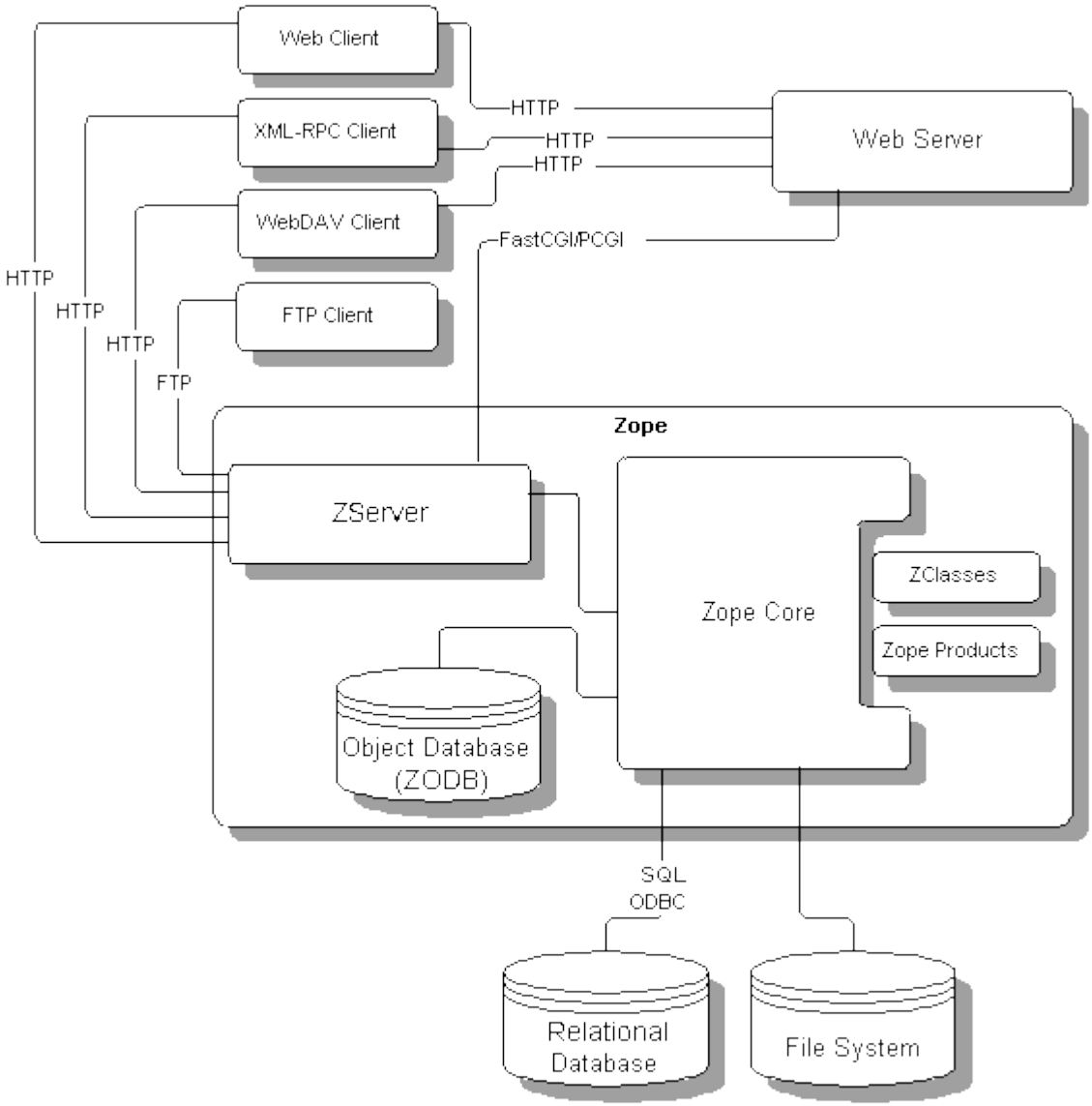
```
if type(zerrenda) != type([]):
      raise EzDaZerrenda
for elementua in zerrenda:
      # Badaezpada elementuren bat zenbakia ez bada
      try:
            print elementua + 1
      except:
            raise EzDaZenbakia, 'Elementu bat ez da zenbakizkoa'
```

4.2. Zope web aplikazioen zerbitzaria

Python programazio lengoaian idatzitako web aplikazioen zerbitzaria da Zope. Objektuei zuzendutako datu-base kudeaketa sistema transakzional bat dauka azpian eta honek ez datuak bakarrik, baita HTML orrialdeak, txantiloi dinamikoak, scriptak eta beste hainbat gauza eduki ditzake.

Zopek integratuta dakartza HTTP, FTP, WebDAV eta XML-RPC zerbitzariak, baina Apache edo beste hainbat zerbitzarirekin elkarbizi daiteke. Hain zuzen ere, normalean, Apache zerbitzari baten atzean kokatzen da Zope zerbitzaria sareari zerbitzu emateko. Irudia 9n ikus daiteke Zoperen arkitektura.

Zopek bi aukera nagusi eskaintzen ditu produktu berriak garatzeko. Alde batetik, hutsetik hasitako produktuak garatu daitezke Python programazio lengoaia erabiliz, eta bestetik Zoperen interfazetik bertatik sor daitezke produktuak, ZClasses deritzanak erabiliz.



Irudia 9: Zoperen arkitektura

Zoperen oinarrizko funtzionalitateak zabaltzen dituzten produktu asko dago sarean, hala nola, albistegiak, kolaborazio atariak, datu-baseekiko konektoreak edo e-salerosketarako produktuak.

Bestalde, egindako edozein produkturen ahalmenak handitu daitezke beraientzako modulu laguntzaileak garatuz (memoria honetan aurkezten den proiektuan eraiki diren inportatzailea eta esportatzailearekin egin dudan bezala) edo jadanik existitzen diren produktuak aldatuz (proiektuan ere eraiki den konpiladorea OpenFlown integratzeko

egin dudan lez).

4.3. XML Pythonez: PyXML paketea

PyXML paketea Python lengoaiarekin XML prozesatzeko Special Interest Group delakoak sortutako paketea da, eta bertan XML tratatzeko aukera ezberdinak eskaintzen dituzten moduluak barneratzen dira.

Pythonek bere oinarrizko instalazioan dakartzan XML tresnak zabaltzen dira pakete honekin eta tresna gehiago eskaintzen du bere tratamendurako. Horregatik erabili dut OpenFlow eta XPDL arteko inportatzailea eta esportatzailea egiteko.

XML fitxategiak tratatzeko bi teknika nagusi daude: DOM eta SAX. Lehenengoak XML dokumentua zuhaitz egiturako datu-mota batean biltzen du eta zuhaitz horretako nodoak atzitzeko metodoak eskaintzen ditu. Bigarrenak XML dokumentuak irakurri ahala egiten du bere lana, irakurritako elementu bakoitzeko metodo bati deituz.

SAX eredua inportatzailea garatzeko orduan erabili dut eta DOM, aldiz, esportazailea garatzeko.

4.3.1. SAX: Simple API for XML

SAX ereduak XML fitxategia irakurri edo *parseatu* egiten du eta aurkitzen duen elementu bakoitzeko eduki kudeatzailearen metodo bati deitzen dio Irudia 10n ikus daitekeen bezala (*ContentHandler* da eduki kudeatzailearen izena aipatutako irudian). Parseatzaileak etiketa bat aurkitzen duenean gertaera bat suertatzen dela esaten da. SAX ereduaren oinarriak [Meggison]en aurki daitezke.

Pythonen SAX inplementazioak, *DefaultHandler* izeneko klase bat eskaintzen du, eta berau erabiltzeko, klase hori heredatu eta erabili nahi diren metodoak berridatzi behar dira. Metodo horien artean garrantzitsuenak *startDocument*, *startElement*, *endElement* eta *characters* dira.

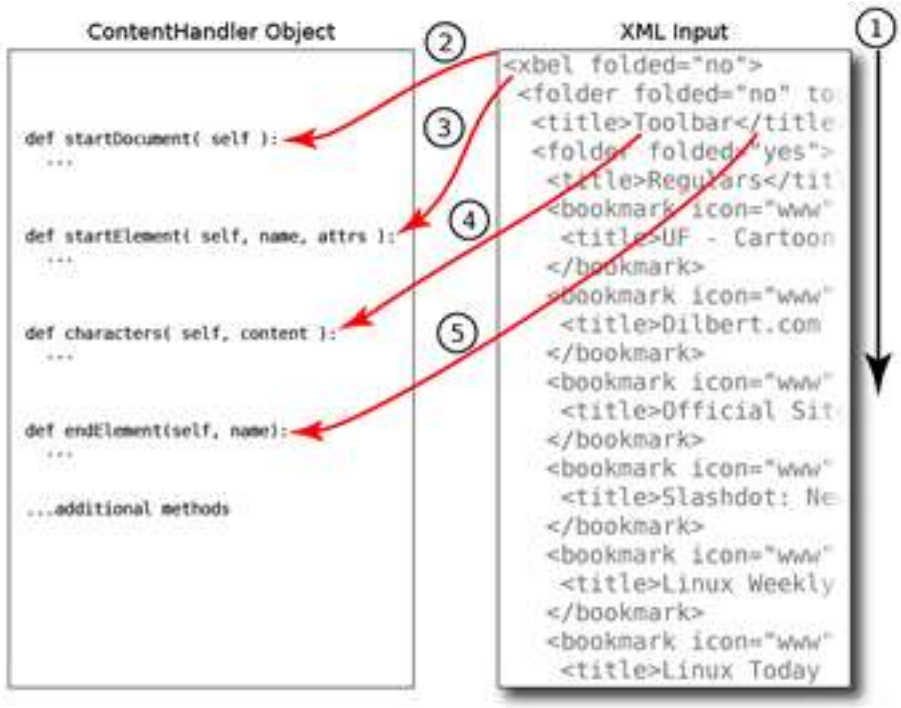
Parseatzaileak fitxategia hasieratik irakurtzen du behin bakarrik (1)[1] eta irakurri ahala deitzen die eduki kudeatzailean inplementatutako metodoei. Dokumentuaren hasiera

1 Testuko zenbakizko erreferentziek Irudia 10eko zenbakiei egiten diete erreferentzia

detektatzen duenean *startDocument* metodoari deitzen dio (2). Etiketa baten hasiera detektatzean (3) eduki kudeatzailean definitutako *startElement* funtzioari deitzen dio parametro bezala etiketa horren informazioa pasatuz. Irudia 10ko adibidean, horrelako dei bat egingo zukeen:

```
handler.startElement('xbel', Attributes({'folded':'no'}))
```

Lehenengo parametroa etiketaren izena eta bigarrenean atributuak eta beren balioak, azken hauen datu-mota 4.1.2. atalean azaldutako hiztegi bat kapsulatzen duen *Attributes* izeneko datu-mota bat izanik.



Irudia 10: SAX prozesamendua (Iturria [Fountain, 2004])

Etiketen arteko testua aurkitzen duenean (4), *characters* metodoari deitzen dio parametro bezala aurkitutako testua pasatuz. Hala ere, funtzio honekin kontuz ibili behar da. Pentsatzekoa da aurkitzen duen testu osoa funtzio dei baten bakarrik tratatzea, adibidez, irudiko adibidean, *handler.characters("Toolbar"),* baina ez dauka zertan horrela izan behar, karaktereen kudeaketa zatika egitea gerta daiteke. Horregatik metodo

honen lana aurkitutako testua gordetzera mugatu behar da.

Testu hau izanda gerta daiteke beherago adierazten diren deiak egitea.

```
<p id="hasiera" lang="eu">Amaia ez da oraindik etorri. Noiz zen
Bartzelonatik etortzekoa?</p>

handler.startDocument()
handler.startElement('p',Attributes({'id':'hasiera', 'lang':'eu'}))
handler.characters('Amaia ez da oraindik etorri. ')
handler.characters('Noiz zen Bartzelonatik etortzekoa')
handler.characters('? ')
handler.endElement('p')
handler.endDocument()
```

Dei hauek nola egiten diren, zatika edo aurkitutako testu guztia batera, parseatzailearen inplementazioaren araberakoa da eta ezin da jakin exekuzio zehatz batean nola egingo duen, ondorioz kontuan hartu beharrekoa da *characters* metodoa inplementatzerakoan.

Bukera etiketa bat aurkitzen duenean (5) *endElement* funtzioari dei egiten dio parseatzaileak, parametro bezala itxitako etiketaren izena pasatuz. Adibideko kasuan *endElement("title")* deia egingo zukeen.

4.3.2. DOM: Document Object Model

DOM eredua, World Wide Web Consortium erakundeak sortutako APIa da, XML dokumentuak atzitu eta aldatzeko edozein programazio lengoaiatarako balio duena [DOM IG]. DOM inplementazio batek XML dokumentua zuhaitz bat balitz bezala erakusten du eta APIaren bidez bere egiturako edozein zati zuzenean atzitzea ahalbidetzen du.

SAX ereduan ez bezala, eredu honetan XML dokumentuaren edukia memorian oso-osorik gordetzen da eta horrek dokumentu handiekin lan egitean arazoak sor ditzake. Bestalde, dokumentua zuhaitz batean edukita, metodo baten deiarekin, egitura osotik informazioa lor daiteke. Adibidez, dokumentuan dauden <*p*> etiketa guztiak lortu nahi baditugu nahikoa dugu hau egitea.

```
document.getElementsByTagName('p')
```

Gainera, lortzen ditugun etiketak, nodo osoak dira, <*p*> horren barruan beste etiketaren bat badago hori ere itzuli egingo digu. Demagun testu hau daukagula *testua* izeneko aldagai batean:

```
<div id="text">
      <p> Urtebeteko ibilbide horretan pauso luze eta ziurrak eman
ditu: <em>21.000 ale inguru saltzen ditu</em> eta <strong>66.500 bat
dira irakurleak.</strong></p>
       <em><strong>Berria</strong></em>
</div>
```

<*em*> etiketapean dagoena lortu nahi badugu zera egingo dugu:

```
# modulua inportatu
>> from xml.dom import minidom
# dokumentua sortu
>> doc = minidom.Document()
>> doc = minidom.parseString(testua)
>> doc
<xml.dom.minidom.Document instance at 0x40586eec>
# Zerrenda batean lortu <em> etiketa duten zatiak eta inprimatu
>> em = doc.getElementsByTagName('em')
>> for i in em:
>>    print i.toxml()
<em>21.000 ale inguru saltzen ditu</em>
<em><strong>Berria</strong></em>
# Gauza bera <strong> etiketekin
>> strong = doc.getElementsByTagName('strong')
>> for j in strong:
>>    print j.toxml()
<strong>66.500 bat dira irakurleak.</strong>
<strong>Berria</strong>
```

PyXML paketeak eskaintzen dituen DOM inplementazio ezberdinak, dokumentu bat parseatzeaz gain, dokumentuetan aldaketak egiteko aukera eskaintzen dute, eta ondorioz dokumentu berriak hutsetik abiatuta sortzeko aukera ere bai.

Horretarako egin behar duguna, dokumentua sortzea da ondoren zuhaitzaren elementuak sortu eta edukiz elikatzeko: azpi-elementu gehiago sartuz, atributuak gehituz, testu-nodoak gehituz, ... Azkenik dokumentuari lehenengo sortutako elementua

gehituko diogu.

```
>> from xml.dom import minidom
# dokumentua sortu
>> xmldoc = minidom.Document()
>> erroa = xmldoc.createElement('erroa')
# atributuak gehitu
>> erroa.setAttribute('Id', 'erroa')
>> erroa.setAttribute('Eguna', '2003-12-31')
>> elem1 = xmldoc.createElement('semea1')
>> elem1_1 = xmldoc.createElement('semearenSemea')
# testu nodo bat sortu
>> text1_1 = xmldoc.createTextNode('Semearen semearen testua')
# testu nodoa gehitu nodo arrunt bati
>> elem1_1.appendChild(text1_1)
<DOM Text node "Semearen s...">
>> elem1_1.toxml()
'<semearensemea>Semearen semearen testua</semearensemea>'
>> elem2 = xmldoc.createElement('semea2')
# nodo bati azpinodo bat gehitu
>> elem1.appendChild(elem1_1)
<DOM Element: semearensemea at 0x4051d3ec>
# erroari gehitu nodo bat
>> erroa.appendChild(elem1)
<DOM Element: semea1 at 0x4051d36c>
# erroari beste nodoa gehitu
>> erroa.appendChild(elem2)
<DOM Element: semea2 at 0x404d5f4c>
# dokumentuari erroa gehitu
>> xmldoc.appendChild(erroa)
<DOM Element: erroa at 0x4051d1cc>
>> print xmldoc.toprettyxml(encoding='UTF-8')
<?xml version="1.0" encoding="UTF-8"?>
<erroa Eguna="2003-12-31" Id="erroa">
        <semea1>
                <semearensemea>
                        Semearen semearen testua
                </semearensemea>
        </semea1>
        <semea2/>
</erroa>
```

4.4. Zope Page Templates

([Kamath]-etik eta [Latteier, Pelletier, McDonough and Sabaini, 2003]-ko 10 eta 14 ataletatik eta C eranskinetik egokituta)

Zope Page Templates (ZPT) delakoak web orrialdeak garatzeko tresnak dira. Programatzaile eta diseinatzaileei lankidetzan orrialde dinamikoak egiten laguntzen diete, azken hauei euren lana, ohiko lan-tresnak alde batera utzi gabe egiten utziz.

ZPTek aplikazioaren aurkezpena eta barnean duen negozioaren logika banatzeko malgutasuna ematen digute, bata aldatzen ari garela bestean interferentziarik sortu gabe.

Dinamikotasun hori lortzeko etiketa guztiek *tal:* aurrizkia daramate, azken finean eta produktuetan integratzeko bidea erraza da.XML izen-eremu bat dena, eta WYSIWYG[2] editore gehienek, ulertzen ez dutenez, ez dute kodea aldatuko txantiloia irekitzerakoan. Zopek aldiz, izen-eremu berezi hori duten etiketak interpretatu eta dagozkien balioak sartuko ditu behar diren lekuan.

```
<title tal:content="here/title">Page Title</title>
```

Adibide horretan, *here/title* ebaluatu ostean lortzen dena sartuko du, *title*-ren barneko testu gisa, *Page Title* ordezkatuz. Adibideko *tal:content* bezala, *tal:define*, *tal:condition*, *tal:repeat*, *tal:replace*, *tal:attributes* eta *tal:omit-tag* ere erabil daitezke ZPTak sortzeko. Hurrenez hurren, aldagaiak definitu, baldintzak ezarri, begiztak egin, etiketak ordezkatu, etiketen atributuak aldatu (HTMLko *a* etiketaren *href* adibidez) eta etiketak kentzeko balio dute.

```
<span tal:repeat="n python:range(10)"
      tal:omit-tag="">
  <p tal:content="n">1</p>
</span>
```

2 "What you see is what you get", editore baten ikusi, eta gero lortzen dena berdinak direnean erabiltzen da.

Adibideak, *range(10)* funtzioaren exekuzioak itzultzen duen elementuen zerrendaren gainean begizta bat egiten du (0-tik hasita lehenengo 10 zenbakiak), une bakoitzean n aldagaian balioa utziz, eta balio bakoitzeko <*p*> etiketa artean zenbakia gordez *tal:content* etiketari esker. Hasierako <*span*> etiketako *tal:omit-tag* etiketarekin, emaitzean <*span*> etiketa ikusi nahi ez dugula adierazten dugu. Ondorioz, goiko ZPTak sortutako HTMLaren iturburua ondokoa litzateke:

```
<p>0</p> <p>1</p> <p>2</p> <p>3</p> <p>4</p>
<p>5</p> <p>6</p> <p>7</p> <p>8</p> <p>9</p>
```

ZPT tresna hau, konpiladorearen modulua garatzerakoan erabili dut bere emaitzak OpenFlowren kudeaketa orrialdeetan ateratzeko, tresna honekin idatzitako fitxategiak Zopeko produktuen kudeaketa interfazeak dinamikoki sortzeko erabil baitaitezke webguneak garatzeko erabiltzeaz gain.

5. OPENFLOW

Memorian aurkezten den proiektuaren garapenean OpenFlow produktua erabili dut. Hala ere, beste lan-fluxu sistema batzuk ere aztertu ditut, eta azterketa horren ondorioz sortutako txostena E eranskin bezala gehitu diot memoriari.

OpenFlow, Icube enpresa italiarrak sortutako produktua da, ekintzetan oinarritutako lan-fluxuak inplementatzeko Zope plataformarako produktua. Workflow Management Coalition [WfMC] erakundearen estandarrak eta nomenklatura jarraitzen ditu bere atalak izendatzean eta Zope plataformak eskaintzen dituen tresnei esker malgutasun handikoa da.

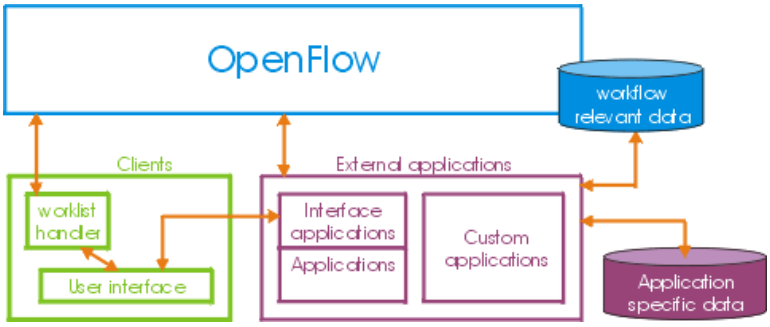
Nahiz eta garatzaileek ez duten erakunde horrekin zerikusirik, WfMC erakundearen lanetan oinarrituta dago OpenFlowren inplementazioa eta horrek asko frogatuta dagoen eredu bat erabiltzea suposatzen du, dituen onurekin.

Web bidezko interfazea eskaintzen du eta horrek lan-fluxua inplementatzeko webak eskaintzen dituen zerbitzu guztiak erabiltzea ahalbidetzen du, hala nola, posta elektronikoko mezuak bidaltzea, enpresaren urruneko ordezkaritzaren lan-fluxuekin lan egitea, ... Gainera Zope plataformarako eginda dagoenez, bertako produktu eta baliabide guztiak erabil daitezke, albisteak argitaratzeko, egindako lanaren historia ikusteko, lan-fluxuaren inguruko txostenak inprimatzeko, ...

Irudia 11n ikus daiteke OpenFlowren garatzaileek OpenFlowren ahalmenak laburtzeko erabiltzen duten grafikoa. Alde batetik, bezeroek erabiltzailearen interfazea erabiliz lan-zerrenden bidez atzitzen dute OpenFlow euren lanak egiteko eta bestetik OpenFlown definitutako aplikazioak erabil ditzake.

Aplikazioek lan-fluxuaren inguruko informaziorako atzipena ere badute beren lana era egokian egiteko.

Beste alde batetik, software librea da eta iturburua denon esku egonik, garatzaileak nahi edo behar dituen aldaketak edo gehikuntzak egin ditzake produktua hobetzeko.



Irudia 11: OpenFlow Zope barnean

Zope plataformarako produktua izanik, Zope instalatu daitekeen plataforma guztietan erabili daiteke OpenFlow, hala nola, Linux, Windows edo MacOS sistema eragileetan.

WfMC erakundeak erabiltzen duen izendatze sistema bera erabiltzen du OpenFlow produktuak. Horretan oinarrituz, OpenFlowk prozesuetan biltzen ditu lan-fluxuen definizioak. Prozesu hauek dituzte lan-fluxuaren muina izango diren ekintza eta trantsizioak, eta ondorioz ekintzei lotutako aplikazioen definizioa, ekintzen arteko trantsizioen irteera eta sarrera babesak eta rolen definizioa.

Hurrengo ataletan zehar azalduko dut OpenFlowk nola definitzen dituen lan-fluxuekin lan egiteko objektuak, nola sortu behar diren eta zein ezaugarri dituzten.

5.1. Lan-fluxuaren alderdi estatikoa

Atal honetan OpenFlown alderdi estatikoaren inguruko ezaugarriak komentatuko ditut: lan-fluxuaren prozesuak, ekintzak, trantsizioak eta rolak. Osagai hauetako bakoitza nola sortzen den ere azalduko dut.

5.1.1. Prozesuak

Inplementatu nahi den lan-fluxu bakoitzeko prozesu bat sortu behar da OpenFlown eta bertan adierazten dira zeintzuk izango diren lan-fluxuaren ekintzak, ekintzen arteko trantsizioak eta ekintzak burutzeko beharrezko diren rolak.

Prozesuak sortzea Zoperen kudeaketa interfazearen bidez egiten da HTML formulario bat erabiliz. Formulario horrek, OpenFlowren APIko funtzio bati deitzen dio prozesua sortzeko eta ondorioz ez da derrigorrezkoa prozesuak formulario bidez sortzea,

APIko funtzioari era egokioan deitzen dion script bat ere egin daiteke. Era honetan egin daiteke *"erreklamazioak"* izeneko prozesu bat sortzeko (lehenengo lerroa funtzioaren definizioa da eta bigarrenekoa deia):

```
addProcess(id, title='', description='', BeginEnd=None, priority=0,
REQUEST=None)
```

```
wf.addProcess('erreklamazioak', 'Erreklamazioen fluxua', BeginEnd=1)
```

Funtzioari deia egiterakoan ez diogu *description* parametroan baliorik eman, eta *BeginEnd* parametroan 1ekoa pasatu diogu, horrekin prozesua sortzearekin batera automatikoki sortuko ditu prozesuaren hasiera eta bukaerako egoerak, horrela 8.2. atalean azaltzen ditudan arazoak saihestuz. HTML formularioaren bidez sortu izan bagenu, aurreko datu horiek eman izan beharko genituzke.

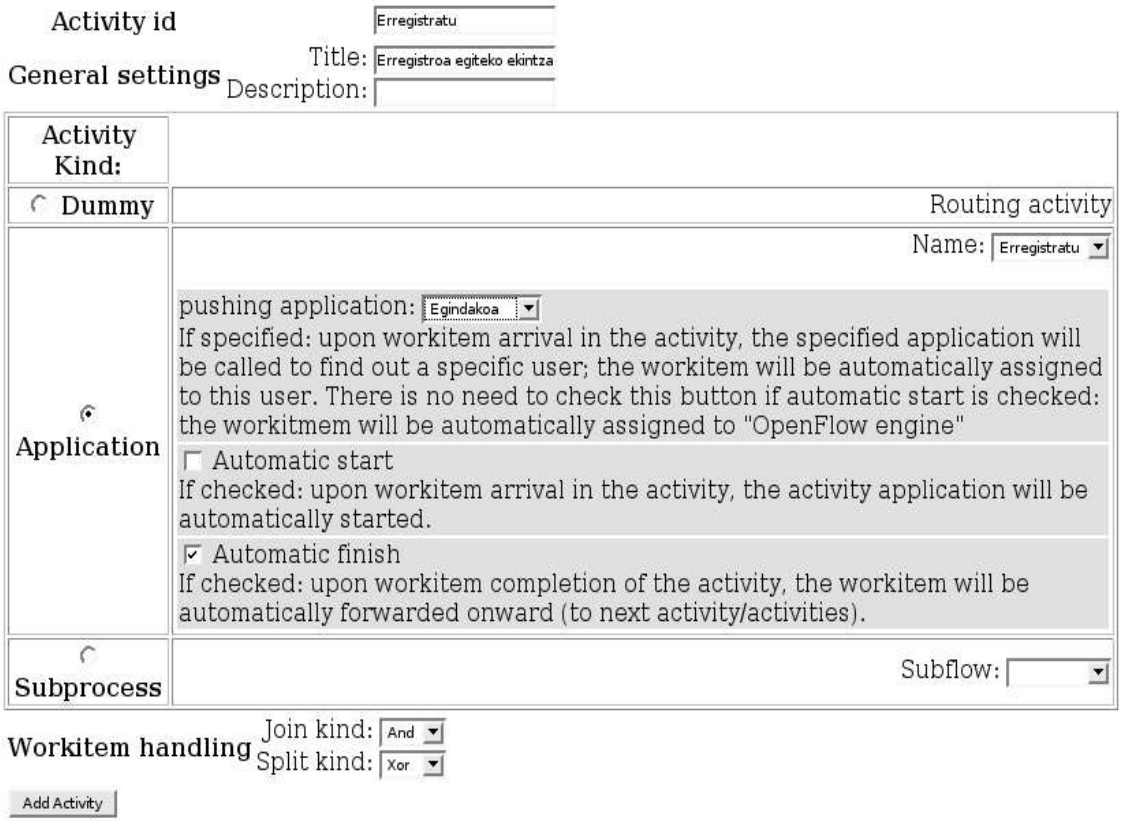
5.1.2. Ekintzak

OpenFlowko ekintzek 3.3.1.1 atalean definitutako ekintzen ezaugarri guztiak betetzen dituzte. Prozesu bati ekintza bat gehitzeko, prozesuaren kudeaketa orrialdeko formularioa bete behar da edo bestela dagokion APIko funtzioari deitu.

Prozesua sortzerakon, identifikadoreaz gain, bere sarrera eta irteeera babesen konfigurazioa nolakoa izango den (*and* ala *xor*) eta zein ekintza mota izango den: ezer egiten ez duena (*route*), azpifluxua (kasu horretan azpifluxua duen prozesuaren izena eman beharko da), edo aplikazio bat lotuta duena. Azken kasu honetan erlazionatutako aplikazioaren izena aukeratu beharko da eta baita ekintzak hasiera edo/eta bukaera automatikoa izango duen ere. Gainera, ekintza zein erabiltzaileri esleitu erabakitzeko aplikazio bat baldin badugu (*push* erako aplikazioa[3]), hori ere adierazi dezakegu.

OpenFlowren kudeaketa interfazeko formularioaren bidez egiten den ekintzaren sorrera ikusi daiteke Irudia 12n.

3 *Push* erako aplikazioak lan-fluxuaren alde dinamikoari dagokion atalean azalduko ditut



Irudia 12: Prozesu baten ekintza bat sortzeko formularioa

Formularioak egiten duen funtzio-deia ondokoa litzateke kasu horretan (lehenengo funtzioaren goiburukoa eta ondoren deia):

```
addActivity(self, id, split_mode='and', join_mode='and',
self_assignable=1, start_mode=0, finish_mode=0, subflow='',
push_application='', application='', title='', parameters='',
description='', kind='standard', REQUEST=None)

process.addActivity('Erregistratu', split_mode='xor', join_mode='and'
self_assignable=0, start_mode=0, finish_mode=1, subflow='',
push_application='Egindakoa', application='Erregistratu',
title='Erregistroa egiteko ekintza', parameters='')
```

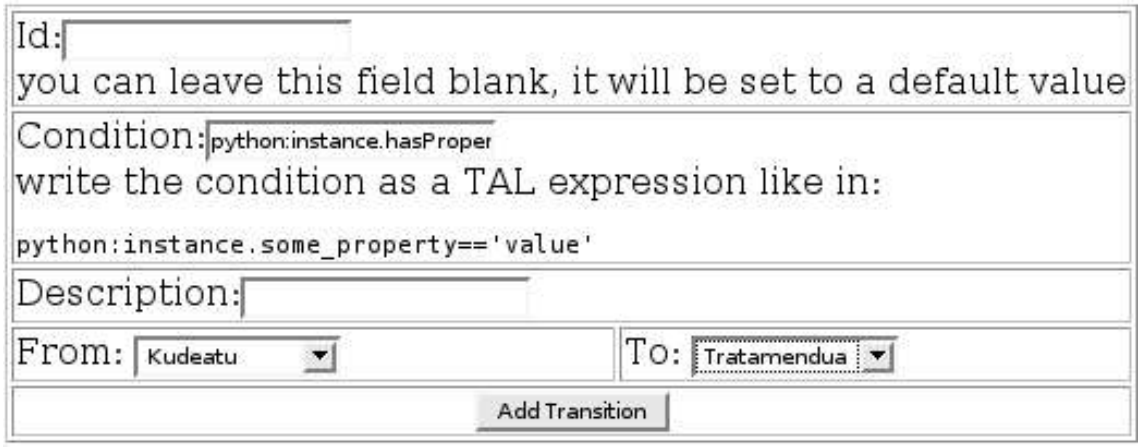
Self_assignable parametroak, 1 balioa hartzen du, ekintza norbaiti esleitzeko aplikaziorik, hau da, *push* aplikaziorik, ez badago. Gainera *parameters* parametroaren balioa beti hutsa izango da, formularioak ez baitu aukerarik ematen horretarako balioak sartzeko. Hala ere, eta nahiz eta APIko funtzioari zuzenean deituz *parameters* horretan

balioak sar ditzakegun, OpenFlowk ez ditu erabiltzen. OpenFlowren aurreko bertsioetan *parameters* hori erabili egiten zen eta orain erabili ez arren, iturburu kodetik kendu gabe dago.

5.1.3. Trantsizioak

Trantsizioek hurrengo ekintzara pasatu ala ez erabakitzeko baldintzak izan ditzakete. Une batean instantzia batek trantsizio honetatik igaro nahi badu, OpenFlow arduratuko da erabakia hartzeko baldintza ebaluatzeaz.

Trantsizioak sortzeko, formularioaren bidez egin dezakegun aurreko kasuan bezala edo bestela APIko metodoei zuzenean deituz.



Irudia 13: Trantsizioa formularioaren bidez sortzen

Irudiko deia metodoarekin egindako deira itzuliz gero honakoa edukiko genuke (lehenengo goiburukoa eta gero deia):

```
addTransition(self, id, From, To, condition=None, description=None,
REQUEST=None)
```

```
wf.addTransition('','Kudeatu', 'Tratamendua',
condition='python:instance.hasProperty('onartua'))
```

Baldintzak *TALES*[4] erako espresioetan idatzi behar dira. Espresio hauek Zopeko

4 Tag Attribute Language Expression Syntax

Zope Page Templates txantiloi sisteman erabiltzen dira Zope objektuak atzitu edo baldintzak ebaluatzeko (ikusi [Latteier, Pelletier, McDonough and Sabaini, 2003]-ko C eranskina erreferentzia osorako eta 4.4. atala *Zope Page Templates*-en azalpenerako).

Trantsizioetan sistema hori erabiltzen da, horrela uneko instantziaren atributuak aztertu baitaitezke baldintzan. Adibidez, demagun uneko instantziak *errepikatuta* izeneko atributu bat baldin badu trantsizio batetik ezin dela pasatu, hori adierazteko trantsizioaren baldintza bezala *not:instance.hasProperty('errepikatuta')* jarri beharko litzateke. Uneko instantzia *instance* bidez adieraziko litzateke eta *hasProperty* metodoak, parametro bezala pasatutako izena duen atributuren bat duenentz itzuliko luke.

TALES adierazpenen erabilera hemen azaltzen dena baino zabalagoa den arren, trantsizioetan erabiliko den sintaxia ez da normalean oso konplexua izango, esaterako *activity, instance, workitem, process* edo *openflow* erabil daitezke, hurrenez hurren, uneko ekintza, instantzia, lan-itema, prozesua edo OpenFlow objektua bera eta bere metodoak atzitu edo erabiltzeko.

```
not:instance.hasProperty('eginda')
python:instance.getProperty('erreklamaziokopurua') > 5
```

Lehenengo baldintzak, uneko instantziak *'eginda'* izeneko propietate bat ez duela egiaztatzen du. Bigarrenak aldiz, instantziaren *'erreklamaziokopurua'* propietatearen balioa 5 baino handiagoa dela egiaztatzen du. Bigarrenak *python* aurrizkia daukala ikus daiteke, hori TALES espresioetan, Python lengoaiako adierazpenak idazteko erabiltzen da. Esate baterako, eragiketa matematikoak dituzten adierazpenak edo espresioen arteko konparazioak egin daitezke.

5.1.4. Rolak

Ekintza bakoitza zein pertsonak aurrera eraman dezakeen mugatzeko, ekintza bakoitzari rol bat esleitu diezaiokegu OpenFlown. Rol horiek, Zopeko rol arruntak dira eta Zopen ohiko moduan sortu behar dira.

Ekintza bati bi eratako rolak esleitu diezazkiokegu: *pull* eta *push* erakoak. Ekintza bati *pull* erako rol bat esleitzen diogunean, ekintza hori egiteko eduki behar den rola zein den definitzen dugu. Aldiz, *push* erako rol bat esleitzean, ekintza bat zein rol duen pertsonari eslei dakiokeen esaten dugu. Esleipen hori geroago azalduko ditudan *push* aplikazioek automatikoki egingo dute.

Roles

Role	Pushable activities	Pullable activities
Administraria	not assigned	assigned
Alorburua	not assigned	not assigned
Anonymous	not assigned	not assigned
Authenticated	not assigned	not assigned
DBConnection	not assigned	not assigned
Idazkaria	assigned	assigned
Irakaslegoa	not assigned	not assigned
KalitateArduraduna	not assigned	not assigned
KexaKudeatzailea	not assigned	not assigned
KexenArduraduna	not assigned	not assigned
Manager	not assigned	assigned
Owner	not assigned	not assigned
Shipper	not assigned	not assigned
Sistema	not assigned	assigned
Tutorea	not assigned	not assigned
Zuzendaria	assigned	assigned
Zuzendariordea	not assigned	not assigned

Irudia 14: Pull eta push rolak esleitzen

Rolak gehitzea Zoperen interfazetik egin behar da ohiko moduan edo bestela script bidez Zoperen APIari deituz, horrela:

```
wf._addRole('Administraria')
```

Behin beharrezko rolak sortuta, Irudia 14n ikusten den OpenFlowren *Role* kudeaketa fitxa erabiliz, ekintza bakoitzari dagokion rola esleituko diogu.

5.2. Lan-fluxuaren alderdi dinamikoa

Alderdi estatikoa aztertu ostean, OpenFlowko lan-fluxu baten exekuzioak dituen ezaugarriak azalduko ditut: instantziak eta lan-itemak zer diren eta zertarako balio duten eta ekintzei lotutako aplikazioak zer diren eta nola idatzi, hain zuzen ere.

5.2.1. Instantziak eta lan-itemak

OpenFlowk, lan-fluxuaren exekuzioa, instantzia deritzon objektu baten errepresentatzen du. Instantzia batek lan-itemak biltzen ditu eta azken finean hauek dira exekuzioaren informazioa gordetzen dutenak.

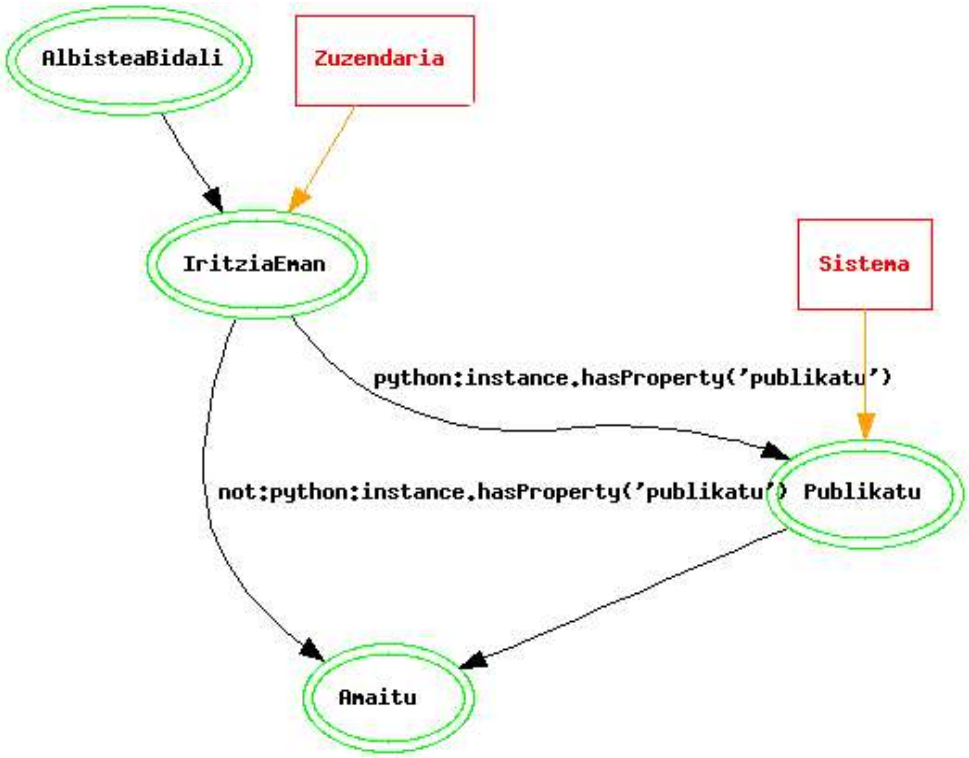
Lan-fluxuaren exekuzio bat martxan jartzen denean instantzia bat sortu behar da. Instantzia ekintza desberdinetatik pasatzen doan heinean, lan-itemak sortzen ditu eta bertan erregistratuta gelditzen da exekuzioari buruzko hainbat informazio. Hala nola, zein ordutan iritsi den ekintza horretara, zein ekintzatatik datorren, zein erabiltzailek exekutatu duen, noiz amaitu den ekintza horretan egin beharrekoa edo zein ordutan bidali den hurrengo ekintzara. Lan-fluxua pasatzen den ekintza bakoitzeko lan-item bat sortzen du eta lan-item guztiak instantzia batean biltzen ditu OpenFlowk.

Lan-fluxu baten exekuzioan zehar eta ekintza batetik beste batera pasatu garela adierazteko erabili behar den APIan, zein lan-itemekin lanean ari garen esan behar diogu sistemari. Hori horrela da, instantzia jakin bat une berean bi ekintzatan edo gehiagotan egon daitekeelako (*and* motako irteera babes baten ondorioz adibidez). Kasu horretan ezin diogu esan *"bidali instantzia hurrengo ekintzara"*, ez bailuke jakingo zein bidali, horregatik sortzen da lan-itemen beharra OpenFlown.

5.2.2. Aplikazioak

Ekintza bat aurrera eramateak aplikazio bat exekutatzea dakar, 3.3.1.3. puntuan azaldu denez. Aplikazioa URL batez dei daitekeen edozer gauza izan daiteke OpenFlown. Normalean formularioak dituzten HTML orrialdeak izango diren arren, beste edozer gauza ere izan daitezke. Hori horrela eginez, Zoperekin lan egitearen

ahalmena ikus daiteke, Zopeko objektuak URL bidez dei baitaitezke eta ondorioz aplikazio gisa erabil daitezke OpenFlown. Horrela, aplikazio gisa datu-base baten SQL sententzia bat exekutatzen duen Pythonez idatzitako script bati, bilaketak egiteko produktua indexatzen duen funtzioari edo albistegi baten albiste berri bat gehitzeko erabiltzen den metodoari dei dakioke.



Irudia 15: Albistegi bateko lan-fluxua

Demagun Interneteko albistegi batean albisteak publikatzeko lan-fluxu bat daukagula, bere grafoa Irudia 15n ikus daitekeena izanik. Erabiltzaile batek albiste bat bidaltzen du albistegian publikatzeko eta horretarako formulario bat bete behar du publikatu nahi duen testuarekin. Formulario hori betetzea eta bidaltzean egiten den prozesamendua, *AlbisteaBidali* ekintzari dagokion aplikazioa exekutatzea izango litzateke.

Ondoren albistegiaren zuzendariak, testuari buruzko bere iritzia emango du eta publikatu nahi den testua eta berak emandako iritzia jasotzen dituen datu-sorta datu-

basean sartuko ditu hori egiten duen SQL sententzia batekin eta instantziari *'publikatu'* propietate bat gehituko dio testua publikagarria dela irizten badu. Prozedura hori betetzea, *IritziaEman* ekintzari lotutako aplikazioa exekutatzea izango da.

Albistea publikagarria baldin bada, sistema bera arduratuko da publikazioa automatikoki egiteaz, hori egiten duen metodoari automatikoki deituz, parametro gisa publikatu beharreko testua eta gainontzeko datuak pasatuz. Aplikazio automatiko hori *Publikatu* ekintzari lotuta egongo litzateke.

OpenFlowk bi aplikazio mota definitzen uzten du: *pull* eta *push* motako aplikazioak. Lehenengoak orain arte azaldutako aplikazio arruntak dira, ekintza bati lotutako aplikazioak eta ekintza burutzeko exekutatu behar direnak. Grafikoki ikusteko, kutxa batetik hartuko balira bezala egiten delako deitzen zaie *pull*. Adibidez, aurreko adibidea hartuz, albistegiko zuzendariak orrialde baten ikus ditzake jendeak publikatzeko bidali dituen albiste guztiak eta nahi duena hartuko du bere iritzia emateko.

Push erakoak, aldiz, lana automatikoki esleitzeko erabiltzen dira. Lehen esan dudan bezala, OpenFlown ekintza batzuk rol zehatz bat duten erabiltzaileek eraman ditzakete aurrera, horretaz gain instantzia konkretu baten ekintza zehatz bat zein erabiltzailek egin behar duen ere definitu daiteke. Lan hori eskuz egin daitekeen arren, APIko *assignWorkitem* funtzioari deituta, erabiltzaile baten izena itzultzen duen aplikazio batekin automatizatu daiteke.

Adibidez, demagun gure albistegiak zuzendari bat baino gehiago duela eta bakoitzak egindako lana orekatu nahi dutela. Era honetako aplikazio bat definitu daiteke: albiste berri bat dagoen bakoitzean, datu-basetik informazioa atera eta une horretara arte lan gutxien egin duen zuzendariari lana automatikoki esleitu. Zuzendaria sistemara sartzen denean, lan bat egiteke daukala abisatuko lioke sistemak. *Push* izena daukate, norbaitek erabiltzaile bati aplikazioa *"bultzatzen"* edo "*bidaltzen"* diolako.

Push aplikazioa egiteko, dagokion ekintza nori esleitu erabakiko duen funtzioa inplementatu behar da Zopeko objektu baten, Pythonez idatzitako script baten, SQL sententzia bat exekutatzen duen metodo baten, ... Edozein modutan ere, eta aplikazioa

instantzia ekintza horretara iristen den momentuan exekutatuko denez, ez du erabiltzailearekiko elkarrekintzarik izan behar. Aplikazioak erabiltzaile baten izena itzuliko du eta OpenFlowk automatikoki esleituko dio ekintza hori erabiltzaileari.

5.2.3. Lan-zerrendak

Lan-fluxuaren erabiltzaileei egiteke dituzten lanak era egokian erakutsi behar zaizkie eta horretarako bide arruntena lan-zerrendak antolatzea izaten da. Sistemako erabiltzaile bakoitzari egiteke dituen lanen zerrenda erakutsi behar zaie bere lana egin dezaten.

Lan-zerrendak sortzeko, lan-fluxu kudeaketa sisteman une bakoitzean dauden instantzia guztien informazioa behar dugu, uneoro horietakoren bat norbaitek aktibatuta duen ala ez jakiteko.

Horretarako *ZCatalog* deritzon produktu bat darabil OpenFlowk. *ZCatalog* Zope barneko bilatzaile bat dela esan daiteke. Objektu ezberdinak motaren arabera, beraiek duten edukiaren arabera edo erabiltzaileak definitutako indizeen arabera sailkatu ditzake. Horretaz baliatuz OpenFlowko egitura guztiak (prozesuak, ekintzak, trantsizioak, lan-itemak eta instantziak) automatikoki indexatzen dira *ZCatalog* baten [Zope Catalog].

Lan-zerrenda sortzeko garaian *ZCatalog* egitura horretatik lor dezakegu informazioa erabiltzaileari egiteke edo hasita baina bukatu gabe dituen lanen zerrenda erakusteko. Esate baterako, erabiltzaile bati erakutsi beharreko lan-zerrendan honako instantzien informazioa jar dezakegu (kasu bakoitzaren ostean informazio hori katalogotik ateratzeko erabili daitekeen kode zatiak jarri ditut, *Catalog* izanik katalogoaren izena):

- Ekintza egikaritu dezaketen rolen artean erabiltzailearenak dituztenak eta gainera oraindik inork hartu gabe daudenak (inaktiboak).

```
wis1 = Catalog(meta_type='Workitem', pull_roles=username.getRoles(),
status=['inactive'], actor='')
```

- Ekintza egikaritu dezaketen rolen artean erabiltzailearenak dituztenak eta inaktibo izanda berari esleituta daudenak.

```
wis2 = Catalog(meta_type='Workitem', pull_roles=username.getRoles(),
status=['active'], actor=username)
```

- Ekintza egikaritu dezaketen rolen artean erabiltzailearenak dituztenak eta aktibo egonda berari esleituta daudenak.

```
wis3 = Catalog(meta_type='Workitem', pull_roles=username.getRoles(),
status=['inactive'], actor=username)
```

Katalogoari kontsulta egiterakoan, zein motako objektuak nahi ditugun esan behar diogu eta gainera propietateren baten balio zehatz bat izan behar badute, hori ere zehaztu egin behar diogu.

Behin katalogotik informazio hori lortuta, *wis1*, *wis2* eta *wis3* zerrendetan ditugun objektuen informazioa erakutsi behar dugu pantailan eta bakoitzari lotutako aplikazioari lotura egin, behar diren parametroak (instantziaren, lan-itemaren eta OpenFlow objektuaren identifikadoreak gutxienez) URLan pasatuz, HTTPko GET metodoa erabiliz. Adibidez:

```
for workitem in wis1:
      i_id = workitem.instance_id
      w_id = workitem.id
      app_url = wf.getApplicationUrl(i_id,w_id)
      link_url = '<a href="%s?i_id=%s&w_id=%s&o_id=%s">%s</a><br/>' %
(app_url, i_id, w_id, wf.id, i_id)
      ret = ret + link_url
return ret
```

Python script horrek *wis1* zerrendan gordetako elementu bakoitzeko dagokion lotura sortzen du parametro gisa instantziaren, lan-itemaren eta OpenFlowren identifikadoreak pasatuz.

Lotura sortzean erabiltzen den *%s* erabiltzearen nomenklatura, C programazio lengoaian printf eta antzeko funtzioetan erabiltzen den berbera da. String batean testu edo zenbakiren bat sartu nahiez gero *%s* edo *%d* jarriz markatzen da eta string-aren bukaeran *%* ikurraren ondoren jartzen dira hutsune horiek beteko dituzten balioak.

5.2.4. Salbuespenen kudeaketa eta egoera bereziak

Lan-fluxu sistema batek ahal den moldagarriena izan behar du. Era erraz batean kudeatu behar dira gerta daitezkeen egoera bereziak eta lan-fluxua aldatzen ere utzi behar du.

Instantzia baten exekuzioan zehar egoera bereziak gerta daitezke, hala nola, baldintzaren bat ez betetzea edo arazo teknikoetatik at dagoen zerbait gertatzea, uneko instantziaren exekuzioa behin-behingoz alde batera uztera derrigortuz. Hori kudeatzeko API funtzioak erabil daitezke OpenFlown, horretarako daude *falloutWorkitem, endFallinWorkitem* edo *fallinWorkitem* funtzioak. Funtzio hauek, lan-itema egoera berezi batean uzten dute eskuz egin beharreko aldaketa edo bestelako egiaztapen batzuen zain.

Egiaztapen edo aldaketa horiek nola egiten diren ez du OpenFlowk zehazten eta ez du horretarako mekanismorik eskaintzen kasuistika oso zabala delako. Horregatik, askotan, kasu berezietaz arduratzen den kudeatzaile bat, pertsona bat azken finean, definitzen da lan horiek aurrera eramateko.

Adibidez aurreko ataleko lan-zerrendetan, aplikazioa exekutatzeko URLarekin batera, lan-itema salbuespen egoeran uzten duten funtzioetarako deien loturak ager daitezke sistemaren erabiltzaileak egoera berezi bat gertatu dela uste badu.

Beste adibide bat, lan-itema automatikoki salbuespen egoerara bidaltzen deneko kasua izan daiteke. Esaterako, har dezagun Interneteko albistegiko lan-fluxua (ikusi Irudia 15). Demagun *Publikatu* ekintza egiten duen aplikazioak, albistea argitaratzeko metodoari deitzen dionak, erroreren bat detektatzen duela programaren exekuzioan salbuespen bat altxatu delako. Kasu horretan lan-itema salbuespen edo *fallout* egoerara bidaliko du.

```
try:
    # galdera exekutatu eta dena ondo badoa propietatea gehitu,
    # lana bukatutzat eman eta hurrengora bidali
    context.sqlConnection.execute(galdera)
    if publikatu != 'Ez':
        # publikagarria denez, hori dioen
```

```
        # propietate bat gehitu instantziari
        instance.manage_addProperty('publikatu',1,boolean)

    wf.completeWorkitem(instance_id, workitem_id)
    wf.forwardWorkitem(instance_id, workitem_id)
    return 'Albistea era egokian gorde da datu basean \
            eta automatikoki publikatu da'
except:
    # salbuespena altxatu da, ondorioz salbuespen
    # egoerara bidaliko dugu lan-itema
    wf.falloutWorkitem(instance_id, workitem_id)
    return 'Erroreren bat gertatu da eta albistea \
            salbuespen egoeran gelditu da'
```

Lan-item baten aurreko egoera berreskuratzeko *fallinWorkitem* funtzioari deitu behar zaio eta lan-itemak zein lekutatik aurrera jarraituko duen adierazi. OpenFlowk salbuespen egoeran dagoen lan-item bat lan-fluxuko beste edozein ekintzatatik jarraitzea baimentzen du, APIko funtzioari era egokian deituz. Gainera, behin baino gehiagotan dei diezaiokegu funtzio honi lan-item berberaren datuak emanez, horrela lan-itema ekintza bat baino gehiagotan berrabiarazi eta fluxu paralelo bat abiatuz.

Albisteen argitalpenaren adibidearekin jarraituz, salbuespen egoeran dauden lan-itemen zerrenda erabiltzaile berezi batek ikusiko duela suposa dezagun. Kasu horretan, bere lan-zerrendan, egoera berezian dauden lan-itemen zerrenda ikusiko du eta *IritziaEman* edo *Amaitu* ekintzetara bidali ahal izango ditu lan-itemak. Horretarako horrelako deiak egingo lituzke:

```
wf.fallinWorkitem(instance_id, workitem_id, process_id, activity_id)
wf.endFallinWorkitem(instance_id, workitem_id)
```

Lan-itema beste ekintza batera bidaltzerakoan, ekintza horren identifikadorea (kasu honetan *IritziaEman* edo *Amaitu*) eta berau zein prozesutakoa den ere adierazi behar zaio funtzioari deitzerakoan. Horretaz gain, salbuespen egoera bukatu dela adierazi behar dugu *endFallinWorkitem* funtzio horri deituz. Hau horrela egitean, lan-item horren exekuzioa eta salbuespen egoera bukatu egin dela adierazten dugu, gainera, salbuespen egoera batetik datorrenez, ez du bere exekuzioak jarraituko, hau da, ez da aktibatuko lan-fluxuaren definizioari jarraituz hurrengo aktibatu beharko litzatekeen

ekintza.

Laburpen gisa, honako hiru pauso hauek eman behar dira salbuespenak kudeatzerakoan:

1. Lan-itema salbuespen edo *fallout* egoerara bidali *falloutWorkitem*-i deituz.
2. Lan-itema nahi adina aldiz errekuperatu *fallinWorkitem* funtzioari nahi adina aldiz deituz.
3. Salbuespen egoeraren bukaera seinaleztatu *endFallinWorkitem* funtzioari deituz

Egoera berezietaz gain, askotan jadanik martxan dagoen lan-fluxu baten aldaketak egin behar izaten dira. OpenFlowk lan-fluxuaren grafoa edozein unetan aldatzen uzten du instantzietan inolako aldaketarik egin gabe eta instantziak eta euren informazioa galdu gabe. Aldaketak egin ahala gelditzen dira finkatuta sisteman eta instantzia berriak berehala joan daitezke aldatutako ekintzetan zehar inolako arazorik sortu gabe. Gainera ikutu ez diren ekintzetan dauden instantziak egin diren aldaketetaz ez dira ohartzen.

Bestalde, instantzia bat ezabatzen dugun ekintza batean baldin badago, instantzia ez da desagertzen, egoera berezi batean gelditzen da, *fallout* egoeran hain zuzen ere. Gero dagokion APIko funtzioari deituz berreskura daiteke instantzia, nahi den ekintzan utziz.

Salbuespenen kudeaketa eta lan-fluxuaren berdiseinuen inguruan, OpenFlow oso moldagarria dela esan daiteke.

5.2.5. Workflow Relevant Data

Lan-fluxuaren exekuzioan zehar, *workflow relevant data* delakoa mantendu behar du lan-fluxu sistemak [WfMC] erakundearen estandarraren arabera. Hori, une bakoitzean erabaki zuzenak hartzeko beharrezko informazioa da, eta adibidez, trantsizio bateko baldintza ebaluatzeko erabil daiteke. OpenFlowk ez dauka informazio hori errepresentatzeko modu zehatzik, azken finean erabiltzailearen esku uzten baitu nondik hartu informazio hori: uneko instantziaren propietateetatik, Zopeko objekturen batetik ...

Hala ere, informazioa exekuzio konkretu bati buruzkoa bada, datu horiek errepresentatzeko modurik egokiena instantziaren propietateak erabiltzea da. Adibidez erreklamazioa onartu duen pertsonaren arabera geroagoko ekintza baten lanen bat egin

behar bada, datu hori instantziari dagokio eta bertan gordetzea litzateke egokiena. Bestalde, adibidez, lana lan-fluxuaren erabiltzaileen artean era orekatuan banatzen duen *push* erako aplikazio baterako, egokiagoa izango litzateke bakoitzak egiten duen lana datu-base batean gordeta izatea.

5.2.6. Bestelakoak

Lan-fluxuetan ohikoa den beste gauza bat, ekintzak burutu ahal izateko denbora mugak ezartzea da. Adibidez espediente baten kudeaketarako lan-fluxuan, *"jakinarazpen publikorako epea zabaltzen denetik 7 eguneko epean inork ez badu kexarik aurkeztu, espedientearen tramitazioak aurrera jarraituko du automatikoki"*, baldintza izan dezakegu.

Denborazko baldintza horiek zuzenean ezartzea ez da posible OpenFlown. Hala ere Zoperi esker lor daiteke. Alde batetik horretarako zehazki garatuta dauden produktu batzuk erabiliz edo bestela, Linux/Unix erako sistema eragileak erabiliz gero, *cron* deabruarekin ataza bat inplementatuz.

5.3. Nola erabili OpenFlow aplikazioen inplementazioan

Atal honetan OpenFlown lan-fluxu baten inplementazioa egiteko pauso garrantzitsuenak azalduko ditut, C eranskinean zerrendatutako APIko funtzioak erabiliz.

OpenFlowrekin lan-fluxu batzuk inplementatu ostean, gehienetan patroi hauek jarraitu izan ditut eta uste dut egokiak direla era erraz batean aplikazioak diseinatzeko. Adibide bezala inplementatu ditudan lan-fluxu gehienek erabiltzen dituzten aplikazioak formularioak izaten dira. Aplikazio bat "exekutatzea" formulario bat betetzea izango da.

Azalpenak emateko Python lengoaian idatzitako scriptak erabiliko ditut.

Behin inplementatu beharreko lan-fluxuaren diseinua eginda dugula, prozesu hori OpenFlown islatu beharko dugu, banan banan ekintzak eta trantsizioak OpenFlowren kudeaketa infterfazetik sortuz, *OpenFlow Process Editor* tresna erabiliz edo prozesu,

ekintza eta trantsizioak sortzeko APIko metodoei deituz.

Lehenengo modua da sinpleena eta OpenFlow produktua bakarrik edukita egin daitekeena. Formularioen bidez prozesu, ekintza eta trantsizioen inguruko datuak eskatzen ditu OpenFlowk eta ondoren datu horiek erabiltzen ditu dagokion metodoari deituz behar diren objektuak sortzeko.

Bigarren tresna, [Icube]-tik deskargatu daiteke eta ekintza eta trantsizioen sorrera era grafiko batean sortzea ahalbidetzen du. Horretarako Zoperako produktu bat eta sistema eragilean instalatu behar den software bat eskatzen ditu. Edozein modutan, gauza sinpleak egiteko balio du, hala nola, ekintzak eta trantsizioak sortu, rolei ekintzak esleitu eta lan-fluxua era grafiko batean ikusteko. Ekintzak konfiguratzea (aplikazioak, sarrera eta irteera babesak, ...) edo trantsizioen baldintza aldatzea, kudeaketa formularioen bidez egin behar da.

OpenFlowren metodoei zuzenean deitzea lan nekagarriena da baina egokia izan daiteke beste objektu batzuetan errepresentatuta dauden lan-fluxuak era automatikoan OpenFlown integratzeko. Adibidez, metodo hau erabili dut XPDL fitxategi batetik lan-fluxuen egitura OpenFlowra inportatzeko.

Lan-fluxuaren diseinua egitean ez gara aplikazioetaz arduratuko, lan-fluxuaren grafoa era egokian diseinatuko dugu (sarrera eta irteera babesak kontuan hartuz) eta lan bakoitza dagokion rolari esleituko diogu.

5.3.1. Aplikazioak sortu

Aplikazioak OpenFlowren kudeaketa interfazearen bidez sortu behar dira eta beren identifikadorea eta aplikazioaren URLa eman.

Ondoren emandako URLan, aurretik existitzen ez bada, aplikazioa programatu beharko da dagokion objektua sortuz: HTML formularioa, Python scripta edo dena delakoa.

Ondoren aplikazioa esleitu nahi diogun ekintzaren kudeaketa fitxara joan behar gara eta aplikazio hori esleitu.

Adibide gisa, aurreko atalean erabili dudan Interneteko albistegiaren adibidea erabili

ko dut. Bertako *IritziaEman* ekintzari lotutako aplikazioaren kode zatiak emanez.

5.3.2. Instantzien hasieraketa

Lan-fluxu bat hastean lehenengo egin behar den pausoa, lan-fluxuaren instantzia bat sortzea da. Horretarako zein prozesuren exekuzioa nahi dugun jakin behar dugu eta funtzioari parametro bezala izen hori pasatu. Kasu honetan ezin dut *IritziaEman* ekintzaren adibidea erabili, ez baita lan-fluxu lehen ekintza, ondorioz lan-fluxu horretako lehenengo ekintzaren kodea erabiliko dut.

```
(...)
# OpenFlow objektuaren erreferentzia jaso
wf = getattr(context, openflow_id)
instance_id = wf.addInstance('Publikazioa', '','', '', 0)
(...)
```

Momentu honetan instantzia sortuta dago eta *instance_id* aldagaian daukagu bere identifikadorea. OpenFlowk sortzen ditu identifikadore horiek sistemaren ordua erabiliz baina OpenFlowren ondorengo bertsioetan instantzia bati beste era bateko identifikadorea esleitzeko aukera izango da.

Instantzia sortu ostean, martxan jarri behar dugu, ekintzarik ekintza bere bidea has dezan:

```
(...)
wf.startInstance(instance_id)
(...)
```

5.3.3. Lan-itemak erabiltzen

Behin instantzia sortuta dagoenean edo aurreko lan-itemak bere lana bukatu duenean, uneko lan itemarekin lanean hasi behar gara. Lehenengo eta behin lan itema aktibatu egin behar dugu. Lan-itemen funtzionamenduak desberdintasunak ditu ekintzak konfiguratuta dituen hasiera eta bukaera moduen arabera, eta kasu horiek ere azaldu egingo ditut.

Ekintzaren hasiera automatikoki egiteko konfiguratuta badago, lan-itema automatikoki aktibatuko du sistemak eta erabiltzaile gisa *openflow-engine* ezarriko dio, lan-itemaren aktibazioa sistemaren esku gelditu dela adierazteko.

Ekintzaren hasiera eskuz egin behar bada, aplikazioa exekutatzerakoan adibidez, eskuz seinaleztatu behar diogu sistemari, era honetan APIko funtzioari deituz:

```
(...)
wf.activateWorkitem(instance_id, workitem_id, username)
(...)
```

Aplikazioaren exekuzioan sortutako daturen bat instantziari propietate gisa gehitu nahi izanez gero, hau da gure kasua albistegian, lehenik instantziaren objektua lortu behar dugu eta ondoren propietatea gehitu:

```
(...)
if publikatu != 'Ez':
        # publikagarria denez, hori dioen
        # propietate bat gehitu instantziari
        instance.manage_addProperty('publikatu',1,boolean)
(...)
```

Aplikazioarekin egin beharreko lana amaitu dugunean, lan-itema gehiago ez dugula erabiliko adierazi behar dugu. Hemen ere bi kasu bereizten dira lan-itema hasieratzean bezala. Ekintza automatikoki amaitzeko konfiguratu baldin badugu, lan-itema bukatu dela seinaleztatzen dugunean (*completeWorkitem* metodoari deituz) beste egitekorik ez dugu izango. Bestalde, eskuz amaitu behar bada, hurrengo ekintzara bidaltzeko metodoari deitu beharko diogu.

```
(...)
wf.completeWorkitem(instance_id, workitem_id)
wf.forwardWorkitem(instance_id, workitem_id)
(...)
```

Metodo guztiei instantzia eta lan-itemaren identifikadoreak pasatu behar zaizkie parametro gisa eta beren balioak sistematik hartu behar dira. Ekintzen hasiera automatikoa baldin bada, parametro horiek eta OpenFlow objektuaren identifikadorea

automatikoki pasatzen dizkio aplikazioari dei egitean. Hasiera eskuz egin behar bada, normalean lan-zerrenda baten klik eginda, parametro horiek URLan pasatu beharko dira, lan-zerrendak nola osatu azaldu dudan atalean ikusten den moduan.

Ekintza automatikoki hasteko konfiguratuta baldin badago, ordea, lan-itemaren, instantziaren eta OpenFlow objektuaren identifikadoreak sistemak pasatzen dizkio aplikazioari automatikoki.

5.3.4. Adibide osoa

Erabili dudan adibidearen kode osoa dator jarraian.

```
# Zopek REQUEST aldagaian gordetzen ditu objektuei deia
# egiterakoan pasatzen diren parametroak

# Era berean 'container' eta 'context' aldagai berezien
# bidez, uneko scripta barnean daukan karpetaran erreferentzia
# eta scripta exekutatzen den ingurunearen inguruko erreferentziak
# gordetzen ditu

# Ez diogu APIko activateWorkitem metodoari deitu behar.
# Aplikazio hau automatikoki hasteko konfiguratuta dagoelako

request = container.REQUEST

# instantzia, lan-item eta openflow objektuaren
# erreferentziak lortu
instance_id = request.i_id
workitem_id = request.w_id
openflow_id = request.o_id

# OpenFlow objektuaren erreferentzia lortu APIko funtzioei
# deitu ahal izateko
wf = getattr(context, openflow_id)

# instantziaren erreferentzia jaso atributu gisa
# publikatuko den ala ez gehitzeko
instance = wf.getInstance(instance_id)

# Argitalpenaren datuak jaso
eguna      = request.eguna
zuzendaria = request.zuzendaria
egilea     = request.egilea
izenburua  = request.izenburua
testua     = request.testua
```

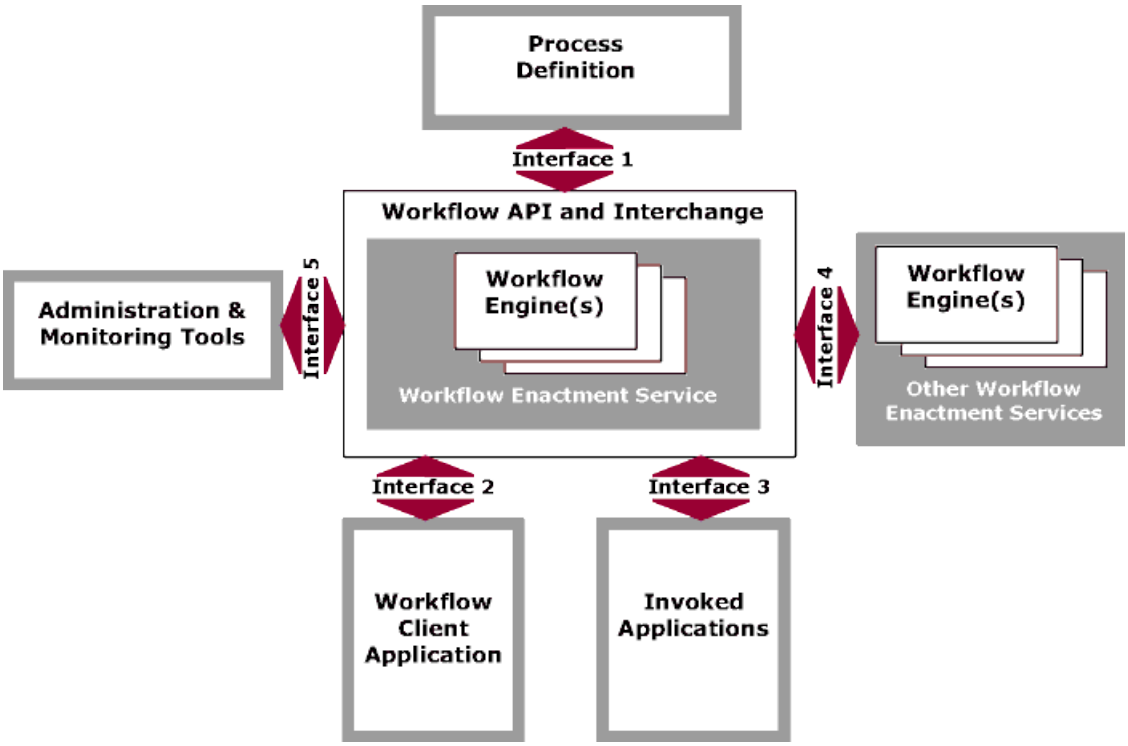
```
# Zuzendariaren iritzia jaso
iritzia = request.iritzia

# Formularioko publikatu aukeran Bai ala Ez erantzun duen jaso
try:
    # Checkbox moduko eremua denez aukeratu gabe
    # utzi bada ez da eskaeran bidaltzen
    publikatu = request.publikatu
except:
    publikatu = 'Ez'


sqlGaldera  = 'INSERT INTO publikazioak(eguna, zuzendaria, iritzia, '
sqlGaldera += 'egilea, izenburua, testua, publikatu) VALUES '
sqlGaldera += '("%s", "%s", "%s", "%s", "%s", "%s", "%s")'

# definitutako hutsuneen ordezkapena egin dagozkion balioak sartuz
galdera = sqlGaldera % (eguna, zuzendaria, iritzia, \
                        egilea, izenburua, testua, publikatu)

try:
    # galdera exekutatu eta dena ondo badoa propietatea gehitu,
    # lana bukatutzat eman eta hurrengora bidali
    context.sqlConnection.execute(galdera)
    if publikatu != 'Ez':
        # publikagarria denez, hori dioen
        # propietate bat gehitu instantziari
        instance.manage_addProperty('publikatu',1,boolean)

    wf.completeWorkitem(instance_id, workitem_id)
    wf.forwardWorkitem(instance_id, workitem_id)
    return 'Albistea era egokian gorde da datu basean \
            eta automatikoki publikatu da'
except:
    # salbuespena altxatu da, ondorioz salbuespen
    # egoerara bidaliko dugu lan-itema
    wf.falloutWorkitem(instance_id, workitem_id)
    return 'Erroreren bat gertatu da eta albistea \
            salbuespen egoeran gelditu da'
```

6. XPDL LENGOAIA

XPDL lengoaiari buruzko kapitulu honetan, lengoaia honek zertarako balio duen eta zein funtzionalitate eskaintzen dituen azalduko da.

6.1. Sarrera

Workflow Management Coalition erakundeak argitaratutako estandar bat da XPDL lengoaia, WfMC erakundearen lan-fluxuen erreferentzia ereduaren 1go interfazearen espezifikazioa dena hain zuzen ere (ikusi Irudia 16 eta [WfMC-RM, 1995]) .



Irudia 16: Lan-fluxuen erreferentzia eredua WfMCren arabera [WfMC]-tik aterata

Ereduak lan-fluxuaren alde estatikoa eta dinamikoaren arteko bereizketa garbi bat egiten du, estatikoan bere indarrak bilduz. Ereduaren helburua, lan-fluxuen definizioa atzitu eta egiteko eredu amankomun bat eskeintzea da, leku eta sistema ezberdinetan gordetako lan-fluxuak beste sistema batera erraz garraiatu eta han inportatzeko.

Eskaintzen duen beste funtzionalitate bat, lan-fluxu sistema baten kudeatzaileak

berari interesatzen zaizkion atributuak gehitzeko aukera da. Horrela estandarrean adierazi ezin diren funtzionalitate gehigarriak erakusteko.

6.2. Definitzen dituen etiketa nagusiak eta esanahia

Atal honetan, XPDLren definizioak eskaintzen dituen etiketa nagusien azalpena egingo dut. Erreferentzia osoa ikusteko, A eranskinean XPDL definitzen duen XML-Schema fitxategiaren edukia dago eta [WfMC-XPDL, 2002] dokumentuan informazio gehiago lor daiteke.

6.2.1. Package

Etiketa hau da XPDL dokumentu baten erro-etiketa. Bertan biltzen dira erlazionatuta dauden eta aplikazio edo erabiltzaile berberak dituzten prozesu ezberdinak.

Package etiketa batek, bere identifikadoreaz gain, bere barnean *PackageHeader* etiketa bat izan behar du bertan erabili den XPDLren bertsioa, fitxategia sortu deneko ordu eta eguna eta sortu duen tresnaren informazioa biltzen duena.

```
<Package Id="OpenFlow">
      <PackageHeader>
            <XPDLVersion>1.0</XPDLVersion>
            <Vendor>openflow2xpdl</Vendor>
            <Created>Wed Jan 23 17:47:21 2004</Created>
      </PackageHeader>
      (...)
</Package>
```

6.2.2. Participant

Paketearen definizioaren ostean, bertan errepresentatuko diren lan-fluxuen erabiltzaileen informazioa bildu behar da. Bertan, erabiltzailearen identifikadorea eta izena atributuez gain, *ParticipantType* etiketa bat agertu behar da eta bere *Type* atributuan erabiltzailearen mota. Erabiltzaile motak rolak (ROLE), pertsonak (HUMAN), sistemak (SYSTEM), departamentu bat (ORGANIZATIONAL_UNIT), baliabide bat (RESOURCE) edo baliabide multzo bat (RESOURCE_SET) izan daitezke.

Horrelakoa litzateke rol bati legokion etiketa:

```
<Participant Id="Administraria" Name="Administraria">
      <ParticipantType Type="ROLE"/>
</Participant>
```

Erabiltzaileak definitzen dituzten *Participant* etiketa hauek, *Participants* etiketa nagusiago baten azpian bildu behar dira.

6.2.3. Application

Ekintzetan erabiliko diren aplikazioak aurretik erazagutu egin behar dira, atal honetan hain zuzen ere. Identifikadore eta izenaz gain, bere parametro formalen definizioa (parametro zenbakia, mota eta sarrerako, irteerakoa ala sarrera-irteerakoa den) edo kanpoko erreferentzia bat eman behar dugu. Kanpoko erreferentzia batekin, beste lekuren batean erazagututa dagoen aplikazio bat erabiliko dugula adierazten ari gara. Adibidez erabiliko dugun aplikazioa web zerbitzu bat baldin bada eta bere WSDL dokumenturako helbidea baldin badugu, helbide hori jarriko dugu kanporako erreferentziaren lekuan.

Application etiketa hauek *Applications* goragoko mailako etiketa baten barruan joan behar dira.

```
<Applications>
   <Application Id="Erregistratu">
      <FormalParameters>
```

```
        <FormalParameter Id="workflowID" Index="0" Mode="IN">
            <DataType><BasicType Type="STRING"/></DataType>
        </FormalParameter>
      <FormalParameter Id="instanceID" Index="1" Mode="IN">
            <DataType><BasicType Type="STRING"/></DataType>
      </FormalParameter>
      <FormalParameter Id="workitemID" Index="2" Mode="IN">
       <DataType><BasicType Type="STRING"/></DataType>
      </FormalParameter>
     </FormalParameters>
  </Application>
  (...)
</Applications>
```

6.2.4. WorkflowProcess

Hau da lan-fluxuari buruzko informazioa bilduko duen etiketa. Bere barnean definitu behar dira prozesu zehatz baten ekintza eta trantsizioak. *Activities* etiketaren azpian bilduko dira ekintzak eta *Transitions*-en azpian trantsizioak.

```
<WorkflowProcesses>
      <WorkflowProcess Id="Adibidea" Name="Adibidea">
            (...)
      </WorkflowProcess>
</WorkflowProcesses>
```

6.2.5. Activity

Lan-fluxuaren prozesu bateko lan-unitatearen definizioa dator *Activity* etiketan, ekintza batena hain zuzen ere.

Bere etiketaren barnean definituko dira ekintza hori nork eramango duen aurrera, zein motatakoa izango den, aplikaziorik lotuta izango duen ala ez, irteera eta sarrera babesak zein diren, ...

Ekintzaren barruan, lehenengo eta behin, inplementaziorik baldin badu hau nolakoa den esan behar da, *Implementation* etiketarekin. Alde batetik, aplikazio arruntak ditugu, kasu horretan, *Tool* etiketarekin markatuko dugu bere *Type* atributuan aplikazio bat dela adieraziz. Ondoren aplikazioari pasatu beharreko parametroak agertu behar dira *ActualParameters* etiketaren barnean bakoitzaren balioa emanez. Pasatutako parametro

kopuruak eta motak bat etorri behar dute XPDL fitxategiaren hasieran jarri diren aplikazioek zituzten parametro formalen definizioarekin.

```
<Activity Id="Hasiera" Name="Hasiera">
   <Implementation>
      <Tool Id="Erregistratu" Type="APPLICATION">
            <ActualParameters>
                  <ActualParameter>workflowID</ActualParameter>
                  <ActualParameter>instanceID</ActualParameter>
                  <ActualParameter>workitemID</ActualParameter>
            </ActualParameters>
      </Tool>
   </Implementation>
(...)
</Activity>
```

Ekintzak lotutako aplikaziorik ez badauka, *Implementation* etiketaren barruan *No* etiketa bat jartzea nahikoa da. Antzeko zerbait gertatzen da ekintzaren inplementazioa azpi-fluxu batek egiten badu.

Kasu horretan *SubFlow* dela markatu beharko da eta behar izanez gero, azpi-fluxuari pasatu beharreko parametroak jarri behar dira. Azpi-fluxuaren identifikadorearekin batera, bere exekuzioa sinkronoa ala asinkronoa den esan behar da. Exekuzio sinkronoak azpi-fluxua amaitu arte ekintza ez dela bukatutzat emango esan nahi du. Asinkronoarekin, ez da itxarongo azpi-fluxua amaitu arte ekintza bukatutzat emateko.

Ekintza burutu dezakeen erabiltzailearen inguruko informazioa *Performer* etiketa baten barnean testu hutsez jarriko da. Erabiltzailearen informazioak fitxategiaren hasieran definitu den erabiltzaile batekin bat etorri behar du.

```
(...)
<Performer>Administraria</Performer>
(...)
```

Ondoren ekintzaren hasiera eta amaiera moduak jarri behar dira *StartMode* eta *FinishMode* etiketen barruan. Bere balio posibleak *Automatic* edo *Manual* etiketekin definitiko dira.

```
(...)
<StartMode> <Manual/> </StartMode>
<FinishMode> <Automatic/> </FinishMode>
(...)
```

Ekintzari dagokion informazioarekin amaitzeko, sarrera eta irteera babesen informazioa jarri behar da *TransitionRestrictions* etiketa baten barnean. Alde batetik sarrera babesa jarri behar da *Join* etiketa baten *Type* atributuan AND edo XOR markatuz, motaren arabera. Irteera babesarekin antzeko gauza bat egin behar da *Split* izeneko etiketan. Kasu honetan *TransitionRefs* etiketa bat gehitu behar da barnean ekintza horretatik irteten diren trantsizio guztien identifikadoreekin.

Ondorengo adibidean erakusten den ekintzak, *xor* erako sarrera babesa eta *and* moduko irteerakoa duela esango genuke. Gainera irteeran bi trantsizio ditu, bata *Hasiera_Kudeatu* izenekoa eta bestea *Hasiera_Amaiera*.

```
(...)
<TransitionRestrictions>
      <TransitionRestriction>
           <Join Type="XOR"/>
           <Split Type="AND">
                  <TransitionRefs>
                    <TransitionRef Id="Hasiera_Kudeatu"/>
                    <TransitionRef Id="Hasiera_Amaiera"/>
                  </TransitionRefs>
            </Split>
      </TransitionRestriction>
</TransitionRestrictions>
(...)
```

6.2.6. Transition

Transitions etiketaren barnean kokatu behar dira prozesuari dagozkion trantsizioak eta, baldin badituzte, beraiei lotutako baldintzak. Trantsizioak bi nodoren arteko arku gisa uler daitezke, eta bere atributuetan adierazi behar da nondik nora pasatzea ahalbidetzen duten. A ekintzatik B ekintzara eramaten duen baldintzagabeko trantsizio bat horrela adieraziko litzateke. *From* eta *To* atributuetan adierazitako ekintzen izenek, aurreragoko *Activity* etiketa batzuetako *Id* atributuan jarritako izenekin bat etorri behar

dute.

```
<Transition Id="A_B" Name="Atik Bra joateko" From="A" To="B"/>
```

Baldintzadun trantsizio bat, Kudeatu-tik Ezeztatu-ra joateko:

```
<Transition Id="Ez" Name="Ez onartu" From="Kudeatu"  To="Ezeztatu">
      <Condition Type=”CONDITION”>
            not:python:instance.hasProperty('onartua')
      </Condition>
</Transition>
```

6.2.7. ExtendedAttribute

Etiketa hau, lan-fluxu sistema bakoitzaren garatzaileek XPDLk eskaintzen dituen definizioetatik haratago joateko aukera izan dezaten erabiltzen da. Sistema bakoitzak, erabiliko dituen etiketa gehiago definitu ditzake, XPDL estandarraren muina aldatu gabe eta bere sistemak dituen ezaugarrietara hobeto doitzeko. *ExtendedAttributes* etiketa, edozein entitatetan erabil daiteke.

Merkatuan dagoen tresna bat baino gehiagok, editore grafiko baten bidez sortzen ditu XPDL fitxategiak, lan-fluxua era grafikoan adieraziz. Tresna horiek, *ExtendedAttributes* etiketa barnean gorde ditzakete, adibidez, grafoa pantailan marrazterakoan objektu bakoitza joan behar den lekuaren koordenatuak. Hori egiten du adibidez, JaWE edo Visio programarentzat CapeVisions-ek sortutako programa gehigarri batek [CapeVisions].

Adibidez WfMC erakundeak, XPDL estandarraren dokumentuan azaltzen duen adibidea Visiorako aipatutako programa gehigarriarekin eginda dago eta honelako etiketak sortzen ditu (*Activity* etiketaren barruan):

```
<ExtendedAttributes>
      <ExtendedAttribute Name=”Coordinates”>
            <xyz:Coordinates xpos=”347” ypos=”435”/>
      </ExtendedAttribute>
</ExtendedAttributes>
```

JaWE tresnarekin sortutako XPDL fitxategi baten, honelako informazioa agertzen da A*ctivity* etiketa baten barruan:

```
<ExtendedAttributes>
      <ExtendedAttribute Name="ParticipantID" Value="Anonymous"/>
      <ExtendedAttribute Name="XOffset" Value="590"/>
      <ExtendedAttribute Name="YOffset" Value="30"/>
</ExtendedAttributes>
```

Tresna bakoitzak bere barruko beharretara egokitzen ditu atributu hauek. Horrek alde batetik gauza zehatzagoak egitea ahalbidetzen du, bestalde, beste sistema batera guk sortutako XPDL fitxategia bidaltzean ezingo da aprobetxatu *ExtendedAttribute* bezala sartutako funtzionalitate hori, estandarra ez den modu batean adierazten delako.

6.3. Adibidea

Aurreko atalean azaldutako XPDL zatiak adibide oso baten nola integratzen diren ikusteko, XPDL fitxategi oso baten edukia erakusten dut jarraian, eta aurretik azaldutako zatiak markatu egin ditut.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package Id="OpenFlow" xmlns="http://www.wfmc.org/2002/XPDL1.0"
xmlns:xpdl="http://www.wfmc.org/2002/XPDL1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wfmc.org/2002/XPDL1.0
http://wfmc.org/standards/docs/TC-1025_schema_10_xpdl.xsd">
      <PackageHeader>
            <XPDLVersion>1.0</XPDLVersion>
            <Vendor>openflow2xpdl</Vendor>
            <Created>Fri Jun 18 12:54:32 2004</Created>
      </PackageHeader>
      <ConformanceClass GraphConformance="NON_BLOCKED"/>
      <Script Type="text/pythonscript"/>
      <Participants>
            <Participant Id="Administraria" Name="Administraria">
                  <ParticipantType Type="ROLE"/>
            </Participant>
            <Participant Id="Idazkaria" Name="Idazkaria">
                  <ParticipantType Type="ROLE"/>
            </Participant>
      </Participants>
      <Applications>
```

```
        <Application Id="Erregistratu">
          <FormalParameters>
            <FormalParameter Id="workflowID" Index="0" Mode="IN">
             <DataType> <BasicType Type="STRING"/></DataType>
            </FormalParameter>
            <FormalParameter Id="instanceID" Index="1" Mode="IN">
             <DataType> <BasicType Type="STRING"/></DataType>
            </FormalParameter>
            <FormalParameter Id="workitemID" Index="2" Mode="IN">
             <DataType> <BasicType Type="STRING"/></DataType>
            </FormalParameter>
          </FormalParameters>
        </Application>
        <Application Id="Kudeaketa">
         <FormalParameters>
          <FormalParameter Id="workflowID" Index="0" Mode="IN">
             <DataType><BasicType Type="STRING"/></DataType>
           </FormalParameter>
          <FormalParameter Id="instanceID" Index="1" Mode="IN">
              <DataType><BasicType Type="STRING"/></DataType>
          </FormalParameter>
          <FormalParameter Id="workitemID" Index="2" Mode="IN">
              <DataType><BasicType Type="STRING"/></DataType>
           </FormalParameter>
         </FormalParameters>
        </Application>
    </Applications>
    <WorkflowProcesses>
       <WorkflowProcess Id="Adibidea" Name="Adibidea">
         <ProcessHeader/>
         <Activities>
          <Activity Id="Hasiera" Name="Hasiera">
            <Implementation>
              <Tool Id="Erregistratu" Type="APPLICATION">
               <ActualParameters>
                <ActualParameter>Adibidea</ActualParameter>
                <ActualParameter>instanceID</ActualParameter>
                <ActualParameter>workitemID</ActualParameter>
               </ActualParameters>
              </Tool>
            </Implementation>
            <Performer>Administraria</Performer>
            <StartMode><Manual/></StartMode>
            <FinishMode><Manual/></FinishMode>
            <TransitionRestrictions>
              <TransitionRestriction>
                  <Join Type="AND"/>
                  <Split Type="AND">
                      <TransitionRefs>
```

```
                    <TransitionRef Id="Hasiera_Kudeatu"/>
                  </TransitionRefs>
              </Split>
            </TransitionRestriction>
          </TransitionRestrictions>
        </Activity>
        <Activity Id="Kudeatu" Name="Kudeatu">
          <Implementation>
           <Tool Id="Kudeaketa" Type="APPLICATION">
            <ActualParameters>
                <ActualParameter>Adibidea</ActualParameter>
                <ActualParameter>instanceID</ActualParameter>
                <ActualParameter>workitemID</ActualParameter>
            </ActualParameters>
           </Tool>
          </Implementation>
          <Performer>Idazkaria</Performer>
          <StartMode><Manual/></StartMode>
          <FinishMode><Automatic/></FinishMode>
          <TransitionRestrictions>
            <TransitionRestriction>
                <Join Type="AND"/>
                <Split Type="XOR">
                  <TransitionRefs>
                   <TransitionRef Id="Kudeatu_Tratamendua"/>
                   <TransitionRef Id="Kudeatu_Bukaera"/>
                  </TransitionRefs>
                </Split>
             </TransitionRestriction>
          </TransitionRestrictions>
        </Activity>
        <Activity Id="Tratamendua" Name="Tratamendua">
          <Implementation><No/></Implementation>
          <Performer>Administraria</Performer>
          <StartMode><Manual/></StartMode>
          <FinishMode><Manual/></FinishMode>
          <TransitionRestrictions>
             <TransitionRestriction>
                <Join Type="AND"/>
                <Split Type="AND">
                  <TransitionRefs>
                    <TransitionRef Id="Tratamendua_Bukaera"/>
                  </TransitionRefs>
                </Split>
             </TransitionRestriction>
          </TransitionRestrictions>
         </Activity>
         <Activity Id="Bukaera" Name="Bukaera">
          <Implementation><No/></Implementation>
          <StartMode><Automatic/></StartMode>
```

```
            <FinishMode><Automatic/></FinishMode>
            <TransitionRestrictions>
               <TransitionRestriction>
                  <Join Type="XOR"/>
                  <Split Type="AND">
                        <TransitionRefs/>
                  </Split>
               </TransitionRestriction>
            </TransitionRestrictions>
          </Activity>
        </Activities>
        <Transitions>
          <Transition From="Hasiera" Id="Hasiera_Kudeatu" Name=""
To="Kudeatu"/>
          <Transition From="Kudeatu" Id="Kudeatu_Tratamendua"
Name="None" To="Tratamendua">
           <Condition>python:instance.hasProperty('onartua')
</Condition>
          </Transition>
          <Transition From="Kudeatu" Id="Kudeatu_Bukaera" Name="None"
To="Bukaera">
           <Condition>not:python:instance.hasProperty('onartua')
</Condition>
          </Transition>
          <Transition From="Tratamendua" Id="Tratamendua_Bukaera"
Name="" To="Bukaera"/>
        </Transitions>
      </WorkflowProcess>
   </WorkflowProcesses>
</Package>
```

7. XPDL INPORTATZAILEA ETA ESPORTATZAILEA

Proiektuan garatutako bi moduluetako baten azalpena dator atal honetan. OpenFlow produktua eta XPDL lengoaiaren arteko inportazio eta esportazio lanak egiten dituen moduluarena hain zuzen ere.

7.1. Sarrera

OpenFlow produktuak ez dauka [WfMC] erakundeak estandarizatutako XPDL lengoaiara bere baitan dituen prozesuak esportatzeko modu automatizaturik. Izan ere, OpenFlowren garatzaileek komentatu ziguten proiektuaren hasieran hau zela beren produktuaren beharretako bat.

Gaur egun OpenFlow produktuarekin sortutako lan-fluxuak XPDL estandarrera itzuli behar badira eskuz edo beste tresnaren bat erabiliz egin behar da lana. Adibidez, lehenago aipatu dudan JaWE tresna erabiliz, lan-fluxuaren adierazpen grafikoa egin eta ondoren XPDL fitxategia lortzeko. Gauza bera gertatzen da XPDL formatuan errepresentaturik dagoen lan-fluxu bat OpenFlowra inportatzeko, eskuz egin behar da lan guztia.

Modulu hauen beharra OpenFlowren egileek eurek nabarmendu digute beraiekin izandako posta elektronikoko mezuen bitartez eta lan-fluxuen munduan lehenengo pausuak emateko hasiera interesgarria iruditu zaigu hasieratik. OpenFlowri bere ahalmenak handitzeaz gain, lan-fluxu sistema ezberdinen arteko prozesuen trukea ahalbidetzeko, tresna beharrezkoak dira modulu hauek. XPDLren helburuei jarraiki, jendea produktu hau erabiltzen has dadin hasiera ona izan daiteke jadanik existitzen diren lan-fluxuen definizioak era erraz batean OpenFlowra pasatzea ahalbidetzea, trantsizioa gogorregia izan ez dadin.

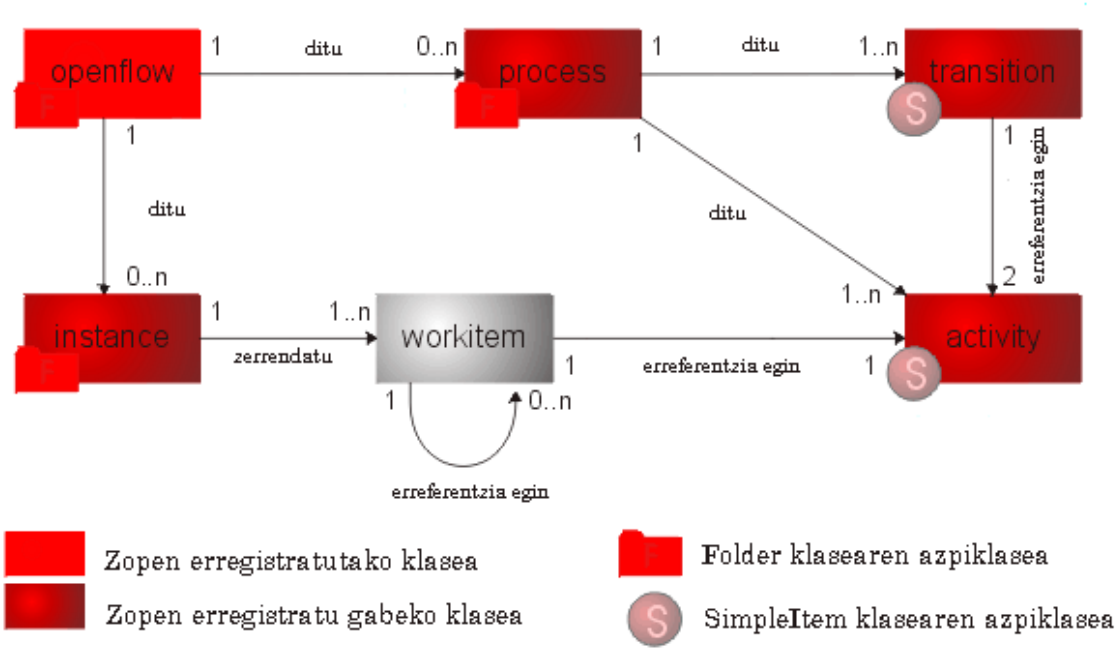
Garatutako moduluak OpenFlow produktuaren garatzaileei bidali zaizkie, baina memoria hau idazteko orduan oraindik ez digute bere funtzionamenduari buruzko erantzun edo iritzirik eman, edonola ere moduluek hobekuntzak edo beste era bateko aldaketak behar badituzte, prest gaude OpenFlow produktuaren garatzaileekin

elkarlanean aritzeko.

7.2. OpenFlow eta XPDL formatuen ezberdintasunak

OpenFlow Zope plataformarako produktua izanik, bere errepresentazioa objektuen bidez egiten du, objektuei zuzendutako programazioaren ikuspegiari jarraiki, eta objektu horien iraunkortasuna Zoperen objektu datu-baseak (ZODB deritzanak) bermatzen du.

OpenFlowren klase diagrama ikus daiteke Irudia 17n.



Irudia 17OpenFlowren klase diagrama([Icube]tik egokituta)

Folder eta *SimpleItem* klaseak, Zopek produktuak garatzeko eskaintzen dituen klase bi dira. *Folder-ekin*, beste objektu batzuk eduki ditzaketen klaseak defini daitezke. *SimpleItem*-ekin, aldiz, Zopen oinarrizkoak diren funtzionalitateak dituzten klaseak sor daitezke. Funtzionalitate horiei esker, objektuei Zoperen interfaze nagusitik kudeatuak izateko gaitasuna ematen die, hala nola, objektuak kopiatu eta itsatsi, egindako aldaketak desegiteko gaitasuna eman, ...

OpenFlow objektua edukiontzi bezala ulertu behar da. Bere baitan gordetzen baitu

lan-fluxuen definizioa (process edo prozesuak) eta exekuzioen informazioa (instance edo instantziak). Aldi berean, prozesu eta instantziak ere edukiontziak dira, lehenengoek, trantsizio eta ekintzak dituzte eta bigarrenek lan-itemen informazioa gordetzen dute, hau da, exekuzioaren pauso bakoitzaren informazioa.

XPDL formatuak aldiz, XML lengoaiaren dialekto bat izaki, testu fitxategi baten gordetzen du lan-fluxuen errepresentazioa, A eranskinean dagoen XPDL formatuaren espezifikazioari jarraiki.

OpenFlow eta XPDLren errepresentazio formatu ezberdinak ikusteko bi adibide jarriko ditut, ekintza bat eta trantsizio bat nola adierazten diren bietako bakoitzean.

XPDLn era honetan adieraziko den ekintza bat,

```
<Activity Id="Bukaera" Name="Bukaera">
      <Implementation><No/></Implementation>
      <StartMode><Automatic/></StartMode>
      <FinishMode><Automatic/></FinishMode>
      <TransitionRestrictions>
         <TransitionRestriction>
            <Join Type="XOR"/>
            <Split Type="AND">
                  <TransitionRefs/>
            </Split>
         </TransitionRestriction>
      </TransitionRestrictions>
</Activity>
```

OpenFlown, prozesu baten barnean dagoen objektu gisa errepresentatuko litzateke eta hauexek lirateke bere atributuen balioak:

```
id = 'Bukaera'
split_mode = 'and'
join_mode = 'xor'
self_assignable = 1
start_mode = 'automatic'
finish_mode = 'automatic'
subflow = ''
kind = 'standard'
application = ''
push_application = ''
title = 'Bukaera'
parameters = ''
```

```
description = ''
```

Trantsizio bat XPDLz:

```
<Transition From="Hasiera" Id="Hasiera_Kudeatu" Name="" To="Kudeatu"/>
```

OpenFlown, trantsizio objektuaren atributuak hauek lirateke:

```
id = 'Hasiera_Kudeatu'
from = 'Hasiera'
to = 'Kudeatu'
condition = ''
description = ''
```

Formatuen bihurketa egiten duten moduluek berezitasun hauek kontuan eduki behar dituzte eta OpenFlow osatzen duten klase guztien atributu eta metodoen definizioa eta darabiltzaten parametroak eskura eduki behar dituzte beren lana era egokian egiteko. Horretarako OpenFlowren APIa erabili behar da, memoriaren D eranskinean ikusi daitekeena hain zuzen ere.

Lan horretarako OpenFlow produktuaren iturburu kodea eskuragarri izatea ere garrantzitsua da, funtzionamenduaren edozein zalantza izanez gero bertara jotzeko gainera iturburua denon esku dago, OpenFlow GNU[5] elkargoaren GPL[6] lizentziapean banatzen delako.

Moduluen inplementazioaren diseinuaren aldetik, XPDLtik inportatzeko moduluaren diseinua eta OpenFlowtik esportatzekoarena desberdindu behar dira. Lehenengoa inplementatzeko, bi aukera identifikatu ditut. Batetik, fitxategia behin irakurri, bitarteko errepresentazio batera itzuli eta ondoren OpenFlowren funtzioak erabiliz prozesuak, ekintzak eta gainontzeko egiturak sortu. Bigarren aukera, fitxategia irakurtzea eta irakurri ahala OpenFlowko objektuak sortzea da. Bi hurbilpen horiek XML dokumentuak tratatzeko dauden bi aukera nagusiak erabiliz inplementatu daitezke (aurreko atal batean azaldutako DOM eta SAX, hurrenez hurren).

5 GNU: GNU's Not UNIX http://www.gnu.org
6 GPL: GNU General Public License http://www.gnu.org/copyleft/gpl.html

OpenFlowren egitura XPDLra pasatzeko, formatu honen egitura aurrean eduki behar da eta beharrezko datuak ekarri formatuak eskatzen duen ordenean: lehenengo goiburukoa, gero erabiltzaileak eta aplikazioak, gero prozesu bakoitzaren ekintza eta trantsizioak, eta abar. Ondorioz, diseinua, atal bakoitzeko informazioa ekartzen duen metodo bat inplementatzea litzateke, OpenFlowren metodoak eta atributuak erabiliz informazioa atera eta XPDL fitxategia sortzeko. Fitxategia sortzeko aukera ezberdinak daude, fitxategira zuzenean informazioa idaztea edo bitarteko egitura baten gorde eta ondoren idaztea.

7.3. OpenFlow – XPDL esportatzailea

Esportatzailea egiteak dakarren arazo nagusia, XPDL fitxategia sortzea da eta hori egiteko aukera nagusi bi landu ditut: fitxategia OpenFlow-ko datuak irakurri ahala zuzenean sortzea eta DOM eredua jarraitzea fitxategia gordetzeko.

7.3.1. Estrategia

Lehenengo aukera OpenFlowren objektuak lortu eta zuzenean testu fitxategi arrunt baten beharrezko informazioa idaztea da, baina aukera hori ia hasieratik baztertu egin nuen, arazo asko ikusten bainizkion. Batez ere XML etiketak eta atributuak fitxategian zuzen idazterako orduan.

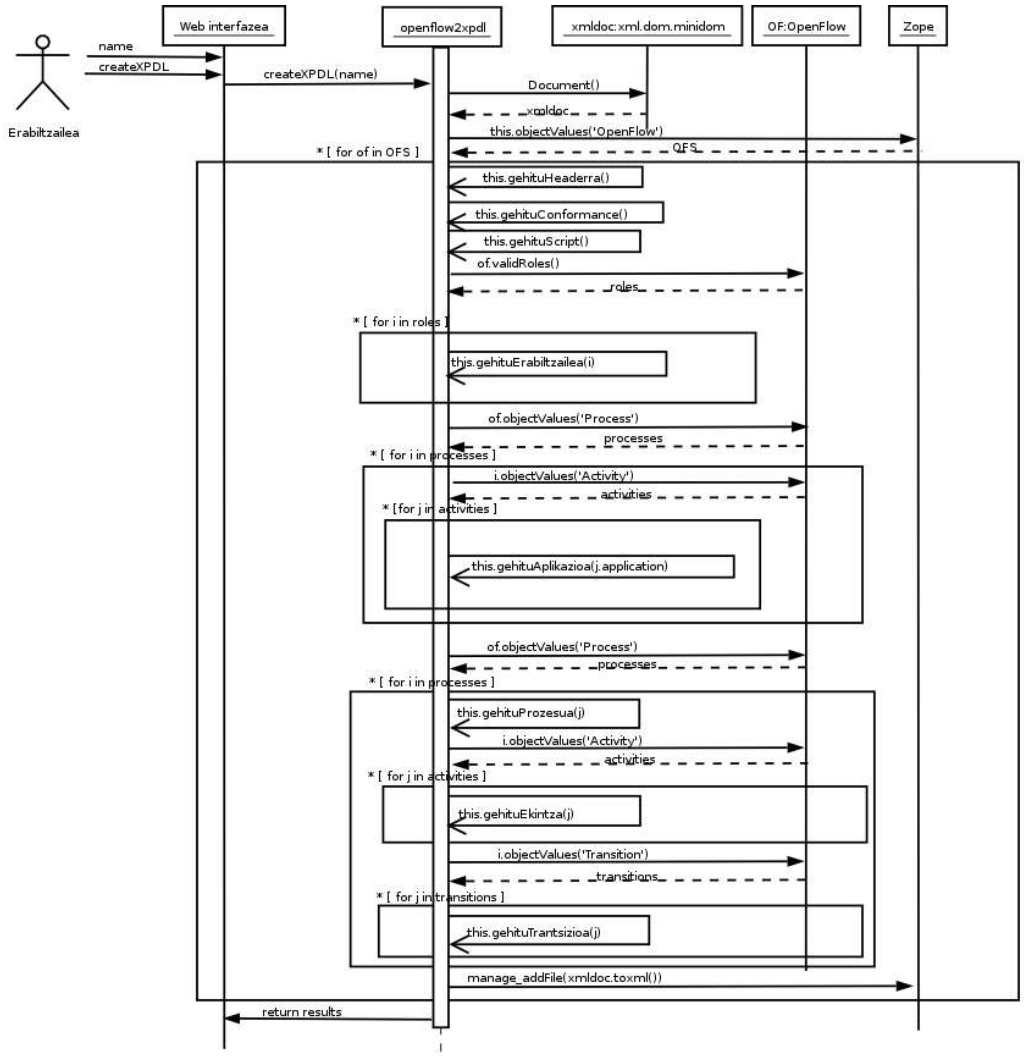
Hori dela eta Python lengoaiarako existitzen den PyXML paketeak [SIG XML Py] eskainitako tresnak erabiltzea erabaki dut. PyXML paketeak, 4.3. atalean azaldu denez, Python lengoaian XML dokumentuak tratatzeko moduluen bilduma eskaintzen ditu.

Modulu hau erabiltzea beharrezkoa da XML dokumentuekin lan egiteko, liburutegi estandarrean datozen tresnak txiki geratzen baitira, horregatik DOM eredua inplementatzen duen *minidom* modulua erabili dut esportatzailea garatzeko.

DOM ereduak, existitzen den XML fitxategi bat parseatu eta bere edukia zuhaitz egitura baten gordetzeaz gain, zuhaitzari adar berriak gehitzea ahalbidetzen du, eta hori aprobetxatu dut modulua inplementatzeko. Zuhaitz huts bat sortzen dut hasieran eta OpenFlowtik beharrezko informazioa atera ahala gehitzen diot zuhaitzari.

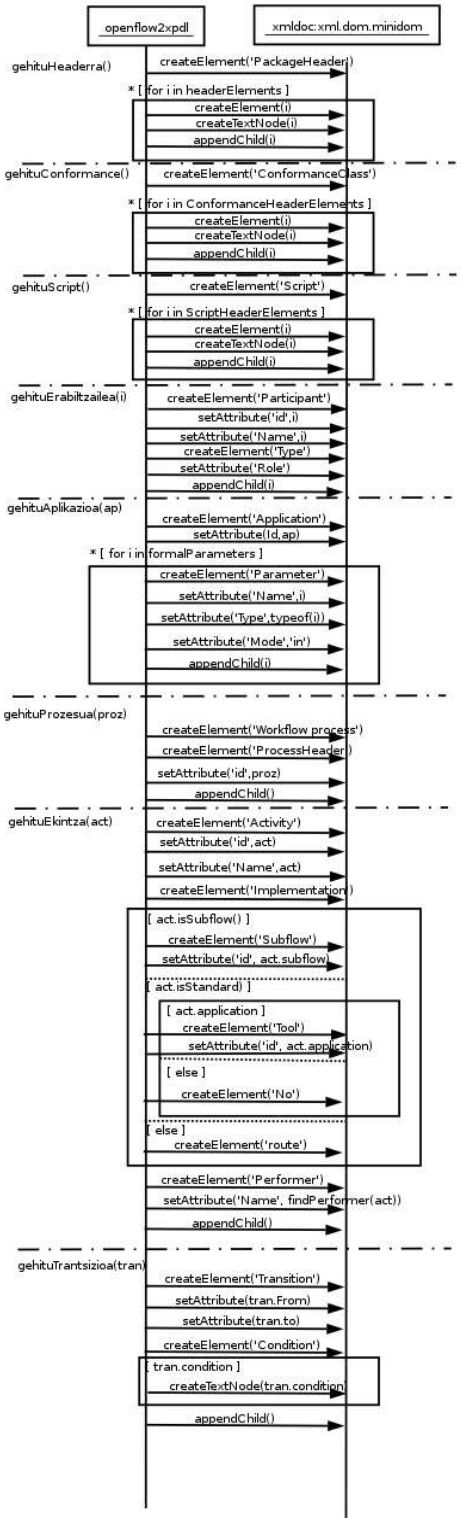
Sortutako dokumentuko etiketen ordena eta habiatzea esanguratsua denez, garrantzitsua da informazioa OpenFlowtik zein ordenatan lortu XPDL dokumentu zuzena sortzeko, ondorioz XPDL fitxategi baten egiturak eta bertako etiketen ordenak mugatzen du informazioa nola atera OpenFlowtik.

7.3.2. Sekuentzia diagrama



Irudia 18: Esportatzailearen sekuentzia diagrama (1)

Esportazioaz arduratzen den moduluaren sekuentzia diagramak ikus daitezke Irudia 18 eta Irudia 19n. Garbitasuna dela eta, eta sekuentzia diagrama hobeto ulertzeko, lehenengoan funtzio batzuk erabili eta bigarrenean azaldu egin ditut.



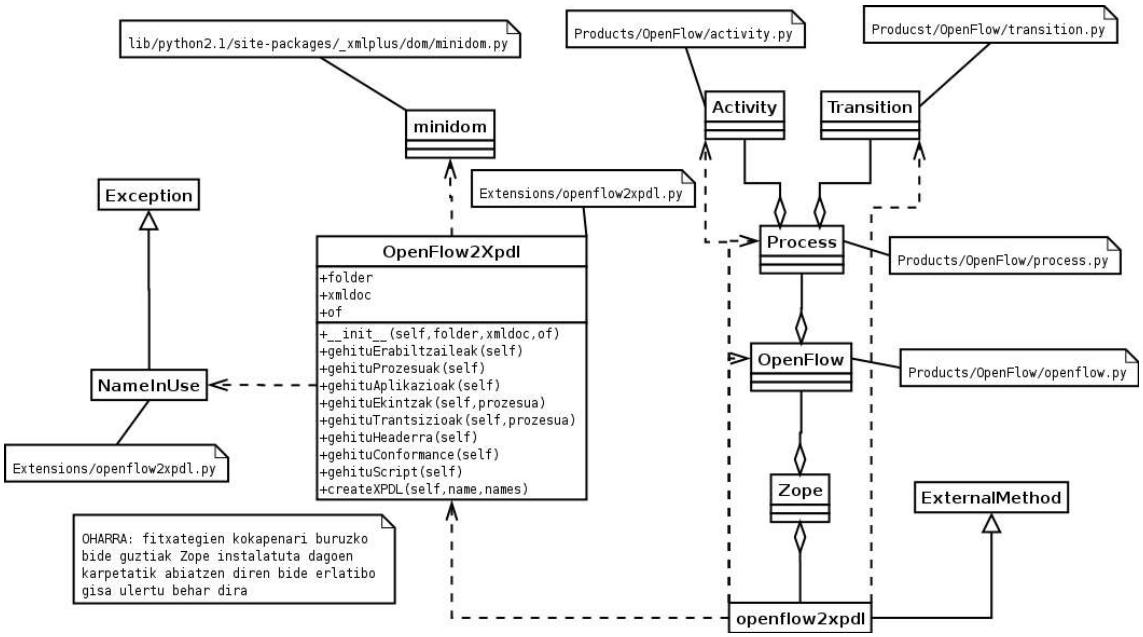
Irudia 19: Esportatzailearen sekuentzia diagrama

7.3.3. Inplementazioa

Esportatzailea Python lengoaiaz idatzitako modulu bakar baten dago, *openflow2xpdl.py* fitxategian, hain zuzen ere. Bertan definitzen da esportazioaz arduratzen den klasea eta baita klaseak altxa dezakeen *NameInUse* izeneko salbuespena ere.

Zopetik klasea atzitu ahal izateko *ExternalMethod* motako objektu bat sortu behar da, sortutako klasea erabiltzen duen funtzio bati lotuta. Hau da, klaseaz gain, funtzio bat idatzi behar da, eta hau da OpenFlow objektu bakoitza hartu, esportazioaz arduratzen den klasearen instantzia bat sortu eta esportazioa egiten duena. Funtzio horren inplementazioa ere, klasearena bezala, *openflow2xpdl.py* fitxategian dago.

Esportazioaz arduratzen den klaseak, NameInUse izeneko salbuespena altxatzen du, sortu nahi den izeneko objektu bat jadanik existitzen denean.



Irudia 20: Esportazailearen klase diagrama

7.3.4. Aurkitutako arazoak eta hartutako erabakiak

Lehenengo eta behin OpenFlow bertako aplikazioen definizioan aurkitu ditut arazoak. OpenFlow-ko aplikazioak URL bidez dei daitekeen edozer gauza izan daitezke eta gainera OpenFlown ez da definitzen zein diren aplikazio horri pasatu beharreko parametroak; bestaldetik XPDL fitxategia sortzeko beharrezkoa da parametro horiek ezagutzea, aplikazio bakoitzari dagokion atalean espezifikatu behar delako.

Hori dela eta, OpenFlow-rekin egindako proba batzuen ostean eta produktuaren garatzaileek webgunean ematen duten informazioan oinarrituta, aplikazio automatikoei defektuz hiru parametro pasatzen zaizkiela ikusi dut: instantziaren, lan-itemaren eta OpenFlow objektuaren identifikadoreak. Guztiak karaktere kateak dira eta sarrera motako parametroak dira. OpenFlowk aplikazio automatikoei pasatzen dizkien parametroak behatuz, gutxienez ondo funtzionatzeko aplikazioek behar dituzten datu minimoak zein diren jakin daiteke. Ondorioz, nahiz eta ez duen zertan horrela izan, aplikazio guztiei parametro horiek bakarrik pasatzen zaizkiela errepresentatzen dut XPDL fitxategian.

Automatikoak ez diren aplikazioei nahi adina parametro pasatu ahal zaie dei egitean hauen izen eta balioak HTTPko GET edo POST metodoak erabiliz, gainera aplikazio berari behin baino gehiagotan deitu ahal zaio parametro ezberdinekin. Parametro horiek zein diren inon zehazten ez denez, XPDL fitxtegia sortzean ezin da jakin zeintzuk pasatzen diren. Hori dela eta, automatikoak ez diren aplikazioei ere, automatikoak direnei OpenFlowk pasatzen dizkien parametroak pasatzen zaizkiela kontsideratu eta XPDL fitxategia sortzean horrela egin dut.

Beste alde batetik XML tratatzeaz arduratzen den *minidom* moduluak, fitxategia idazterakoan arazoren bat duela konturatu naiz. Hori dela eta, eta [PyXML-SIG] taldearen posta zerrendan bila jardun ostean eta bertako aholkuei jarraituz, moduluko *expatbuilder.py* fitxategian aldaketa bat egin behar izan dut. Aldaketa hau egin arren, kasu batzuetan ez zuen idazketak ondo funtzionatzen eta *minidom* moduluaren inplementazioan beste aldaketa bat egin behar izan dut *minidom.py* fitxategian (egin

beharreko aldaketak erabiltzailearen eskuliburuan, B eranskinean, daude azalduta).

XPDL-ren espezifikazioak etiketa berezi bat definitzen du, produktu garatzaile bakoitzak bere sistemaren ezaugarriren bat XPDL fitxategian islatu nahi badu horrela egin dezan. Etiketa hori *ExtendedAttributes* da (ikusi 6.1.7 atala). Ematen zaion erabilera garatzaile bakoitzaren esku dagoenez, lan-fluxuak sistema batetik bestera eramaterakoan bere benetako balioa ez da handia, beste produktuaren garatzaileek ez baitute jakingo zer egin etiketa horretan adierazitako balioekin. Bestalde, ez dut aurkitu OpenFlowren zein funtzionalitate adierazi etiketa horietan eta gainera etiketa horiek ez dira derrigorrezkoak XPDL fitxategiak baliozkoak izan daitezen, hau da, hautazkoak dira. Hori dela eta *ExtendedAttributes* etiketarik ez sortzea erabaki dut.

Azkenik, XPDL fitxategia zuzenean idatzi beharraren arazoa saihesten dut *minidom* modulua erabiliz. Moduluak eskaintzen dituen metodoak erabiliz, zuhaitzaren egitura sortzen dut, etiketen izenak, atributuen informazioa eta testu hutsezko zatiak bertan sartuz, XML dokumentu baten atributuen balioak komatxo bikoitzen artean sartu behar direla arduratu gabe. Horrela, sortu nahi den dokumentuaren edukia zuhaitzean era egokian sortuta dagoenean, zuhaitza testu arruntera itzultzeko metodoa deitzen dut eta edukia fitxategi baten gorde. Pauso hau ematean, metodo hori arduratzen da atributuak, balioak, etiketen izenak eta gainontzekoak era egokian idazteaz.

Hau da, XPDL fitxategia sortzeko informazioa lortu ahala, datu-egitura baten sartzen ditut datuak adibidean azaltzen den bezala.

```
>> xmldoc = minidom.Document()
>> semea = xmldoc.createElement('semea')
>> element= xmldoc.createElement('Erroa')
>> element.setAttribute('Id','OpenFlow')
>> element.appendChild(semea)
>> xmldoc.appendChild(element)
```

Informazio guztia zuhaitzean dagoenean, helburu fitxategia ireki eta zuhaitzaren edukia txertatzen dut bertan.

```
>> xmldoc.toprettyxml(encoding='UTF-8')
```

```
<xml version="1.0" encoding="UTF-8">
<Erroa Id='OpenFlow'>
      </semea>
</Erroa>
```

XPDL fitxategian lan-fluxuaren erabiltzaileen datuak ere adierazi behar dira. Estandarra dakarren dokumentuak, erabiltzaile mota ezberdinak erabiltzen ditu, baina OpenFlown rolak eta erabiltzaile arruntak bakarrik definitzen dira. Ondorioz hauen informazioa txertatu dut XPDL fitxategian. Hala ere gerta daiteke XPDL fitxategian ez agertzea OpenFlow darabilten erabiltzaile guztien datuak. Honen zergatia Zoperen funtzionamenduan dago. Zopek heredentzia mekanismoa dauka bere funtzionamendurako eta karpeta bateko objektu bat eskatzen badiogu eta karpetan ez baldin badago, goragoko mailara joaten da bere bila.

Demagun, Irudia 21n adierazten den egitura bat daukagula Zope barnean.

Irudia 21: Karpeta egitura Zopen

Irudian ikusten den *acl_users* objektuan egoten dira erabiltzaileak definituta. Ondorioz *Adibidea* karpeta atzitu nahi duen erabiltzailea bertako *acl_users* objektuan egon behar da erregistratuta. Ez badago, Zoperen heredentzia mekanismoak *Adibidea* gainean dagoen karpetako *acl_users* objektuan begiratzen du erabiltzailea han dagoen ikusteko. Aurkitzen ez duen artean gorantz joaten jarraitzen du azken mailara iritsi arte, han aurkitzen ez badu errorea ematen du. Horregatik OpenFlow dagoen karpetako

acl_users objektuan definitutako erabiltzaileak XPDL fitxategira gehitzerakoan, lan-fluxua erabiltzen duten erabiltzaile guztiak ez gehitzea gerta daiteke, batzuk heredentzia mekanismoari esker, goragoko *acl_users* baten erregistratuta egon daitezkeelako.

Azkenik, 5.3.5 atalean azaldutako *Workflow Relevant Data*-ren inguruan, esan OpenFlowk ez daukala hori adierazteko modu zehatzik, normalean instantzien propietateak erabiltzen baitira lan horretarako. Horregatik ez dut sortu *Workflow Relevant Data* hori adierazteko erabiltzen den etiketarik.

7.3.5. Proba kasuak

Esportatzaileak sortutako fitxategien zuzentasuna egiaztatzeko JaWE [Enhydra Jawe] eta Shark [Enydra Shark] tresnak erabili ditut.

JaWE, lan-fluxuen adierazpen grafikoa egiten duen Javaz idatzitako tresna bat da. Lan-fluxuaren grafoa eta gainontzeko informazioa eskuz sartuta, XPDL fitxategiak sortzen ditu. Ondorioz OpenFlown adierazitako lan-fluxuak tresna honetan eskuz sartuz XPDL fitxategiak sortzen ditu, eta hauek proiektu honetan garatutako tresnak sortutakoekin alderatu daitezke azken hauen zuzentasuna egiaztatzeko.

JaWE tresnak ez dauka OpenFlowrekin zerikusirik, beste software bat da eta bere helburua lan-fluxuen diseinatzaileari lana erraztea da, lan-fluxua era grafikoan sortzea ahalbidetuz. Behin adierazpen grafikoa edukita XPDL fitxategia sortzen du.

Alderantzizko lana ere egiten du, XPDL fitxategia emanda bertan adierazitako lan-fluxuak interpretatu, grafoa eraiki eta bere informazioa erakusten du, horrela proiektu honetan garatutako moduluarekin sortutako XPDL fitxategiak ireki daitezke JaWErekin, adierazten dutena eta adierazi nahi dena berdinak direnentz egiaztatzeko.

Shark, aldiz, Java programazio lengoaian idatzitako lan-fluxu zerbitzari bat da eta bere barnean JaWE tresnaren bertsio hobetu bat erabiltzen du. Shark-eko JaWEren bertsioak, XPDL fitxategiak irekitzean bere barne errepresentaziora moldatu eta zerbitzarian erabiltzeko prest uzten ditu. Honek, beste tresnekin sortutako XPDL fitxategiak lan-fluxu kudeaketa sisteman erabiltzeko gaitasuna ematen dio tresnari, garatutako moduluek OpenFlowri ahalmen hori eman nahi dioten neurri berean.

Fitxategiak interpretatzerakoan, bi tresnek, fitxategiaren zuzentasunari buruzko informazioa ematen dute, ekintzen identifikadoreak ondo dauden, lan-fluxuak WfMC-ren estandarrak jarraitzen dituen ala ez eta abar esanez.

Proiektu honetan garatutako esportatzaileak ez du lan-fluxuaren zuzenketa edo egiaztapen lanik egiten eta ondorioz gerta daiteke esportatutako fitxategiak, XPDL sintaxian zuzena izan arren, WfMC-ren estandarrak ez jarraitzea eta JaWE edo Shark moduko tresna batek fitxategiak errore semantikoak dituela esatea. Hala ere hori ez da esportazio lanaren ondorioa, jatorrizko lan-fluxuak ere ez baititu, aipatutako kasu horretan, WfMC-ren estandarrak jarraitzen.

7.4. XPDL – OpenFlow inportatzailea

Proiektu honetan garatutako bigarren moduluaren azalpena dator orain. OpenFlowtik XPDL lengoaiarako esportatzailearekin egin dudan bezala, inportatzailearen garapenaren inguruko datu eta ezaugarriak bilduko ditut atal honetan.

Esportatzailearekin gertatzen zen bezala, OpenFlow produktuak ez dauka XPDL lengoaia estandarretik lan-fluxuak inportatzeko gaitasunik, ondorioz beste sistema batzuetatik OpenFlowrako migrazioa zailduz.

7.4.1. Estrategia

Modulua inplementatzeko garaian, XPDL fitxategia parseatu behar da eta bertatik informazioa erauzi, bertan adierazitakoa OpenFlow era egokian txertatzeko. XML fitxategiak parseatzeko garaian bi teknika nagusi daude: SAX eta DOM (4.3.1. eta 4.3.2. ataletan azaldu ditudanak).

Bi ereduen arteko aukeraketa egiterakoan SAXen alde egin dut arrazoi nagusi birengatik. Batetik, fitxategia gutxika-gutxika irakurtzen du eta irakurri ahala sortzen ditu eredua inplementatu duen kudeatzailea ohartarazteko gertaerak, DOM ereduak egiten duenaren guztiz aurkako zentzuan. DOMek fitxategia goitik behera irakurketa bakarrean parseatu eta bere edukia zuhaitz egitura batean eskaintzen du. Bestetik, SAX eredua erabiliz, fitxategia parseatu eta beharrezko informazioa lortu ahala, sor daitezke

OpenFlowko egiturak eta pertsonalki errazagoa iruditu zait SAXekin lan egiteko modua DOMekin lan egitekoa baino.

7.4.2. Sekuentzia diagrama

Modulu honen sekuentzia diagraman ezin da zuzenean adierazi zein izango den programaren sekuentzia, ezin baita aurrez jakin zein izango den metodoak deitzeko ordena. Lehenago azaldu denez, SAX ereduak, XML dokumentuan aurkitzen dituen elementuen arabera gertaera batzuk sortzen ditu eta eredua inplementatu duen kudeatzailean dagozkien metodoei deitzen die. Ondorioz, parseatzaileari emandako XML dokumentuaren arabera dei ezberdinak sortuko dira eta ez dago dei horiek sekuentzia batean jartzerik. Adibide batekin erakutsiko dut azaldutakoa.

Demagun ondoko XML dokumentuak dauzkagula.

```
<pertsona id=”bat”>
      <semea id=”1”>
            <biloba id=”1”>Aitor</biloba>
            <biloba id=”2”>Nerea</biloba>
      </semea>
      <semea id=”2”/>
</pertsona>

<pertsona id=”bi”>
      <semea id=”1/>
</pertsona>
```

Lehenengo dokumentuan honako dei sekuentzia hau izango genuke:

```
etiketaIreki('pertsona'), etiketaIreki('semea'), etiketaIreki
('biloba'), testua('Aitor'), etiketaItxi('biloba'), etiketaIreki
('biloba'), testua('Nerea'), etiketaItxi('biloba'), etiketaItxi
('semea'), etiketaIreki('semea'), etiketaItxi('semea'), etiketaItxi
('pertsona')
```

Bigarren dokumentuak beste dei sekuentzia hau sortuko zukeen:

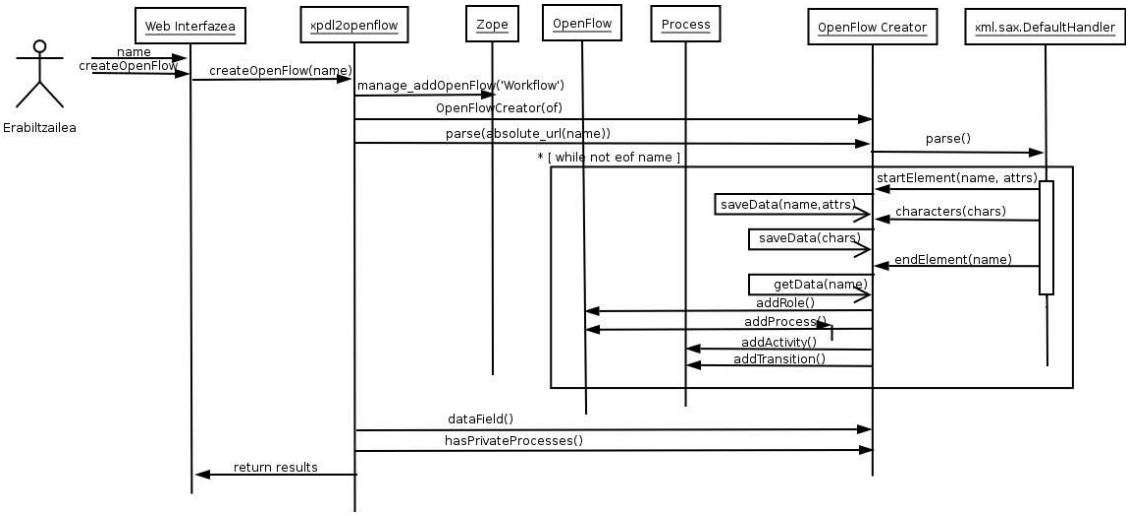
```
etiketaIreki('pertsona), etiketaIreki('semea'), etiketaItxi('semea'),
etiketaItxi('pertsona')
```

Dokumentuaren arabera, egindako deien ordena ez da berdina izango eta ondorioz zaila da hori sekuentzia diagrama baten islatzea, hori dela eta, sekuentzia diagraman deiak ordena honetan gertatuko balira bezala jokatu dut: etiketa ireki, testua, etiketa itxi. Etiketa arrunt bat tratatzeko era normalena izan ohi baita. Hala ere garbi eduki behar da deiek ez dutela zertan horrela gertatu.

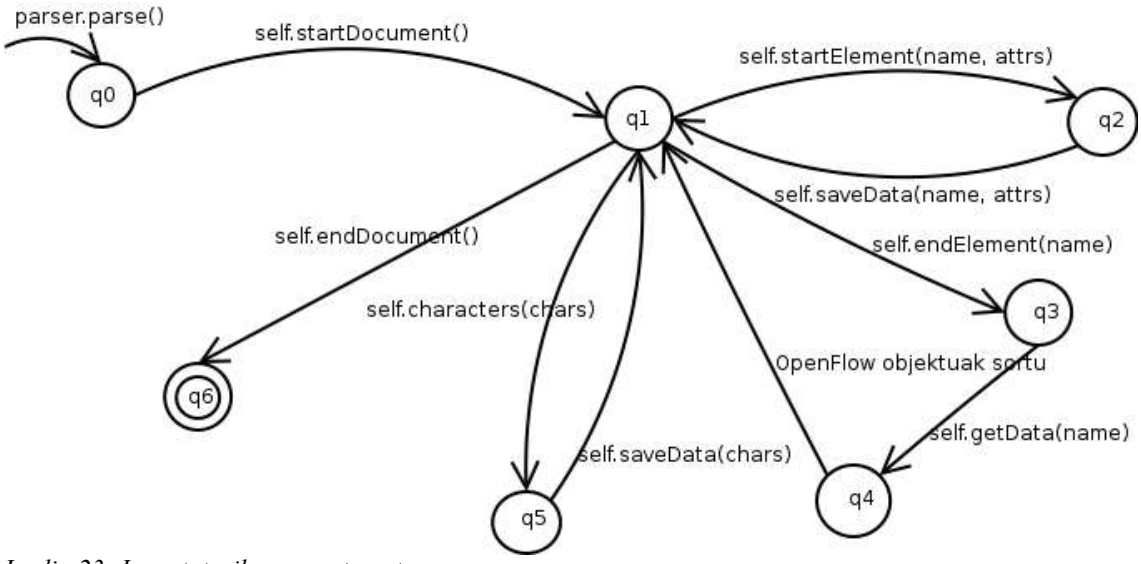
Era berean, OpenFlow-ko metodoen deiak ere ezin dira zehazki ordenean adierazi, horregatik ordena logiko baten adierazi ditut: prozesua sortu, ekintzak gehitu eta trantsizioak gehitu. Sekuentzia diagraman, Irudia 22n ikus daitekeen bezala, *saveData* izeneko metodo bat erabili dut informazioa gordetzen dudala adierazteko.

Horregatik, eta funtzionamendua hobeto ulertzeko, automata bat ere erabili dut gertatzen diren deiak errazago ulertzeko. Automata hori Irudia 23n ikus daitekeena da.

Automatak *q0* bezala identifikatutako hasiera egoera dauka eta parseatzailea lanean hastean egoera horretan jartzen da. Inportatu beharreko den unean hasten da lana *q1* egoerara pasatuz. Egoera honetatik, eta parseatzaileak deitzen dituen metodoen arabera aldatzen da egoeraz. Azkenean, parseatu beharreko dokumentua amaitzean amaitzen da bere lana.



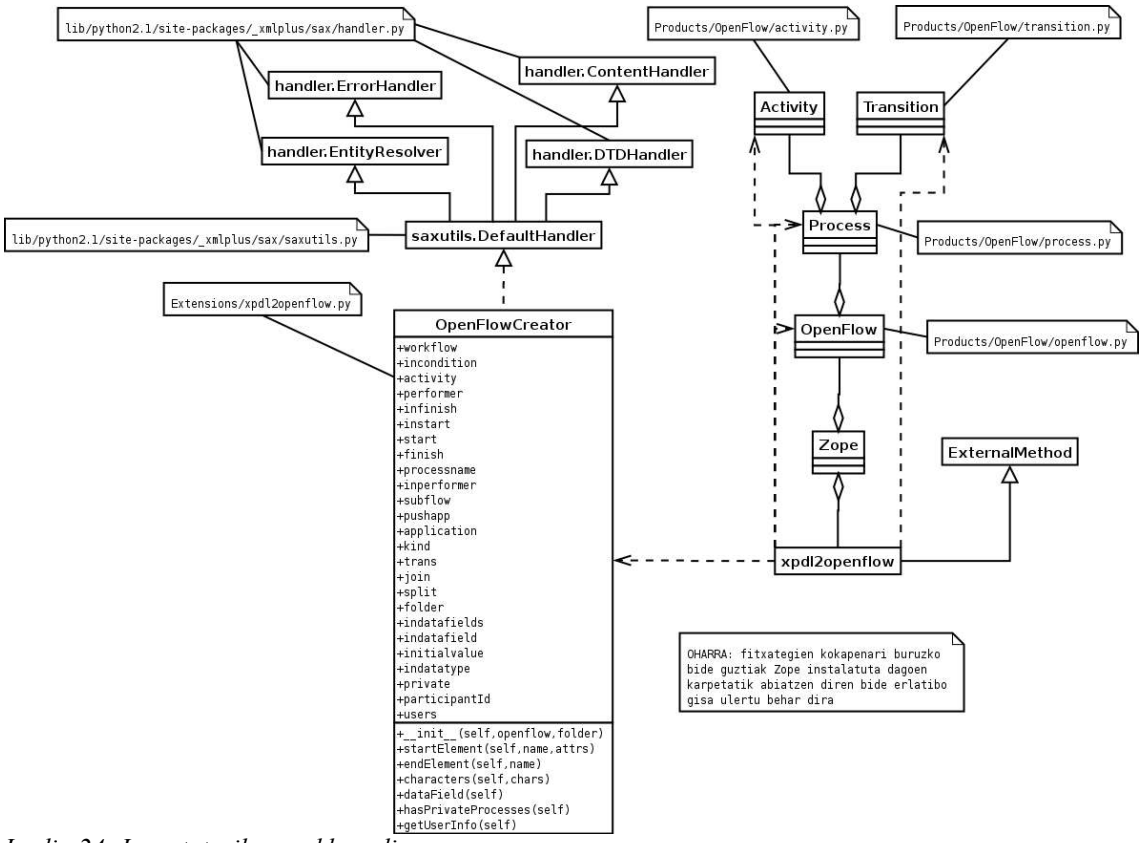
Irudia 22: Inportatzailearen sekuentzia diagrama



Irudia 23: Inportatzailearen automata

7.4.3. Inplementazioa

Irudia 24ko klase diagraman ikusten den bezala, *xpdl2openflow.py* izeneko fitxategi baten dago inportazioaz arduratzen den modulua. Moduluak Pythonerako modulu gehigarrien erabilpena egiten duenez (aipatutako PyXML moduluaren erabilpena hain zuzen ere), berau fitxategi sisteman kokatu behar da, Zope instalazioaren barneko *Extensions* karpetan, esportatzailearekin egin behar den bezala.



Irudia 24: Inportatzailearen klase diagrama

Ondoren Zoperen interfazetik *External Method* objektu bat sortu behar da eta fitxategi sistemako modulurako erreferentzia egin bertatik exekutatu behar den funtzioaren izena jarriz.

Ondorioz xpdl2openflow.py fitxategian gauza bi aurki ditzakegu: batetik, inportazioaz arduratzen den klasea, hau da, SAX eredua inplementatzen duen klasea eta bestetik, External Method objektutik deitu behar den funtzioa, hau arduratuko da aurreko klasearen instantzia sortu eta parseatu beharreko fitxategia lehenengoari emateaz. Azkenik egiaztapen batzuk egingo ditu eta erabiltzaileari aurkitutako arazo eta erabakien berri emango dio nabigatzailearen interfazearen bidez.

7.4.4. Aurkitutako arazoak eta hartutako erabakiak

Lan-fluxuan parte hartzen duten erabiltzaileen inguruko erabakia hartu behar izan

dut, batez ere kontuan hartuz XPDL formatuan erabiltzaile mota bat baino gehiago definitu daitekeela (ikusi 6.1.2 atala). Hori dela eta 'ROLE' eta 'HUMAN' motako erabiltzaileak, rol bat eta erabiltzaile arrun bat adierazten dutenak hain zuzen ere, dira zuzenean OpenFlown erabiltzeko Zopera gehitzen diren bakarrak. Beste erabiltzaile moten informazioa ('SYSTEM' edo 'RESOURCE' erakoak adibidez), gorde egiten da eta inportazioa bukatzean, erabiltzaileari informazio hori erakutsi egiten zaio nabigatzailearen interfazearen bidez, erabiltzaile mota horiek fitxategian errepresentatuta daudela baina OpenFlown adierazi ezin direla jakin dezan.

Beste alde batetik, eta esportatzailean hartutako erabakiei jarraiki, *ExtendedAttributes* etiketei jaramonik ez egitea erabaki dut, ez baitakit zer adierazi nahiko duen fitxategia sortu duenak bertan esandakoarekin.

Esportatzailean egin dudan bezala, XPDL fitxategian adierazitako *Workflow Relevant Data* ez dut OpenFlowra inportatu OpenFlown ez baitago hori adierazteko modu garbirik. Horren ordez, inportazio lana bukatzen denean, erabiltzaileari adierazten zaio nabigatzailean XPDL fitxategian Workflow Relevant Data hori dagoela esanez eta bertan adierazten diren datuak pantailaratuz, erabiltzaileak datuok kontuan har ditzan lanean hasi baino lehen.

XPDLn ez dago prozesuen hasiera eta bukaera ekintzak zein diren adierazteko modurik, hori dela eta inportatutako prozesuak ezin dira zuzenean erabili, hasiera eta bukaera ekintzak zein diren adierazi behar delako. Datu honen berri ere erabiltzaileari ematen zaio inportazioa bukatzean.

Aplikazioak ere, sortuta egon arren, ez daude inplementatuta, XPDLn aplikazioen definizioa soilik egin behar delako. Hori dela eta lanean hasi baino lehen aplikazioen inplementazioa egin behar dela ere abisu gisa esaten zaio erabiltzaileari eta gainera aplikazio bakoitzari pasatu beharreko parametroen izenak eta mota ere erakusten zaizkio, hauek inplementatzean kontuan har dezan.

7.4.5. Proba kasuak

Inportatzailearen probak egiteko, WfMC erakundeak XPDL estandarraren

espezifikazioan ematen duen lan-fluxu baten adibidea erabili dut nagusiki. Fitxategi hori JaWE tresnarekin ireki dut grafoa ikusteko eta ondoren inportatzailearekin sortutako lan-fluxua OpenFlow-ren webgunean [Icube] deskargatu daitekeen OpenFlow Process Editor tresnak sortutako grafoarekin alderatu dut eta lortutako emaitzak positiboak izan dira. Bestalde, nik sortutako beste lan-fluxu batzuekin ere probak egin ditut eta guztiekin emaitza onak lortu.

Tresna honek OpenFlow-n prozesu baten ekintza eta trantsizioak era erraz eta grafiko batean sortu eta aldatzea eskaintzen du, lan-fluxuaren ezaugarri nagusienak oso erraz aldatzea onartuz.

7.5. Inplementazioa

Moduluen inplementazioa Python programazio lengoaiaz egin dut eta bakoitza fitxategi baten gorde dut fitxategi sisteman, Zoperen objektu datu-basetik kanpo. Horrela egitearen zergatia, segurtasun arazoetan aurkitu behar da: Zope barruko Python scriptetatik ezin baitira defektuzko instalazioan ez datozen Python pakete gehigarriak atzitu, eta nik PyXML pakete gehitu eta erabili dut moduluak garatzeko orduan.

Horretarako Zopek *External Method* deritzon objektu mota bat dauka eta objektu mota honen bidez Zope instalatuta dagoen fitxategi sisteman kokatutako scriptak atzigarri jar daitezke Zope interfazearen bidez.

Zoperen kudeaketa web bidez egiten da eta ondorioz, webguneko script eta metodoak fitxategi sistemarako atzipenik gabe sortu eta editatu daitezke. Segurtasun arau gisa, Zoperen interfazean bertan dauden objektuetarako atzipena dute bertan sortutako script eta metodoek, *External Method* motako objektuek izan ezik. Hauek fitxategi sistemako karpeta jakin batean kokatutako Python scripten zubi gisa jokatzen dute era arruntean erabil ezin daitezkeen modulu gehigarriak erabili ahal izateko.

Erreferentziatutako modulu horiek fitxategi sisteman kokatu behar direnez, horrek beste segurtasun maila bat eskaintzen du, webgunea kudeatu behar duen pertsonak ez baitu fitxategi sistemarako atzipenik behar bere lana egiteko, web interfazearen bidez egiten baitu bere lana.

Garatzaileen ideia zera da: Zope interfazerako atzipena duen jendeak, kontrolatutako gauzak bakarrik egin ahal izatea, bertan sor daitezkeen Python scriptak modulu konkretu batzuk erabil ditzaten utziz (karaktere kateen inguruko funtzioak, funtzio matematikoak eta ausazko zenbakiak sortzeko modulua).

8. KONPILADOREA

Modulu honen lana, gaizki konfiguratuta edo diseinatuta dauden lan-fluxuak aztertzea izango da akatsak non dituzten jakitera emateko. Lan-fluxuan zerbait ondo diseinatuta ez badago, demagun ekintza batek bukaerakoa izan gabe ez duela irteerako trantsiziorik, orduan instantzia ekintza horretara iristean, blokeatuta gelditzuko da egoera berezi batera pasatuz eta ez du errorerik emango. Hori gertatuz gero, eskuz egin behar dira sistemak egoera egokia berreskura dezan egin beharreko aldaketak. Erabiltzaileak horrelako antzematea da modulu honen helburua.

Kapituluan zehar kasu berezi horiek zein diren, hauek gertatzean OpenFlowren portaera zein den eta une bakoitzean zer egin beharko litzatekeen azalduko dut.

8.1. Diseinu okerreko kasuen azterketa eta OpenFlowren portaera

Atal honetan, salbuespen egoerak edo arazoak ematen dituzten egoerak zein diren eta bakoitzerako OpenFlowk zein portaera duen azalduko dut.

8.1.1. Prozesuen hasiera eta bukaera egoerak ezarri gabe egotea

Prozesu batek hasiera egoera ezarri gabe baldin badauka, ezingo da martxan jarri. Bukaera egoera ezarri gabe baldin badauka, aldiz, instantziak ez du jakingo bere lana bukatu den ala ez, eta ondorioz egoera berezi batean gelditzuko da.

Hasierako ekintza zein den definitu gabe egonik instantzia martxan jartzean, APIko *startInstance* metodoari deitzean adibidez, Zoperen errore orrialdea agertzen da. Hala ere, ez du gerta daitekeenaren mezurik ematen, Zoperen errore estandar bat baizik.

Bukaera ekintza definitu gabe dagoenean, instantzia *fallout* deritzon salbuespen egoeran gelditzen da, bertatik ateratzeko salbuespen egoeretatik irteteko modu arrunta erabili behar da dagokion metodoari deituz.

Errore hauek ekiditen saiatzeko, prozesua sortzerakoan, hasiera eta bukaerako ekintzak automatikoki sortu eta definitzeko aukera ematen du OpenFlowk. Hala ere, erabakia diseinatzailearen esku dago, eta hasiera edo bukaerako ekintza ez duten lan-fluxuak inplementatzea gerta daite, exekutaraztean aurretik aipatutako errorea sortuz.

8.1.2. Egoerek sarrera edo irteera trantsiziorik ez edukitzea

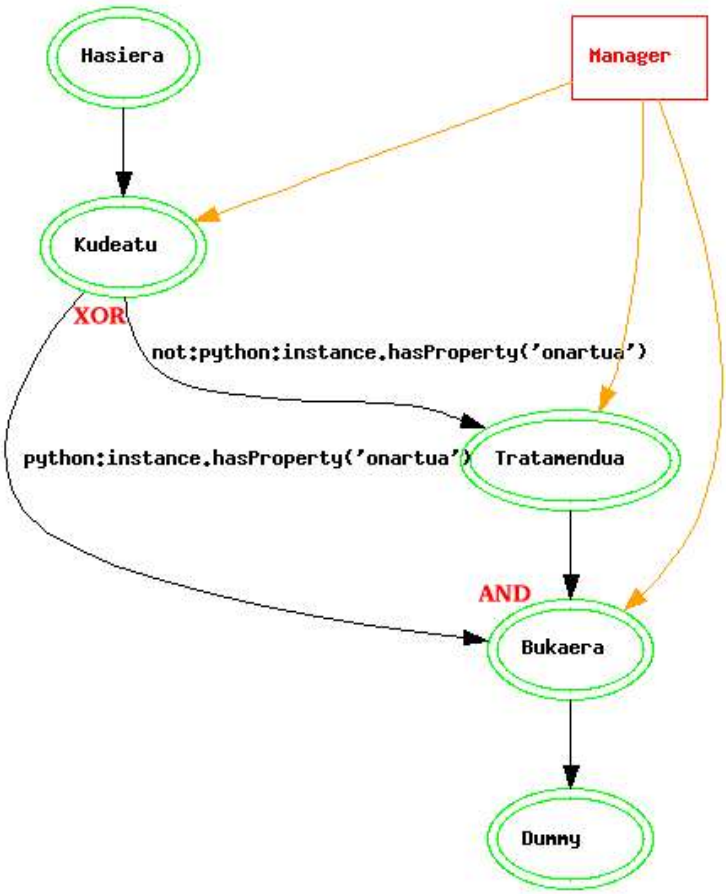
Sarrera trantsiziorik gabeko ekintzetara instantziak ezingo dira iritsi, hala ere salbuespenen kudeaketa egiten ari bagara, komenigarria izan daiteke horrelako egoera bat izatea, egoera zehatzen batean dauden instantziak egoera berezi horretara eskuz eraman eta bertatik berriz abiatzeko. Irteera trantsiziorik gabeko ekintzetan, ordea, instantziak bertan gelditzuko dira eta ezingo dute aurrerantz jarraitu. Azken kasu hau, bukaera definitu gabe duten prozesuen antzekoa da.

Ekintza batek sarrera trantsiziorik ez badu eta ez bada prozesuaren hasierako ekintza, ezin da errore mezurik eman instantziak ez baitira inoiz bertara iritsiko.

Ekintzaren batek irteerako trantsiziorik definituta ez badu eta bukaerako egoera ez baldin bada, instantzia *fallout* salbuespen egoeran gelditzen da bertara heltzean, eta salbuespenak berreskuratzeko jarraitu behar diren pauso berdinak eman behar dira instantzia bertatik berreskuratzeko. Kasu hau, prozesu baten bukaera egoera definitu gabe dagoeneko kasuaren antzekoa da.

8.1.3. *and* eta *xor* moduko sarrera eta irteera babesen kontrola

Ekintza baten irteera trantsizioak era esklusiboan, *xor* eran, irteten badira eta *and* eran konfiguratutako ekintza batean biltzen bada, instantziak ezingo du jarraitu, beste adarretatik etorri beharko liratekeen ekintzen zain gelditzuko baita.



Irudia 25: and eta xor babesen adibidea

Irudia 25ko lan-fluxuan gaizki konfiguratutako babesen adibide bat daukagu. *Bukaera and* gisa konfiguratuta dagoenez bere bi sarrera trantsizioetatik (*Tratamendua* eta *Kudeatu* ekintzetatik datozenetatik) lan-itemak heltzeko zain gelditukо da, *Dummy* ekintzara pasatu aurretik. Lan-fluxu horren instantzia baten historia begiratzen badugu, *Bukaera* egoerari dagokion lan-itema, beste bat heltzeko zain dagoela ikus daiteke. Lan-item berria sortu dela ikus daiteke baina blokeatuta dago (ikusi Irudia 26 eta Irudia 27), inoiz iritsiko ez den lan-item baten zain.

Instantzia lan-itemen baten zain blokeatuta baldin badago, ezin dugu egoera horretatik atera. Egin dezakegun gauza bakarra instantzia hori bertan behera uztea da eta horrekin exekuzio horri buruzko informazio guztia galduko litzateke. Horregatik da komenigarria horrelako gertaerak gerta daitezkeela aurreikustea eta erabiltzailea

ohartaraztea nahi ez diren portaerak ager ez daitezen.

Irudia 26: Lan-itema blokeatuta beste bat iristeko zain

Id	3
Activity	Bukaera (in process *Adibidea1*)
Actor	
From	['2']
Status	blocked
To	[]
Events	• creation (2004/06/28 09:36:28.590 GMT+2)

Irudia 27: Lan-itema sortu egin da baina blokeatuta dago

8.2. Diseinua eta inplementazioa

Diseinua eta inplementazioa bi zatitan banatu dut. Alde batetik, aurreko atalean azaldutako lehenengo bi kasuen inplementazioa egin dut eta beste alde batetik hirugarren kasuarena. Lehenengo biek antzerako tratamendua behar dute, eta erabiltzailearentzat errazagoa da bi kasuei dagokien informazioa toki batetik atzitzea. Azken kasuak aldiz, gaizki konfiguratutako sarrera eta irteera babesenak, beste inplementazio bat eskatzen du eta horregatik banatu dut bestetik bere inplementazioa.

8.2.1. Gaizki konfiguratutako egoeren kontrola

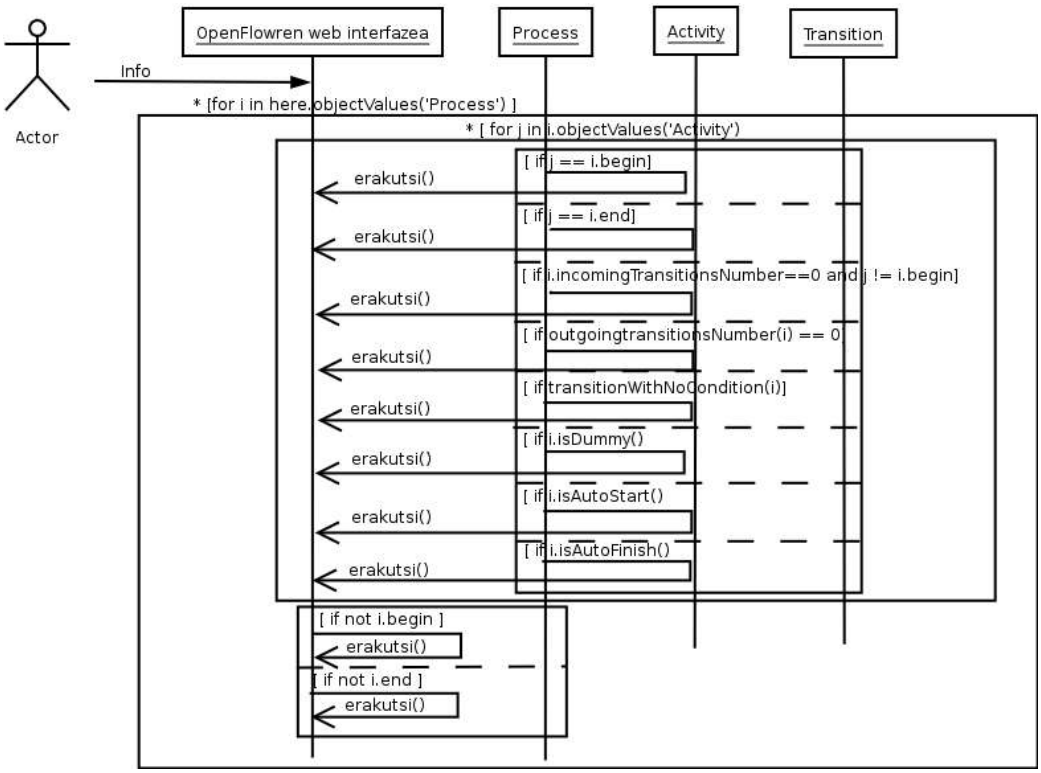
Lehenengo bi kasuen inplementaziorako, prozesu eta ekintzen metodoak erabili ditut. Metodo horietaz baliatuz, lortzen dut prozesu eta ekintzetatik behar den informazioa. Bere sekuentzia diagrama Irudia 28koa da.

Sekuentzia diagraman ikusten diren metodo batzuk irudia sinplifikatzearren neuk jarritakoak dira eta jarraian azaltzen dut esan nahi dutena.

- *outgoingTransitionsNumber(act):* metodo honekin *act* ekintza honetatik abiatzen den

trantsiziorik dagoen ala ez itzultzen du. Inplementatzerakoan dauden trantsizio guztiak lortu behar dira eta bertatik irteten direnak dauden ala ez lortu.

- *transitionWithNoCondition(act):* metodo honekin *act* ekintzatik trantsizio bat baino gehiago irteten bada hauek baldintzarik ezarrita duten egiaztatzen du.
- *erakutsi:* metodo honek dagokion baldintza betetzen den egiaztatu ostean erabiltzaileari eman beharreko informazioa erakusten du OpenFlowren interfazean.

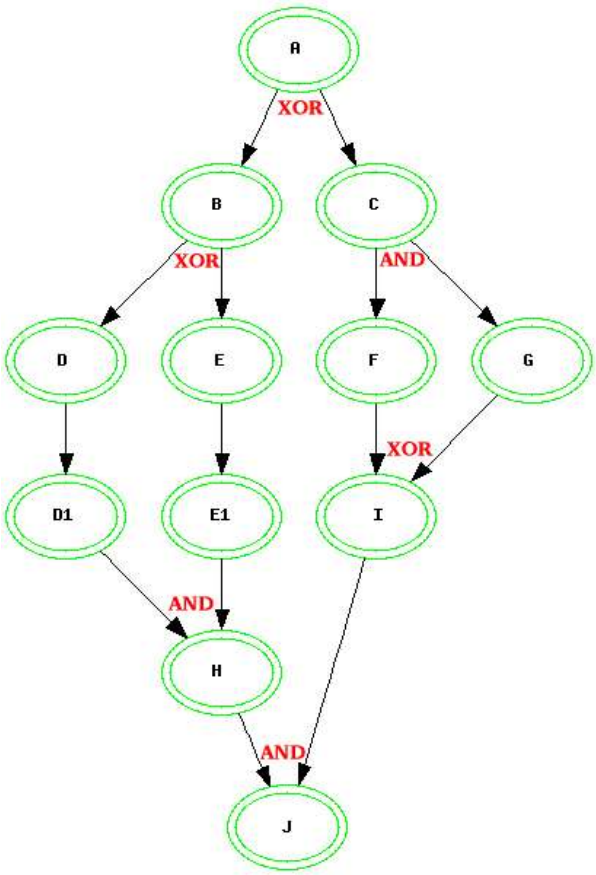


Irudia 28: Lehenengo zatiaren sekuentzia diagrama

8.2.2. Gaizki konfiguratutako babesen kontrola

Hasieran aipatu dugun hirugarren kasurako, hots, gaizki konfiguratutako sarrera eta irteera babesen kasurako bi aukera bereiztu ditut.

1. *xor* eran banatzen den ekintzatik irteten diren trantsizioetatik zuzenean zein ekintzatara iristen garen lortu eta ekintza horiek elkarren artean irisgarriak diren egiaztatzen dut. Irudia 25eko adibidea hartuz, *Kudeatu*tik bi trantsizio irteten dira: bata *Tratamendua* ekintzara eta bestea *Bukeara*ra. Kasu horretan *Tratamendua*tik *Bukaera*ra edo alderantziz, heltzerik posible den begiratuko litzateke.



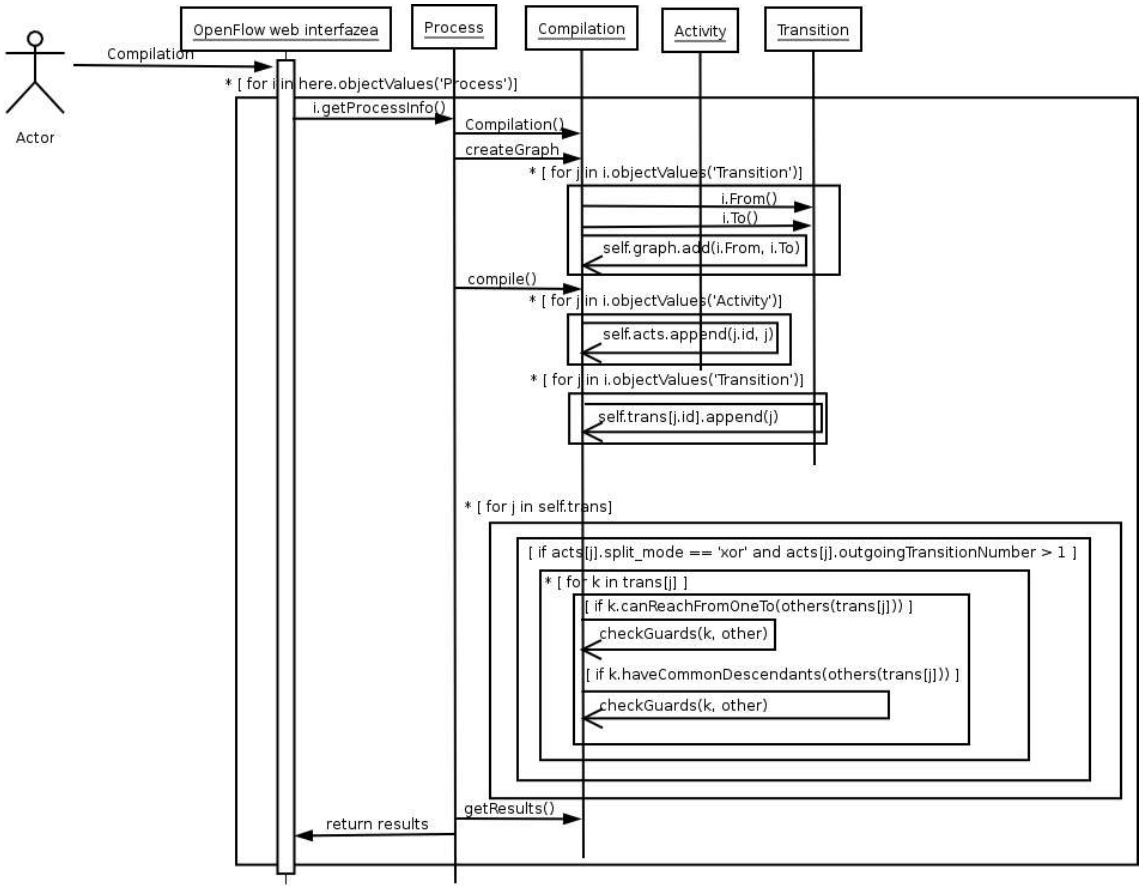
Irudia 29: Gaizki konfiguratutako babesen adibidea

2. Ekintza batetik banatzen diren trantsizioen bidez ekintza berdinetara iristerik badagoen begiratzen dut, hau da, ondorengo amankomunik dutenentz egiaztatzen

dut. Irudia 29ko lan-fluxua da horren adibide. *A* ekintzatik irteten diren trantsizioetatik *B* eta *C* ekintzetara irits daiteke zuzenean. Bi horietatik irisgarri diren ekintzen artean bat bakarrik dute amankomunean, *J*. Orduan *A* eta *J* artean babesak ondo konfiguratuta dauden ala ez egiaztatu behar da. Kasu honetan gaizki konfiguratuta daude. Gauza bera egin behar da *B* eta *H,* eta *C* eta *I* artean.

Gaizki konfiguratutako babesen kasuari dagokion sekuentzia diagrama ikus daiteke Irudia 30en. Sekuentzia diagrama horretan sinplifikatzearren erabili ditudan metodo batzuek azalpena emango dut hurrengo parrafoetan.

- *checkGuards(from,to):* Funtzio honek *to* bezala pasatutako ekintza *and* moduan konfiguratuta dagoen, sarrera trantsizio bat baino gehiago duen eta *from* ekintza konpilazioaren emaitzetan dagoen egiaztatzen du, babesa gaizki konfiguratuta dagoen detektatzeko.



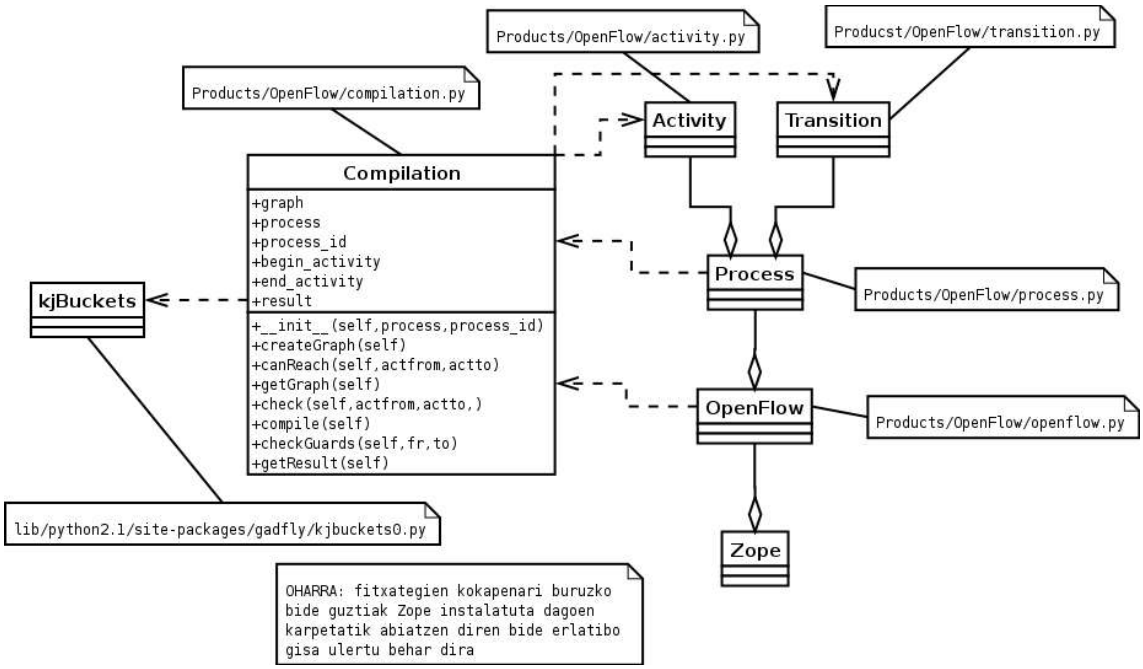
Irudia 30: Konpiladorearen sekuentzia diagrama

- *from.canReachFromOneTo(other):* Gaizki konfiguratutako babesak detektatzeko lehenengo kasua betetzen den egiaztatzen du, hau da, ekintza batetik (*from*) besteetara (*others*) irits daitekeen.
- *from.haveCommonDescendants(other)*. Bigarren kasua betetzen den detektatzeko funtzioa. Bi ekintzek (*from* eta *other*) ondorengo amankomunik duten egiaztatu eta horrela bada ondorengo hori edo horiek itzultzen ditu.

8.2.3. Klase-diagrama

Gaizki konfiguratutako sarrera eta irteera babesen kasurako, ekintza eta trantsizioek osatzen duten grafoaren gainean lan egitea erabaki dut. Horretarako Python programazio lengoaiarako *kjbuckets* [Watters] modulua erabili dut. Modulu honek grafo, multzo eta

hiztegi datu-motak inplementatzen ditu. Irudia 31n ikus daiteke konpiladorearen klase diagrama.



Irudia 31: Konpiladorearen klase diagrama

Klase diagraman ikus daitekeen bezala, OpenFlow eta Process klaseek darabilte konpiladorearen klasea eta klase hau da grafoen inplementazioa daukan *kjbuckets* modulua erabiltzen duena grafoaren errepresentazioa gordetzeko. Konpiladoreak Activity eta Transition klaseak erabiltzen ditu prozesu baten errepresentatuta dauden ekintza eta trantsizioen informazioa lortzeko.

Inplementaziorako, hasiera baten Datu Egiturak eta Algoritmoak II ikasgaian grafoak korritzeko ikasitako metodoak erabili nituen: zabalerako eta sakonerako korritzeak.

Zabalerako korritzeak grafoa hasierako nodo batetik abiatuz korritzen du, nodo horren ume guztiak zerrenda baten gehituz eta bakoitza banan-banan tratatuz. Lana modu sekuentzialean egiten du, ume bakoitzeko bere semeak ere zerrenda beraren bukaeran gehituz.

Sakonerako korritzeak, aldiz, umeen umeak tratatzen ditu lehenengo, grafoaren sakonereraino joanez eta ondoren hasierako nodoaren hurrengo umetik lanean jarraituz.

Hala ere ez nuen lortu horiek erabiliz identifikatutako kasu guztiak detektatzea. Horregatik datu-egitura hori erabiltzea birplanteatu egin behar izan nuen.

OpenFlowk trantsizio eta ekintzen informazioa bere barnean gordetzen duenez, datu horiek zuzenean erabiltzea erabaki nuen. Hala ere, grafoa sortu egiten dut laguntza gisa erabiltzeko, nodo batetik zein nodotara irits daitekeen jakiteko, horretarako *kjbuckets* moduluko grafo datu-egiturak metodo batzuk eskaintzen baititu.

Grafoa zuzenean erabili beharrean, Pythoneko hiztegi baten (ikusi 4.1.2 atala) gordetzen ditut ekintzen identifikadoreak eta ekintza objektuak eta beste baten ekintza bakoitzetik irteten diren trantsizioetatik joanda irisgarri diren nodoak. Adibidez, Irudia 25n azaldutako lan-fluxuan, ondoko datu-egiturak izango genituzke:

```
ekintza_hiztegia = {'Hasiera':<activity object>, 'Kudeatu':<activity
object>, 'Tratamendua':<activity object>, 'Bukaera':<activity object>,
'Dummy':<activity object>}

trantsizio_hiztegia = {'Hasiera':['Kudeatu'], 'Kudeatu':
['Tratamendua', 'Bukaera'], 'Tratamendua':['Bukaera'], 'Bukaera':
['Dummy']}
```

Behin datu horiek edukita, trantsizioen hiztegiaren elementu bakoitzeko, dagokion ekintzak *xor* erako irteera babesa eta irteera trantsizio bat baino gehiago duen egiaztatzen dut. Ondoren identifikatutako bi kasuak betetzen diren ala ez begiratzen dut, horrela bada bukaerako ekintza *and* moduan konfiguratuta dagoen begiratzeko.

8.2.4. Inplementazioa

Inplementaziorako OpenFlow produktuan aldaketa txiki batzuk egin behar izan ditut, aurretik azaldutako kasuen informazioa OpenFlowren kudeaketa interfazeak berak emateko.

Zoperen interfaze nagusitik produktuak kudeatzean, objektu bakoitzak kudeatze orrialde bat baino gehiago du fitxatan antolatuta eta fitxa horietako bi sortu ditut, Irudia 32n ikusten diren *Info* eta *Compilation*, lehenengoa lehen kasu bien inplementaziorako eta bigarrena azkenengo kasurako.

Irudia 32: Zoperen kudeaketa fitxak

Fitxa horien inplementaziorako 4.4. atalean azaldutako *Zope Page Templates* deritzanak erabili ditut, OpenFlow objektuaren metodoak atzitu eta erabiltzeko eta jasotako informazioa interfazean erakusteko. Fitxen inplementazioa *Info.zpt* eta *Compilation.zpt* eta izeneko fitxategietan dago.

Erabiltzaileak moduluak erabiltzeko orduan bi lekutatik atzitu ahalko du moduluak sortutako informazioa, aipatutako kudeaketa fitxak bi lekutan sortu baitut. Alde batetik, OpenFlow objektuaren kudeaketa fitxa gisa, objektuan errepresentatuta dauden prozesu guztien inguruko informazioa lor daiteke. Bestetik, prozesu zehatz baten kudeaketa fitxen artean ere gehitu ditut berri hauek, eta bertatik prozesu horren informazioa aztertu daiteke. Bigarren zati hau inplementatzeko, lehenengo kasuan erabilitako fitxategiak berrerabili ditut inplementazioaren lerro batzuk aldatuta, prozesu guztiak begiratu beharrean kokatuta dauden prozesuko informazioa hartu ahal izateko.

8.3. Aurkitutako arazoak

Izan dudan arazo nagusiena *and* eta *xor* babesak gaizki konfiguratuta daudela nola detektatu asmatzea izan da. Aurrerago esandako moduan, lehenengo grafoen zabalera eta sakonerako korritzeekin saiatu nintzen baina ez nuen egin nahi nuena lortu. Azkenenean *xor* erako ekintzetatik irteten diren trantsizioak aztertuz lortu dut informazioa lortzea.

Kudeaketa fitxak inplementatzeko erabili dudan *Zope Page Templates* tresnarekin ere arazoak izan ditut, batez ere bere erabilpena ikasterakoan, ez baita oso intuitiboa. Kudeaketa fitxak HTML dokumentuak dira azken finean, baina ZPTei esker dinamikotasuna lortzen dute. Dinamikotasun hori HTML etiketetan txertatzen diren hitz gako batzuei esker lortzen da, eta programazio lengoaia bat izango balitz bezala, sintaxi zehatz bat daukate eta askotan bertako erroreak aurkitu eta konpontzea ez da batere

erraza izaten.

9. ONDORIOAK

Atal honetan proiektuarekin atera ditudan ondorioak azalduko ditut. Lehenik kudeaketari dagozkion ondorioak, bigarrenik proiektuari buruzkoak eta azkenik pertsonalki proiektu honen inguruko nire iritzi eta balorazio pertsonala zein diren azalduko dut.

9.1. Kudeaketaren ondorioak

Proiektuaren plangintza egitea ez da lan erraza izan niretzat. Alde batetik, horrelako proiektuen kudeaketa eta garapenean esperientziarik ez izateak eragin handia izan baitu, bestetik maila honetako proiektu bat bakarkako lanean aurrera eraman dudan lehen aldia izan baita.

Plangintza guztiz osatu gabe zegoela hasi zen proiektua eta horren ondorioz aldaketa batzuk egin behar izan ditut beranduago. Gainera proiektuaren garapena nolabait atzeratu egin zen otsaila eta martxoa inguruan, alde batetik egindako kontsultei OpenFlow produktuaren garatzaileek ez zietelako erantzunik ematen eta bestetik enpresaren beste proiektu baterako lan-fluxu batzuk OpenFlowrekin inplementatzen aritu nintzelako.

OpenFlowren garatzaileen erantzunak beharrezkoak ziren, alde batetik garatutako XPDL – OpenFlow inportatzailea eta esportatzailea probatu eta euren emaitzak jaso nahi nituelako hobekuntzak eta beharrezko aldaketak egin ahal izateko. Beste alde batetik, hasiera batean hitzartutako salbuespenen kudeatzailea garatzeko euren iritziak eta produktuak bete behar zituen ezaugarriak ere jakin egin behar nituen, hasieratik ez baitzidaten honen inguruko daturik eman.

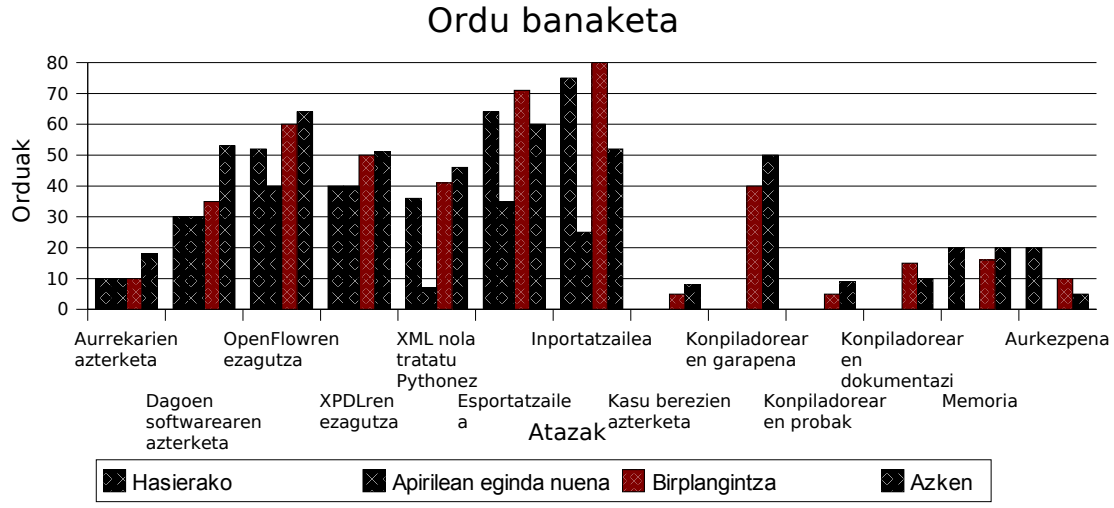
Horrekin batera etorri zen, enpresaren beste proiektu baterako lan-fluxu batzuen inplementazioa nola egingo litzatekeen aztertzea, eta horretan aritu nintzen urtearen lehenengo hilabeteetan.

Ondoren [PAFlow] eta Shark [Enhydra Shark] lan-fluxu zerbitzariak instalatu eta probatzen aritu nintzen, zituzten ezaugarriak hobeto aztertzeko. Batez ere Shark probatu

nuen, esportatzaileak sortutako XPDL fitxategien zuzentasuna probatzeko erabili nuen.

Planifikatutako orduen betetze mailari buruz, esan planifikazioari hobekuntzak egin ahal zaizkiola, batez ere, ez bainuen uste azterketaren alderdia izango zela denbora gehien hartuko zidana. Azterketaren alderdia neketsuagoa izan da, batetik gai honen inguruan Interneten aurki daitekeen informazioa ingelesez dagoelako eta bestetik, ezagutzen ez den gai baten inguruko testu teknikoak ulertzea erraza izan ohi ez delako, batez ere beste hizkuntza batean idatzitako testuak badira. Inplementazioan sartutako orduak uste baino gutxiago izan dira, uste baino erraztasun handiagoa izan baitut Python eta Zope erabiliz moduluak garatzerakoan.

Irudia 33n ikusi daiteke zein alde egon den hasierako plangintza, birplangintza eta sartutako ordu kopuru errealaren artean.



Irudia 33: Plangintza eta sartutako ordu kopuruaren arteko alderaketa

Irudian agertzen ez dena, hasierako plangintzan datorren salbuespen kudeatzaileari dagokion ordu kopurua da, hasierako plangintzan 127 orduko lana balioetsi nuen fase horrentzat baina apirileko birplangintzan fase horren ordez konpiladorea garatzea erabaki nuen, eta hori bai agertzen da aurreko grafikoan.

Taula 2n ataza bakoitzean sartutako ordu kopurua ikus daiteke.

Ataza	*Ordu kopurua*
Aurrekarien azterketa	18
Dagoen softwarearen azterketa	53
OpenFlowren ezagutza	64
XPDLren ezagutza	51
XML nola tratatu Pythonez	46
Esportatzailea	60
Inportatzailea	52
Kasu berezien azterketa	8
Konpiladorearen garapena	50
Konpiladorearen probak	9
Konpiladorearen dokumentazioa	10
Memoria	20
Aurkezpena	5
Guztira	**446**

Taula 2: Ataza bakoitzean sartutako ordu kopurua

9.2. Proiektuaren ondorioak

Merkatuan dauden lan-fluxu sistemak aztertuta atera dudan ondorio nagusia, lan-fluxuen mundua oraindik garapenean dagoen merkatua dela da. Azkenaldian lan-fluxu sistemak zabaltzen ari dira, enpresa, ikastetxe eta bestelako erakundeak ISO kalitate ziurtagiriak lortu nahian, beren lan prozesuak estandarizatzen eta kontrolpean eduki nahian dabiltzalako.

Hori da merkatuan lan-fluxu sistema asko egotearen arrazoietako bat. Beste arrazoi bat lan-fluxuen estandarizazio lanen zabalpen eza da eta ondorioz produktu estandar bat ez egotearena. Nahiz eta [WfMC] eta [BPMI] erakundeak lan horretan ibili, oraindik bere emaitzak ez daude guztiz merkaturatuta.

OpenFlow da, Zope plataforman, lan-fluxuak erabiltzeko tresna interesgarriena,

eskaintzen duen estandarren jarraipen eta malgutasunaren ikuspuntutik. Malgutasun horri esker lor daiteke, adibidez, Italiako Administrazio Publikoaren Informatika Erakundeak [CNIPA] aurrera daraman PAFlow proiektua, memoria honetan aurkezten den proiektuan erabili den OpenFlow lan-fluxu sistemarekin garatutakoa.

Software askearen garapenaren aldetik eta estandarizazio lanetan lagunduz, garatu da memoria honetan aurkezten den software moduluetako bat. OpenFlow eta XPDL formatu estandarraren arteko bihurketarako balio duena. OpenFlow, [WfMC] erakundearen lanen ondorioetan oinarrituta sortu zuten, eta helburua estandarrak jarraitzen dituen kode irekiko lan-fluxu sistema bat sortzea da eta horretarako beharrezko ikusten zuten garatzaileek bihurketa lanetarako tresna bat bertan integratzea.

Konpilazio lanaren helburuak ere bete dira, lan-fluxuak diseinatzerakoan egiten diren akatsen inguruan abisatzen baitu orain produktuak. Gainera, konpilazioaz arduratzen den tresna, OpenFlow produktuaren kudeaketa fitxa bezala integratzea lortu da, lan-fluxua diseinatzerakoan klik bakar batekin arazoak ematen dituzten kasuak aztertzea ahalbidetuz. Gainera, konpilazioaren emaitzak ematen diren fitxetan bertan, aztertutako prozesuetarako loturak jarri ditut, aldaketak egiteko bidea errazagoa izan dadin.

9.3. Balorazio eta iritzi pertsonala

Memorian azaltzen den proiektuaren garapenean erabili dudan Zope plataforma eta bere oinarrian dagoen Python lengoaian trebatzea oso interesgarria izan da, lan-fluxuen mundua aldiz, enpresa eta beste erakundeentzako gauzak era egoki eta ordenatuan egiteko tresna izan arren, oraindik jendeak erabiltzeko tresna zaila delako susmoa hartu dut.

OpenFlow produktuaren garapenaren aldetik, tresna interesgarria dela deritzot eta iturburu kode aldetik oso garbia eta aldatzeko edo bakoitzaren beharretara moldatzeko erraztasunak eskaintzen dituena. Horretaz gain, bere APIko funtzioen zerrenda, dituzten parametroekin eta funtzio bakoitzak egiten duenaren azalpen batekin, laburtuta lor daiteke garatzaileen webgunean. Bestalde, dokumentazioa da bere puntu ahula, ez

baitago asko. Produktuarekin datorren apurra laguntza gisa integratuta dator eta gainontzekoa webgunean kontsulta daiteke. Bestalde, posta-zerrendaren bidez OpenFlow darabilen komunitatea harremanetan jartzea alde positibo bezala hartu behar da, nahiz eta azken asteetan mugimendu gutxi ibili.

Proiektua enpresa batean garatzeak ere alderdi positiboak izan ditu nire iritziz, bertan lan nola egiten den ikasi eta lankideekiko harremanean batez ere, karrera bukaerako proiektua denez, hau bukatu ostean egin beharko dudanera hurbildu nauelako, batez ere.

Software askearen munduan parte hartzea, esperientzia atsegina izan da niretzat, hainbat produkturen posta zerrenda eta foroetan parte hartzeak eta bertatik kode irekiaren munduak nola funtzionatzen duen ikasteak ondorio positiboak baitakartza nire ustez. Batez ere, beste norbaitek idatzitako kodea aztertu eta aldatzeko askatasun osoa ematen du, produktuetan aldaketa edo hobekuntzak bakoitzaren neurrira egiteko. Gainera inplementatutako moduluak OpenFlowren garatzaileen posta zerrendan argitaratuko ditut inork erabili nahi baditu horrela egin dezan. XPDL eta OpenFlow arteko bihurketa egiten duen modulua jadanik badute OpenFlowren garatzaileek baina ez didate berari buruzko ezer esan oraindik.

Dokumentazio eta kudeaketa lanetan ere esperientzia hartu dut, batez ere gauzak egin ahala dokumentatzeko orduan, gerorako utziz gero gauza asko ahaztu edo galdu egiten baitira.

Laburbilduz, pozik nago egindako lanarekin, batez ere Python lengoaia eta Zope plataformaren inguruko ezagupenekin.

10. BIBLIOGRAFIA ETA ERREFERENTZIAK

[Allen, 2001] Allen, Rob (2001). *Workflow: An Introduction*. Open Image Systems Inc. United Kingdom http://www.wfmc.org/standards/docs/Workflow_An_Introduction.pdf

[BPMI] Business Process Management Initiative. 1155 S. Havana Street, #11-311 Auroar, CO 80012. USA http://www.bpmi.org

[Brown, 2002] Brown, Martin C. (2002). *XML Processing with Perl, Python, and PHP*.

[CapeVisions] CapeVisions. *XPDL Extension for Visio* http://www.capevisions.com/tech/wxml.shtml

[CNIPA] Centro nazionale per l'informatica nella pubblica amministrazione. Via Isonzo, 21/B. 00198 Roma. Italia http://www.cnipa.gov.it

[CS] CodeSyntax Internet Solutions. Azitain industrialdea, 3-K Eibar http://www.codesyntax.com

[DOM IG] XML DOM Interest Group. Document Object Model. W3C. http://www.w3.org/DOM/

[Downey, Elkner and Meyers, 2003] Downey, A., Elkner, J. and Meyers, C. (2003). *How to Think Like a Computer Scientist: Learning with Python*. http://greenteapress.com/thinkpython/

[Enhydra Jawe] Enhydra (2004). *Java Workflow Editor*. http://jawe.objectweb.org

[Enhydra Shark] Enhydra (2004). *Shark Workflow Server*. http://shark.objectweb.org

[Faasen] Faasen, Martijn. *Formulator product for Zope* http://www.zope.org/Members/infrae/Formulator

[Feith] Feith Systems and Software, Inc. 425 Maryland Drive, Fort Washington, PA 19034. USA. http://www.feith.com

[Flowring] Flowring Technology Corp. 12F, No.120, Sec2, Gongdao 5th Rd., Hsinchu City 300, Taiwan (R.O.C.) http://www.flowring.com

[Fountain, 2004] Fountain, Derek (2004). *SAX processing in Python*. http://www.devchannel.org/webserviceschannel/04/05/25/

1414203.shtml

[Icube] Icube. *OpenFlow – OpenSource Workflow Management System* http://www.openflow.it

[Integrated Workflow] Integrated Workflow. Suite 18, 39-51 Highgate Raod. London. NW5 1RS. United Kingdom http://www.integratedworkflow.com

[Kamath] Kamath, H. (2002) ZPT Basics. *Devshed.com* http://www.devshed.com/c/a/Zope/ZPT-Basics-part-1 http://www.devshed.com/c/a/Zope/ZPT-Basics-part-2 http://www.devshed.com/c/a/Zope/ZPT-Basics-part-3 http://www.devshed.com/c/a/Zope/ZPT-Basics-part-4

[Kiepuszewki, 2002] Kiepuszewki, Bartosz (2002). *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows. Dissertation presented to the Faculty of Information Technology Queensland University of Technology in fulfilment of the requirements for the degree of Doctor of Philosophy.*

[Latteier, Pelletier, McDonough and Sabaini, 2003] Latteier, A., Pelletier, M., McDonough, C. and Sabaini, P. (2003). *The Zope Book (2.6 Edition)*. http://zope.org/Documentation/Books/ZopeBook/2_6Edition/

[Lemmi, 2002] Lemmi, Ricardo (2002). *OpenFlow ver. 0.8 Manuale dello sviluppatore*. http://www.zope.it/Members/rlemmi/manuale_OpenFlow.pdf

[Marzal & Gracia, 2003] Marzal, A., Gracia. I. *Introducción a la programación con Python.* Departamento de Lenguajes y Sistemas Informáticos. Universitat Jaume I Castelló. Apuntes para la asignatura Metodología y Tecnología de la Programación del primer curso de la Ingeniería Informática e Ingeniería Técnica de Informática de Gestión

[McDonough, Pelletier and Hathaway, 2003] McDonough, C., Pelletier, M. and Hathaway, S. (2003) *The Zope Developer's Guide (2.4 Edition)*. http://zope.org/Documentation/Books/ZDG/current/

[Megginson] Megginson, D. Simple API for XML. http://www.saxproject.org/

[PAFlow] PAFlow. *Software libero per il protocollo informatico e la gestione dei flussi documentali in una Pubblica Amministrazione*. Icube

[Pilgrim, 2004] Pilgrim, Mark (2000-2004). *Dive Into Python*. http://diveintopython.org

[Plesums, 2002] Plesums, Charles (2002). *Introduction to Workflow*. Computer Sciences Corporation, Financial Services Group. http://www.wfmc.org/information/introduction_to_workflow02.pdf

[Prior, 2003] Prior, Carol (2003). *Workflow and Process Management*. Maestro BPE Pty Limited, Australia http://www.wfmc.org/information/Workflow_and_Process_Management.pdf

[Robotiker] Robotiker. Gestión de flujos de trabajo y gestión documental avanzada. *Robotiker: Revista de tecnologías emergentes. Número 0.* http://revista.robotiker.com/articulos/articulo3/pagina1.jsp

[SIG XML Py] Special Interest Group for XML Processing in Python http://www.python.org/sigs/xml-sig/ http://pyxml.sourceforge.net/

[Van Rossum] Van Rossum, Guido (1990-2004). Python Programming Language. Amsterdam http://www.python.org

[W3C] W3C http://www.w3.org/XML/Schema

[Watters] Watters, A. *Set and Graph Datatypes for Python: kjbuckets* http://starship.python.net/crew/aaron_watters/kjbuckets/kjbuckets.html http://gadfly.sourceforge.net/kjbuckets.html http://sourceforge.net/projects/gadfly/

[WfMC-RM, 1995] Workflow Management Coalition. *Workflow Reference Model* (1995) TC00-1003

[WfMC-XPDL, 2002] Workflow Management Coalition. *XML Process Definition Language* (2002) WFMC-TC-1025

[WfMC] Workflow Management Coalition. 2436 N. Federal Highway, No. 374. Lighthouse Point, Florida 33064, USA. http://www.wfmc.org

[Zope Catalog] Zope Corporation. *ZCatalog*

http://www.zope.org/Documentation/How-To/ZCatalogTutorial

[Zope] Zope Corporation (1996-2004). *Z Object Publishing Environment*. http://www.zope.org http://www.zope.com

A ERANSKINA: XPDL FORMATUA

Eranskin honetan XPDL formatuaren definizioa dakarren XML-Schema fitxategia aurkezten da. Formatuari buruzko informazio osagarria lortu nahiez gero, gomendagarria da [WfMC] erakundearen webgunea eta [WfMC XPDL, 2002] dokumentua kontsultatzea .

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.wfmc.org/2002/XPDL1.0"
xmlns:xpdl="http://www.wfmc.org/2002/XPDL1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:element name="Activities">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Activity" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Activity">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
            <xsd:element ref="xpdl:Limit" minOccurs="0"/>
            <xsd:choice>
               <xsd:element ref="xpdl:Route"/>
               <xsd:element ref="xpdl:Implementation"/>
               <xsd:element ref="xpdl:BlockActivity"/>
            </xsd:choice>
            <xsd:element ref="xpdl:Performer" minOccurs="0"/>
            <xsd:element ref="xpdl:StartMode" minOccurs="0"/>
            <xsd:element ref="xpdl:FinishMode" minOccurs="0"/>
            <xsd:element ref="xpdl:Priority" minOccurs="0"/>
            <xsd:element ref="xpdl:Deadline" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element ref="xpdl:SimulationInformation"
minOccurs="0"/>
            <xsd:element ref="xpdl:Icon" minOccurs="0"/>
```

```
            <xsd:element ref="xpdl:Documentation" minOccurs="0"/>
            <xsd:element ref="xpdl:TransitionRestrictions"
minOccurs="0"/>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Name" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ActivitySet">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Activities" minOccurs="0"/>
            <xsd:element ref="xpdl:Transitions" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ActivitySets">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ActivitySet" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ActualParameter" type="xsd:string"/>
   <xsd:element name="ActualParameters">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ActualParameter" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Application">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
            <xsd:choice>
               <xsd:element ref="xpdl:FormalParameters"/>
```

```
               <xsd:element ref="xpdl:ExternalReference"
minOccurs="0"/>
            </xsd:choice>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Name" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Applications">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Application" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ArrayType">
      <xsd:complexType>
         <xsd:group ref="xpdl:DataTypes"/>
         <xsd:attribute name="LowerIndex" type="xsd:NMTOKEN"
use="required"/>
         <xsd:attribute name="UpperIndex" type="xsd:NMTOKEN"
use="required"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Automatic">
      <xsd:complexType/>
   </xsd:element>
   <xsd:element name="BasicType">
      <xsd:complexType>
         <xsd:attribute name="Type" use="required">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="STRING"/>
                  <xsd:enumeration value="FLOAT"/>
                  <xsd:enumeration value="INTEGER"/>
                  <xsd:enumeration value="REFERENCE"/>
                  <xsd:enumeration value="DATETIME"/>
                  <xsd:enumeration value="BOOLEAN"/>
```

```
                  <xsd:enumeration value="PERFORMER"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="BlockActivity">
      <xsd:complexType>
         <xsd:attribute name="BlockId" type="xsd:NMTOKEN"
use="required"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Codepage" type="xsd:string"/>
   <xsd:element name="Condition">
      <xsd:complexType mixed="true">
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="xpdl:Xpression"/>
         </xsd:choice>
         <xsd:attribute name="Type">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="CONDITION"/>
                  <xsd:enumeration value="OTHERWISE"/>
                  <xsd:enumeration value="EXCEPTION"/>
                  <xsd:enumeration value="DEFAULTEXCEPTION"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ConformanceClass">
      <xsd:complexType>
         <xsd:attribute name="GraphConformance">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="FULL_BLOCKED"/>
                  <xsd:enumeration value="LOOP_BLOCKED"/>
                  <xsd:enumeration value="NON_BLOCKED"/>
               </xsd:restriction>
            </xsd:simpleType>
```

```
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Cost" type="xsd:string"/>
   <xsd:element name="CostUnit" type="xsd:string"/>
   <xsd:element name="Countrykey" type="xsd:string"/>
   <xsd:element name="Created" type="xsd:string"/>
   <xsd:element name="DataField">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:DataType"/>
            <xsd:element ref="xpdl:InitialValue" minOccurs="0"/>
            <xsd:element ref="xpdl:Length" minOccurs="0"/>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Name" type="xsd:string"/>
         <xsd:attribute name="IsArray" default="FALSE">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="TRUE"/>
                  <xsd:enumeration value="FALSE"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="DataFields">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:DataField" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="DataType">
      <xsd:complexType>
         <xsd:group ref="xpdl:DataTypes"/>
      </xsd:complexType>
```

```
   </xsd:element>
   <xsd:group name="DataTypes">
      <xsd:choice>
         <xsd:element ref="xpdl:BasicType"/>
         <xsd:element ref="xpdl:DeclaredType"/>
         <xsd:element ref="xpdl:SchemaType"/>
         <xsd:element ref="xpdl:ExternalReference"/>
         <xsd:element ref="xpdl:RecordType"/>
         <xsd:element ref="xpdl:UnionType"/>
         <xsd:element ref="xpdl:EnumerationType"/>
         <xsd:element ref="xpdl:ArrayType"/>
         <xsd:element ref="xpdl:ListType"/>
      </xsd:choice>
   </xsd:group>
   <xsd:element name="Deadline">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="DeadlineCondition" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="ExceptionName" minOccurs="1"
maxOccurs="1"/>
         </xsd:sequence>
         <xsd:attribute name="Execution">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="ASYNCHR"/>
                  <xsd:enumeration value="SYNCHR"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="DeclaredType">
      <xsd:complexType>
         <xsd:attribute name="Id" type="xsd:IDREF" use="required"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Description" type="xsd:string"/>
   <xsd:element name="Documentation" type="xsd:string"/>
   <xsd:element name="Duration" type="xsd:string"/>
   <xsd:element name="EnumerationType">
```

```
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:EnumerationValue"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="EnumerationValue">
      <xsd:complexType>
         <xsd:attribute name="Name" type="xsd:NMTOKEN"
use="required"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ExtendedAttribute">
      <xsd:complexType mixed="true">
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:any processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:choice>
         <xsd:attribute name="Name" type="xsd:NMTOKEN"
use="required"/>
         <xsd:attribute name="Value" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ExtendedAttributes">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ExtendedAttribute" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ExternalPackage">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="href" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ExternalPackages">
```

```
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ExternalPackage" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ExternalReference">
      <xsd:complexType>
         <xsd:attribute name="xref" type="xsd:NMTOKEN"
use="optional"/>
         <xsd:attribute name="location" type="xsd:anyURI"
use="required"/>
         <xsd:attribute name="namespace" type="xsd:anyURI"
use="optional"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="FinishMode">
      <xsd:complexType>
         <xsd:choice>
            <xsd:element ref="xpdl:Automatic"/>
            <xsd:element ref="xpdl:Manual"/>
         </xsd:choice>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="FormalParameter">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:DataType"/>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Index" type="xsd:NMTOKEN"/>
         <xsd:attribute name="Mode" default="IN">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="IN"/>
                  <xsd:enumeration value="OUT"/>
                  <xsd:enumeration value="INOUT"/>
               </xsd:restriction>
            </xsd:simpleType>
```

```
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="FormalParameters">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:FormalParameter" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Icon" type="xsd:string"/>
   <xsd:element name="Implementation">
      <xsd:complexType>
         <xsd:choice>
            <xsd:element ref="xpdl:No"/>
            <xsd:element ref="xpdl:Tool" maxOccurs="unbounded"/>
            <xsd:element ref="xpdl:SubFlow"/>
         </xsd:choice>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="InitialValue" type="xsd:string"/>
   <xsd:element name="Join">
      <xsd:complexType>
         <xsd:attribute name="Type">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="AND"/>
                  <xsd:enumeration value="XOR"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Length" type="xsd:string"/>
   <xsd:element name="Limit" type="xsd:string"/>
   <xsd:element name="ListType">
      <xsd:complexType>
         <xsd:group ref="xpdl:DataTypes"/>
      </xsd:complexType>
```

```
</xsd:element>
<xsd:element name="Manual">
   <xsd:complexType/>
</xsd:element>
<xsd:element name="Member">
   <xsd:complexType>
      <xsd:group ref="xpdl:DataTypes"/>
   </xsd:complexType>
</xsd:element>
<xsd:element name="No">
   <xsd:complexType/>
</xsd:element>
<xsd:element name="Package">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:PackageHeader"/>
         <xsd:element ref="xpdl:RedefinableHeader" minOccurs="0"/>
         <xsd:element ref="xpdl:ConformanceClass" minOccurs="0"/>
         <xsd:element ref="xpdl:Script" minOccurs="0"/>
         <xsd:element ref="xpdl:ExternalPackages" minOccurs="0"/>
         <xsd:element ref="xpdl:TypeDeclarations" minOccurs="0"/>
         <xsd:element ref="xpdl:Participants" minOccurs="0"/>
         <xsd:element ref="xpdl:Applications" minOccurs="0"/>
         <xsd:element ref="xpdl:DataFields" minOccurs="0"/>
         <xsd:element ref="xpdl:WorkflowProcesses" minOccurs="0"/>
         <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
      <xsd:attribute name="Name" type="xsd:string"/>
   </xsd:complexType>
</xsd:element>
<xsd:element name="PackageHeader">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:XPDLVersion"/>
         <xsd:element ref="xpdl:Vendor"/>
         <xsd:element ref="xpdl:Created"/>
         <xsd:element ref="xpdl:Description" minOccurs="0"/>
         <xsd:element ref="xpdl:Documentation" minOccurs="0"/>
         <xsd:element ref="xpdl:PriorityUnit" minOccurs="0"/>
```

```
            <xsd:element ref="xpdl:CostUnit" minOccurs="0"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Participant">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ParticipantType"/>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
            <xsd:element ref="xpdl:ExternalReference" minOccurs="0"/>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Name" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="ParticipantType">
      <xsd:complexType>
         <xsd:attribute name="Type" use="required">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="RESOURCE_SET"/>
                  <xsd:enumeration value="RESOURCE"/>
                  <xsd:enumeration value="ROLE"/>
                  <xsd:enumeration value="ORGANIZATIONAL_UNIT"/>
                  <xsd:enumeration value="HUMAN"/>
                  <xsd:enumeration value="SYSTEM"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Participants">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Participant" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
```

```
<xsd:element name="Performer" type="xsd:string"/>
<xsd:element name="Priority" type="xsd:string"/>
<xsd:element name="PriorityUnit" type="xsd:string"/>
<xsd:element name="ProcessHeader">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:Created" minOccurs="0"/>
         <xsd:element ref="xpdl:Description" minOccurs="0"/>
         <xsd:element ref="xpdl:Priority" minOccurs="0"/>
         <xsd:element ref="xpdl:Limit" minOccurs="0"/>
         <xsd:element ref="xpdl:ValidFrom" minOccurs="0"/>
         <xsd:element ref="xpdl:ValidTo" minOccurs="0"/>
         <xsd:element ref="xpdl:TimeEstimation" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="DurationUnit">
         <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
               <xsd:enumeration value="Y"/>
               <xsd:enumeration value="M"/>
               <xsd:enumeration value="D"/>
               <xsd:enumeration value="h"/>
               <xsd:enumeration value="m"/>
               <xsd:enumeration value="s"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
   </xsd:complexType>
</xsd:element>
<xsd:element name="RecordType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:Member" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="RedefinableHeader">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="xpdl:Author" minOccurs="0"/>
         <xsd:element ref="xpdl:Version" minOccurs="0"/>
```

```
            <xsd:element ref="xpdl:Codepage" minOccurs="0"/>
            <xsd:element ref="xpdl:Countrykey" minOccurs="0"/>
            <xsd:element ref="xpdl:Responsibles" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="PublicationStatus">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="UNDER_REVISION"/>
                  <xsd:enumeration value="RELEASED"/>
                  <xsd:enumeration value="UNDER_TEST"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Responsible" type="xsd:string"/>
   <xsd:element name="Responsibles">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Responsible" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Route">
      <xsd:complexType/>
   </xsd:element>
   <xsd:element name="SchemaType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Script">
      <xsd:complexType>
         <xsd:attribute name="Type" type="xsd:string" use="required"/>
         <xsd:attribute name="Version" type="xsd:string"
use="optional"/>
         <xsd:attribute name="Grammar" type="xsd:anyURI"
```

```
use="optional"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="SimulationInformation">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Cost"/>
            <xsd:element ref="xpdl:TimeEstimation"/>
         </xsd:sequence>
         <xsd:attribute name="Instantiation">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="ONCE"/>
                  <xsd:enumeration value="MULTIPLE"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Split">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:TransitionRefs" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Type">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="AND"/>
                  <xsd:enumeration value="XOR"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="StartMode">
      <xsd:complexType>
         <xsd:choice>
            <xsd:element ref="xpdl:Automatic"/>
            <xsd:element ref="xpdl:Manual"/>
         </xsd:choice>
```

```
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="SubFlow">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ActualParameters" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:string" use="required"/>
         <xsd:attribute name="Execution">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="ASYNCHR"/>
                  <xsd:enumeration value="SYNCHR"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TimeEstimation">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:WaitingTime" minOccurs="0"/>
            <xsd:element ref="xpdl:WorkingTime" minOccurs="0"/>
            <xsd:element ref="xpdl:Duration" minOccurs="0"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Tool">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ActualParameters" minOccurs="0"/>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Type">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="APPLICATION"/>
                  <xsd:enumeration value="PROCEDURE"/>
```

```
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Transition">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Condition" minOccurs="0"/>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="From" type="xsd:NMTOKEN"
use="required"/>
         <xsd:attribute name="To" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Name" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TransitionRef">
      <xsd:complexType>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TransitionRefs">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:TransitionRef" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TransitionRestriction">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Join" minOccurs="0"/>
            <xsd:element ref="xpdl:Split" minOccurs="0"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TransitionRestrictions">
```

```
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:TransitionRestriction"
minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Transitions">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Transition" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TypeDeclaration">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:group ref="xpdl:DataTypes"/>
            <xsd:element ref="xpdl:Description" minOccurs="0"/>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:ID" use="required"/>
         <xsd:attribute name="Name" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TypeDeclarations">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:TypeDeclaration" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="UnionType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:Member" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
```

```
   <xsd:element name="ValidFrom" type="xsd:string"/>
   <xsd:element name="ValidTo" type="xsd:string"/>
   <xsd:element name="Vendor" type="xsd:string"/>
   <xsd:element name="Version" type="xsd:string"/>
   <xsd:element name="WaitingTime" type="xsd:string"/>
   <xsd:element name="WorkflowProcess">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:ProcessHeader"/>
            <xsd:element ref="xpdl:RedefinableHeader" minOccurs="0"/>
            <xsd:element ref="xpdl:FormalParameters" minOccurs="0"/>
            <xsd:element ref="xpdl:DataFields" minOccurs="0"/>
            <xsd:element ref="xpdl:Participants" minOccurs="0"/>
            <xsd:element ref="xpdl:Applications" minOccurs="0"/>
            <xsd:element ref="xpdl:ActivitySets" minOccurs="0"/>
            <xsd:element ref="xpdl:Activities" minOccurs="0"/>
            <xsd:element ref="xpdl:Transitions" minOccurs="0"/>
            <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
         <xsd:attribute name="Name" type="xsd:string"/>
         <xsd:attribute name="AccessLevel">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="PUBLIC"/>
                  <xsd:enumeration value="PRIVATE"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="WorkflowProcesses">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="xpdl:WorkflowProcess" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="WorkingTime" type="xsd:string"/>
```

```
   <xsd:element name="XPDLVersion" type="xsd:string"/>
   <xsd:element name="Xpression">
      <xsd:complexType mixed="true">
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:any processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
         </xsd:choice>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

B ERANSKINA: ESPORTAZIO ETA INPORTAZIO MODULUEN ESKULIBURUA

Modulu hauen instalaziorako pauso bi eman behar dira. Lehenengo, oinarria instalatu behar da: Zope web aplikazioen zerbitzaria, OpenFlow produktua eta moduluen funtzionamendurako XML tratatzeko tresnak dakartzan paketea. Bigarrenik, moduluak eurak instalatu behar dira eta web bidezko atzipena ahalbidetu.

1. Instalazioa prestatu

Esportatzailearen eta inportatzailearen instalazioa egiteko honako produktu hauek instalatuta egon behar dira sisteman:

1. Zope web aplikazioen zerbitzaria. http://www.zope.org helbidetik deskarga daiteke eta bertan daude instalazio argibideak (2.6.1 bertsioa erabili dut).
2. Zoperako OpenFlow produktua. http://www.openflow.it helbidetik deskarga daiteke eta bertan daude instalazio argibideak (1.2.0 bertsioa erabili dut). OpenFlow produktuak era eraginkorrean lan egin dezan (instantzia asko daudenean adibidez), BTreeFolder2 produktua erabiltzea gomendagarria da. http://hathawaymix.org/Software/BTreeFolder2 helbidetik eskura daiteke.
3. Expat XML parseatzailea. http://sourceforge.net/projects/expat/ helbidetik deskarga daiteke eta bertan daude instalazio argibideak (1.95.7 bertsioa erabili dut). Expat, C lengoaian idatzitako parseatzaile bat da eta PyXML moduluak erabiltzen du XML formatuko fitxategiak parseatzeko. Ez da derrigorrezkoa.
4. PyXML modulua Zoperen instalazio barneko Python-i instalatzeko. http://pyxml.sourceforge.net helbidetik deskargatu daiteke eta bertan daude instalazio argibideak (0.8.3 bertsioa erabili dut).
5. PyXML moduluan aldaketa bi egin behar dira, dokumentatuta baina oraindik konpondu gabe dauden arazo bi saihesteko:
    1. $ZOPEHOME/lib/python2.1/site-packages/_xmlplus/dom/expatbuilder.py

fitxategian, 765. lerroan zera gehitu behar da:

```
if uri is None:
      uri = ""
```

2. $ZOPEHOME/lib/python2.1/site-packages/_xmlplus/dom/minidom.py

   fitxategian, 304. lerroan, _write_data funtzioan dauden hiru lerroak, if egitura baten sartu behar dira. Honela geldituko da:

```
if data:
      data = data.replace("&", "&amp;").replace("<", "&lt;")
      data = data.replace("\"", "&quot;").replace(">", "&gt;")
      writer.write(data)
```

2. Moduluen instalazioa burutu

Moduluen instalazioa ere pauso bitan egiten da. Lehenengo, moduluak dituzten fitxategiak zerbitzarian kokatu behar ditugu. Honek bere arrazoia dauka: garatutako moduluek PyXML paketea erabiltzen dute eta segurtasun arazoengatikvezin dira zuzenean Zope objektuen artean kokatu.

Zopeko objektuen artean sortu daitezkeen Python scriptek ez dute Python liburutegi guztietarako atzipena, hori dela eta bertatik erabili ezin daitezkeen liburutegiak darabiltzaten moduluak aparteko fitxategietan idatzi behar dira eta External Method deritzen objektuen bidez atzitu. Aparteko fitxategi horiek zerbitzarian kokatu behar direnez, hori egiteko baimena duzula suposatzen dute Zope garatzaileek eta horregatik uzten dute horrelako scriptekin liburutegi horiek atzitzen. Zope objektu gisa idatzi daitezkeen scriptak idazteko ez dago zertan zerbitzarirako atzipenik eduki, web interfazearen bidez sortu eta Zoperen objektuen datu-basean gordetzen direlako scriptok.

Hori dela eta moduluak zerbitzarian kokatu ostean External Method izeneko objektuak sortu behar ditugu zerbitzariko fitxategi sisteman kokatu ditugun scriptak atzitzeko. Azken finean, scriptak atzitzen dituzten zubiak bagenitu bezala ulertu behar

ditugu External Method-ak.

Behin horiek sortuta, erabili ahal izateko, HTML formularioak sortuko ditugu erabilpena errazteko. Hala ere hori ez da derrigorrezkoa, External Method-aren URLa eta bertan parametroak jarriz atzi ditzakegulako metodoak.

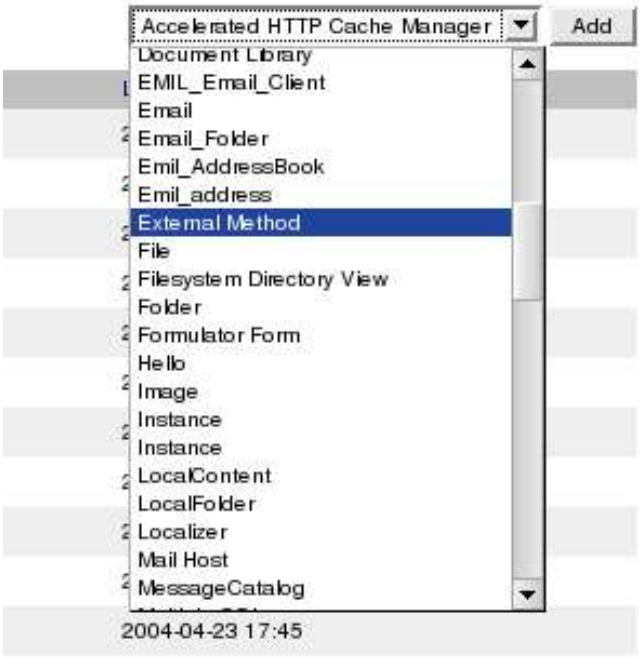
Azken finean lortu beharrekoa zera da: External Method objektuari *name* izeneko parametroan fitxategi baten izena pasatzea eta inportatzailearen kasuan, sortuko den OpwnFlow objektuaren izena *wfname* parametroan. URL bidez zuzenean horrela egin daiteke HTTP protokoloaren GET metodoa erabiliz:

```
http://www.zerbitzaria.org/esportatzailea?name=izena.xpdl
http://www.zerbitzaria.org/inportatzailea?name=izena.xpdl&wfname=Adib
```

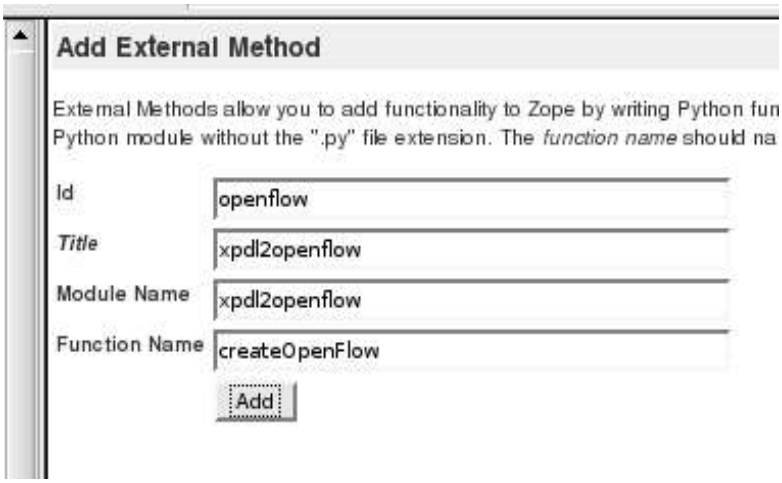
1. Moduluak (*xpdl2openflow.py* eta *openflow2xpdl.py* fitxategiak) $ZOPEHOME/Extensions katalogora kopiatu.
2. Zoperen interfazean sartu eta ExternalMethod deritzon objektu bat sortu modulu bakoitzeko, moduluaren izena (*openflow2xpdl* edo *xpdl2openflow*) jarriz eta bakoitzean deitzen zaion metodoaren izena jarriz (*sortuXPDL* edo *createOpenFlow*). Ikusi Irudia 34 eta Irudia 35.
3. Zoperen interfazetik DTML Method edo PageTemplate erako objektu bat sortu eta HTML formulario bat idatzi testu kutxa batekin. Testu kutxaren izena (*name* atributua) *"name"* izan behar da, dagokion funtzioari deitzean izen horretako parametroan pasatu behar baitzaio fitxategiaren izena. Formularioaren *action* atributuan, sortutako External Method-aren izena jarri behar da (*xpdl* esportatzailean eta *openflow* inportatzailean, kasu konkretu honetan). Inportatzailearen kasuan, gainera, beste testu kutxa bat egon beharko da sortuko den OpenFlow objektuaren izena idazteko. Testu kutxa horren izena *wfname* izan behar da.

   Oharra: Memoriarekin batera banatu den CDan dagoen *Dokumentazioa.zexp* fitxategia erabiliz 2 eta 3 puntuetako External Method objektuak eta HTML formularioak automatikoki inportatu daitezke. Fitxategi hori

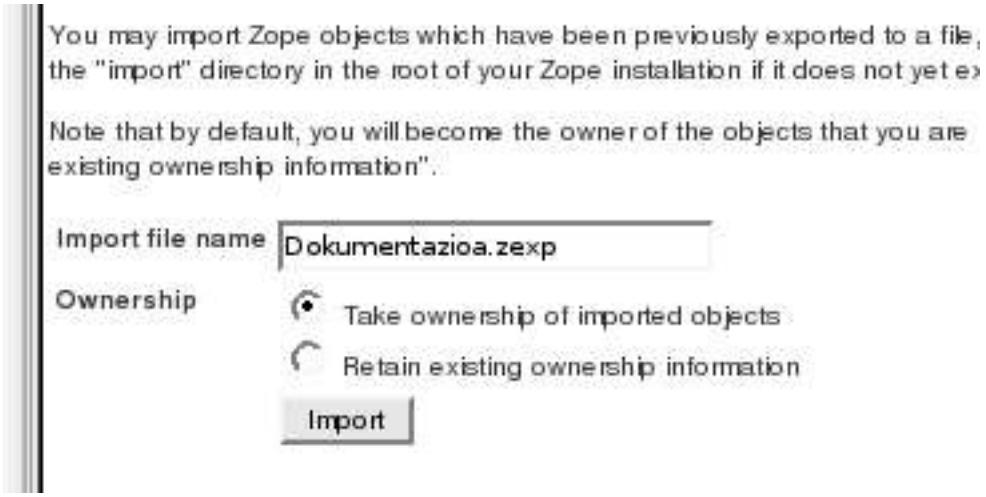
$ZOPEHOME/import katalogoan kokatu behar da eta ondoren Zoperen interfaze nagusian *Import/Export* botoia erabiliz inportatu. Horrela *Dokumentazioa* izeneko karpeta bat sortuko da eta barruan dago lan-fluxu baten adibidea OpenFlow objektu batean, XPDL fitxategi baten adibidea, External Method-ak eta HTML formularioak (Irudia 36).



Irudia 34: External Method objektua sortu



Irudia 35: External Method-en inportatzailearen informazioa sartu



Irudia 36: Fitxategia Zope-ra inportatzen

4. Moduluak funtzionatzen ikusteko, sortutako HTML formularioak erabili behar dira inportatu nahi den fitxategiaren izena jarriz edo esportatzean sortuko den fitxategiaren izena adieraziz.
5. Esportazioa eta inportazioa egin ostean, sortutako XPDL fitxategiak edo OpenFlow objektuak, Zoperen interfaze nagusitik eskuratu behar dira. Hala ere, moduluek, egindako lanaren emaitzak edo gertatu diren erroreak erakusten dituzte

formularioaren erantzun gisa sortzen den orrialdean.

C ERANSKINA: KONPILADOREAREN INSTALAZIOA

Modulu hauek instalatzeko OpenFlow produktuaren iturburu kodean aldaketa batzuk egin behar dira eta Zope instalazioko Python-i *kjbuckets* modulua gehitu behar zaio konpiladoreak funtziona dezan. Aldaketak egiteko baimena OpenFlow produktuaren lizentziak berak ematen du, GNU elkargoaren GPL lizentziapean banatzen baita produktua.

1. Instalazioa

1. Zope eta OpenFlow produktuak instalatu B eranskinean aipatzen den moduan.
2. Zope instalazioko Python-i ($ZOPEHOME/bin/python) *kjbuckets* modulua instalatu, konpiladoreak grafoak erabiltzen dituenez, datu-mota hori inplementatzen duen modulua erabili ahal izateko. Modulu hori *gadfly* paketearen barruan dator eta http://sourceforge.net/projects/gadfly/ helbidetik deskargatu daiteke, instalazio argibideekin batera.
3. Modulua osatzen duten fitxategiak dituen trinkotutako fitxategia, OpenFlow produktua dagoen karpetan destrinkotu. Honako fitxategi hauek sortuko dira:
   - *compilation.py* fitxategia OpenFlowren erro katalogoan
   - Konpiladorea eta lan-fluxuen inguruko informazioa ematen dituzten fitxak (*Info.zpt* eta *Compilation.zpt* fitxategiak) *zpt/openflow* eta *zpt/process* katalogoetan.
   - Zoperen interfazetik ikusgai izango den laguntza, help katalogo barruko *Info.stx*, *Compilation.stx*, *and-xor.png* eta *and-xor2.png* fitxategiak.
4. Moduluak erabili ahal izateko honako aldaketa hauek egin OpenFlowren fitxategietan:
   - *openflow.py* fitxategian:

     175. lerroan, produktuaren kudeaketa fitxetatik deitu beharreko fitxategiak zein diren definitzen duenean, ondorengo lerroak gehitu:

```
security.declareProtected('Manage OpenFlow', 'Info')
Info = PageTemplateFile('zpt/openflow/Info', globals())

security.declareProtected('Manage OpenFlow', 'Compilation')
Compilation = PageTemplateFile('zpt/openflow/Compilation', globals())
```

211. lerrotik aurrera, fitxak definitzen dituenean zera gehitu:

```
{'label':'Info',
'action':'Info',
'help':('OpenFlow', 'info.stx')},

{'label':'Compilation',
'action':'Compilation',
'help': ('OpenFlow', 'compilation.stx')}
```

- *process.py* fitxategian:

  73. lerroan, kudeaketa fitxetatik deitu beharreko orriak zein diren definitzen duenean:

```
security.declareProtected('Manage OpenFlow', 'Info')
Info = PageTemplateFile('zpt/process/Info', globals())

security.declareProtected('Manage OpenFlow', 'Compilation')
Compilation = PageTemplateFile('zpt/process/Compilation', globals())
```

87. lerrotik aurrera, fitxak definitzean:

```
{'label':'Info',
'action':'Info',
'help':('OpenFlow', 'info.stx')},

{'label':'Compilation',
'action':'Compilation',
'help': ('OpenFlow', 'compilation.stx')}
```

Aldaketa hauekin, OpenFlow produktuari fitxa berriak gehitzen dizkiogu eta fitxa bakoitza zein fitxategitan dagoen adierazten diogu. Horretarako lehenengo fitxategiak zein diren adierazten diogu (bai *openflow.py* eta bai *process.py* fitxategiko lehenengo

aldaketarekin) eta ondoren fitxa bera definitzen dugu, bere izena, erabiliko duen fitxategia (lehenengo pausuan definitutakoa) eta laguntza fitxategia zen diren esanez.

Horretaz gain *process.py* izeneko fitxategian funtzio bat gehitu behar dugu. Funtzio hau izango da prozesuari dagokion informazioa erakustean kudeaketa interfazeak deituko duena:

```
def getProcessInfo(self, process_id):
      """ analyzes process looking for unproperly configured
          'and' and 'xor' guards """
      comp = Compilation(self, process_id)
      comp.createGraph()
      comp.compile()
      if comp.getResult():
            ret = ''
            for fr,to in comp.getResult().items():
                ret += '<p>Problem from: %s to %s</p>' % (fr,to)
            return ret
      else:
            return '<p>No problem!</p>'
```

Gainera, process.py fitxategiaren lehenengo lerroetan, beharrezko moduluak inportatzen dituen lekuan, lerro hau gehitu behar da:

```
from compilation import *
```

2. Erabilpena

Moduluaren funtzionamendua ikusteko OpenFlow produktuaren *Info* eta *Compilation* kudeaketa fitxetan klik egin behar da. Bertan agertuko da OpenFlown inplementatuta dauden prozesuen informazioa (*Info* fitxan klik eginez gero) eta konpilazioaren emaitza (*Compilation*) fitxan klik eginez gero.

Prozesu zehatz baten kudeaketan ere Info eta *Compilation* fitxak daude, hauen bidez prozesu zehatz horren inguruko informazioa eta konpilazioaren emaitza ikusi daitezke.

Fitxen inguko laguntza, goialdean ezkerrean agertzen den *Help!* loturan klik eginda lortzen da, bai OpenFlowren fitxetatik edo prozesuen fitxetatik.

D ERANSKINA: OPENFLOWren APIa

Jarraian datozen tauletan OpenFlow produktuaren klase ezberdinen APIa ikus daiteke.

1. OpenFlow klasea

openflow Products.OpenFlow.openflow A openflow folder contains all the processes of the openflow			
Funtzioa	Zope baimena	Parametroak	Azalpena
Applications	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
Instances	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
Processes	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
Roles	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
Updates	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
__ac_permissions__	('Manager',)		tuple
__init__	('Manager',)	self, id	function
activateWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, actor=None, REQUEST= None	declares the activation of the specified workitem of the given instance
addApplication	-- explicit -- ('Manager',)	self, name, link, REQUEST= None	adds an application declaration

addInstance	-- explicit -- ('Manager',)	self, process_id, customer='', comments='', title='', activation=0, priority=0, REQUEST= None	adds an a process definition instance
addProcess	-- explicit -- ('Manager',)	self, id, title='', description='', BeginEnd=No ne, priority=0, REQUEST= None	adds a new process
addRoleWithActivit iesPullable	-- explicit -- ('Manager',)	self, role, process	Add a role
addRoleWithActivit iesPushable	-- explicit -- ('Manager',)	self, role, process	Add a role
assignWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, actor, REQUEST= None	Assign the specified workitem of the specified instance to the specified actor (string)
callAutoPush	-- explicit -- private	self, instance_id, workitem_id, REQUEST= None	function

changeWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, push_roles=None, pull_roles=None, event_log=None, process_id=None, activity_id=None, blocked=None, priority=None, workitems_from=None, workitems_to=None, status=None, actor=None, graph_level=None, REQUEST=None	use this API to modify anything of a fallout workitem usable only if workitem status is `fallout`
chooseFallin	-- explicit -- ('Manager',)		HTMLFile with bindings and support for __render_with_namespace__
completeSubflow	('Manager',)	self, instance_id, workitem_id	function
completeWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, REQUEST=None	declares the completion of the specified workitem of the given instance

deleteApplication	-- explicit -- ('Manager',)	self, app_ids=None, REQUEST=None	removes an application
deleteInstance	-- explicit -- ('Manager',)	self, inst_ids=None, REQUEST=None	removes specified instance
deleteProcess	-- explicit -- ('Manager',)	self, proc_ids=None, REQUEST=None	removes specified process
deleteProcessWithActivitiesPullableOnRole	-- explicit -- ('Manager',)	self, role, process	Delete the link between a role and activities of a process
deleteProcessWithActivitiesPushableOnRole	-- explicit -- ('Manager',)	self, role, process	Delete the link between a role and activities of a process
deleteRoleWithActivitiesPullable	-- explicit -- ('Manager',)	self, role	Delete a role
deleteRoleWithActivitiesPushable	-- explicit -- ('Manager',)	self, role	Delete a role
editActivitiesPullableOnRole	-- explicit -- ('Manager',)	self, role, process, activities=None, REQUEST=None	Edit the link between a role and activities of a process
editActivitiesPushableOnRole	-- explicit -- ('Manager',)	self, role, process, activities=None, REQUEST=None	Edit the link between a role and activities of a process

editApplication	-- explicit -- ('Manager',)	self, name, link, REQUEST= None	edits an application declaration
endFallinWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, REQUEST= None	ends the exceptional state of the given workitem (of the specified instance)
fallinWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, process_id, activity_id, REQUEST= None, coming_from =None	the exceptional specified workitem (of the specified instance) will be put back in the activity specified by process_id and activity_id; workitem will still be in exceptional state: use endFallinWorkitem API to specify the end of the exceptional state
falloutWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, REQUEST= None	drops the workitem (of the specified instance) in exceptional handling
fixCreateProcessInstancesContainer	('Manager',)	self	this is a fix moving OF 1.0.6 to OF 1.1
fixMoveInstancesInTheirContainer	('Manager',)	self	this is a fix moving OF 1.0.6 to OF 1.1
forwardWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, path=None, REQUEST= None	instructs openflow to forward the specified workitem
getActivitiesPullableOnRole	-- explicit -- ('Manager',)	self	function
getActivitiesPushableOnRole	-- explicit -- ('Manager',)	self	function

getApplicationUrl	-- explicit -- ('Manager',)	self, instance_id, workitem_id	Return application definition URL relative to instance and workitem
getDestinations	('Manager',)	self, instance_id, workitem_id, path=None	function
getEnvironment	('Manager',)	self, instance_id, workitem_id	function
getInstance	-- explicit -- ('Manager',)	self, instance_id	returns a given instance
getNeededFixes	('Manager',)	self	finds the fixes the current openflow needs
getNextTransitions	-- explicit -- ('Manager',)	self, instance_id, workitem_id	Returns the list of transition that the given workitem (of the specified instance) will be routed on
getPIContainer	-- explicit -- ('Manager',)	self	returns the instance container
getPullRoles	('Manager',)	self, process_id, activity_id	function
getPushRoles	('Manager',)	self, process_id, activity_id	function
getSubflowWorkitem	('Manager',)	self, instance_id, workitem_id	function
icon	('Manager',)		string
inactivateWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, REQUEST=None	declares the inactivation of the specified workitem of the given instance
index_html	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL

isEnd	('Manager',)	self, process_id, activity_id	function
linkWorkitems	('Manager',)	self, instance_id, workitem_from_id, workitem_to_id_list	function
listApplications	-- explicit -- ('Manager',)	self	List application declaration; returns a list of dictionaries with keys: name, link
manageDummyActivity	-- explicit -- ('Manager',)	self, instance_id, workitem_id	function
manageWorkitemCreation	('Manager',)	self, instance_id, workitem_id	function
manage_addApplicationForm	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
manage_addInstanceForm	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
manage_addProcessForm	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
manage_afterAdd	-- explicit -- private	self, item, container	function
manage_editActivitiesPullableOnRole	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
manage_editActivitiesPushableOnRole	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
manage_editApplicationForm	-- explicit -- ('Manager',)		HTMLFile with bindings and support for __render_with_namespace__
manage_options	('Manager',)		tuple

manage_pushWorkitem	-- explicit -- ('Manager',)		HTMLFile with bindings and support for __render_with_namespace__
meta_type	('Manager',)		string
meta_types	('Manager',)		tuple
resumeInstance	('Manager',)	self, instance_id=None, REQUEST=None	suspend a specified instance
resumeWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, REQUEST=None	declares the resumption of the specified workitem of the given instance
runFixes	('Manager',)	self, fixes=None, REQUEST=None	runs the selected fixes
sendWorkitemsToException	('Manager',)	self, process_id, activity_id	function
startAutomaticApplication	-- explicit -- ('Manager',)	self, instance_id, workitem_id	function
startInstance	-- explicit -- ('Manager',)	self, instance_id, REQUEST=None	Starts the flowing of the process instance inside the process definition
startSubflow	-- explicit -- private	self, instance_id, workitem_id, REQUEST=None	function
suspendInstance	-- explicit -- ('Manager',)	self, instance_id=None, REQUEST=None	suspend a specified instance

suspendWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, REQUEST= None	declares the suspension of the specified workitem of the given instance
terminateInstance	-- explicit -- ('Manager',)	self, instance_id= None, REQUEST= None	terminate a specified instance
unassignWorkitem	-- explicit -- ('Manager',)	self, instance_id, workitem_id, REQUEST= None	Unassign the specified workitem
usersAssignableTo	-- explicit -- ('Manager',)	self, process_id, activity_id	List all user name assignable to activity in the process

Taula 3: OpenFlow klasearen APIa

2. Process klasea

process Products.OpenFlow.process A process is a collection of activities and transitions. The process map is given by the linking of activities by transitions. Each process instance is described by a instance			
Funtzioa	Zope baimena	Parametroak	Azalpena

Setting	-- explicit -- ('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
__ac_permissions__	('Manager', 'Anonymous')		tuple
__init__	('Manager', 'Anonymous')	self, id, title='', description='', BeginEnd=None, priority=0	function
addActivity	-- explicit -- ('Manager',)	self, id, split_mode='and', join_mode='and', self_assignable=1, start_mode=0, finish_mode=0, subflow='', push_application='', application='', title='', parameters='', description='', kind='standard', REQUEST=None	adds the activity and eventually sets the process begin and end activity
addTransition	-- explicit -- ('Manager',)	self, id, From, To, condition=None, REQUEST=None	adds a transition
edit	-- explicit -- ('Manager',)	self, begin=None, end=None, title=None, description=None, priority=None, REQUEST=None	changes the process settings
icon	('Manager', 'Anonymous')		string
index_html	('Manager',)		HTMLFile with bindings and support for __render_with_namespace__

manage_addActivityForm	-- explicit -- ('Manager',)		HTMLFile with bindings and support for __render_with_namespace__
manage_addTransitionForm	-- explicit -- ('Manager',)		HTMLFile with bindings and support for __render_with_namespace__
manage_delObjects	-- explicit -- ('Manager',)	self, ids=[], REQUEST=None	override default method to handle better the redirection
manage_options	('Manager', 'Anonymous')		tuple
meta_type	('Manager', 'Anonymous')		string
meta_types	('Manager', 'Anonymous')		tuple

Taula 4: Process klasearen APIa

3. Instance klasea

instance Products.OpenFlow.instance Even though it is called instance it is more than that: it is the collection of all instances related to a given process instance.			
Funtzioa	Zope baimena	Parametroak	Azalpena
__ac_permissions__	('Manager', 'Anonymous')		tuple

__init__	('Manager', 'Anonymous')	self, process_id, activity_id, customer, title='', comments='', priority=0	function
addWorkitem	('Manager', 'Anonymous')	self, process_id, activity_id, blocked, push_roles=[], pull_roles=[]	function
getActiveWorkitems	-- explicit -- ('Manager',)	self	returns all active workitems
getJoiningWorkitem	('Manager', 'Anonymous')	self, process_id, activity_id	function
icon	('Manager', 'Anonymous')		string
index_html	('Manager',)		Zope wrapper for filesystem Page Template using TAL, TALES, and METAL
isActiveOrRunning	('Manager', 'Anonymous')	self	function
manage_options	('Manager', 'Anonymous')		tuple
meta_type	('Manager', 'Anonymous')		string
meta_types	('Manager', 'Anonymous')		tuple
setPriority	('Manager', 'Anonymous')	self, value	function
setStatus	('Manager', 'Anonymous')	self, status, comment='', actor=''	function

Taula 5: Instance klasearen APIa

4. Activity klasea

activity Products.OpenFlow.activity Each activity is responsible for doing something and then forwarding the instance			
Funtzioa	Zope baimena	Parametroak	Azalpena
__ac_permissions__	public		tuple
__init__	public	self, id, split_mode='and', join_mode='and', self_assignable=1, start_mode=0, finish_mode=1, subflow='', push_application='', application='', parameters='{}', title='', description='', kind='standard'	function
edit	('Manager',)	self, split_mode=None, join_mode=None, self_assignable=None, start_mode=None, finish_mode=None, subflow=None, push_application=None, application=None, title=None, description=None, kind=None, REQUEST=None	changes the activity settings
getIncomingTransitionsNumber	public	self	returns all the process transition objects that go to the specified activity
index_html	('Manager',)		HTMLFile with bindings and support for __render_with_namespace__

isAutoFinish	-- explicit -- ('Manager',)	self	returns true if the activity finish mode is automatic
isAutoPush	-- explicit -- ('Manager',)	self	returns true if the activity push mode is automatic
isAutoStart	-- explicit -- ('Manager',)	self	returns true if the activity start mode is automatic
isDummy	-- explicit -- ('Manager',)	self	returns true if the activity is a dummy
isSelfAssignable	-- explicit -- ('Manager',)	self	returns true if the activity is assignable to self
isStandard	-- explicit -- ('Manager',)	self	returns true if the activity is of `standard` kind
isSubflow	-- explicit -- ('Manager',)	self	returns true if the activity is a subflow
meta_type	public		string
title_or_id	-- explicit -- ('Manager',)	self	function

Taula 6: Activity klasearen APIa

5. Transition klasea

transition Products.OpenFlow.transition Links two activities			
Funtzioa	Zope baimena	Parametroak	Azalpena
__ac_permissions__	public		tuple
__init__	public	self, id, From, To, condition='', description=''	function
edit	('Manager',)	self, condition, From, To, description, REQUEST=None	function

manage_editTransitionForm	-- explicit -- ('Manager',)		HTMLFile with bindings and support for __render_with_namespace__
meta_type	public		string

Taula 7: Transition klasearen APIa

6. Workitem klasea

workitem Products.OpenFlow.workitem describes a single workitem of the history graph			
Funtzioa	Zope baimena	Parametroak	Azalpena
__ac_permissions__	public		tuple
__init__	public	self, id, instance_id, process_id, activity_id, blocked, priority=0, workitems_from=[], workitems_to=[], push_roles=[], pull_roles=[]	function
addEvent	public	self, event, comment=''	function
addFrom	public	self, id	function
addTo	public	self, id_list	function
assignTo	public	self, actor, by=None, comment=''	function

edit	-- explicit -- ('Manager',)	self, instance_id=None, process_id=None, activity_id=None, blocked=None, priority=None, workitems_from=None, workitems_to=None, status=None, actor=None, graph_level=None	function
endFallin	public	self	function
getEventLog	public	self	function
isActiveOrInactiveOn	public	self, process_id, activity_id	function
meta_type	public		string
setArrivalTime	public	self, activity_id, comment	function
setGraphLevel	public	self, graph_level	function
setStatus	public	self, status, comment='', actor=''	function
unblock	public	self, value=1, comment='', actor=''	function

Taula 8: Workitem klasearen APIa

E ERANSKINA: AZTERTUTAKO LAN-FLUXU SISTEMAK

Eranskin honen helburu nagusia, webean aurkitutako lan-fluxu kudeaketa sistema ezberdinen aurkezpen bat egitea da, era laburbildu batean bakoitzaren ezaugarri nagusiak azalduz.

1. ProFlow

[Integrated Workflow] enpresaren produktua da ProFlow eta dokumentuen kudeaketaren inguruko lan-fluxuak inplementatzea du helburu.

Produktuaren helburu nagusiena enpresak bere ohiko lan-tresnekin lanean jarraitu ahal izatea da, sistema berri batera migrazioa prestatu ordez. Horretarako Microsoft etxearen mahaigaineko softwarean (Microsoft Office produktuarekin, alegia) integratzen da, bertan sortutako dokumentu bat botoi bat sakatuz lan-fluxuan integratzeko, beste software bat ikasten ibili beharrean.

Integrazio honetaz gain, web bidezko sistema ere eskaintzen du Proflowk eta web zerbitzuetan oinarritutako inplementazioa duenez, programatzaileek, bere APIa erabiliz, lan-fluxu sistemaren funtzionalitateak beste produktu batzuetan integratu ditzakete

Microsoft etxearen .NET teknologiarekin garatutako produktua da eta Windows plataforman funtzionatzen du soilik, bezero zein zerbitzariak Windows XP edo 2000 sistema eragileetan instalatu behar dira eta produktuak darabilen datu-base kudeaketa sistema SQL Server da. Web bidezko atzipenerako ere Microsoft etxearen Internet Explorer web arakatzailea erabili behar da.

Enpresa txikientzat ostatze sistema bat eskaintzen dute, enpresak zerbitzaria instalatu behar ez dezan eta zerbitzarirako atzipena Internet bidez egitea ahalbidetzen dute.

Eskaintzen dituen funtzionalitateen artean, nabarmentzekoak dira lan-fluxua editatzeko modu grafikoa daukala, posta elektroniko bidezko jakinarazpenak bidaltzeko ahalmena duela, segurtasun kopia sistema integratzen duela eta lan-fluxuaren inguruko txostenak Microsoft Excel formatuan ateratzen dituela. Gainera lan-fluxua bera XML formatuan esportatu dezake, produktuaren dokumentazioan txantiloia deitzen zaio, eta

espezifikazio berdinak jarraituz idatzitako beste txantiloi bat inportatu daiteke sistemara. Txantiloia era erraz batean alda daiteke kode gehiegi idatzi gabe, esate baterako web edo Visual Basic aplikazioen bidez, egin behar dena ProFlowren azpian dagoen web-zerbitzua atzitzea da.

Webgunean produktuaren erakustaldi bat daukate eta gainera salmenta osteko laguntza eta arazo konponketarako zerbitzua eskaintzen dute.

Produktuaren prezioa, urte eta pertsonako, 180 € ingurukoa da.

2. Agentflow

Java programazio lengoaian idatzitako produktu hau Taiwango [Flowring] enpresak garatutakoa da. Javaz idatzita dagoenez, sistema eragile eta datu base desberdinekin funtzionatzen du eta gainera [WfMC] erakundearen espezifikazio eta gomendioak jarraitzen ditu honek definitzen dituen interfaze guztiekin bateragarri izateko.

Bai [WfMC] eta baita [BPMI] erakundearen kide da produktu hau garatu duen enpresa, eta lan-fluxuen estandarizazioaren bidean lanean ari da bi erakunde hauekin.

Garatzailearetzat ingurune grafikoa eskaintzen du produktuak, eta baita aplikazioen garapenerako formularioak sortzeko diseinatzailea, sistemaren ahalmenak zabaltzeko interfaze ahaltsu batez gain. Guztia J2EE, Java, JDBC eta XML teknologiak erabiliz garatuta.

Lan-fluxu sistemaren motoreak, uneko exekuzioak monitorizatzeko tresna dauka eta aurreko exekuzioen historia fitxategiak kudetzeko aukera ere eskaintzen du. Horretaz gain, erabiltzaileentzako lan-iterfazeak ditu, era erraz batean lan egin dezaten, gainontzeko partehartzaileei mezuak bidali, beren menpeko exekuzioak monitorizatu edo lan-abisuak hainbat modutan bidali ahal izateko. Web bidezko interfazea ere eskaintzen du, langileek edonondik lan egin ahal dezaten.

Webgunearen bidez ez dute eskaintzen produktuaren erakustaldirik eta ordainpeko sistema da, preziorik esaten ez duten arren. Produktuaren demo bat lortu ahal izateko eurekin harremanetan jarri arren, ez dute erantzunik eman.

3. OpenFlow

OpenFlow, [Icube] enpresa italiarraren produktua da eta Zope plataformarako garatuta dago. Ekintzetan oinarritutako lan-fluxu sistema inplementatzen du Python lengoaian garatutako produktu honek. Zoperako produktua izanik, hainbat plataformatan funtzionatzen du, hala nola, Windows, Linux edo MacOS sistema eragileetan.

Zope plataformarako garatutako sistema denez web bidezkoa da, kudeaketa zein prozesu berrien definizioa eta lana bera web bidez egiten da. Prozesuaren definizio eta rolen esleipena ere web bidez egiten dira.

Sistemaren malgutasuna eta egoera berezien kudeaketa erraztu behar dela diote Openflowren egileek, horretarako lan-fluxuaren egoera bereziak kudeatzeko sistema bat eskaintzen du OpenFlowk. Prozesuak lan-fluxuaren exekuzio garaian aldatzeko aukera ere ematen du, sortutako informazioa galdu gabe.

Gainera produktua GPL lizentziapean banatzen da eta garatzaileek behar dituzten aldaketak egiteko bere iturburu koderako atzipena hasieratik daukate. Bestalde, ez dauka lan-fluxuen adierazpen grafikoa egiteko tresnarik integratuta, beren webgunean eskaintzen duten produktu gehigarri bat erabili behar da horretarako. OpenFlowk ez dauka erabiltzaileei egiteke dituzten lanen berri abisuak bidaltzeko sistemarik, hala ere, Zope plataformako beste produktu batzuk erabiliz lor daitezke funtzionalitate horiek.

Produktua dohainekoa da, eta alderdi negatibo gisa esan daiteke dagoen dokumentazio apurra italiera eta ingelesez dagoela eta OpenFlowren sortzaileek ez dutela salmenta ondoko zerbitzurik eskaintzen. Hala ere, posta-zerrenda bat daukate antolatuta OpenFlow darabilen jendearekin inplementazio arazoak edo produktuak izan beharko lituzkeen beste ezaugarri batzuen inguruan eztabaidatzeko.

4. PAFlow

[PAFlow] ez da lan-fluxu kudeaketa sistema bat, erabiltzaile zehatz bati garatutako tresna bat baizik. Produktuaren garapena OpenFlow sortu duen Icube enpresa italiarrarena da eta bezeroa Italiako Administrazio Publikoaren Informatikarako

Erakundea [CNIPA] da.

Produktuak OpenFlow darabil lan-fluxu sistema gisa eta PostgreSQL datu-base kudeaketa sistema gisa. Produktua garatzaileen webgunetik deskargatu daite, Zoperi instalatu behar zaion produktu gehigarrien zerrendarekin batera. Instalazio lana astuna da eta produktua beraren funtzionalitateak AIPA erakundearen estandarretara mugatzen dira. Hala ere, dokumentazio guztia italieraz dago eta zaila da erakundearen estandarrei buruz ezer jakin gabe produktua erabiltzen ikastea.

Produktuaren garapenarekin batera, *Paflow Janitors* deitu duten proiektu bat jarri dute martxan. Bere helburua ingelesezko dokumentazioa sortu eta produktuan erabilitako kodea ingelesera itzultzea da, ondoren produktua internazionalizatu (*i18n*[7]) eta lokalizatu (*l10n*[8]) ahal izateko.

5. Workflow IQ

[Feith] enpresak garatutako produktu honek, lan-fluxu sistemarako web bidezko atzipena eskaintzen du, bezeroan instalatu daitekeen softwareaz gain.

Programazio lanik behar izan gabe, lan-fluxuak modu grafikoan sortu eta kudeatzeko aukera ematea da bere helburua, erabiltzailearen lana erraztu asmoz. Bide horretatik, egiteke dauden lanen inguruko abisuak, egindako lanaren kontabilitatea edo uneko lan-fluxuaren exekuzioen monitorizazioa ezer programatu behar izan gabe egin daitezke. Horretaz gain, datu-base kudeaketa sistemekin konektatzeko aukera dauka, lan-fluxuaren inguruko datuak bertan gordetzeko.

Egin beharreko lanak automatikoki esleitzen ditu langileak oporretan edo kanpoan baldin badaude eta lan fluxua monitorizatzen du gauza bakoitza zein egoeratan dagoen ikusteko. Horrela nahi izanez gero, dokumentuak behar diren lekuetara bidaltzea automatikoki egiten du pertsonen eskuhartzerik gabe. Gehien bat dokumentuen kudeaketarako sortutako produktua da baina ez da hori bere helburu bakarra.

7 Ingelesezko *internationalization* terminoaren laburdura, hasierako i eta bukaerano n-ren artean 18 karaktere daudela adieraziz.

8 Ingelesezko *localization* terminoaren laburdura, hasierako l eta bukaerako n-ren artean 10 karaktere daudela adieraziz.

Webgunean ez dute informazio gehiegi eskaintzen ezta bere prezioa ere.

6. DCWorkflow

Zope plataformarako garatutako produktua da hau ere, baina aurretik azaldutako OpenFlowrekin desberdintasun nagusi bat dauka: hura ekintzetan oinarritutakoa bada, hau egoeretan oinarritutako da eta *CMF* edo *Content Management Framework* delakorako lan-fluxua da, eta tresna horrekin egindako atarietako dokumentuen lan-fluxuak kudeatzeko erabiltzen da.

7. SPRINT

Kalitate sistema baten kudeaketa eta kontrolera zuzendutako produktu bat da Lotus Notes eta Lotus Domino software tresnen gainean oinarrituta.

Edizio grafikoa dauka, e-mail bidezko abisuak, langile bakoitzak lanak kontsulta ditzaketa, erabiltzaile eta rolen kudeaketa, intranet/internet bidezko argitarapena, ...

Hiru modulu nagusitan banatzen da: lan fluxuen edizio grafikoa, lan fluxuen jarraipena eta kudeaketa, dokumentuen kudeaketa aurreratua. Hauetaz gain, enpresa edo behar zehatzago bakoitzerako modulu berriak txerta dakizkioke.

Robotiker enpresaren webgunean ([Robotiker]) dago produktuaren informazioa, baina ez du prezioaz ezer esaten eta ez dago produktua probatzeko aukerarik.

8. WFTK

Produktu hau probatzeko aukerarik ez dut izan exekuzio errore bat ematen duelako programa martxan jarri bezain laster. C programazio lengoaiaz idatzitako programa denez, plataforma ezberdinetan funtzionatzen du, gainera bere informazio guztia XML formatua erabiliz gordetzen duenez, informazio trukerako aukera polita eskaintzen du.

Momentuz modulu txiki bat da eta beste modulu batzuekin integratu behar da. Oraindik garapenean dago eta etorkizunerako mugarri gisa, Zope plataformarekin integratzeko asmoa daukate. Produktuari buruzko dokumentazioa badagoen arren, berau

bertsio zaharren ingurukoa da eta ez dirudi proiektuak azken aldian mugimendurik izan duenik

9. Beste batzuk

Aipatutako lan-fluxu kudeaketa sistemetaz gain, beste hainbat aurki daiteke Interneten zehar. Hauetako asko, azken aldian indartsu dabilen web-zerbitzuen teknologia erabiliz garatutakoak dira, bai Java programazio lengoaia erabiliz eta baita Microsoftek garatutako .NET teknologiarekin ere.

Enpresa eta erakunde askotan, neurrira sortutako aplikazioak erabiltzen dira, inplementatu behar dituzten lan-fluxuak legez estandarizatuta daudelako. Hau da, adibidez, udalen kasua. Gai bakoitzeko espedienteen kudeaketan legez ezarritako pauso eta epe batzuk bete behar baitituzte. Horren haritik, Gipuzkoako udaletxe askotan, Gipuzkoako Foru Aldundiak neurrira sortutako tresna bat darabilte.

Bestalde gaur egun, ikastetxe askotan erabiltzen dira Lotus Notes eta Lotus Domino tresnak. Hauen bidez, talde-lanerako tresna ahaltsua osatu daiteke, bai dokumentuen kudeaketarako. bai beste era bateko lan-fluxuetarako. Tresnan bertan integratzen den datu-basearekin elkarlanean erabil daiteke, lan-fluxuaren une bakoitzean sortzen diren datuak sartu eta kontsultatzeko.

10. Laburpen taula

Aurreko ataletan azaldu ditudan lan-fluxu sistema ezberdinen ezaugarri nagusiak biltzen dituen taula bat aurkezten dut jarraian.

	Kudeaketa grafikoa	***Software librea***	***WfMCren estandarrak jarraitu***	***Teknologia***	***Bestelakoak***
ProFlow	Bai	Ez	Ez	.NET	Dokumentuen kudeaketarako prestatuta
Agentflow	Bai	Ez	Bai	Java	WfMCren estandarizazio lanetan parte hartzen du enpresak
OpenFlow	Tresna gehigarri batekin	Bai	Bai	Zope eta Python	Dokumentazio gutxi
PAFlow	Bai	Bai	Bai	Zope eta Python	OpenFlown oinarritutako produktua
WorkflowIQ	Bai	Ez	Ez	?	Programazio lana murriztu nahi du
DCWorkflow	Ez	Bai	Ez	Zope	Egoeretan oinarritutako lan-fluxu sistema
SPRINT	Bai	Ez	?	Lotus Notes eta Domino	Kalitate sistemen kudeaketarako prestatuta
WFTK	Ez	Bai	?	C	Proiektua geldirik dago

Taula 9: Lan-fluxu sistemen ezaugarrien laburpena