UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Sistema de monitoratge autoadaptable heterogeni i distribuït

Autor:
Joaquim MOTGER DE LA ENCARNACIÓN

Director:
Xavier FRANCH GUTIERREZ

Codirector:
Marc ORIOL HILARI

Treball de final de grau presentat sota el marc del

Grau d'Enginyeria Informàtica

en l'especialitat de

Enginyeria del Software

Universitat Politècnica de Catalunya

Abstracte

Facultat d'Informàtica de Barcelona
Enginyeria de Serveis i Sistemes de la Informació

Grau d'Enginyeria Informàtica

Sistema de monitoratge autoadaptable heterogeni i distribuït

per Joaquim MOTGER DE LA ENCARNACIÓN

El monitoratge consisteix en la tècnica d'observació i control dels sistemes software amb l'objectiu de garantir la seva fiabilitat, la qualitat del servei (Quality of Service, QoS), la seguretat i altres característiques dels sistemes software pròpies de la seva execució en temps real. El monitoratge proporciona, entre d'altres resultats, la informació que permet a un sistema software auto-adaptable modificar la seva execució davant la violació d'uns certs valors sobre aquestes característiques, tals com per exemple un increment elevat del temps de resposta d'un sistema o la detecció d'un nombre elevat d'errors per unitat de temps. De la mateixa manera, els sistemes de monitoratge (com a sistemes software que són) requereixen també poder adaptar la seva execució per satisfer la seva fiabilitat. En aquest context, com podem dotar a un sistema de monitoratge de capacitats auto-adaptables?

En base a aquesta premisa, aquest Treball de Final de Grau consisteix en el disseny i implementació d'un sistema de monitoratge autoadaptable, heterogeni i distribuït que integra un conjunt de monitors de naturalesa diversa i permet, mitjançant la gestió i adaptació de diagrames UML, la seva reconfiguració de forma automàtica. La proposta i el desenvolupament plantejats en aquest projecte es validen mitjançant casos d'ús reals i, en especial èmfasi, amb la seva integració dins el marc de SUPERSEDE, un projecte del programa Horizon 2020 enfocat a la gestió del cicle de vida dels serveis software i les aplicacions, amb l'objectiu de millorar l'experiència final de l'usuari en l'ús d'aquests sistemes.

Índex

Índex de figures

Índex de taules

1 Introducció

El present document consisteix en la memòria del Treball de Final de Grau (TFG) del Grau en Enginyeria Informàtica titulat *Sistema de monitoratge autoadaptable, heterogeni i distribuït*. Com a projecte realitzat a la cloenda dels estudis de grau, el desenvolupament i presentació d'aquest projecte tenen dos objectius principals.

En primer lloc, la consolidació dels coneixements adquirits durant el transcurs del grau. Aquests coneixements engloben des de la qüestió tècnica i específica de la matèria, amb especial èmfasi en els conceptes i aprenentatges relacionats amb l'especialitat d'Enginyeria del Software (tals com el disseny de components software), fins a aspectes relacionats amb la gestió, realització i documentació de projectes complets, pràctics i funcionals, dels quals aquest TFG n'és un exemple. Al llarg dels capítols que composen aquesta memòria, la justificació, explicació i demostració de les tasques realitzades i els conceptes tractats s'exposen amb la rigurositat adequada a un document acadèmic d'aquesta categoria, demostrant l'assoliment d'aquests coneixements amb la major claredat possible.

Per altra banda, aquest projecte pretèn presentar-se com un treball d'investigació, recerca i desenvolupament amb valor propi, dins d'un àmbit i context determinats, amb un objectiu pràctic i aplicable. Més enllà del caire acadèmic, els productes i resultats generats com a conseqüència de la realització d'aquest projecte (components software, disseny i implementació de sistemes, documentació, etc.) esdevenen elements amb valor propi, amb expectatives d'ús i possibilitats d'expansió dins del seu propi context.

Per tal de satisfer aquests dos objectius, aquest projecte presenta el següent propòsit: dissenyar, implementar, gestionar, testejar, validar i mantenir un sistema de monitoratge que satisfaci els criteris d'autoadaptabilitat, heterogeneïtat i distribució (conceptes que s'aprofundiran més endavant). Sota aquesta temàtica, i amb les consideracions prèviament establertes, s'assoliran tant l'objectiu de consolidació de coneixements com la generació d'uns resultats que puguin ser presentats pel seu valor propi i independent.

2 Contextualització

2.1 Presentació i justificació de la temàtica

En les darreres dècades els sistemes software han evolucionat fins al punt d'esdevenir elements clau i imprescindibles de les activitats primàries de qualsevol empresa, organització o institució. La gestió de la informació, els protocols i controls de seguretat, els processos de negoci, etc., són els reptes als quals els *Chief Information Officers* (CIO) de moltes empreses s'han d'enfrontar. Aquests reptes i els seus resultats depenen, en gran mesura, del comportament dels sistemes software que entren en joc dins aquestes activitats. Addicionalment, la quantitat de productes software exposats com a serveis o aplicacions mòbils ha incrementat dràsticament. Fet que deriva en el sorgiment d'una gran varietat de contexts i entorns d'execució entre els grans volums d'usuaris que aquests sistemes poden tenir.

És per aquest motiu que eventualment ha anat prenent força un concepte basat en l'estudi i control de qualitat dels sistemes software: el monitoratge. Com a part de la vida professional d'un enginyer de software, la supervisió i control dels components i sistemes amb els què treballa és un concepte clau amb el qual, d'una forma o altra, ha d'estar familiaritzat. Però el problema que plantegem aquí va més enllà: després del repte de monitorar els sistemes, ens hem de plantejar com dissenyar, gestionar i adaptar aquest monitoratge.

Els reptes que aquestes tasques plantegen i que aquest projecte treballa són diversos. Entre d'altres, cal valorar el disseny i les característiques tècniques dels monitors, la seva configuració i la capacitat d'adaptabilitat. En relació amb aquest últim aspecte, també cal valorar com s'emmagatzemen i es gestionen els detalls relacionats amb la configuració dels monitors, i establir interaccions de la manera més genèrica possible per facilitar l'extensibilitat.

Tal i com veurem més endavant a l'apartat *2.3. Estat de l'art*, existeix una àmplia recerca que actualment treballa i desenvolupa projectes en relació a aquest àmbit. El potencial d'estudi que ofereix resulta d'un alt interès a causa de la possibilitat de recerca i síntesi i als diferents aspectes i criteris sobre els quals es pot treballar.

Així, tant com estudiant com a futur professional del sector de l'enginyeria del software, es poden contemplar diversos criteris per treballar en aquesta temàtica:

- Un aprofundiment en els coneixements de l'enginyeria i els sistemes software
- Treball i recerca en conceptes de control de qualitat, fiabilitat i millora de l'experiència de l'usuari
- Possibilitat de col·laborar i aprofundir en un tema de recerca d'actualitat dins l'enginyeria de serveis i els sistemes d'informació

- Plantejament d'un projecte complet que pugui servir a tercers interessats en l'estudi de sistemes de monitoratge autoadaptatius

2.2 Identificació dels stakeholders

Les diferents fases que engloben aquest projecte deriven en l'obtenció d'un producte final, orientat a la seva aplicació pràctica. Com a tal, els documents generats i els components dissenyats i implementats esdevenen productes propis dins el context del monitoratge de sistemes software. Com a tals, aspectes que s'exposaran al llarg d'aquest document (el disseny i implementació d'una arquitectura genèrica pels monitors, la gestió de les configuracions, etc.) poden resultar d'utilitat per a agents externs a la pròpia autoria del projecte.

Podem considerar, per tant, que podrà ser una eina d'interès pels principals stakeholders, que vindrien a ser:

- **Desenvolupadors i enginyers software**. Aquells agents al càrrec del control de qualitat de sistemes softwares de diverses naturaleses. Els conceptes treballats, el plantejament de problemàtiques, i el producte generat, poden aportar valor de qualitat al sector treballant, per una banda, en la síntesi i recopilació de la informació actual, i per altra banda, aportant propostes i solucions pròpies basades en l'experiència del desenvolupament del projecte.
- **Gestors de projecte i experts en Sistemes d'Informació**. La gestió de la informació, el tractament i el seu potencial poden resultar afectats gràcies a la capacitat de recol·lecció de dades del sistema de monitoratge, així com els criteris de revisió i control de qualitat, que permeten analitzar i obtenir informació fiable.
- **Usuaris finals dels sistemes monitorats**. De forma indirecta, es veuran afectats degut a les conseqüències del monitoratge dut a terme per aquest sistema de monitoratge o d'altres derivats dels conceptes treballats al llarg d'aquest projecte.

2.3 Estat de l'art

Per tal de plantejar les necessitats i projeccions del treball, així com les vies de desenvolupament del projecte, és necessari conèixer quina és la situació del monitoratge autoadaptatiu de sistemes software en la recerca actual.

En primer lloc, cal conéixer l'entorn referent als sistemes software autoadaptatius: és a dir, aquells que seran l'objectiu de monitoratge del nostre sistema de monitoratge (que no deixa de ser un altre sistema software autoadaptatiu). És interessant veure com la recerca i la investigació planteja, de forma pràcticament correlacionada, els conceptes de sistema autoadaptatiu i monitoratge.

De fet, documents de caràcter acadèmic tals com [1] plantegen una arquitectura d'autoadaptabilitat per sistemes software basada en la supervisió o monitoratge. Tal com podem veure a la figura 2.1, destaquem dos components principals: l'aplicació principal, o **Main Application Function**, equivalent al sistema software (servei web, aplicació mòbil, etc.) encarregat d'una funcionalitat específica i sobre el qual volem

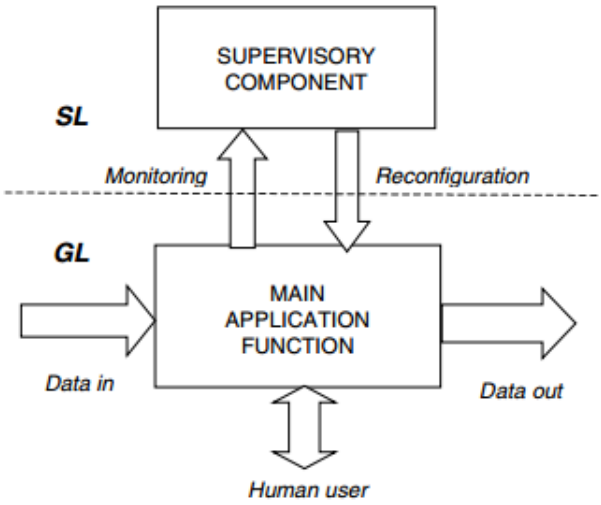

FIGURA 2.1: Arquitectura de sistemes autoadaptables monitorats

realitzar el control de qualitat; i el component supervisor, o **Supervisory Component**, que s'encarrega d'executar aquest control de qualitat.

La interacció i arquitectura plantejada en la figura és relativament senzilla. El component software a avaluar forma part del domini del sistema a avaluar (*ground-level*, GL). En la seva activitat, aquesta aplicació o sistema rep una sèrie de dades (que dependrà de la naturalesa i objectiu del sistema), i produeix uns resultats en base a aquest *input*. Paral·lelament, com qualsevol sistema orientat a l'ús, existeix una interacció sistema-usuari. Més enllà del domini i les característiques pròpies dels sistemes software, es presenta l'entorn corresponent al sistema de supervisió o monitoratge (*supervisory-level*, SL). En aquest nivell trobem des d'un punt de vista lògic el component de supervisió o monitor, que interacciona amb l'aplicació principal de dues formes diferents:

- **Monitoratge**. Procés d'interacció entre el component principal i el monitor on el primer envia informació al segon. L'aplicatiu produeix, com a conseqüència de la seva activitat, informació i dades que envia com a *output* al monitor. Aquesta informació ha d'estar prèviament definida i estructurada, de tal manera que el monitor sigui capaç d'interpretar-la.
- **Reconfiguració**. El monitor procesa les dades que rep com a input del monitoratge i, segons els criteris d'adaptabilitat establerts, aplica els canvis o reconfiguracions pertinents en el sistema software monitorat. D'aquesta manera, la lògica encarregada d'interpretar la informació i prendre decisions d'adaptabilitat (SL) queda totalment separada de la lògica del domini de l'aplicatiu (GL).

Partint d'aquest punt, aquest document estableix les premisses del monitoratge i les necessitats de reconfiguració d'aquests tercers sistemes softwares, la naturalesa dels quals és diversa i heterogènia. Noti's que, de fet, l'anterior disseny presenta una arquitectura genèrica aplicable a qualsevol entorn software, independentment de la naturalesa del domini monitorat. Tot i així, existeix una àmplia recerca especialitzada: publicacions com [2] es centren en analitzar els requisits, necessitats i funcions

principals del monitoratge i reconfiguració de sistemes i recursos al núvol (p.e. serveis web).

La documentació i bibliografia referent als sistemes software autoadaptables i als sistemes de supervisió i monitoratge és molt àmplia. Tot i així, si focalitzem la recerca a la problemàtica a resoldre, és a dir, l'autoadaptabilitat i reconfiguració dels sistemes de monitoratge, no trobem un treball tan profunditzat i específic com en el cas prèviament explicat.

En alguns documents, com per exemple el ja esmentat [2], es mencionen criteris del disseny de la capa de supervisió, entre els quals entren en joc, p.e., factors com la previsió d'errors a la capa de l'aplicatiu principal, o ve esdeveniments/disparadors inesperats. I, com és lògic, una reacció i canvis per part del sistema de monitoratge.

En altres documents, tals com [3], es defineixen models autònoms d'autoadaptabilitat i reconfiguració autònoma, basats en tècniques de detecció d'esdeveniments i canvis en el propi sistema. Conceptes com l'anàlisi de l'estat del sistema, el monitoratge i l'execució d'una reconfiguració entren en joc dins d'aquest pla.

2.4 Projecte SUPERSEDE

Com a part de l'estat de l'art i punt de partida pel desenvolupament del sistema, cal introduïr el projecte SUPERSEDE[4]. Aquest projecte forma part del *Horizon 2020 Programme*[5], un programa de recerca i innovació financiat i gestionat per la Unió Europea. Actualment, compta amb la participació de diverses empreses, fundacions i universitats, entre les quals s'inclou la pròpia UPC.

Aquest projecte planteja una proposta del cicle de vida i la gestió dels serveis software i les aplicacions, amb l'objectiu final similar al plantejat com a premisa d'aquest Treball de Final de Grau: millorar la qualitat de l'execució dels sistemes software i, en conseqüencia, l'experiència de l'usuari final en l'interacció amb aquests sistemes.

Dins aquest cicle de vida, orientat al control de qualitat dels sitemes software, es proposen 4 fases:

1. **Col·lecció**. L'obtenció i emmagatzematge de dades que puguin resultar d'interès pel control de qualitat. La naturalesa d'aquestes dades (així com el format i altres criteris) dependran de l'objectiu d'aquest anàlisi i el tipus de dades tractat. Així, aquestes poden incloure des de dades purament analítiques (p.e. % de disponibilitat del sistema) o bé contextuals (p.e. missatges o continguts introduïts al sistema).

2. **Anàlisi**. Les dades obtingudes en la fase anterior tenen un significat, una informació que el sistema ha de ser capaç d'extreure i comprendre. En aquesta fase, les dades es transformen en coneixement en relació a l'estat del sistema, a través de diverses tècniques analítiques, de nou en funció del marc d'estudi i el context. Per exemple, es podrien valorar tècniques d'anàlisi de llenguatge natural per estudiar les valoracions d'usuaris introduïdes a un sistema.

3. **Decisió**. El coneixement produït a l'anterior fase genera la capacitat de prendre decisions de millora i actuació sobre el sistema software. És a dir, deriva en una

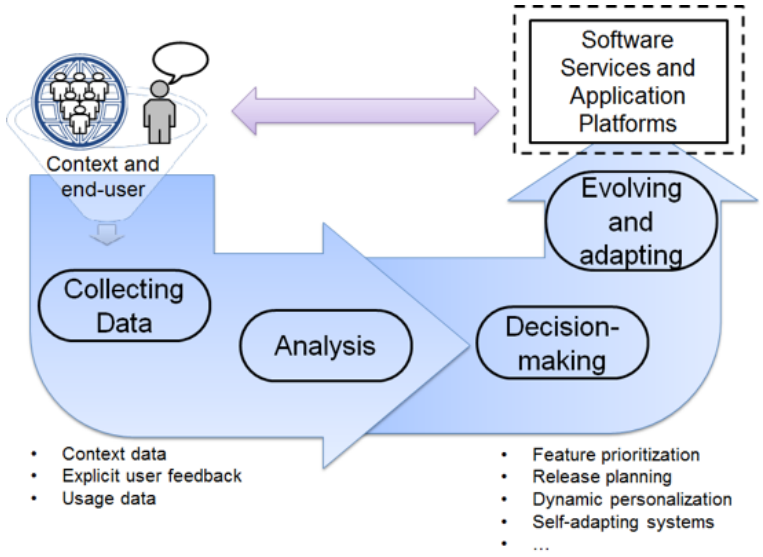

FIGURA 2.2: Cicle de vida i entorn proposat per SUPERSEDE

o vàries propostes d'adaptació. Les eines de presa de decisions entren en joc en aquesta fase, rebent com a entrada la informació i, a partir dels criteris i paràmetres definits en relació a aquesta informació, es produeix la proposta d'adaptació del sistema.

4. **Adaptació**. Un cop el sistema ha estat capaç de produir de forma automàtica una proposta de millora o adaptació del sistema, aquesta s'aplica sobre el component monitorat amb l'objectiu de millorar l'experiència de l'usuari. Arribats a aquest punt, es tanca el cicle de control de qualitat, reflectint la transformació de l'input de la fase de col·lecció, les dades, fins a l'output d'aquesta darrera fase, l'adaptació del sistema.

La figura 2.2 resumeix aquest cicle de vida i les característiques del context de les seves fases. Tal i com es pot observar, la principal font de dades amb les quals aquest control de qualitat es nodreix es l'experiència de l'usuari final, el context on s'executa aquesta aplicació o sistema i les dades generades del propi ús i funcionament del sistema. És, per tant, un aspecte clau l'obtenció de **dades en execució real**, que seran les que ens permetin aplicar aquest cicle regularment. Regularitat que serà crucial per satisfer l'objectiu d'anàlisi de la qualitat del sistema: només amb dades actuals i constants serem capaços de conèixer l'activitat real del nostre sistema i actuar en conseqüència.

Aquest cicle de vida no és més que una descripció del cicle *MAPE-k* [6][7], un cicle de vida del software orientat a la retroalimentació i adaptació de l'activitat classificat en aquestes quatre fases: ***M**onitoring*, ***A**nalysis*, ***P**lan* i ***E**nactment*.

En aquest context identifiquem per tant 4 sectors o subsistemes amb un objectiu específic que, integrats, serveixen a un propòsit genèric. Si ens plantegem com encaixa aquest model dins el nostre tema d'estudi (és a dir, l'adaptabilitat dels sistemes de monitoratge) podem establir una relació directa amb les fases de **col·lecció** de dades i **adaptació**. Dins aquesta primera fase de col·lecció, necessitem definir un sistema capaç d'obtenir aquestes dades, en funció dels sistemes a controlar i de l'interès que tinguem sobre aquests sistemes. Aquest sistema de **monitoratge** estarà compost per

un conjunt definit de monitors, encarregats de recollir aquesta informació. Per altra banda, si volem dotar a aquest sistema de monitoratge d'adaptabilitat, i per tant, garantir un control de qualitat sobre aquests monitors, necessitem establir un sistema que gestioni l'activitat dels monitors i defineixi adaptacions a aplicar sobre aquests monitors.

Pel desenvolupament d'aquest projecte, ens centrarem en aquestes dues parts: el sistema encarregat de la fase de col·lecció de dades (monitors), i el sistema encarregat de gestionar i aplicar les adaptacions sobre aquest sistema de col·lecció. De la mateixa manera, es treballa la integració entre aquests dos components, per generar com a resultat final l'objectiu d'aquest projecte: un sistema de monitoratge autoadaptable, heterogeni i distribuït.

3 Objectius

Definit el context, l'àrea d'estudi i una aproximació a l'estat de l'art actual d'aquest projecte, cal definir amb el màxim nivell de detall quins seran els objectius principals, així com els objectius específics i l'abast, per tal d'introduïr els conceptes treballats durant el desenvolupament del mateix.

3.1 Objectiu general

L'objectiu principal d'aquest projecte consisteix en la implementació d'un sistema software orientat al monitoratge d'altres sistemes softwares. Aquest sistema haurà de complir 3 característiques principals: ser autoadaptable, heterogeni i distribuït. A continuació procedim a explicar en detall què entendrem per aquestes característiques dins el context d'aquest projecte, en base a la contextualització i els conceptes explicats anteriorment:

1. **Autoadaptable**. El sistema de monitoratge generat ha d'estar dotat de capacitats d'adaptabilitat de la seva execució en temps real. Mitjançant la gestió i control de la seva activitat, els diferents monitors han d'oferir eines d'adaptació orientades al control de qualitat del propi sistema. Per fer-ho, caldrà tenir en compte dos punts que es desenvoluparan més endavant: en primer lloc, la dotació dels monitors d'aquestes eines d'adaptació; en segon lloc, el disseny i implementació dels components necessaris per gestionar les adaptacions.

2. **Heterogeni**. El sistema constarà d'un conjunt de monitors de naturalesa variada i permetrà, mitjançant un disseny i una arquitectura prou genèrica, la integració de nous monitors de diversa índole. Per tant, el sistema haurà d'estar capacitat per gestionar els diversos tipus de monitors tot i les seves diferències en aspectes com el sistema monitorat, la naturalesa del monitoratge, les necessitats de configuració, etc. L'objectiu d'aquesta característica és que el resultat final sigui el més aprofitable i reusable possible.

3. **Distribuït**. El sistema haurà de permetre desplegar els diferents monitors i els components d'adaptabilitat de forma distribuïda i, per tant, tenir la capacitat de desplegar els diferents components com a elements independents dins el nostre sistema genèric.

Els detalls tècnics de l'assoliment d'aquests 3 objectius es desenvoluparan al llarg d'aquesta memòria.

3.2 Objectius específics

En base a l'objectiu general prèviament establert, cal definir una sèrie d'objectius específics que ens permetran assolir-lo definint unes vies prou clares com per a facilitar el desenvolupament del projecte. Procedim, doncs, a enumerar aquests objectius:

OBJ1. Definir una planificació pel desenvolupament del projecte en funció dels requisits.

OBJ2. Dissenyar una arquitectura software adequada a les necessitats (sistema autoadaptable, heterogeni i distribuït).

OBJ3. Implementar el sistema de monitoratge.

OBJ4. Implementar el sistema d'adaptació dels monitors.

OBJ5. Generació d'un producte final usable, que pugui ser desplegable (segons els criteris de distribució) i validat per un escenari d'ús específic.

OBJ6. Configurar i definir l'entorn de desenvolupament i d'ús del sistema.

OBJ7. Assegurar la qualitat i fiabilitat del sistema mitjançant la validació dels components i del sistema integrat.

OBJ8. Seguir una metodologia de desenvolupament i validació del sistema.

OBJ9. Definir una sèrie de casos d'ús per mostrar la usabilitat i comportament real del sistema.

OBJ10. Documentar i justificar l'evolució del projecte.

Aquests objectius engloben les dues vessants d'aquest projecte, ja especificades anteriorment: la generació i documentació d'un Treball de Final de Grau, i el disseny i la implementació del sistema descrit. En qualsevol cas, aquests objectius específics defineixen les "metes finals"d'aquest projecte. Per garantir-ne i comprendre el desenvolupament fins a assolir-los, cal definir les tasques i, per tant, l'abast específic d'aquest projecte.

3.3 Abast del projecte

Els objectius específics prèviament identificats ens donen una visió acurada de l'abast del nostre projecte i les tasques a realitzar. Tot i així, és important reflectir de forma explícita l'abast d'aquest projecte, enumerant els requisits (o dit d'una altra manera, les tasques o necessitats a satisfer) i delimitant el nostre projecte. Ens basarem per tant en els següents punts:

- Realitzar una recerca bibliogràfica (basada en l'estat de l'art) per assentar les bases i el context del desenvolupament del projecte.
- Dissenyar, implementar i documentar un disseny arquitectònic software que satisfaci l'objectiu general i els tres criteris (autoadaptabilitat, heterogeneïtat i distribució) del nostre sistema de monitoratge.
- Dissenyar, implementar i documentar el sistema d'adaptabilitat dels monitors i realitzar la integració amb els mateixos.
- Definir un cas d'ús o escenari que ens permeti validar les funcionalitats del sistema amb exemples mitjançant l'execució real.
- Dissenyar i implementar un *dashboard* que permeti visualitzar l'activitat del sistema d'adaptabilitat.

Per contra, cal especificar quines parts no considerarem com a part de l'abast del projecte:

- El desenvolupament d'un sistema de presa de decisions i anàlisi (***A*nalize** dins el cicle *MAPE-k*); assumirem que la responsabilitat d'aquest component serà assumida per sistemes tercers, que es comunicaran amb el nostre sistema generat
- Reconfiguracions i adaptacions complexes. Per assegurar la viabilitat del projecte i una validació satisfactòria, establirem un cas d'ús bàsic que ens permeti validar els components, i a arrel d'aquest escenari, es plantejarà com a desenvolupament futur l'extensió d'aquestes reconfiguracions. Tot i així, en tot moment el disseny del sistema anirà orientat a suportar aquests casos més complexos, per assegurar el potencial del sistema.

4 Gestió i desenvolupament

Abans d'entrar en els detalls del projecte, necessitem definir sota quins criteris i quines pràctiques realitzarem la gestió i el desenvolupament del projecte.

4.1 Metodologia de desenvolupament

Les necessitats i requisits específics del projecte aniran fortament relacionades amb la recerca i l'avenç del propi transcurs del projecte. Si bé els objectius específics queden clars, les tasques a desenvolupar aniran evolucionant dinàmicament. Per aquesta raó, en aquest cas serà adequat seguir una metodologia de desenvolupament àgil. I, en concret, es seguirà una simplificació de la metodologia Kanban[8].

En base als requisits establerts inicialment durant la planificació del projecte, i al llarg del seu transcurs, s'aniran generant una sèrie de tasques que seguiran el següent *workflow*:

1. **To do**. Identificat com el *backlog* del projecte, anirem definint les tasques a realitzar conforme aquestes es vagin identificant.
2. **Analysis**. Un cop identificades les tasques, la primera fase per cadascuna d'elles seran les relacionades amb disseny i anàlisi des del punt de vista de requisits i de disseny software.
3. **Development**. Finalitzat l'anàlisi de la tasca, es procedeix a la seva implementació.
4. **Test**. Validació del component o tasques realitzades segons els seus requisits funcionals.
5. **Deploy**. Desplegament a l'entorn de producció (per aquest projecte, servidor localitzat a un dels *partners* del projecte) i prova del component en aquell mateix desplegament.
6. **Done**. Estat de finalització del desenvolupament d'aquella tasca. En aquest punt la seva validació ha finalitzat.

Per garantir la integritat del sistema, cadascuna d'aquestes tasques serà desenvolupada fora de l'entorn de producció, en un entorn (o branca) separats. D'aquesta manera, el desenvolupament no afectarà al producte provisional generat en cada moment del desenvolupament del projecte. I, alhora, permetrà fer un seguiment més exacte de l'estat de cada tasca o funcionalitat a implementar.

S'ha considerat millor opció a, per exemple, alternatives àgils com Scrum, degut a diversos factors. P.e., a Kanban les entregues o releases són constants, i no acotades temporalment[9]. Considerarem més important, per tant, la metodologia basada en el producte final.

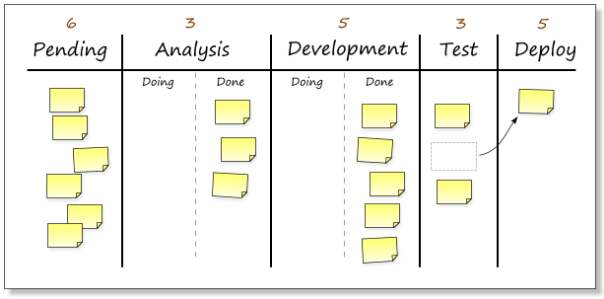

FIGURA 4.1: Imatge representativa de la metodologia Kanban

4.2 Recursos

Per definir les necessitats de recursos per satisfer la realització del projecte, els classificarem segons el següent criteri: recursos **humans**, recursos **materials** i recursos **software**.

Recursos humans

Pel domini de nostre projecte, basat en 1 desenvolupador principal (estudiant), considerarem les seves hores de treball com recursos humans. Estimarem, i considerant els següents aspectes:

- El TFG es correspon a 18 ECTS (3 crèdits ECTS GEP + 15 crèdits ECTS TFG) [10]
- 1 ECTS = 25-30 hores de treball [11]
- Durada aproximada TFG = 22 setmanes

Podem estimar, per tant, 30 hores/ECTS * 18 ECTS / 22 setmanes = 25 hores aproximadament de desenvolupament setmanal del TFG.

Recursos materials

Els recursos materials per aquest projecte són relativament senzills. Bàsicament:

- **Portàtil Lenovo G-50**. Màquina principal amb la qual es durà a terme el projecte
- **Materials d'impressió**. Necessaris per documentació, impressió de memòria, etc.

Recursos software

Tot i que aquests es discutiran amb més detall al capítol *5. Eines de desenvolupament*, podem identificar inicialment la necessitat d'alguns recursos software principals, tals com:

- **Gestor de versions**. El codi (emmagatzemament, desenvolupament i evolució) requereix un control i manteniment, motiu pel qual s'estableix com a necessitat una eina d'aquest tipus

- **IDE**. És necessari l'ús d'un entorn de desenvolupament integrat per desenvolupar el projecte, dissenyar l'arquitectura, realitzar la implementació, configurar l'entorn, etc.
- **Sistema de compilació automàtic**. Necessari per gestionar la compilació i les dependències dels projectes.
- **Framework desenvolupament web**. Segons les necessitats que es defineixin al llarg del projecte, es triarà una tecnologia específica per desenvolupar el dashboard definit com a requisit al projecte.

Les opcions triades per suplir les necessitats d'aquests recursos i altres recursos software identificats com a necessaris es plantejaran més endavant.

4.3 Planificació temporal

Segons les necessitats d'aquest projecte i les seves característiques, classificarem el desenvolupament en 4 fases: la planificació, el disseny, la implementació, i la documentació i entrega del projecte.

4.3.1 Descripció de fases

Planificació

En primer lloc cal una fase inicial o fase de planificació. L'objectiu principal d'aquesta fase del projecte és definir els aspectes bàsics que definiran la naturalesa del projecte: els objectius, la metodologia de treball, el desenvolupament, els requisits, la planificació, etc. És a dir, tot allò relacionat amb l'establiment de les premisses sobre les quals ens basarem per desenvolupar el nostre projecte.

Aquesta fase del projecte va fortament lligada al desenvolupament del curs GEP, juntament amb altres activitats. En definitiva, les tasques a realitzar són les següents:

- **P1.** Investigació i recerca en quant al monitoratge i l'adaptabilitat de sistemes software.
- **P2.** Definir l'abast i el context del projecte.
- **P3.** Definir les fases i tasques del projecte, així com una planificació temporal de desenvolupament.
- **P4.** Preparar una gestió econòmica i una anàlisi de sostenibilitat.
- **P5.** Establir els requisits i necessitats d'acord a l'especialitat d'Enginyeria del Software.
- **P6.** Definir les eines de desenvolupament i els llenguatges amb els que treballar i implementar el sistema i el dashboard.
- **P7.** Definir els casos d'ús per validar i implementar el sistema.

Disseny

Conforme la fase de planificació avanci, podrem començar a realitzar el disseny del nostre sistema des d'un punt de vista de requisits i també arquitectònic (en referència a arquitectura del software). És a dir: l'objectiu d'aquesta fase és passar dels conceptes i objectius definits a la planificació a un "mapa" o "esquema" que serveixi de guia pel desenvolupament del projecte (subjecte, per descomptat, a possibles canvis i adaptacions al llarg del desenvolupament).

Principalment, les tasques a incloure en aquesta fase són:

- **D1.** Definir els entorns de configuració sota els quals es desenvoluparà el projecte
- **D2.** Dissenyar l'arquitectura software del sistema de monitoratge i dels monitors
- **D3.** Definir els criteris d'adaptabilitat amb els quals es dotarà al sistema
- **D4.** Configurar un entorn de configuració d'acord amb els criteris definits
- **D5.** Documentar els avanços referents a aquesta fase (memòria)

Implementació

Aquesta fase tindrà la major càrrega de feina de tot el projecte. Partirà d'uns criteris i objectius ben definits i estructurats a partir de les anteriors fases. Les tasques es correspondran principalment a totes aquelles tasques d'implementació i de testing del nostre sistema.

Per tant, identificarem com a tasques:

- **I1.** Implementació de l'arquitectura genèrica del sistema de monitoratge.
- **I2.** Implementació del sistema de monitors.
- **I3.** Implementació del sistema d'adaptabilitat.
- **I4.** Integració dels sistemes.
- **I5.** Implementació d'un dashboard que permeti visualitzar l'activitat del sistema
- **I6.** Generació de documentació referent al desenvolupament i la implementació (documentació d'APIs, README per desplegar el sistema, manual d'usuari del dashboard, etc.)
- **I7.** Ampliació del sistema (afegir monitors, ampliar dashboard) en funció de les necessitats i/o de la disponibilitat temporal
- **I8.** Redacció i ampliació de la memòria

Fase final

Finalment, identificarem una darrera fase del projecte que servirà de cloenda per preparar l'entrega i defensa final de la feina realitzada, així com tancar possibles tasques pendents i assegurar el correcte funcionament del sistema i la generació d'un producte final adequat.

Les tasques principals seran:

- **F1.** Testing de les funcionalitats del sistema
- **F2.** Comprovació de la satisfacció dels objectius i requisits
- **F3.** Finalització i revisió de la memòria
- **F4.** Preparació de la defensa final

4.3.2 Previsió d'alternatives i pla d'acció

La planificació temporal i de tasques plantejada, així com el consum de recursos, contempla un desenvolupament del treball regular i sense imprevistos. Tot i així, hem de considerar alternatives al desenvolupament fruït d'imprevistos, desviacions o altres factors no contemplats dins de la normalitat. Identificarem, per tant, les possibles següents desviacions:

- **Increment del nº d'hores necessàries.** Aquest problema pot ser derivat per diverses causes (inhabilitació temporal del desenvolupador, dificultats tècniques en el desenvolupament, etc.). Això pot provocar, per una banda, una necessitat de més hores, i per altra banda, més dedicació per unitat de temps.

  Pla d'acció. La tasca corresponent a la implementació del dashboard serà adaptada en funció de l'estat del projecte arribat al moment. És a dir: al tractar-se d'un requisit molt flexible en quant a la seva complexitat (podem aspirar a un dashboard molt complet i multifuncional, o bé establir els requisits mínims per satisfer els criteris d'acceptació del TFG), podem permetre'ns retallar hores d'aquesta tarda, prioritzant aspectes crucials (com p.e. la implementació dels monitors).

- **Avaria en el hardware.** És possible que durant el desenvolupament del projecte el hardware utilitzat (en aquest cas, el portàtil Lenovo G-50) pateixi alguna avaria. Al tractar-se de la principal eina de desenvolupament, això pot afectar als terminis de desenvolupament.

  Pla d'acció. Dues mesures complementàries: per una banda, en tot moment es mantindran diverses còpies de seguretat de tots els artefactes generats (documentació, software, entorns de configuració...) per garantir-ne la recuperabilitat; per altra banda, disposarem d'un entorn de treball alternatiu (un sistema operatiu instal·lat a un disc dur extern) on podrem seguir el desenvolupament del nostre projecte paral·lelament mentre l'avaria es soluciona.

- **Canvis en els requisits del projecte.** Podem suposar que el desenvolupament pràctic del projecte ens durà a concloure nous canvis necessaris en quant als requisits prèviament establerts.

  Pla d'acció. En primer lloc, sotmetre a constant revisió el projecte per tal d'evitar al màxim que es produeixi un canvi de requisits significatiu. Per ferho, constantment es revisaran requisits, nivell de viabilitat, i satisfacció envers els terminis establerts. Si, per contra, es produeix un canvi significatiu inevitable, el nº d'hores total del projecte (aproximats) permet un petit increment fruit d'aquesta desviació, per tal de garantir que la resta de tasques es duen a terme correctament.

4.4 Viabilitat

Un dels principals objectius del Treball de Final de Grau és plasmar la capacitat de l'estudiant de generar, mitjançant els coneixements obtinguts, un producte o projecte real, amb una utilitat i uns objectius aplicables al nostre entorn. Per aquest motiu, i com estudiants, cal assumir la responsabilitat del projecte i estudiar-ne la seva viabilitat en dos sentits. Per una banda, la seva viabilitat econòmica, mitjançant una estimació pressupostària dels costos del projecte en un àmbit professional. Per altra banda, el seu impacte econòmic, social i ambiental des d'un punt de vista de sostenibilitat.

4.4.1 Estimació pressupostària

Per simplificar al màxim l'estudi dels costos i, alhora, clarificar i estudiar-ne la justificació, procedirem a identificar i estimar els diferents costos associats al projecte segons el seu tipus.

Despeses directes

Entraran dins la classificació de despeses directes aquelles despeses derivades directament de la realització d'activitats previstes pel desenvolupament del projecte. Per una major precisió, relacionarem directament les activitats definides a l'anterior entregable amb els costos directes.

Per aquest cas, farem les següents assumpcions:

- En el projecte intervindrà el desenvolupador (l'alumne) com a agent principal desenvolupament).
- L'estimació del cost serà de 15€/hora pel desenvolupador, en base a la informació actual que podem trobar referent a aquest aspecte pels rols de programador junior i altres càrrecs relacionats als rols que l'alumne assumirà.

Despeses indirectes i amortitzacions

Considerarem les següents despeses indirectes i amortitzacions:

- **Impressions a paper**. Considerarem 150 fulls / memòria, a 3 memòries a entregar per la defensa final + 1 de provisional; com a afegit, 200 fulls per articles, documentació... fan un total de 800 fulls.
- **Electricitat**. Cost i consum en base a referències de característiques de portàtil i preu estàndard de companyies elèctriques.
- **Amortització**. portàtil Lenovo G-50. Cost en base a compra; percentatge d'amortització en base a les hores útils de vida aproximada i les hores de rendiment esperades de programació, documentació, etc.
- **Software**. El desenvolupament serà basat en software lliure i, per tant, sense despesa addicional, però el considerarem com a part del pressupost per possibles desviacions.

ACTIVITAT	HORES TOTALS	COST TOTAL
Planificació	**149**	**2285,00€**
Abast i context	30	450,00€
Planificació	12	180,00€
Gestió econòmica i sostenibilitat	12	180,00€
Requisits d'especialitat	12	180,00€
Recerca de la temàtica	38	570,00€
Eines de desenvolupament	20	350,00€
Definir casos d'ús	25	375,00€
Disseny	**102**	**1530,00€**
Entorn de configuració	15	225,00€
Arquitectura software	32	480,00€
Criteris d'autoadaptabilitat	25	375,00€
Integració continuada	20	300,00€
Documentació memòria	10	150,00€
Implementació	**203**	**3675,00€**
Implementació arquitectura genèrica	30	450,00€
Implementació monitors	60	900,00€
Implementació sistema d'adaptabilitat	70	1050,00€
Integració sistemes	20	300,00€
Implementació dashboard	25	375,00€
Documentació de components	15	225,00€
Ampliació del sistema	15	225,00€
Documentació memòria	10	150,00€
Fase final	**85**	**1275,00€**
Finalització tasques	10	150,00€
Testing	20	300,00€
Comprovació satisfacció	15	225,00€
FInalització memòria	25	375,00€
Preparació defensa final	15	225,00€
Total	**539**	**8765,00€**

TAULA 4.1: Costos directes

Contingències

Afegirem, en base als costos directes i indirectes prèviament desglosats, un 15% sobre el total en concepte de contingències.

Imprevistos

Identificarem 2 imprevistos en base a la planificació:

- **Prolongació de les hores / ampliació del termini**. Contemplarem la possibilitat de requerir més temps de l'estimat per acabar el projecte, considerant una desviació de fins a 50 hores (que podríem considerar la dedicació aproximada de dues setmanes a mitja jornada).
- **Avaria de hardware**. Problemes en l'ús del material hardware (en aquest cas, exclusivament el portàtil).

CONCEPTE	COST UNITARI	UNITATS	COST
Impressions a paper	0,03€/full	800 fulls	24,00€
Electricitat	0,20€/kWh	225 kWh	45,00€
Amortització portàtil Lenovo G-50	450,00€/portàtil	0,15 (amortitzat)	67,50€
Software	0,00€/mes	5 mesos	0,00€
Total			**136,50€**

TAULA 4.2: Costos indirectes i amortitzacions

CONCEPTE	COST BASE	% CONTINGÈNCIES	TOTAL
Costos directes	8765,00€/full	15	1314,75€
Costos indirectes	136,50€/kWh	15	20,475€
Total			**1335,225€**

TAULA 4.3: Contingències

Per determinar la probabilitat de l'increment horari, utilitzarem el document de referència *Chaos Manifesto* [12], que utilitza dades reals per determinar la probabilitat de causes de riscos en el desenvolupament de projectes, tals com l'increment del temps necessari.

CONCEPTE	COST UNITARI	UNITATS	COST TOTAL	% PROBABILITAT	TOTAL
Ampliació del termini	12,00€/hora	50 hores	600,00€	47.8	286,80€
Avaria del hardware	450,00€/ordinador	1 ordinador	450,00€	5	22,50€
Total					**309,30€**

TAULA 4.4: Imprevistos

Pressupost global

Un cop valorat costos indirectes, costos indirectes i amortitzacions, contingències i possibles imprevistos, podem donar una versió completa de l'estimació pressupostària per la realització del projecte.

Per tant, el pressupost final és de **10.546,025€**.

4.4.2 Control de gestió

El pressupost exposat ja inclou com a part de la partida destinada una part generada per imprevistos i desviacions amb possibilitats de produir-se i que pretenen precisament realitzar un control i manteniment de la gestió i evolució del projecte i els recursos (veure apartats 2.1.3. i 2.1.4.).

Tot i així, podem considerar oportú establir uns mecanismes, o tasques específiques, dedicades al control periòdic que permetin fer un seguiment de l'activitat de gestió de projecte i, per tant, no només preveure de forma teòrica aspectes com desviacions pressupostàries, sinó detectar al llarg de l'evolució del projecte quan això succeeixi.

Per fer-ho es proposa realitzar un control de desviacions durant la transició de fases; és a dir, treballarem amb un model de plantilla que ens calculi una sèrie de desviacions (en funció de diversos criteris), que al finalitzar cada fase ens permeti obtenir un feedback objectiu i ràpid de possibles desviacions respecte al pressupost

CONCEPTE	COST
Costos directes	8765,00€
Costos indirectes	136,50€
Contingències	1335,225€
Imprevistos	309,30 142,50€
Total	**10.546,025€**

TAULA 4.5: Resum global del pressupost

inicial.

Per fer-ho, i basant-nos en la bibliografia utilitzada a GEP, farem servir els següents indicadors:

- **Desviament de mà d'obra en preu** = (cost estimat - cost real) * consum hores real
- **Desviament en la realització d'una tasca en consum** = (consum estimat - consum real) * cost real
- **Desviament total en la realització de tasques** = cost total estimat tasca - cost total real tasca
- **Desviament total de despeses fixes** = total costos fixes pressupostat - total costos fixes real

Considerarem aquests 4 indicadors, ja que ens seran els més útils per detectar desviacions, p.e., en quant a la realització de les activitats descrites al Gantt, en termes tant de dedicació en quantitat d'hores total com per tasca, i també en aspectes com despeses fixes (p.e. amortitzacions). Mitjançant la comprovació d'aquests indicadors, podrem veure el grau de desviament i actuar en conseqüència segons les necessitats.

4.4.3 Sostenibilitat econòmica, social i ambiental

Procedim a fer un anàlisi de les 3 dimensions de la sostenibilitat en referència a aquest projecte, per posteriorment avaluar fent servir la matriu de sostenibilitat el grau de satisfacció d'aquest àmbit en funció dels criteris establerts.

Dimensió econòmica

La dimensió econòmica està satisfactòriament treballada gràcies al pressupost prèviament exposat, basat en dades objectives i específiques (p.e., activitats reals a realitzar durant el projecte), que inclou despeses materials i humanes. Aquests costos i temps de dedicació inclouen aspectes crítics tals com possibles desviacions i imprevistos, i una assignació proporcional dels recursos assignats a la rellevància de cada tasca. Es tracta d'un projecte realitzat amb el cost mínim, però suficient (tenint en compte sempre que és necessari afegir extres per desviacions), garantint la seva satisfacció però sense despeses innecessàries, el que garanteix la seva viabilitat econòmica.

Dimensió social

L'objectiu principal del treball és aprofundir en el control de qualitat i monitoritatge de sistemes software mitjançant el desenvolupament de software lliure reaprofitable. Tal i com vam veure a l'entregable 1 (a l'apartat Estat de l'art), existeix marge d'investigació i treball en aquest àmbit, i l'àmplia gama de serveis software poden extreure un cert benefici en base als avenços (o si més no, la recerca i síntesi) que aquest projecte pugui aportar. Des del punt de vista dels desenvolupadors (usuaris reals d'aquest projecte, ja que seran els que l'utilitzaran), aportem noves eines i facilitem criteris d'autoadaptabilitat per monitors de control de qualitat. També, però, tindrà conseqüències en els usuaris dels sistemes monitorats, ja que la recol·lecció de dades d'aquests està orientada a la millora de la qualitat dels serveis oferts per aquests sistemes. Aquest és un aspecte que cada vegada requereix més profunditat, motiu pel qual podem considerar l'existència d'una necessitat dins el mercat actual. Podem considerar que no existeixen col·lectius afectats negativament.

Dimensió ambiental

Els recursos plantejats tant pel desenvolupament del projecte com el consum necessari per la seva vida útil són mínims, i inclouen únicament aquells derivats del manteniment d'un sistema software. No existirà contaminació destacada més allà de la generada pel consum d'electricitat del dispositiu portàtil a utilitzar pel desenvolupament del projecte o la impressió de papers (que es limitarà al mínim necessari). Es tracta, a més, d'un projecte que té per objectiu ser reaprofitat per tercers projectes (plantejant estructures, arquitectures, monitors, etc. reaprofitables).

Matriu de sostenibilitat

En base als anteriors criteris establerts, assignarem les següents puntuacions a la matriu de sostenibilitat, considerant únicament la Planificació per cadascuna de les 3 dimensions:

Sostenibilitat	**Econòmica**	**Social**	**Ambiental**
Planificació	9	8	7

TAULA 4.6: Matriu de sostenibilitat

Podem considerar, juntament amb la informació prèviament exposada, les següents justificacions:

- **Dimensió econòmica**. S'assoleixen satisfactòriament criteris econòmics amb rigor i detall (basat en pressupost) i es presenta informació verídica en quant a costos, optimitzats per un ajustament adequat.
- **Dimensió social**. Tot i que l'impacte pot no ser especialment destacable, sí que té un mercat profitós i aporta beneficis dins el seu sector que el fan un projecte positiu des del punt de vista social.
- **Dimensió ambiental**. No aporta un benefici destacable directe però sí que assoleix els criteris d'eficiència ambiental en quant al consum de recursos o l'empremta ecològica del projecte, que podrà ser reutilitzat per projectes tercers.

5 Visió general del sistema

En aquesta part no es presentaran detalls més enllà de la naturalesa, objectius i funcionalitats generals del sistema i els seus components, ja que aquests es desenvoluparan més endavant, un cop els requisits estiguin definits.

En primer lloc, establim de nou la premisa d'aquest projecte: el **disseny**, la **implementació** i **validació** d'un sistema de **monitoratge** que satisfaci les característiques d'**adaptabilitat**, **heterogeneïtat** i **distribució** (característiques explicades al *Capítol 3. Objectius*). En base al context del projecte SUPERSEDE (presentat al *Capítol 2. Contextualització*), i segons aquest objectiu, el nostre sistema haurà d'incloure dues vessants:

- Un **sistema de monitoratge** de serveis i components software tercers.
- Un **sistema d'adaptabilitat** que permeti adaptar l'activitat del sistema de monitoratge.

El component clau de l'activitat del monitoratge és el que anomenem **monitor**. Un monitor no és més que un component software (independentment de la seva naturalesa o la tecnologia amb la qual està desenvolupat) que interactua amb un component software i col·lecciona informació relacionada amb la seva activitat, tal i com s'explica al *Capítol 2.2. Estat de l'art*. Per tal de generar un sistema de monitoratge dins el nostre projecte, haurem de tenir en compte diversos factors.

En primer lloc, necessitarem definir una **arquitectura genèrica** que ens permeti definir l'estructura i arquitectura bàsica dels monitors que inclourem al nostre projecte. D'aquesta manera, mitjançant criteris que s'estudiaran més endavant, el nostre sistema disposarà d'un component genèric a partir del qual podrem generar **monitors específics**, independentment de la seva activitat en termes específics. Així, garantit la característica d'**heterogeneïtat**, el nostre sistema permetrà la seva extensió mitjançant la implementació de nous monitors que es puguin integrar al sistema.

En segon lloc, haurem de considerar per una banda que aquests monitors han de ser components independents que es puguin desplegar de forma distribuïda i que la seva activitat pugui actuar com a unitat per sí mateixa. Per altra banda, per gestionar la integració del nostre sistema, necessitarem definir components que **integri** aquest conjunt de monitors en un únic punt i sigui capaç de gestionar l'activitat dels mateixos.

Paral·lelament al sistema de monitoratge, necessitem dissenyar i implementar una part del sistema que **gestioni les configuracions dels monitors** (és a dir, les diferents activitats de monitoratge) i pugui gestionar les adaptacions sobre els monitors. Per gestionar tot aquest subdomini del projecte, s'utilitzaran un **conjunt de models UML** amb els quals es modelaran tots els detalls relacionats amb les configuracions

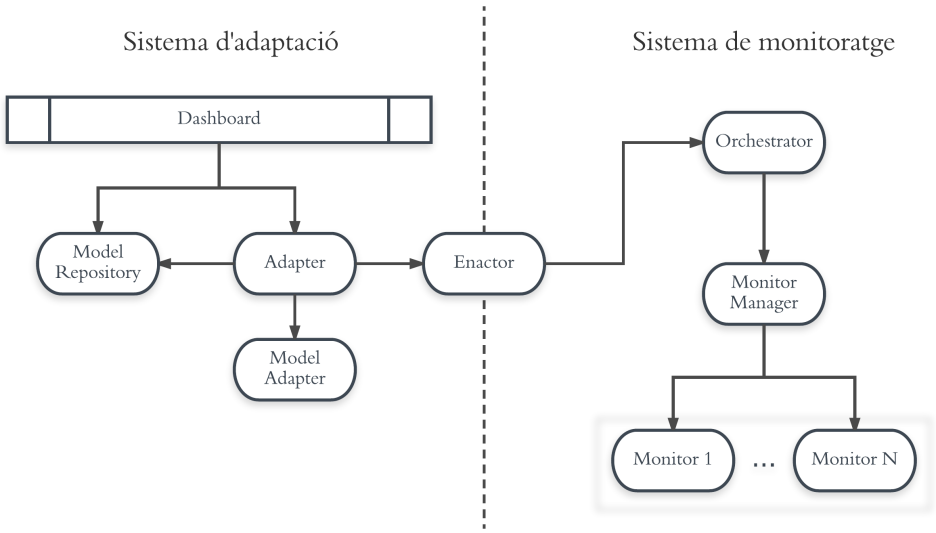

FIGURA 5.1: Disseny genèric del sistema proposat

i les adaptacions dels monitors: configuracions actuals, propostes de noves configuracions, detalls sobre mecanismes d'adaptacions, etc. Mitjançant aquest conjunt de models, que més endavant es detallaran, el sistema podrà **computar i aplicar de forma automàtica adaptacions** sobre els monitors desplegats. Per garantir el funcionament i la validació del sistema, caldrà que aquests dos subcomponents estiguin integrats i es puguin comunicar entre ells, seguint els criteris d'adaptació.

Finalment, com a tasca complementària, el nostre sistema inclourà un *dashboard* consultor que permeti visualitza les diferents adaptacions que el sistema realitza sobre els monitors, per tal de poder observar i validar l'activitat del sistema d'acord amb els requisits establerts.

Definida la visió general del nostre sistema, i abans d'adreçar-nos als requisits específics, podem definir una **proposta de disseny** del sistema (presentada a la *Figura 5.1*) basada en els diferents components que intervindran per dur a terme l'activitat prèviament descrita. En base a aquesta proposta, procedirem a explicar cadascun dels components que intervenen segons els següents criteris: element/s d'entrada o **input**, comportament intern o **action**, i element/s de sortida o **output**.

5.1 Disseny del sistema

Primerament, comencem per explicar els components que formen el **sistema de monitoratge**:

- **Monitor.** Component de naturalesa ja descrita anteriorment, és l'encarregat de realitzar la col·lecció de dades d'un sistema software orientat al control de qualitat d'aquest. Dins el nostre sistema, disposarem d'un conjunt de monitors variats, conjunt que podrà ser extès sota diversos criteris i seguint el marc de la proposta de disseny plantejada.

  - ***Input***. Informació/paràmetres de configuració d'un procés de monitoratge.
  - ***Action***. Procés de la informació per iniciar, aturar o modificar els paràmetres de la configuració d'un procés de monitoratge.
  - ***Output***. Conjunt de dades recol·lectades pels processos de monitoratge actius en el monitor.

- **Monitor Manager.** Encarregat de gestionar l'activitat dels monitors, integrant el conjunt de monitors independents en el sistema sota un únic punt d'entrada, amb una semàntica genèrica. És l'encarregat, per tant, de redireccionar les diferents reconfiguracions (així com l'inici i aturada de processos de monitoratge) als monitors corresponents.

  - ***Input***. Informació/paràmetres de configuració d'un procés de monitoratge per a un monitor específic.
  - ***Action***. Procés de la informació i transformació de la mateixa d'acord amb el monitor corresponent.
  - ***Output***. Informació/paràmetres de configuració transformats i orientats al monitor corresponent.

- **Orchestrator.** Aquest document forma part del context del projecte SUPERSEDE. Dins aquest projecte (presentat al capítol *2.2.1. Projecte SUPERSEDE*), aquest component és el punt d'integració entre el subsistema d'adaptació de sistemes software i el subsistema de col·lecció i anàl·lisi de dades, i actua com a *orquestrador* en un sentit de "pont"redireccional. En el marc del nostre projecte, que s'inclou a SUPERSEDE, aquest component actuarà com a punt entre el subsistema d'adaptació i el sistema de monitoratge

  - ***Input***. Informació/paràmetres de configuració d'un procés de monitoratge per a un monitor específic.
  - ***Action***. Procés de la informació i transformació de la mateixa d'acord amb el propòsit del nostre sistema (configuració de monitors).
  - ***Output***. Informació/paràmetres de configuració transformats i orientats al sistema de monitoratge.

D'aquesta manera, el nostre sistema de monitoratge presenta 3 subcomponents independents que integren l'activitat de monitoratge mitjançant la comunicació entre ells: el sistema rep, a través de l'Orchestrator, peticions d'accions sobre els processos de monitoratge dels monitors. Aquest Orchestrator processa aquesta petició, i la redirecciona al Monitor Manager, qui coneix i controla els diferents monitors que hi ha al nostre sistema. El Monitor Manager s'encarrega d'analitzar la informació, processar-la d'acord al monitor al qual s'ha de redireccionar, i finalment enviar l'ordre de configuració al monitor corresponent. Amb aquesta informació ja processada per tal que el monitor concret pugui entendre-la, aquest adapta (és a dir, reconfigura) un procés de monitoratge existent, o bé en crea un de nou o n'elimina un d'existent, i procedeix amb el procés de monitoratge d'acord amb l'acció realitzada.

Definit els components del sistema de monitoratge, procedim a exposar els components i la interacció del **sistema d'adaptabilitat**:

- **Model Repository.** Aquest component s'encarrega de gestionar la persistència (lectura i escriptura) dels diferents models UML que defineixen les configuracions del nostre sistema de monitoratge. Els detalls d'aquests models UML, la seva sintaxi i el seu ús es descriuran més endavant.
  - ***Input***. Peticions de lectura i escriptura dels models UML.
  - ***Action***. Accions pertinents sobre els models UML.
  - ***Output***. Retorna els models UML demanats d'acord amb la petició o modificació pertinent.
- **Model Adapter.** Component que s'encarrega de realitzar adaptacions sobre els models UML que defineixen l'estat actual de les configuracions dels monitors d'acord amb les peticions d'adaptació que se li apliquen. Aquestes adaptacions sobre els diagrames definits s'apliquen en aquest component de forma aïllada, de manera que la resta del sistema no necessita tenir coneixement del procés tècnic de transformació dinàmica de models UML.
  - ***Input***. Petició de modificació d'un model de configuració amb els models i paràmetres pertinents.
  - ***Action***. Modificació dinàmica del model UML d'acord amb la petició
  - ***Output***. Model UML transformat.
- **Adapter.** És l'encarregat de realitzar l'adaptació del model des d'un punt de vista d'abstracció tècnica, centrant-se en la part semàntica de l'adaptació dels monitors. Mitjançant els models que defineixen el sistema i les possible millores, aquest component estudia i computa de forma automàtica reconfiguracions dels monitors.
  - ***Input***. Lectura dels models del Model Repository per computar la petició d'adaptació de monitors.
  - ***Action***. Analitza els models i les possibles adaptacions per computar modificacions sobre els models de configuració, i demana aquesta modificació al Model Adapter.
  - ***Output***. Model/s de configuració adaptats.

En definitiva, el sistema d'adaptabilitat defineix un *workflow* basat en una petició de reconfiguració a l'Adapter que, mitjançant l'anàlisi dels models que defineixen les configuracions dels monitors (configuracions actuals, propostes de noves configuracions, etc.) computa una modificació real sobre la configuració actual.

En aquest punt, necessitem un últim component que actui de pont entre el sistema d'adaptabilitat i el sistema de monitoratge, de tal manera que les adaptacions realitzades en el sistema d'adaptabilitat sobre els models UML que defineixen l'estat del sistema de monitoratge s'apliquin a aquest darrer. En aquest sentit, s'introdueix el component **Enactor.**

- **Enactor.** Component d'integració entre les adaptacions generades pel sistema i el sistema de monitoratge. Concretament, actua de pont entre l'Adapter, encarregat de gestionar aquestes reconfiguracions, i l'Orchestrator, component del sistema genèric a SUPERSEDE encarregat de gestionar totes les peticions d'adaptabilitat de components software. La seva tasca principal és afegir una capa d'abstracció entre els dos subsistemes, per evitar que aquests hagin de conèixer de l'activitat de l'altre.

- ***Input***. Model UML adaptat generat per l'Adapter amb una nova proposta de configuració del sistema.
- ***Action***. Transformació del model UML en format processable per l'Orchestrator.
- ***Output***. Petició de reconfiguració d'un monitor específic que envia a l'Orchestrator.

A termes genèrics, i sense entrar encara en detalls de disseny intern de cadascun d'aquests components, tenim una proposta inicial genèrica que defineix com ha de ser el nostre sistema, quins components l'han de formar i com s'han de relacionar entre ells per satisfer l'objectiu genèric d'aquest projecte.

Com a punt addicional, es proposa també el disseny d'un **dashboard** basat en una aplicació web senzilla, a través de la qual puguem visualitzar algunes de les dades de les adaptacions generades pel nostre sistema, amb l'objectiu de facilitar el control de l'activitat del sistema i la validació del mateix per una possible demostració.

5.2 Integració de components

La interacció entre els diferents components del sistema haurà de ser un dels elements a tractar en el desenvolupament del projecte. Un dels objectius principals és garantir el màxim desacoblament entre cadascuna d'aquestes interaccions, de tal manera que cadascun dels components puguin ser reaprofitats de forma independentment, i que a més la major part de modificacions en aquests components no afectin a la resta, com a mínim en termes d'interacció.

Per facilitar aquesta integració, el projecte SUPERSEDE ofereix una plataforma anomenada *Integrated Framework* (IF), desenvolupada per un *partner* del projecte. El seu objectiu és oferir una integració de tots els components desplegats al *back-end* del sistema. A nivell tècnic, aquest component ofereix una llibreria amb un conjunt de *proxies* implementats, a través dels quals els diferents components es poden comunicar amb altres components desplegats i afegits al IF.

Per permetre aquesta integració, l'únic requisit que planteja aquesta plataforma és l'exposició dels diferents components com a serveis web RESTful. D'aquesta manera, IF actua com a pont directe sense necessitat de formatar o "mapejar"els paràmetres d'entrada i sortida d'aquestes interaccions, ja que aquest component s'encarrega de fer les transformacions pertinents d'acord amb les necessitats d'interacció.

Veurem més endavant com aquest sistema aprofita aquest framework per facilitar la comunicació i evitar mapejats d'entrada/sortida. Amb aquest objectiu, serà necessari exposar alguns dels components com a serveis web.

6 Entorn de desenvolupament

El desenvolupament del sistema proposat requereix la integració d'un conjunt de subcomponents independents que, tot i comunicar-se entre ells, presenten una sèrie de característiques tècniques variades. Els requisits de desenvolupament i els entorns sobre els quals aquests components s'han de desenvolupar dependran de la naturalesa i els objectius de cadascun d'aquests. I, en termes genèrics, la gestió del sistema requerirà l'ús d'eines que ens facilitin aquest comportament.

Com a punt de partida al desenvolupament i exposició dels diversos components i tecnologies utilitzades, plantegem les tecnologies i elements bàsics que formen part del desenvolupament del projecte per, a partir d'aquí i al llarg dels següents capítols, exposar les tecnologies (llibreries, frameworks, etc.) que integrarem a aquestes per assolir els objectius.

6.1 Tecnologies utilitzades

Procedim a identificar les tecnologies bàsiques, amb les seves versions corresponents:

- **Git / GitHub [13].** Com a software de control de gestions i repositori s'utilitza **git** i **GitHub**, respectivament. La completesa i maduresa de git en la seva actualitat, així com les funcionalitats oferides per GitHub i la comoditat de la seva gestió, les fan candidates ideals per a realitzar el desenvolupament del projecte.
- **Java 8 [14].** Pràcticament la totalitat del projecte i els seus components s'han implementat utilitzant llenguatge Java. Concretament, la darrera versió Java 8, degut a les millores i la correcció d'alguns bugs que suposa respecte la seva anterior versió, Java 7.
- **Eclipse IDE for Java Developers (Neon 4.6.2) [15].** IDE i versió corresponents utilitzats pel desenvolupament dels components. S'ha considerat com l'opció ideal per una banda, per facilitar la compatibilitat i integració amb el projecte SUPERSEDE i els altres components, i per altra banda per la senzillesa i la integració de diferents plug-ins i eines que faciliten el desenvolupament.
- **Eclipse Modeling Tools (Neon 4.6.2) [16].** IDE complementari al desenvolupament orientat al desenvolupament de projectes de creació i edició de models UML mitjançant l'ús de tecnologies associades que es detallaran més endavant.
- **Gradle 2.13 [17].** Davant la necessitat de gestionar les dependències i la compilació dels diferents components, s'ha triat Gradle com a opció preferent.
- **Spring Framework 4.3.9 [18].** Framework popularment conegut orientat al desenvolupament d'aplicacions basades en Java. L'utilitzarem especialment orientat a:

  - Disseny i implementació de serveis web RESTful
  - Desenvolupament d'aplicacions web senzilles
  - Gestió de persistència d'aplicacions

- **UML2 5.0.0 [19].** Llibreria que encapsula la implementació per la gestió dinàmica mitjançant la plataforma Eclipse de models UML. L'utilitzarem per la lectura i escriptura dels models del nostre sistema de monitoratge.

- **Papyrus 2.0.2 [20].** Framework que ofereix un entorn de modelació gràfic dels models UML, els seus components i les seves propietats. Especialment útil per la visualització de models i el seu tractament per facilitat la seva lectura i anàlisi.

- **MySQL 5.6 [21].** Sistema de gestió de base de dades relacional que utilitzarem pel desenvolupament i gestió de les bases de dades implementades per aquest projecte.

6.2 Implementació i artefactes generats

Tot el codi dels components implementats per aquest projecte (així com aquells implementats per tercers dins SUPERSEDE i utilitzats pel desenvolupament d'aquest) es troba al client web GitHub. A continuació es poden consultar els enllaços on figura tot el codi de cadascun d'aquests components, tant per la seva documentació com per obtenir i desplegar algun d'aquests per separat.

Sistema de monitoratge

Directori arrel
https://github.com/supersede-project/monitor_feedback/tree/develop_quim-motger/

- **Monitor genèric**
  /monitoring

- **Twitter Monitor**
  /monitors/twitter

- **Google Play Monitor**
  /monitors/googlePlay

- **App Store Monitor**
  /monitors/appStore

- **Monitor Manager**
  /monitormanager

- **Orchestrator**
  /orchestrator

Sistema d'adaptabilitat

Directori arrel
https://github.com/supersede-project/dyn_adapt/tree/develop/Enactment/

- **Model Repository**
  /components/adapter/model/eu.supersede.dynadapt.modelrepository.manager
  /components/adapter/model/eu.supersede.dynadapt.modelrepository

- **Adapter**
  /components/adapter/model/eu.supersede.dynadapt.adapter

- **Model Adapter**
  /components/adapter/model/eu.supersede.dynadapt.modeladapter

- **Dashboard**
  /components/adapter/model/adaptation-dashboard

7 Sistema de monitoratge

En aquest punt tenim la base necessària per procedir a exposar el treball realitzat des d'un punt de vista de disseny software i implementació dels diferents components. Agafant com a referència la solució proposada al *Capítol 5. Visió general del sistema*, procedirem a desenvolupar els detalls tècnics de cadascun dels components que integren aquest sistema. Començarem per explicar els detalls relacionats amb el disseny i la implementació dels monitors.

7.1 Monitors

Recordem que, dins el nostre context, un **monitor** consisteix en un **component software** autònom amb una activitat regular orientada al control de qualitat d'un altre sistema software. Aquest control de qualitat es basa en una col·lecció de dades obtingudes a través d'aquest segon sistema, que ens aporten informació pròpia del context monitorat. Amb aquestes dades, un sistema capacitat per processar i analitzar aquestes dades, es poden generar suggerències de modificacions. Aquesta darrera part, però, queda fora de l'abast del nostre projecte, que centrarà l'activitat dels monitors en la seva tasca principal: la col·lecció de dades sota una sèrie de criteris específics.

En general, per tant, volem que el nostre sistema disposi d'un conjunt de monitors heterogenis (i, per tant, de naturalesa i comportament diferents), que puguin ser capaços de gestionar **processos de monitoratge** de forma paral·lela. És a dir: cadascun d'aquests monitors ha de ser capaç d'inicialitzar processos de monitoratge que s'executin en paral·lel i en segon pla, col·lectin dades d'acord als criteris de cadascun d'aquests processos, i les redireccionin a un tercer component software, encarregat del seu anàlisi.Per tant, necessitem que cadascun d'aquests monitors satisfaci els següents requisits funcionals:

1. **Inicialització de procés de monitoratge.** El monitor ha de poder rebre una petició per inicialitzar un nou procés de monitoratge amb una sèrie de paràmetres de configuració que defineixin aquest procés de monitoratge.

2. **Modificació de procés de monitoratge.** Donat un procés de monitoratge ja existent, el sistema ha de permetre la seva reconfiguració. És a dir: el sistema ha de permetre modificar els paràmetres d'aquest procés i, en conseqüència, alterar el comportament del procés (d'acord amb els criteris que es presenten a continuació.

3. **Aturada de procés de monitoratge.** Donat un procés de monitoratge ja existent, el sistema ha de permetre la seva aturada. Davant aquesta petició, el procés s'atura, i per tant es deixen de recol·lectar dades sota aquells criteris de monitoratge.

7.1.1 Especificacions tècniques i funcionals

Tal i com establiem com a objectiu principal del projecte, aquest sistema de monitoratge ha de ser **heterogeni**. La conseqüència principal d'aquesta característica és que necessitem definir un sistema que contempli que cadascun d'aquests requisits funcionals es garanteixen en la integració dels monitors implementats, i que per tant esdevenen casos d'ús complets i satisfactoris del nostre context. Per fer-ho, i donat que els monitors seran el principal punt de variabilitat del nostre sistema, hem d'afegir un cert nivell d'abstracció, un **desacoblament** entre els detalls específics de cadascun dels monitors, que ens resulten indiferents per la resta del sistema.

Per tant, el primer pas que hem de realitzar és **dissenyar una arquitectura** i uns **criteris de configuració genèrics** que satisfacin dos criteris: primerament, que ens permetin integrar tots els monitors implementats sota aquesta proposta al nostre sistema; i en segon lloc, que permetin una independència suficient com per garantir el criteri d'heterogeneïtat dels monitors. Desenvoluparem aquesta proposta analitzant els següents punts:

1. **Comportament intern del monitor.** Anàlisi de les necessitats i detalls tècnics del funcionament intern del procés de monitoratge.
2. **Redirecció de dades col·lectades.** Especificacions tècniques del mètode de gestió i redirecció de dades.
3. **Configuració dels monitors.** D'acord amb les necessitats anteriors, descriure quins paràmetres necessitarem per configurar els monitors.

Comportament intern del monitor

La implementació d'un monitor representa, des d'un punt de vista semàntic, un component software encarregat de monitorar un component software concret. En aquest context específic, entendrem **monitorar** com la col·lecció periòdica d'un conjunt de dades produïdes de l'execució del sistema software monitorat. En base a aquesta definició, entendrem com a **procés de monitoratge** el cicle següent:

1. Inicialització de les estructures de col·lecció de dades
2. Captura de dades durant el transcurs d'un període de temps determinat
3. Enviament de les dades a un tercer component software

Així, de forma periòdica, un procés de monitoratge recull durant un període de temps (o ***time slot***) específic totes les dades que el monitor ha estat configurat per recollir. Per tal que els monitors es puguin explotar al màxim, és imprescindible que aquests permetin l'**execució en paral·lel** de diversos processos de monitoratge, amb possibles diferències en els seus paràmetres de configuració (p.e., aquest *time slot*).

Davant aquesta proposta, el comportament i potencial d'un monitor queda molt limitat, ja que elements com p.e. el mètode de recollida de dades, o fins i tot les dades recollides, queden molt limitats. En general, és molt possible que un sistema software pugui ser monitorat mitjançant diferents tècniques, com per exemple l'ús d'APIs, llibreries o components externs, etc. Per aquesta raó, si volem permetre que el nostre monitor ofereixi flexibilitat en aquest aspecte, hem de permetre l'ús de diferents eines, o ***tools***, que aquest monitor pot utilitzar indistintament per executar

els processos de monitoratge.

La integració de diverses *tools* dins un monitor ens permeten no només una variabilitat en l'execució de la col·lecció de dades, sinó també una major fiabilitat i qualitat del monitor com a component software. Ens permet reaccionar, entre d'altres, davant escenaris on l'ús d'un sistema de monitoratge específic deixa de funcionar (p.e. una API que no dona resposta), ja que davant la detecció d'aquest error el canvi de *tool* utilitzada ens permet que el monitor no deixi de ser usable.

En resum, necessitem que el monitor sigui capaç de gestionar un nombre indefinit de processos de monitoratge en paral·lel, amb configuracions diferents, i que utilitzin el conjunt de *tools* implementades.

Redirecció de dades

Un dels objectius de les especificacions tècniques dels monitors és permetre la seva integració, en primer lloc, dins el context del nostre projecte, i en segon lloc, a sistemes tercers que permetin l'anàlisi de les dades recollides durant la seva activitat de monitoratge. D'aquesta manera augmentem el valor propi dels components dissenyats en el projecte, reutilitzable en contexts diferents al plantejat. Per aquesta raó, i per completar l'activitat del monitor, hem de contemplar com dissenyar l'enviament i redirecció de les dades que cada monitor recull i formata durant la seva activitat.

Per facilitar aquest aspecte, i permetre també la seva integració dins el sistema general de SUPERSEDE, els monitors integraran la implementació d'enviament de les seves dades a través d'**Apache Kafka** [22]. Es tracta d'una plataforma distribuïda de *streaming* que ofereix la possibilitat de crear i configurar *pipelines* de dades en temps real que actuen com a canal de comunicació entre diverses aplicacions o components software. L'arquitectura és senzilla: un component software, anomenat ***producer***, es comunica amb el servidor Kafka i envia dades de forma periòdica a un *pipeline* específic d'aquest servidor, prèviament configurat i identificament amb el que anomenem Kafka *topic*, un identificador únic d'aquell *pipeline* per aquell desplegament de Kafka. Aquest flux de dades s'encua al servidor, i es redireccionen a uns altres sistemes o aplicacions, anomenats ***consumers***, que reben i processen les dades d'un pipeline específic a mesura que es van enviant i processant.

En general, l'avantatge principal de Kafka i la justificació del seu ús pel nostre context és que permet una integració còmode i fiable entre diferents aplicacions que necessiten comunicar dades de forma periòdica, garantint la seva arribada. Kafka ofereix una sèrie d'APIs per configurar els *producers* i *consumers*, així com una configuració relativament senzilla del propi servidor. Gràcies a aquestes característiques, podem aprofitar els propis monitors per actuar com a *producers* d'un *stream* de dades, que es correspondrà amb les dades monitorades durant els processos d'execució, i distribuïr-les als diferents *pipelines* o Kafka *topics*. Així, davant possibles ampliacions i expansions d'aquest projecte, podem fàcilment incorporar components d'anàlisi gràcies al desacoblament entre la lògica interna del monitor i l'enviament i captura de dades que l'arquitectura de Kafka ens ofereix.

La lògica interna genèrica proposada per la implementació dels monitors serà, per tant, l'ús de l'API de *producer* de Kafka per part dels monitors, pel qual cada procés de monitoratge enviarà de forma periòdica dades a un servidor Kafka específic,

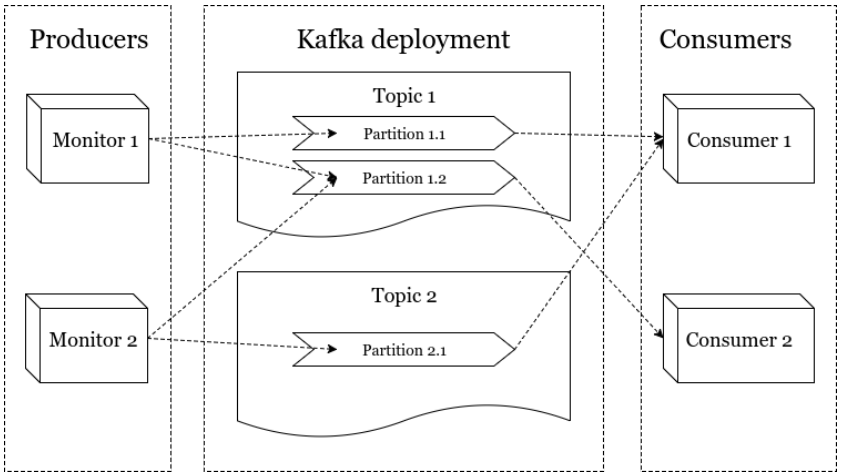

FIGURA 7.1: Exemple d'arquitectura de la comunicació amb Kafka

o Kafka *endpoint*, i dins aquest desplegament, a un *pipeline* o Kafka *topic* específic. A la figura 7.1 s'exposa com funciona aquesta comunicació entre els monitors i els *consumers*, components que poden ser de qualsevol tipus sempre i quant implementin la lògica de *consumers* pròpia e Kafka.

Paràmetres de configuració

Davant les especificacions anteriors, i per garantir el màxim nivell de personalització i configuració dels monitors, cadascun dels processos de monitoratge actius en un monitor ha de permetre definir els paràmetres relacionats amb les possibles variacions i diferències entre aquests processos, tant en temps de creació com durant la seva reconfiguració. En aquest sentit, es proposen els següents paràmetres com a genèrics per a totes les implementacions de monitors:

- ***Time slot***. Expressat en segons, indica la durada de cada període de monitoratge de dades (és a dir, temps que transcorre cada vegada que s'envia un nou *stream* de dades).
- ***Tool name***. Nom de l'eina (*tool*) utilitzada per aquell procés de monitoratge, i que per tant implica la col·lecció d'unes dades específiques utilitzant una tècnica específica.
- ***Kafka endpoint***. Adreça que apunta al servidor on es troba desplegat el sistema Kafka on s'han d'enviar les dades generades, ja sigui *localhost* o URL pública.
- ***Kafka topic***. Identifica el *pipeline* de dades del servidor Kafka on el monitor (*producer* dins el context de Kafka) ha d'enviar les dades.

És possible, tal i com veurem més endavant, que alguns monitors requereixin de paràmetres de configuració addicionals, propis del funcionament intern específic del monitor. Per aquest motiu, a banda de proporcionar una configuració bàsica pels monitors amb els paràmetres anteriors, cal permetre també una extensibilitat en quant a paràmetres de configuració.

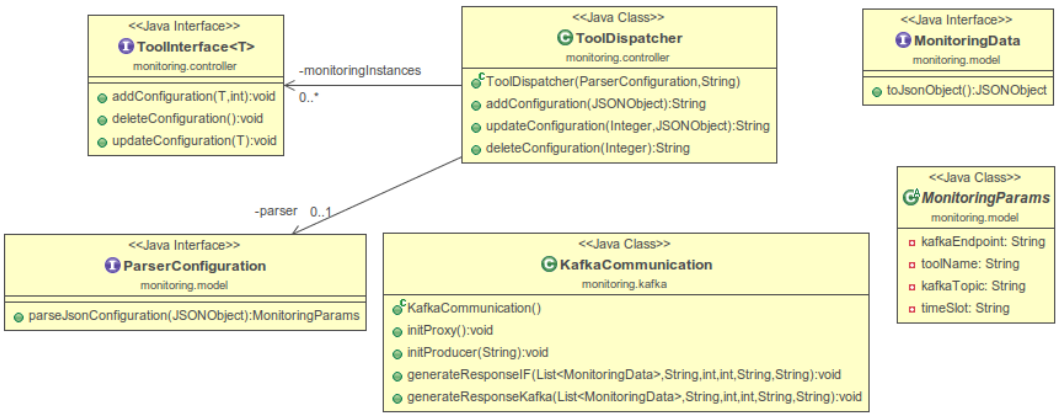

FIGURA 7.2: Arquitectura software genèrica d'un monitor

7.1.2 Disseny i arquitectura genèrica

Es proposa l'arquitectura definida a la figura 7.2 a extendre per cadascuna de les implementacions de monitors. Per entendre aquesta arquitectura, a continuació s'expliquen cadascun dels elements integrats:

- **MonitoringParams**. Classe abstracta que cada monitor ha d'implementar que conté, de base, els paràmetres de configuració dels monitors genèrics per tots aquests. Addicionalment, cada implementació de monitor pot afegir els paràmetres i la lògica associada a aquests que consideri oportuns.
- **ParserConfiguration**. Interfície que cada monitor ha d'implementar i que defineix un mètode per transformar un objecte JSON en una instància de la classe *MonitoringParams* implementada pel propi monitor. L'objectiu és permetre així que el monitor pugui processar JSON com a format de comunicació estàndar per les peticions de configuracions, facilitant el seu ús desplegat com a servei web (especialment útil per l'Integrated Framework dins el context SUPERSEDE, tal i com s'explica al *Capítol 5. Visió general del sistema*).
- **ToolInterface**. Interfície parametritzada amb una especialització de la classe *MonitoringParams* que defineix una instància de procés de monitoratge per una *tool* específica. Defineix els següents mètodes:
  - ***addConfiguration(T)*** -> inicialitza el procés de monitoratge amb els paràmetres definits per la instància de T (subclasse de MonitoringParams)
  - ***deleteConfiguration()*** -> atura el procés de monitoratge i elimina la instància de la *tool*
  - ***updateConfiguration(T)*** -> actualitza els paràmetres de configuració del procés iniciat en segon pla per la instància de la *tool*
- **ToolDispatcher**. Classe que actua com a controlador del monitor, rebent totes les peticions relacionades amb els processos de monitoratge i gestionant les diferents instàncies en execució. Defineix un *ParserConfiguration* per processar la traducció de JSON (format estàndar) a *MonitoringParams* pels següents mètodes:

  - ***addConfiguration(JSONObject)*** -> processa els paràmetres definits al JSONObject i inicialitza una instància de la *tool* corresponent amb els paràmetres associats
  - ***deleteConfiguration(int)*** -> atura el procés de monitoratge identificat per l'id proporcionat
  - ***updateConfiguration(JSONObject, int)*** -> actualitza els paràmetres de configuració definits al JSONObject del procés de monitoratge identificat per int

  Per tal de gestionar la *tool* utilitzada en cada procés de monitoratge (que representarà una nova instància de la implementació de la interfície *ToolInterface*) utilitzant el paràmetre *toolName* de configuració, s'utilitza el patró *reflection* [23]. Aquest patró de disseny software es caracteritza per la modificació en temps d'execució del comportament d'un sistema; i, en el nostre cas, ens interessa per permetre la instanciació de les diferents *tools* utilitzant el seu nom sense necessitat de coneixe'l. Mitjançant l'ús del nom de la tool, i coneixent el *package* on es troben implementades les tools, podem instanciar aquella tool partint únicament del nom, sense necessitat de cap altre tipus de context.

- **MonitoringData**. Interfície que cada monitor implementa amb les dades i format que el monitor genera fruït de la seva activitat, amb la implementació d'un mètode *toJsonObject()* per definir un format genèric de les dades a enviar

- **KafkaCommunication**. Classe que implementa la comunicació amb el servidor de Kafka, i que permet enviar les dades generades per l'activitat de monitoratge. Implementa 4 mètodes segons les dues lògiques de comunicació possibles: la integrada al sistema SUPERSEDE (utilitzant *proxies* de IF), i una configuració personalitzable a un Kafka endpoint específic:

  - ***initProxy()*** -> inicialitza la instància de *proxy* que permet la comunicació amb el Kafka *server* desplegat a IF
  - ***generateResponseIF(List<MonitoringData>)*** -> envia el llistat d'instàncies de dades monitorades a través del *proxy* inicialitzat
  - ***initProducer(String)*** -> inicialitza un *producer* de Kafka que es comunica amb l'*endpoint* especificat
  - ***generateResponseKafka(List<MonitoringData>)*** -> envia el llistat d'instàncies de dades monitorades a través del *producer* inicialitzat

Aquesta és per tant la proposta d'una arquitectura genèrica per la implementació de cadascun dels monitors a integrar en el nostre sistema. Les especificacions tècniques tant des d'un punt de vista de requisits funcionals com requisits arquitectònics o de qualitat queden garantides, així com la seva extensibilitat d'acord amb les característiques específiques necessàries de cada monitor. La proposta genèrica s'implementa en un projecte del qual les implementacions de monitors específics han d'estendre com a subprojecte, utilitzant les funcionalitats de Gradle a tal efecte.

7.1.3 Implementació de monitors

Un cop exposada l'arquitectura i especificacions genèriques dels monitors, el següent pas és procedir a la implementació dels diferents monitors que integrarem al nostre sistema. Aquestes implementacions esdevindran possibles casos d'ús a

executar, però únicament suposen exemples dins d'un ample ventall de possibilitats d'implementació.

En aquest projecte presentem la implementació de 3 monitors classificats en 2 tipus de monitors diferents: **monitors de xarxes socials** i **monitors de botigues d'aplicacions**. Pel primer tipus, es proposa un monitor de la xarxa social **Twitter**. Pel segon tipus, es proposen dos monitors: un monitor de **Google Play** i un altre de l'**App Store**.

Twitter Monitor

L'objectiu d'aquest monitor és supervisar i recol·lectar els tuits publicats a la xarxa social de Twitter durant un període de temps concret (definit al procés de monitoratge) que compleixen una sèrie de característiques específiques, d'acord amb els criteris que poden resultar d'interès en quant a la informació d'aquests tuits.

Per permetre aquesta activitat de monitoratge, i tal i com procedirem amb cadascun dels monitors, necessitem implementar com a mínim una *tool* amb la qual realitzar el procés de monitoratge. Definirem una *tool* anomenada **TwitterAPI** que utilitzarà **twitter4j** [24], una llibreria lliure no oficial de Java que permet utilitzar una arquitectura ja definida per connectar-se a les APIs de Twitter, i entre d'altres a la **Stream API** [25], una API que permet obrir *streams* de dades per capturar i processar tots els tuits publicats en temps real que compleixin una sèrie de característiques específiques. Aquests *streams* s'executaran com a *threads* en segon pla i en paral·lel d'acord amb el nº de processos de monitoratge oberts.

Pel nostre cas, definirem dos paràmetres per filtrar els tuits monitorats: l'**autor** del tuit i l'aparició d'un **conjunt de paraules** específic al contingut del tuit. Per configurar cadascun dels processos de monitoratge d'acord a aquests dos criteris, caldrà afegir els següents paràmetres:

- **accounts** - llistat amb els identificadors dels autors dels quals volem obtenir els tuits
- **keywordExpression** - expressió booleana formada per combinacions AND, OR i NOT de diferents *keywords* que el contingut del tuit ha de satisfer per ser monitorat

La *Stream API* ofereix paràmetres de configuració per realitzar el *tracking* dels tuits que satisfan ambdues condicions. En el cas del primer paràmetre *accounts* no cal realitzar cap transformació, ja que únicament necessitem els noms únics de les comptes dels autors per obtenir els seus tuits. Contràriament, l'API únicament ofereix l'oportunitat de filtrar per combinacions de paraules expressades en la seva *forma normal disjuntiva*, o **FND**. És a dir, expressions del format:

$$X_1\ OR\ X_2\ OR\ ..\ OR\ X_n$$

on X_n és una expressió booleana formada per una combinació indefinida d'operands AND:

$$keyword_1\ AND\ keyword_2\ AND\ ..\ AND\ keyword_n$$

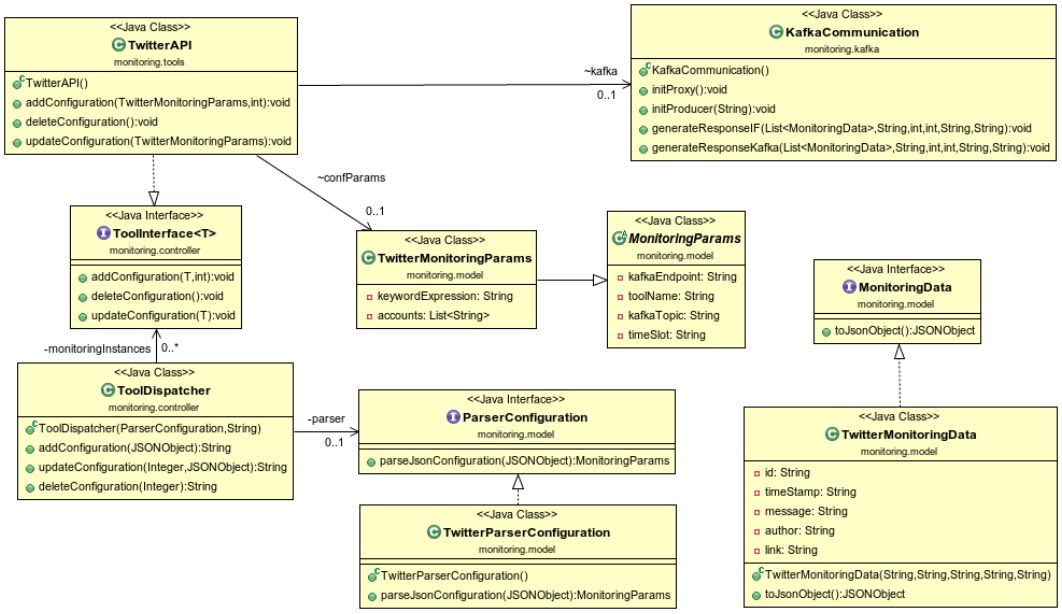

FIGURA 7.3: Arquitectura software del monitor de Twitter

Davant aquest fet, i per evitar forçar un format específic del paràmetre de configuració d'entrada, necessitem implementar una lògica interna pròpia de la *tool* TwitterAPI que transformi qualsevol expressió booleana en la seva expressió FND.

Donats aquests detalls podem utilitzar l'arquitectura genèrica proposada anteriorment per extendre la implementació del monitor de Twitter. Aquesta arquitectura i els components implementats es troben definits a la figura 7.3, on podem veure aquells components reutilitzats de l'arquitectura genèrica, per poder fer una comparativa ràpida sobre les classes i components que ha calgut implementar per poder definir un monitor:

- **TwitterMonitoringParams.** Subclasse de la classe abstracta *MonitoringParams* que hereta per una banda els atributs comuns a tots els monitors (*kafkaEndpoint, toolName, kafkaTopic* i *timeSlot*) i per altra banda en defineix dos nous: la *keywordExpression* i un llistat d'*accounts*.

- **TwitterParserConfiguration.** Implementació de la interfície que defineix la transformació del format d'entrada del monitor (JSONObject) a la instància de *TwitterMonitoringParams* amb la qual el monitor (i en conseqüència la seva *tool*) treballarà.

- **TwitterMonitoringData.** Implementació de la interfície que defineix la transformació de les dades recollides durant un cicle del procés de monitoratge d'una *tool* al format de sortida del monitor (JSONObject).

- **TwitterAPI.** Implementació de la interfície que defineix la lògica de les tres operacions de qualsevol *tool*: iniciar una nova configuració, modificar-ne una d'existent, i eliminar-ne una d'existent. En aquest cas disposem d'una única implementació, però tal i com veurem més endavant, l'arquitectura permet sense problema generar un conjunt de *tools* diferenciades, cadascuna amb les seves característiques i dades internes, que utilitzaran les mateixes implementacions de formats de dades definides anteriorment. Aquest component serà

l'encarregat d'utilitzar *twitter4j* per configurar la crida a la Stream API i obrir el *thread* encarregat d'anar rebent i emmagatzemant els tuits rebut. Addicionalment, serà també responsabilitat d'aquesta tool comunicar-se amb *KafkaCommunication* per utilitzar el mètode de comunicació (a través de IF o personalitzat) que correspongui, d'acord amb les necessitats de la *tool*.

A la figura 7.4 es pot observar un exemple de l'objecte JSON generat a partir de la transformació definida per la implementació de la interfície *MonitoringData*. Aquest objecte serà enviat, a través de la *tool* i utilitzant la classe *KafkaCommunication*, al Kafka *endpoint* i Kafka *topic* definits a la configuració del monitoratge. De les dades generades per la comunicació amb l'API, recollirem: *idItem* (identificador del tuit), *timeStamp* (moment en que s'ha realitzat el tuit), *message* (contingut del mateix), *author* (l'identificador de l'autor), i *link* (enllaç al tuit a la plataforma Twitter).


```
{
    "SocialNetworksMonitoredData": {
        "idOutput": "12345",
        "confId": "67890",
        "searchTimeStamp": "2017-05-01 17:23:00.000",
        "numDataItems": 1,
        "DataItems": [
            {
                "idItem": "6253282",
                "timeStamp": "2017-05-01 17:22",
                "message": "Game on. Big ten network in 10 mins. Hoop for water. Flint we got ya back",
                "author": "@SnoopDogg",
                "link": "https://twitter.com/SnoopDogg/status/734894106967703552"
            }
        ]
    }
}
```


FIGURA 7.4: Exemple dades de sortida generades pel monitor de Twitter

Cal destacar que gràcies al disseny de l'arquitectura proposada anteriorment per la implementació d'un monitor integrat al nostre sistema (o, de fet, sense necessitat que l'objectiu final sigui la seva integració) ha requerit una extensió relativament senzilla. Únicament ha calgut implementar aquells aspectes propis de cada monitor, que en són 4: els paràmetres específics de configuració (1), el mapejat dels paràmetres d'entrada (2), el mapejat de les dades de sortida (3), i el funcionament de les *tools* (en aquest cas, twitter4j) per realitzar els processos de monitoratge (4). D'aquesta manera satisfem un dels principals objectius: dotar al nostre sistema de la màxima extensibilitat i heterogeneïtat possible, amb la possibilitat de personalitzar l'activitat i semàntica de cada monitor partint d'una arquitectura comuna.

Exposició com a servei REST

El següent pas d'acord amb les necessitats tècniques per la integració és exposar el monitor i les seves funcionalitats com un servei web RESTful. D'aquesta manera, mitjançant la documentació d'una API per accedir a les diferents operacions d'alta, baixa i modificació d'un procés de monitoratge, podem integrar els monitors al IF del projecte SUPERSEDE i permetre així el seu accés a través de la plataforma d'integració. En aquest sentit l'ús de JSON com a format de comunicació d'entrada i sortida ens facilita la seva exposició com a servei web, que utilitzarà també el format JSON com a *payload* per fer les crides. La documentació de l'API del monitor de Twitter es pot trobar a l'apèndix **??**, on podem trobar en detall les peticions REST

implementades, així com el format dels *inputs* i *outputs* de cadascuna de les crides.

Per tal de poder concebre com funcionaria la comunicació amb el monitor per iniciar una nova instància de monitoratge, la figura 7.5 mostra un exemple d'objecte JSON utilitzat.

```
{
    "SocialNetworksMonitoringConfProf": {
        "toolName": "TwitterAPI",
        "timeSlot": "30",
        "kafkaEndpoint": "http://localhost:9092",
        "kafkaTopic": "tweeterMonitoring",
        "keywordExpression": "(tweet OR follow) AND (me)",
        "accounts": ["QuimMotger"]
    }
}
```

FIGURA 7.5: Exemple JSON de configuració del monitor de Twitter

En referència a aquesta possible petició, la figura 7.6 mostra un exemple de la resposta (en cas d'èxit en la configuració del monitor) que retorna aquest monitor per la petició anterior. En aquesta figura, definim 2 paràmetres de retorn. Primerament, `idConf`, que conté l'identificador de la nova instància de monitoratge creada (variable indispensable per tal que sigui usable i poder realitzar adaptacions posteriors). En segon lloc `status`, que indica si la petició s'ha realitzat amb èxit o s'ha produït algun error.

```
{
    "SocialNetworksMonitoringConfProfResult":{
        "idConf":"1",
        "status":"success"
    }
}
```

FIGURA 7.6: Exemple amb èxit del monitor de Twitter

Donat el cas que s'hagi produït algun error, el format de resposta anterior no és vàlid (no s'ha creat cap instància i, per tant, cap identificador pot ser produït) ni suficient (no ens aporta informació de quin ha estat l'error). En aquest cas, un exemple de resposta seria l'exposat a la figura 7.7.

```
{
    "SocialNetworksMonitoringConfProfResult":{
        "status":"error",
        "message": "Not a valid JSON configuration object"
    }
}
```

FIGURA 7.7: Exemple resposta amb error del monitor de Twitter

Google Play Monitor

Aquest segon monitor, encarregat del monitoratge de dades de la botiga d'aplicacions dels sistemes operatius Android, **Google Play**, pretén recollir les dades referents

a les crítiques o *reviews* que els usuaris de Google Play fan de les aplicacions que es descarreguen. De manera semblant al monitor de Twitter, però en un context i amb un objectiu diferent, recull dades dels usuaris i els missatges que publiquen al voltant d'un focus específic.

En aquest cas, i aprofitant que tenim definit un cas d'ús simple d'un monitor amb una sola *tool*, podem procedir a explotar l'arquitectura dels nostres monitors i definir un monitor que utilitzi més d'una *tool* per realitzar els processos de monitoratge. En aquest cas, s'ha fet una recerca a la xarxa sobre els diversos sistemes i components que existeixen (d'ús total o parcialment lliure) per realitzar el monitoratge, i s'han triat els dos següents degut a les seves característiques:

- **GooglePlayAPI**. *Tool* que implementa una comunicació en 2n pla, similar al sistema de *threads* emprat pel monitor de Twitter, amb la Google Play Developer API [26]. Diem "similar"degut al fet que aquesta API, que funciona mitjançant un sistema d'autenticació OAuth 2.0, no permet obrir un *stream* de dades com es presentava amb el monitor de Twitter, sino que permet obtenir, mitjançant crides API, les dades referents al conjunt total de *reviews* publicades per una aplicació determinada en el moment de la crida. D'aquesta manera no podem obtenir un *stream* de dades real de forma directa, sino que serà responsabilitat de la pròpia *tool* simular aquest *stream* de dades. Per fer-ho, realitzarem de forma periòdica (segons el *timeSlot* definit) crides a l'API per obtenir aquestes dades, i la *tool* s'encarregarà de recollir i retornar únicament les *reviews* realitzades durant el període de monitoratge. Com a avantatge principal, aquesta *tool* permet fer **fins a 60 crides** en 1 hora, un nombre molt elevat especialment si ho contrastem amb altres eines. Per contra, com a limitació únicament permet obtenir dades d'aplicacions de les quals l'usuari que s'ha autenticat n'és el propietari.
- **GooglePlay-AppTweak**. Aquesta eina permet obtenir als usuaris registrats un ample ventall de dades de les aplicacions publicades a GooglePlay, a través del servei AppTweak [27]. En aquest cas, l'autenticació està basada en crides amb *token*, i el sistema és similar a la *tool* de GooglePlayAPI: necessitem simular dins el comportament de la mateixa *tool* un *stream* de dades parsejant els resultats de la consulta de les *reviews* públiques fins al moment de la crida. En aquest cas, i en contraposició amb l'anterior *tool*, com a avantatge principal aquesta eina permet **obtenir informació de qualsevol app** publicada al mercat, independentment de la seva propietat. Per contra, la seva versió gratuïta (ofereix un servei de pagament) únicament permet realitzar 100 crides al mes. Una xifra que pot resultar baixa però que, considerant la naturalesa de l'entorn monitorat, podem considerar suficient en alguns casos, ja que equivaldria a 3 crides diàries.

La tria d'aquestes dues *tools* es basa en les seves característiques d'ús i limitacions, complementàries entre elles, que ens permeten configurar un monitor prou adaptable a les necessitats del procés de monitoratge. Addicionalment a les consideracions anteriors, les dades obtingudes per ambdues *tools* no són les mateixes (tot i que coincideixen en un alt percentatge), i per tal caldrà gestionar tal i com es comentarà més endavant aquesta irregularitat.

Per aquest cas d'ús, i degut al que hauria de ser l'escenari més habitual (monitorar en temps reals els comentaris que es realitzen sobre una aplicació específica),es

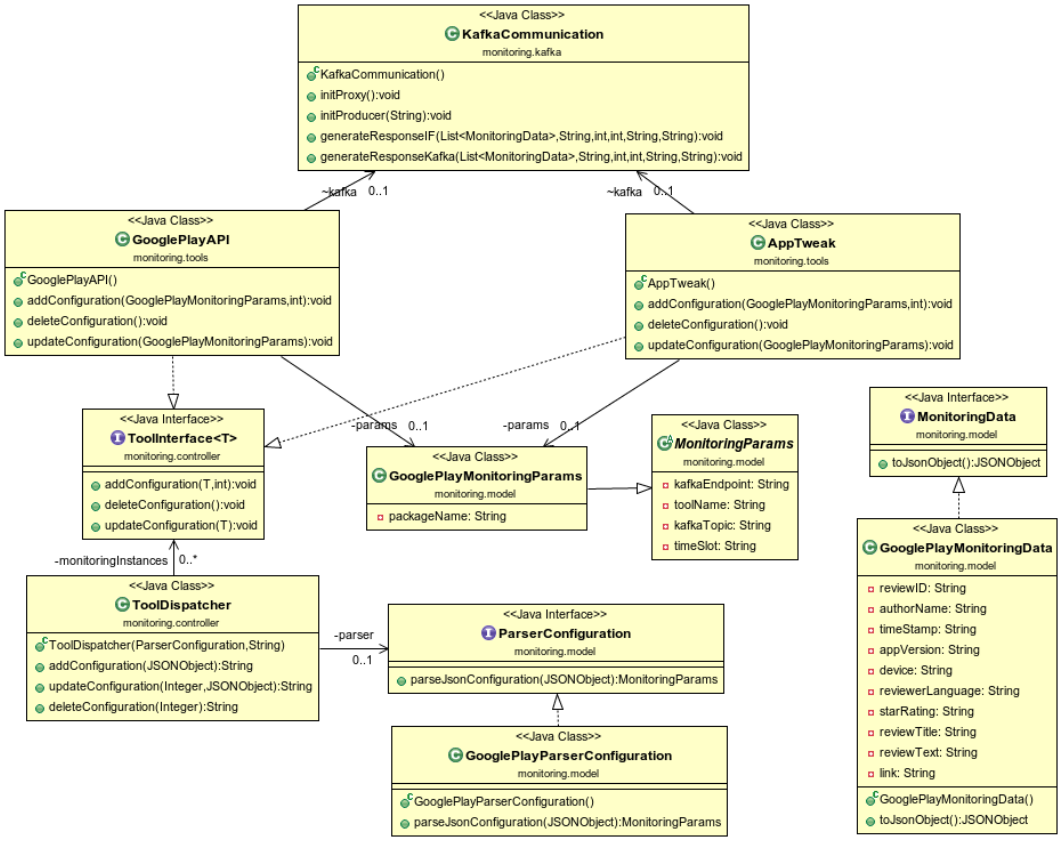

FIGURA 7.8: Arquitectura software del monitor de Google Play

defineix com a paràmetre de configuració la pròpia aplicació a monitorar, identificada pel **nom del *package***:

- **packageName** - identificador únic d'una aplicació publicada a Google Play

La implementació del monitor basada en l'arquitectura genèrica definida presenta diverses similituds en comparació amb el monitor de Twitter, amb la principal diferència que, per aquest cas, tenim 2 *tools* implementades a utilitzar.

- **GooglePlayMonitoringParams.** Subclasse de la classe abstracta *MonitoringParams* que hereta els atributs comuns a tots els monitors, i afegeix com a atribut particular del monitor un string *packageName*.
- **GooglePlayParserConfiguration.** Implementació de la interfície que defineix la transformació del format d'entrada del monitor (JSONObject) a la instància de *GooglePlayMonitoringParams*.
- **GooglePlayMonitoringData.** Implementació de la interfície que defineix la transformació de les dades recollides al format JSON de sortida. En aquest cas hem d'afegir la consideració prèviament introduïda sobre la variabilitat de les dades entre les dues *tools*. Davant aquest punt, podem explotar els avantatges de l'arquitectura presentada, que ens dona dues opcions:

  1. Podem implementar una única classe que estendrà la interfície *MonitoringData* i contindrà tots els paràmetres (comuns i no comuns) de les dades generades durant el procés de monitoratge. En aquest cas, la pròpia

tool crearà instàncies d'objectes monitorats amb les dades de les quals disposi, i aquelles que no pugui definir (i per tant, que prenguin valors buits o nuls) simplement no apareixeran en la transformació a objecte JSON que enviarem al Kafka *endpoint*. D'aquesta manera, amb una única classe satisfem les necessitats del monitor.

2. Alternativament podem modelar un conjunt de classes que implementin la interfície, de manera que cada *tool* utilitzi una d'aquestes implementacions. D'aquesta manera, disposaríem d'una implementació de *MonitoringData* per cada *tool* d'acord amb la informació que proporciona cadascuna d'aquestes.


```
"GooglePlayMonitoredData": {
    "idOutput": "12345",
    "confId": "67890",
    "searchTimeStamp": "2017-05-01 17:23:00.000",
    "numDataItems": 1,
    "DataItems": [
        {
            "reviewID": "5zvcx789f320r458",
            "authorName": "Mike Moreno",
            "timeStamp": "2017-05-01 17:22",
            "appVersion": "5.4",
            "device": "Nexus4",
            "reviewerLanguage": "en",
            "starRating": "5",
            "reviewTitle": "Just 5 minutes",
            "reviewText": " I always say I'll play for 5 minutes which "
            + "turns into 30 then an hour until my battery goes dead. "
            + "Very addictive indeed.",
            "link": "https://play.google.com/store/apps/details?id="
            + "com.rovio.angrybirds&reviewId=Z3A6QU9xcFRPSEI2LUNKWVl5UV"
            + "JaZVpfVkF5bHBXNG5VWV9qWFY5WlpWVFFHejY2LVBrZEh0Qm9odV9zSUh
            + "0ZWpEb0ZLcU11cVU2YnBIYmQ5QlVJVUMzU1E&hl=en"
        }
    ]
}
```


FIGURA 7.9: Exemple dades de sortida generades pel monitor de Google Play

Gràcies al disseny, cada desenvolupador podrà utilitzar l'opció que m'es s'adeqüi a les seves necessitats. La 1a opció pot resultar adequada quan p.e. el subconjunt de dades que volem obtenir sigui independent de la *tool*, i per contra la 2a opció ens farà servei per desacoblar totalment les dades entre diferents tools. Pel nostre cas d'ús, considerarem la 1a opció, ja que dins la integració de SUPERSEDE, l'anàlisi d'aquestes dades serà independent de la *tool* utilitzada.

- **GooglePlayAPI**. Implementació de *ToolInterface* que utilitza l'API de Google Play Developer. Aquesta *tool* implementa una comunicació mitjançant autenticació OAuth 2.0 mitjançant els *tokens* obtinguts al registrar un usuari de Google Play com a *developer*. Per aquest projecte s'ha utilitzat un compte personal de Google Play compartit amb altres estudiants del Grau en Enginyeria Informàtica, per tal de poder probar i validar aquesta *tool* (ja que, tal i com especificat anteriorment, únicament ens permet obtenir dades de les aplicacions de les quals l'usuari n'és l'autor). Degut al gran volum de dades que es poden

generar al demanar les *reviews* d'una aplicació, l'API funciona mitjançant un sistema de paginació. És a dir: la crida REST per obtenir les reviews retorna un subconjunt de mida relativament petita i, addicionalment, un *token* que permet referenciar el següent subconjunt (o pàgina) de reviews. Aquest token s'utilitza per realitzar una nova crida, que conté un nou subconjunt de reviews i, addicionalment, un nou *token* per la següent pàgina. D'aquesta manera, les dades venen paginades i subdividides. És responsabilitat de la *tool* implementar la lògica per realitzar les crides de forma iterativa. Aquest tractament pot ser un procés relativament lent (en termes computacionals). Però això no suposa un trencament amb la filosofia de "fotografiar"el sistema en un moment determinat: en el moment que es fa la crida REST, l'API captura les *públiques* en aquell moment, i construeix els objectes de dades paginats, de tal manera que encara que es triguin uns segons en computar totes les pàgines de dades, aquestes sempre seran una representació del moment en que s'ha fet la primera crida, corresponent al final d'un cicle de monitoratge de durada definida al *timeSlot*.

- **AppTweak**. Implementació de *ToolInterface* que utilitza l'API de AppTweak. A diferència de l'anterior, tant l'autenticació com la lògica per obtenir les dades és molt més senzilla. En el cas de la primera, funciona mitjançant l'ús d'un *token* a la capçalera de la crida REST, únic per usuari a la plataforma. En el cas de la segona, una sola crida REST retorna totes les dades corresponents a les *reviews* d'aquella aplicació.

Exposició com a servei REST

De forma anàloga a l'exposició com a servei del monitor de Twitter, cal dissenyar i implementar un servei REST per desplegar les funcionalitats del monitor de Google Play. La figura 7.10 mostra un exemple d'un possible objecte JSON de configuració del monitor.

```
{
    "GooglePlayConfProf": {
        "toolName": "AppTweak",
        "timeSlot": "30",
        "kafkaEndpoint": "http://localhost:9092",
        "kafkaTopic": "MarketPlace",
        "packageName": "com.facebook.katana"
    }
}
```

FIGURA 7.10: Exemple JSON de configuració del monitor de Google-Play

Tot i que el paràmetre de *toolName* ja l'havíem vist en l'anterior monitor, ja que és necessari per la instanciació de la *tool* utilitzant el patró reflexió, en aquest cas cobra una especial importància, ja que al disposar d'un conjunt de *tools* podem utilitzar els noms d'aquestes per configurar el monitor; en aquest cas, amb les *tools* de **GooglePlay** i **AppTweak**.

App Store Monitor

El segon monitor dins el tipus de monitoratge de *Market Places* és un monitor de l'**App Store**, la botiga d'aplicacions per sistemes operatius mòbils iOS. Des d'un punt de vista conceptual, la premisa és la mateixa que en el cas anterior: obtenir dades en temps real de les *reviews* que es realitzen sobre una aplicació en concret.

Per aquest monitor desenvoluparem també dues *tools* diferents:

- **ITunes-RSS**. *Tool* oficial d'Apple [28] que permet obtenir, en format JSON, dades sobre les darreres *reviews* publicades sobre una aplicació específica, identificada pel que anomenarem *appId*. La limitació principal d'aquesta eina és que només permet obtenir les darreres 500 *reviews* publicades. Per tant, tot i que sabem que el volum de dades sempre estarà controlat sota uns mínims, cal contemplar la possibilitat que durant el *timeslot* definit el nombre de *reviews* sigui major que aquest límit. En aquest cas, estaríem perdent dades de monitoratge„ja que la implementació d'aquesta eina consisteix principalment en la crida periòdica a aquesta API i obtenció de les *reviews* publicades durant el darrer cicle de monitoratge. Aquesta *tool* permet obtenir les *reviews* ordenades temporalment, per tant la cerca, com amb les eines de Google Play, serà prou eficient. Per contra al límit de *reviews*, no presenta límits de comunicacions alhora o intents de comunicació per hora.
- **AppStore-AppTweak**. Utilitza la mateixa eina que Google Play, publicada com a API REST, amb la possibilitat de demanar dades d'aquesta segona botiga d'aplicacions. El funcionament intern és similar al ja explicat per GooglePlay-AppTweak; però cal considerar diferències tècniques en aspectes com el format de les dades obtingudes.

També de forma similar al monitor de Google Play, necessitem definir l'únic paràmetre que serà necessari per la configuració de la *tool*:

- **appId** - identificador únic d'una aplicació publicada a l'AppStore (anàleg a l'identificador de Google Play, però en format numèric)

A la figura 7.11 veiem la representació de l'arquitectura estesa respecte la genèrica per la implementació d'aquest monitor:

- **AppstoreMonitoringParams.** Subclasse de la classe abstracta *MonitoringParams* que hereta els atributs comuns a tots els monitors, i afegeix com a atribut particular del monitor un string *appId*.
- **AppStoreParserConfiguration.** Implementació de la interfície que defineix la transformació del format d'entrada del monitor (JSONObject) a la instància de *AppstoreMonitoringParams*.
- **AppStoreMonitoringData.** Implementació de la interfície que defineix la transformació de les dades recollides al format JSON de sortida. En aquest cas hem d'afegir la consideració prèviament introduïda sobre la variabilitat de les dades entre les dues *tools*.
- **ITunes-RSS**. Implementació de *ToolInterface* que utilitza el RSS de Apple. El funcionament intern es similar al d'altres *tools* que utilitzen una API per obtenir les dades: un cop iniciat el cicle de monitoratge, i mitjançant un *Timer*

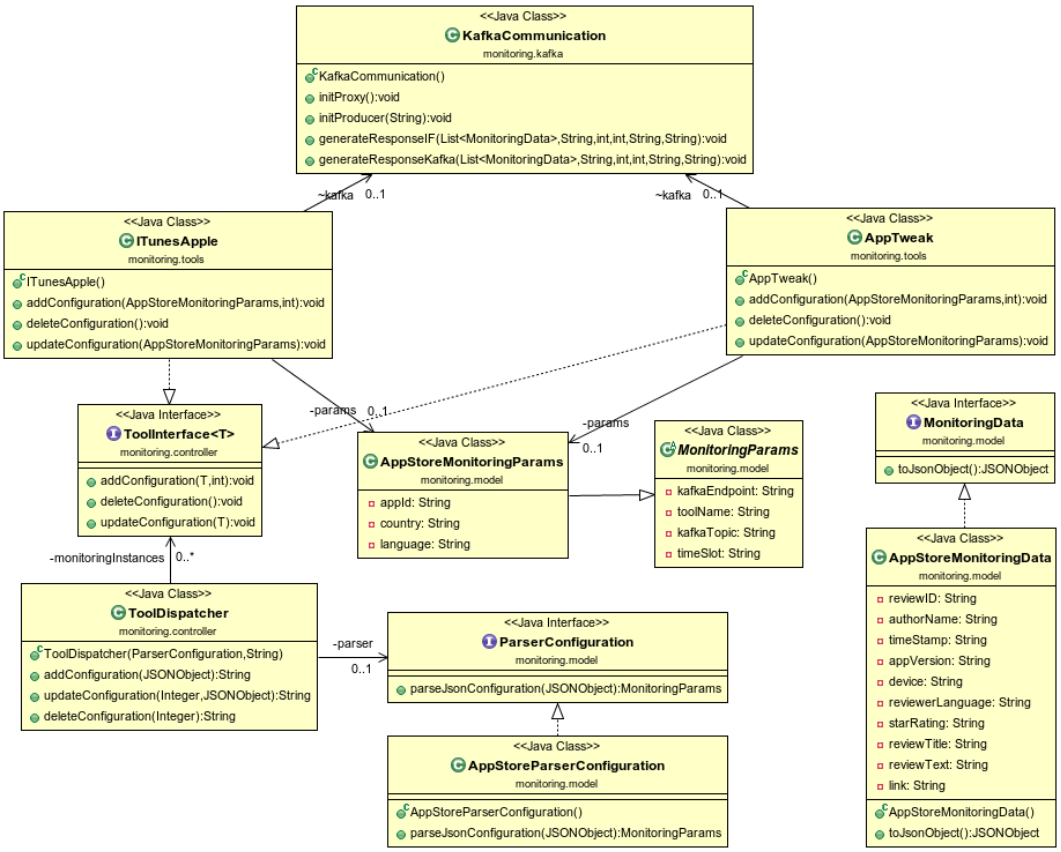

FIGURA 7.11: Arquitectura software del monitor de l'AppStore

executat com a *thread* en segon pla, el monitor s'espera el temps definit pel *timeslot* i, un cop finalitzat, realitza una crida HTTP per obtenir les dades en format JSON per després "parsejar-les". La lògica interna d'aquesta *tool* és relativament més senzilla que la de l'API de Google Play, ja que aquesta segona utilitza un sistema de paginació, mentre que la *tool* RSS de iTunes retorna totes les dades amb una sola crida. Això es deu a la limitació de 500 *reviews*, que assegura que el *payload* retornat per la crida no excedirà una mida específica.

- **AppTweak**. Implementació de *ToolInterface* que utilitza l'API de AppTweak. El funcionament tècnic és el mateix que el ja explicat per Google Play.


```
{
    "AppStoreMonitoredData": {
        "idOutput": "12345",
        "confId": "67890",
        "searchTimeStamp": "2017-05-19 17:23:00.000",
        "numDataItems": 1,
        "DataItems": [
            {
                "reviewID": "5zvcx789f320r458",
                "authorName": "Mike Moreno",
                "timeStamp": "2016-05-25 20:03",
                "appVersion": "5.4",
                "device": "iPhone6",
                "reviewerLanguage": "en",
                "starRating": "5",
                "reviewTitle": "Just 5 minutes",
                "reviewText": " I always say I'll play for 5 minutes which turns into 30 then"
                    + "an hour until my battery goes dead. Very addictive indeed.",
                "link": "https://itunes.apple.com/us/app/apple-store/id375380948?mt=8"
            }
        ]
    }
}
```


FIGURA 7.12: Exemple dades de sortida generades pel monitor de AppStore

Exposició com a servei REST

De forma anàloga a l'exposició com a servei del monitor de Twitter i Google Play, cal dissenyar i implementar un servei REST per desplegar les funcionalitats del monitor de App Store. La figura 7.12 mostra un exemple d'un possible objecte JSON de configuració del monitor.

7.2 Monitor Manager

Partint de l'arquitectura genèrica dels monitors, i l'exemplificació d'aquesta en casos d'ús reals (i per tant a seva implementació), disposem dels primers components essencials per construïr el nostre sistema. En aquest punt disposem d'un subconjunt de monitors que satisfan les seves especificacions tècniques individuals, relacionades amb l'activitat de monitoratge. Però necessitem anar un pas mes enllà en l'evolució del nostre sistema per satisfer els objectius generals d'adaptabilitat, i procedir a desenvolupar components d'integració.

FIGURA 7.13: Exemple JSON de configuració del monitor de AppStore

Per aquest objectiu, el primer pas és el desenvolupament del **Monitor Manager**. Aquest component, tal com el seu nom indica (i com s'ha presentat al *Capítol 5. Visió general del sistema*), s'encarrega de la gestió de tots els monitors desplegats al sistema. Per gestió entenem la satisfacció dels següents requisits funcionals:

1. **Inicialització de procés de monitoratge per a un monitor específic.** El component ha de ser capaç de rebre una petició d'inicialització de procés de monitoratge per un dels monitors integrats al sistema, i redireccionar aquesta petició al monitor corresponent, de manera que aquest pugui processar i executar la petició.

2. **Modificació de procés de monitoratge per a un monitor específic.** Donat un procés de monitoratge existent per a un monitor específic integrat al sistema, el component ha de rebre una petició de reconfiguració d'aquest, processar-la i redireccionar-la al monitor corresponent.

3. **Aturada de procés de monitoratge per a un monitor específic.** Donat un procés de monitoratge existent per a un monitor específic integrat al sistema, el component ha de rebre una petició per aturar-lo, processar-la i redireccionar-la al monitor corresponent.

Com es pot extreure de les funcionalitats anteriors, l'objectiu principal d'aquest component és **processar** i **redireccionar** les peticions relacionades amb accions sobre els monitors desplegats. D'aquesta manera, i enfocat al màxim nivell d'independència entre components, estem afegint un nivell d'abstracció per sobre dels monitors que permetrà la comunicació amb la resta de components del sistema sense necessitat de conèixer els detalls semàntics i tècnics de cada implementació dels monitors.

Per tant, com a part de la tasca en el disseny i desenvolupament d'aquest monitor, caldrà definir un punt d'entrada únic pels 3 tipus d'operacions: **iniciar**, **modificar** i **aturar** un procés de monitoratge en un monitor específic. La lògica interna del Monitor Manager, per tant, s'haurà d'encarregar de processar aquestes peticions, identificar a quin monitor pertoca, i redireccionar-la al mateix.

7.2.1 Especificacions tècniques i funcionals

Les especificacions tècniques en les que ens basarem per dissenyar i implementar el Monitor Manager requereixen satisfer una **integració** que actuï com a pont **únic**

entre components tercers del sistema i els nostres monitors. Seguint els criteris presentats anteriorment, caldrà considerar els següents punts:

1. **INPUT - Petició de configuració de monitor**. Descripció dels paràmetres i el format genèric d'aquests per realitzar la redirecció de la petició.
2. ***ACTION* - Processat de la petició**. Tractament del format genèric dels paràmetres i interpretació d'acord amb el monitor adreçat.
3. ***OUTPUT* - Redirecció al monitor**. Enviament de la petició processada cap al monitor.

Petició de configuració de monitor

La petició de configuració s'ha de rebre en un format genèric que el propi Monitor Manager sigui capaç d'encapsular, identificar el monitor al qual cal enviar-la, i redireccionar-la posteriorment. Per fer-ho, independentment del format, necessitem identificar dos factors claus que el Monitor Manager ha de rebre:

- El **monitor específic** al qual s'ha de redireccionar la petició
- Els **paràmetres de configuració** per aquell monitor: *toolName* + *kafkaEndpoint* + *kafkaTopic* + *timeSlot* + [paràmetres específics]

Respecte al 2n punt, es tracta d'informació formatada que el monitor específic necessita, i que per tant podem reaprofitar i mantenir estructurada tal i com s'ha documentat prèviament. Respecte al 1r punt, en canvi, es tracta d'informació que el Monitor Manager necessita addicionalment per processar la redirecció.

En aquest punt cal decidir si aquesta informació l'afegim de forma implícita a l'objecte JSON de configuració, o bé si busquem una alternativa que ens permeti mantenir l'objecte JSON intacte. En el primer cas, la informació relativa a la configuració queda totalment integrada en un sol objecte JSON que utilitzarem per **redireccionar** i **configurar** el monitor. Però això obliga a modificar el format d'aquest JSON, afegint informació addicional que el monitor no necessita. En el segon cas, en canvi, podem mantenir un objecte de configuració que es propaga entre els diferents components: primer com a *input* del Monitor Manager, i després com a *output* redireccionat com a *input* al monitor específic. Alternativament, però, cal buscar una forma de comunicar al Monitor manager a quin monitor trobarà la *tool* sobre la qual hem d'iniciar un procés de monitoratge.

En aquest sentit aprofitarem la necessitat d'exposició dels components com a serveis RESTful per, mitjançant el propi disseny de la API que implementarà el Monitor Manager, definir aquest paràmetre. En el disseny d'aquesta API, present a l'apèndix **??**, proposem com a paràmetre dins la URL de les 3 crides a implementar (creació, modificació i eliminació) el propi identificador del monitor. Així, exemplificant la crida per crear una configuració sobre el monitor de Twitter, podem veure a la figura 7.14 la URL que defineix el recurs per realitzar aquesta operació (on *monitorName* és el paràmetre de la URL que identifica el monitor a redireccionar), així com un exemple de JSON que rebrà. Com es pot apreciar, aquest presenta els mateixos camps que el JSON definit als monitors.

```
## Monitor [/{monitorName}/configuration]

###Create a new monitor configuration [POST]

+ Request (application/json)

    {
        "SocialNetworks": {
            "toolName": "TwitterAPI",
            "timeSlot": "30",
            "kafkaEndpoint": "http://localhost:9092",
            "kafkaTopic": "tweeterMonitoring",
            "keywordExpression": "(olympics) AND (streaming)"
        }
    }
```

FIGURA 7.14: Exemple JSON de configuració de monitor al Monitor Manager

Amb aquestes dades d'entrada, el Monitor Manager ja és capaç de realitzar la redirecció al monitor corresponent.

Processat de la petició

Un cop definida la informació i el seu format d'entrada necessaris per satisfer les especificacions tècniques, cal avaluar com partim d'aquesta informació a la petició de configuració de monitors.

Ja que la única tasca a realitzar és la redirecció (ja que les dades no cal que siguin tractades), hem de partir del paràmetre *monitorName* per identificar el monitor al qual redireccionar. El component IF exposa, de manera separada, la implementació de classes o *proxies* diferents per cada monitor implementat. És a dir: per cada monitor que vulguem integrar i registrar al nostre sistema, necessitarem afegir-ho a IF per garantir-ne la integració de la comunicació, mitjançant la implementació d'un *proxy* amb els mètodes de comunicació que ofereix. Per tant, la responsabilitat del Monitor Manager serà utilitzar el paràmetre *monitorName* per discernir entre els *proxies* implementats per IF i seleccionar aquell que es correspongui al monitor al qual la petició ha d'anar adreçada.

Redirecció al monitor

Finalment, identificat i instanciat el *proxy* el Monitor Manager executarà una crida *addConfiguration*, *updateConfiguration* o *deleteConfiguration* en funció de l'operació enviada. De nou en aquest sentit s'aprofita l'*input* d'aquest component mitjançant la seva exposició com a servei: cada mètode de creació, modificació i eliminació crida al mètode corresponent del *proxy* definit per *monitorName*. A aquest *proxy* s'enviarà la instància de configuració rebuda d'entrada, reaprofitant exactament el mateix format, mantenint així la uniformitat de les dades desitjada.

7.2.2 Disseny i implementació

En definitiva, d'acord amb les necessitats establertes, la implementació del Monitor Manager es basarà simplement en el disseny i implementació d'un servei REST que

implementi els 3 mètodes definits anteriorment, i que la seva lògica interna s'encarregui simplement d'identificar, a partir del nom del monitor rebut com a paràmetre, el *proxy* que implementa la comunicació amb aquell monitor. A la figura 7.15 podem veure la interfície que defineix els 3 mètodes que exposen les 3 funcionalitats bàsiques se les configuracions de monitors. A partir d'aquesta interfície, només caldrà exposar-lo com a servei web RESTful per la seva integració a IF.

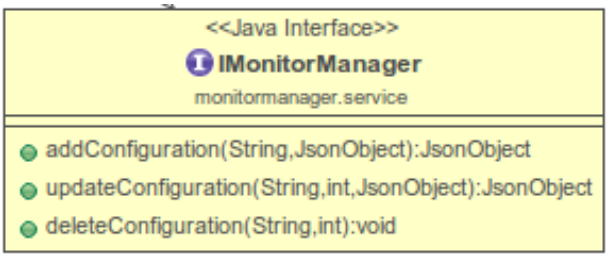

FIGURA 7.15: Disseny de la interfície del Monitor Manager

Cal notar que un dels paràmetres que reben com a paràmetre les 3 peticions ha de ser el nom del monitor al qual va redirigit aquesta petició.

7.3 Orchestrator

Aquest component forma part de la integració d'aquest projecte dins el projecte SUPERSEDE, i per tant el disseny i desenvolupament d'aquest ha estat únicament realitzat com a part d'aquest TFG de forma parcial. Concretament, aquelles parts referents a l'activitat de monitoratge i reconfiguració de monitors.

La seva tasca principal és actuar de pont entre els components encarregats d'aplicar adaptacions (*Enactments*) i aquells encarregats del monitoratge (*Monitoring*), que recordem forma part del cicle MAPE-k descrit anteriorment. Aquest és necessari en el moment que diversos sistemes d'adaptabilitat apliquen modificacions en diversos sistemes encarregats de monitorar dades. Aquest TFG n'és un cas específic. Si, per contra, aquest hagués d'actuar com a sistema independent, l'abstracció garantida pel Monitor Manager, que unifica tots els monitors sota un únic punt d'entrada, ja seria suficient per facilitar la comunicació, el manteniment i l'extensió de components. Per contra, si haguéssim volgut afegir la lògica del Monitor Manager al mateix Orchestrator, i aprofitar la seva necessitat dins SUPERSEDE per estalviar-nos una capa en el sistema, ens veuríem obligats a aplicar modificacions i manteniment a l'Orchestrator, un component genèric d'integració, davant modificacions específiques pel nostre cas d'ús. Per tal d'evitar això, s'ha decidit mantenir aquests dos components per separat, i garantir un desacoblament de components el més alt possible.

Addicionalment, a aquest component se li assigna una responsabilitat superior associada a la gestió i control del sistema de monitoratge i les seves especificacions. Amb això es fa referència a la persistència i control de metadades relacionades amb els monitors, que ens permeten tenir de forma integrada coneixement sobre el nostre sistema. Aspectes com, p.e., els **tipus de monitors** integrats, que engloben un conjunt de monitors treballant sobre una mateixa àrea, o bé les **tools implementades**

per cada tipus de monitor.

7.3.1 Especificacions tècniques i funcionals

En general, per tant, podem concebre l'Orchestrator com un **component d'integració** i un **repositori de metadades**. Per tant, independentment de les funcionalitats que caldrà implementar (que a continuació detallarem), caldrà tenir en compte dues característiques bàsiques:

- Haurà de gestionar la **persistència** de dades relacionades amb els monitors, i per tant caldrà afrontar el **disseny** i la **implementació** d'una base de dades.
- Paral·lelament, caldrà implementar un controlador exposat com a **servei REST** que exposi les seves funcionalitats i, addicionalment, permeti la seva integració a IF.

Per definir els detalls d'aquestes dues branques, necessitem primer definir amb quines metadades estarem treballant i com estructurarem el seu contingut. Definirem, essencialment, tres entitats principals:

1. **Monitor configuration**. Instància ja definida anteriorment, que representa l'execució d'una *tool* amb uns paràmetres de configuració específics.
2. **Monitor tool**. Engloba les metadades associades a aquella *tool* i té associades totes les configuracions en procés d'execució. D'aquesta manera, les dades referents a les configuracions actives per a una *tool* no es troben exclusivament al desplegament del propi monitor, el que suposaria la necessitat d'accedir al propi monitor. Contràriament, aquestes dades es trobaran integrades a l'Orchestrator, el que facilitarà la seva gestió quan es tracti de casos exclusivament de lectura de dades. Aquest darrer propòsit, en qualsevol cas, no entra part del context d'aquest projecte, sinó que està orientat a la integració del TFG amb SUPERSEDE i l'explotació d'aquestes dades.
3. **Monitor type**. Engloba un conjunt de *tools* que pertanyen a monitors d'una mateixa categoria, com per exemple *Social Networks* o *Market Places*. Aquest agrupament per tipus, com en el cas anterior, tampoc és rellevant pel nostre context, però ens ofereix integració de dades que resulta d'interès en termes analítics.

Monitor Type	**Monitor Tool**	**Monitor configuration**
Social Networks	TwitterAPI	TwitterAPI-Conf1
		TwitterAPI-Conf2
		TwitterAPI-Conf3
Market Places	GooglePlayAPI	GooglePlay-Conf1
		GooglePlay-Conf2
	GooglePlay-AppTweak	GooglePlay-AppTweak-Conf1
	iTunesApple	iTunesApple-Conf1
		iTunesApple-Conf2
	AppStore-AppTweak	AppStore-AppTweak-Conf1

TAULA 7.1: Exemple d'instanciació del sistema de monitoratge

Per tal d'entendre bé aquest esquema, podem visualitzar un esquema conceptual a la taula 7.1 que presenta un exemple de constitució d'un sistema de monitoratge en actiu com el que hem anat descrivint al llarg d'aquest projecte. Bàsicament tenim dos tipus de monitors, **Social Networks** i **Market Places**. Pel primer tenim definida una única *tool*, **TwitterAPI**; pel segon, tenim definides 4, dues per cada *market place* (*AppStore* i *GooglePlay*). Fixem-nos en el detall que en cap moment estem contemplant l'entitat **monitor** en aquest esquema. Això es deu al fet que, dins el nostre esquema, l'entitat monitor només és significativa des d'un punt de vista físic, com a punt integrat de desplegament que agrupa un conjunt de *tools*. Podríem considerar que existeixen punts comuns entre *tools* dins un mateix monitor, com p.e. la implementació dels paràmetres d'entrada o de sortida. Però la nostra arquitectura no restringeix aquest aspecte. Dins un mateix monitor, cada *tool* pot implementar aquestes interfícies i classes d'acord amb les seves necessitats. D'aquesta manera, el concepte monitor únicament és significatiu des del punt de vista d'implementació i desplegament físic, no lògic o de metadades. Per tant, no considerarem aquesta entitat dins el nostre esquema conceptual.

Finalment, a la taula podem veure un conjunt d'exemple de configuracions. Aquest no vindrà limitat per cap restricció, i tot i que cada monitor s'ha d'encarregar en última instància de gestionar-ho, serà a través del Orchestrator i el Monitor Manager que es gestionaran les altes, baixes i modificacions d'aquests.

Peticions de configuració del sistema de monitoratge

Mitjançant l'exposició de les funcionalitats com a servei REST, l'Orchestrator haurà de rebre el conjunt de peticions amb dos objectius principals: la **gestió de metadades** del nostre sistema de monitoratge, i la **redirecció de les peticions** associades directament a configuracions del sistema. La definició d'aquests mètodes d'entrada no ve a ser més que el disseny d'un controlador basat en les operacions **CRUD** (***C****reate*, ***R****ead*, ***U****pdate*, ***D****elete*) per les entitats *Monitor Type*, *Monitor Tool* i *Monitor Configuration*. A la taula 7.2 podem visualitzar un resum conceptual dels mètodes que necessitarem definir, i quins paràmetres caldrà definir per cadascun d'ells.

Respecte aquests mètodes, cal considerar els identificadors de cadascuna d'aquestes entitats:

- **Monitor Type**. S'identifica per un **nom** únic
- **Monitor Tool**. S'identifica per un *Monitor Type* (un nom de *Monitor Type*) + un **nom** únic de tool
- **Monitor Configuration**. S'identifica per un *Monitor Type* (un nom de *Monitor Type*) + una *Monitor tool* (un nom de *Monitor Tool*) + un **id** únic de la configuració

En general, tota la informació d'entrada que necessita per executar cadascuna de les operacions es basa en identificar el recurs sobre el qual aplicar l'acció. Aquesta identificació a nivell de configuració, que és la darrera entitat que a nosaltres ens interessa considerar, es basa en un triplet de la forma:

monitorType + monitorTool + idConf

Només en un cas necessitem afegir més informació, que serà pels dos casos a treballar en aquest projecte: la **creació d'una configuració** i la seva **reconfiguració** (o actualització). En aquest cas, i tal i com s'ha definit en l'especificació i disseny dels monitors, caldrà afegir d'una banda aquells paràmetres obligatoris per a tot monitor, i addicionalment aquells propis de cada *tool*.

Actualització de metadades

El comportament intern de l'Orchestrator serà relativament senzill, ja que únicament necessita contemplar l'actualització de metadades i el processament de les peticions realitzades per redirigir-les, si s'escau, al monitor corresponent.

Considerant primer **l'actualització de metadades**, l'Orchestrator actualitzarà d'acord a l'operació executada modificacions a la base de dades d'acord amb **creació**, **actualització** i eliminació de les entitats prèviament definides. En cas que es tracti d'operacions de lectura, només caldrà accedir a la base de dades i llegir les dades associades a la petició realitzada. Aquest aspecte de l'acció interna el considerarem trivial, ja que es tracta únicament de modificacions associades a una BDD relacional (veure més endavant a *Implementació del Orchestrator*), i no resulta d'especial interès per la finalitat del nostre projecte.

Redirecció de peticions

Per altra banda cal definir la **redirecció de peticions** a monitors. Fixem-nos que, al disposar de totes les dades integrades, aquesta únicament caldrà contemplar-la pels casos en els que no només cal realitzar modificacions sobre les pròpies metadades (o bé simplement realitzar lectures), sinó també realitzar canvis relacionats amb l'execució real dels monitors. Això per tant inclourà: **alta** de configuració, **reconfiguració** d'una configuració, i **eliminació** d'una configuració. En aquests 3 casos, l'Orchestrator haurà de realitzar la redirecció amb el Monitor Manager, utilitzant la integració amb IF.

El mateix problema de criteri de redirecció sorgeix que en el cas del Monitor Manager. Necessitem saber quina *tool* va associada a quin monitor, per tal d'indicar al Monitor Manager a quin monitor es troba implementada aquella *tool*. El Monitor Manager requeria aquest paràmetre en la petició, ja que ell no té constància d'aquestes dades. Però a l'Orchestrator sí que disposem d'aquesta metadada, associada a una *tool*, i definida en el moment de creació, tal i com observem també a la taula 7.2. Per tant, a l'hora d'instanciar el *proxy* de IF per comunicar-se amb el Monitor Manager, cridarem al mètode corresponent al Monitor identificat per aquest atribut. D'aquesta manera la redirecció queda satisfeta.

7.3.2 Disseny i implementació

Sense entrar en molt detall en els aspectes d'implementació de les especificacions prèviament descrites, podem trobar els detalls d'aquesta a:

- Disseny dels controladors de domini, definits a la figura 7.16
- Disseny de l'esquema de la **base de dades**, definit a l'apèndix **??**

- Disseny de l**API RESTful**, definit a l'apèndix **??**

Degut a que el comportament i els detalls específics no són d'especial interés pel domini d'aquest projecte, no s'entrarà amb més detall en la informació referent a la implementació tècnica, que s'ha realitzat seguint les instruccions i els requisits tècnics específics del component Orchestrator genèric implementat per *partners* del projecte SUPERSEDE.

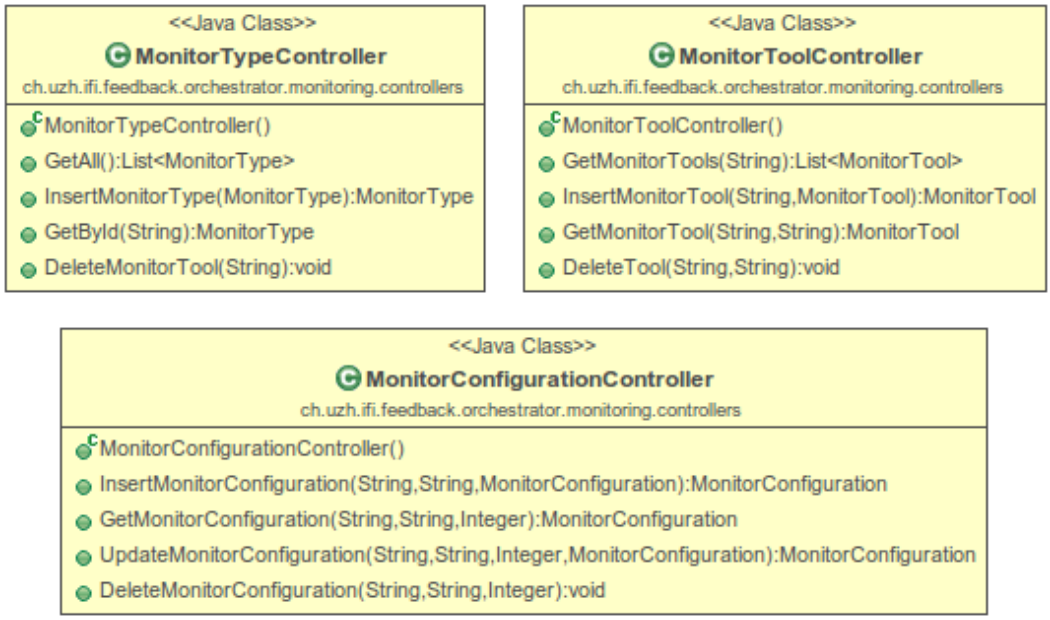

FIGURA 7.16: Disseny dels controladors de l'Orchestrator

Operació	Definició	Paràmetres
Monitor Type		
Llistat de Monitor Types	Retorna el llistat de tots els Monitor Types instanciats al sistema	/
Obté un Monitor Type	Retorna les dades d'un Monitor Type específic	- nomTipus
Alta de Monitor Type	Crea una nova instància de Monitor Type	- nomTipus
Baixa de Monitor Type	Elimina una instància de Monitor Type existent	- nomTipus
Monitor Tool		
Llistat de Tools per Monitor Type	Retorna el llistat de totes les Tools implementades per a un Monitor Type	- nomTipus
Obté una Tool per un Monitor Type	Retorna les dades d'una Tool per un Monitor Type	- nomTipus - nomTool
Alta de Tool per un Monitor Type	Crea una nova Tool per un Monitor Type	- nomTipus - nomTool - nomMonitor
Baixa de Tool per un Monitor Type	Elimina una Tool existent per un Monitor Type	- nomTipus - nomTool
Monitor Configuration		
Llistat de Configurations per una Tool	Retorna el llistat de totes les Configurations actives per una Tool	- nomTipus - nomTool
Obté una Configuration per una Tool	Retorna les dades d'una Configuration per una Monitor Tool	- nomTipus - nomTool - idConf
Alta de Configuration per una Tool	Crea una nova Configuration per una Monitor Tool	- nomTipus - nomTool - timeSlot - kafkaEndpoint - kafkaTopic - [custom]
Modificació d'una Configuration per una Tool	Modifica els paràmetres d'una Configuration existent per una Monitor Tool	- nomTipus - nomTool - idConf - timeSlot - kafkaEndpoint - kafkaTopic - [custom]
Baixa d'una Configuration per una Tool	Elimina una Configuration existent per una Monitor Tool	- nomTipus - nomTool - idConf - timeSlot - kafkaEndpoint - kafkaTopic - [custom]

TAULA 7.2: Llistat de peticions d'entrada del Orchestrator

8 Modelatge UML de configuracions dels monitors

Finalitzada l'especificació de disseny i tècnica dels monitors i les seves configuracions, així com els detalls de la seva implementació, ja tenim definit un sistema de monitoratge que satisfà els criteris d'**heterogeneïtat** i **distribució**. Aquestes característiques queden garantides dins el sistema de monitoratge com a unitat independent.

El següent pas és gestionar una extensió del sistema que permeti gestionar l'adaptabilitat dels monitors, i concretament que aquesta pugui ser automatitzada, sense necessitat de definir explícitament crides a peticions de reconfiguració al Orchestrator. Tot i que queda fora de l'abast el disseny d'un sistema d'anàlisi i detecció automàtica de reconfiguracions a aplicar (l'equivalent al sistema d'**A**nàlisi dins el *MAPE-k*), el nostre objectiu és dotar al sistema de monitoratge d'un sistema d'adaptabilitat, que sigui capaç de processar i computar aquestes reconfiguracions.

8.1 Requisits del modelatge

Per poder definir els models amb els quals hem de treballar, primer necessitem definir les necessitats del nostre sistema. En termes genèrics, el sistema d'adaptabilitat a dissenyar necessita resoldre la següent problemàtica:

*Donada una **instància** del sistema de monitoratge, el sistema ha de rebre una **proposta de reconfiguració** dels monitors, basada en la modificació d'una **característica** específica, aplicant un **conjunt d'accions** sobre els **elements del sistema** determinats.*

Per major aclariment, anem a analitzar el significat de cadascun dels conceptes introduïts en aquesta definició:

1. **Instància del sistema**. Referent al conjunt de **classes** que defineixen els diferents tipus de configuracions persistents al sistema (és a dir, les diferents implementacions de processos de monitoratge associats a les diferents *tools*), i les **instàncies** actives corresponents (o processos de monitoratge actius).

2. **Proposta de reconfiguració**. Necessitem modelar, per una banda, el conjunt de característiques que podem referenciar i editar dins el nostre sistema. Aquestes inclouen, per exemple, la definició de l'atribut *timeSlot* de les instàncies dels monitors de Twitter dins el tipus de monitor SocialNetwork. Per altra banda, necessitem modelar, basat en aquest model de característiques, propostes de configuracions específiques, on es defineixin p.e. valors específics d'aquest *timeSlot*.

3. **Característica específica**. En relació al punt anterior, que engloba un conjunt de característiques modificables en el sistema, necessitem referenciar quina característica volem modificar.

4. **Conjunt d'accions**. A partir de les propostes de configuració, i l'aplicació d'una característica específica, hem de definir les diferents accions que s'han de realitzar sobre la instància actual del sistema. Aquestes accions inclouran, seguint amb el mateix exemple, la modificació p.e. d'un atribut com *timeSlot*.

5. **Elements del sistema**. Necessitem referenciar sobre quins elements del sistema volem aplicar els canvis. És a dir: hem de ser capaços de referenciar, dins el model del sistema, quines instàncies cal modificar i quines no, per tal d'aplicar les accions basades en les característiques anteriors només als que ens interessin.

8.2 Disseny de models UML

Partint d'aquests requisits conceptuals, necessitem definir el conjunt de models amb els quals treballarem per representar tots aquests punts, i permetre així la computació automàtica de canvis dins el nostre sistema. Procedirem presentant cadascun dels models, explicant breument en què consisteixen, i més profundament com s'apliquen al nostre projecte.

La implementació i gestió d'aquests models es realitzarà utilitzant l'eina Papyrus, introduïda anteriorment al *Capítol 5. Eines de desenvolupament*. Concretament, pel tractament i gestió de la seva implementació es farà servir la llibreria UML2, versió 5.0.0.

8.2.1 Base Model

Definirem *Base Model* com un diagrama de classes UML que defineix les diferents classes que representen cadascun dels diferents tipus de configuracions de monitors i les seves instàncies [29]. Conceptualment, per tant, és senzill de concebre dins el nostre context, ja que no ve a ser res més que una representació en UML del sistema. D'acord amb el nostre disseny, les necessitats d'aquest disseny UML són les següents:

- Cal definir una **classe abstracta** que representi l'abstracció genèrica de totes les configuracions: és a dir, que englobi aquelles propietats (atributs) compartits entre totes les *tools* del nostre sistema. En el nostre cas, aquests seran: *timeSlot*, *kafkaEndpoint*, *kafkaTopic*, *toolName* i el propi identificador *id*.

- A continuació cal afegir les implementacions d'aquesta classe abstracta basades en les possibles diferents configuracions que es poden donar d'alta en el nostre sistema. En definitiva: totes aquelles varietats de configuracions segons els diferents paràmetres a definir (que en el nostre sistema ve a ser equivalent a les diferències entre tipus de monitors).

- Per cadascuna d'aquestes implementacions, cal modelar el conjunt de configuracions (és a dir, instàncies de classes), amb els valors dels atributs definits.

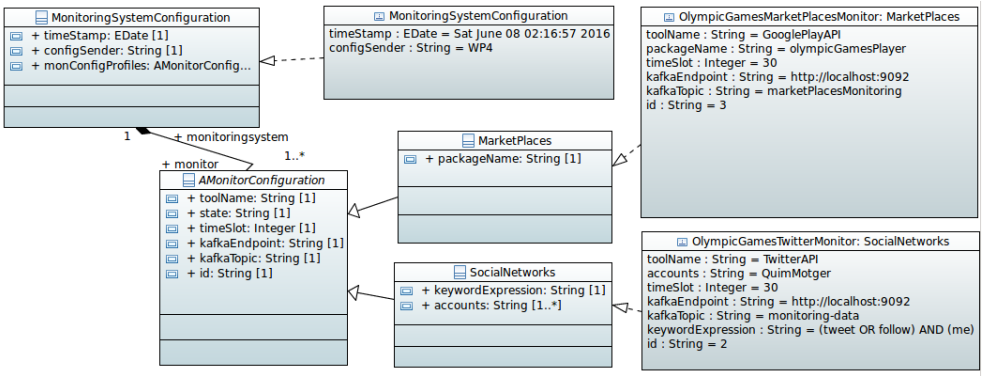

FIGURA 8.1: Exemple de Base Model del sistema de monitoratge

A la figura 8.1 podem visualitzar un exemple de *Base Model* que representa aquesta fotografia d'un sistema on tenim definits el monitor de Twitter i el de Google Play. *AMonitorConfiguration* representa la classe abstracta dels diferents tipus de configuracions; *SocialNetworks* i *MarketPlaces* representen les extensions de configuracions genèriques amb els paràmetres addicionals definits; i finalment, *OlympicGamesMarketPlacesMonitor* i *OlympicGamesTwitterMonitor* són instàncies dels diferents tipus de configuracions (no s'ha afegit el monitor d'AppStore per simplicitat en l'exemplificació). També veiem addicionalment una classe anomenada *MonitoringSystemConfiguration*, en relació d'agregació a *AMonitorConfiguration*, i una instància d'aquesta mateixa. Aquestes són classes establertes com a requisits dins el context SUPERSEDE, i per tant no són necessàries a tenir en compte.

L'objectiu principal de l'adaptabilitat del sistema serà aplicar canvis en aquest *Base Model* que defineix l'estat actual del sistema. Així, el sistema d'adaptabilitat actualitzarà per una banda el *Base Model* actual, aplicant els canvis pertinents, i computarà les diferències amb l'anterior per definir reconfiguracions que s'enviaran al Orchestrator.

8.2.2 Features

Aquests canvis, que anomenem de forma genèrica, són modificacions dins el diagrama de classes del sistema, i per tant inclou aspectes com la modificació del valor d'un atribut d'una instància de configuració. Però aquestes modificacions no poden ser aleatòries: suposant que no considerem el comportament del sistema de ***P**lanificació* dins el *MAPE-k*, i per tant no definim l'autogeneració de noves propostes de configuracions del sistema, necessitem modelar de forma definida una sèrie de casos, o configuracions predefinides, que defineixin un **conjunt de combinacions de característiques** adreçades a aplicar-se en diferents casos. És a dir: necessitem modelar d'alguna forma diferents alternatives per valors dels diferents atributs de les configuracions.

En termes específics, el que estarem fent serà definir un conjunt de propostes que aplicarem per a casos específics. Podríem, per exemple, modelar una proposta de configuració que disminueixi el valor del *timeSlot* quan el volum de dades obtingudes pel monitoratge sigui molt baix, i vulguem rebre aquestes amb menys periodicitat; o bé podríem re-orientar les dades a un *kafkaEndpoint* diferent quan, per

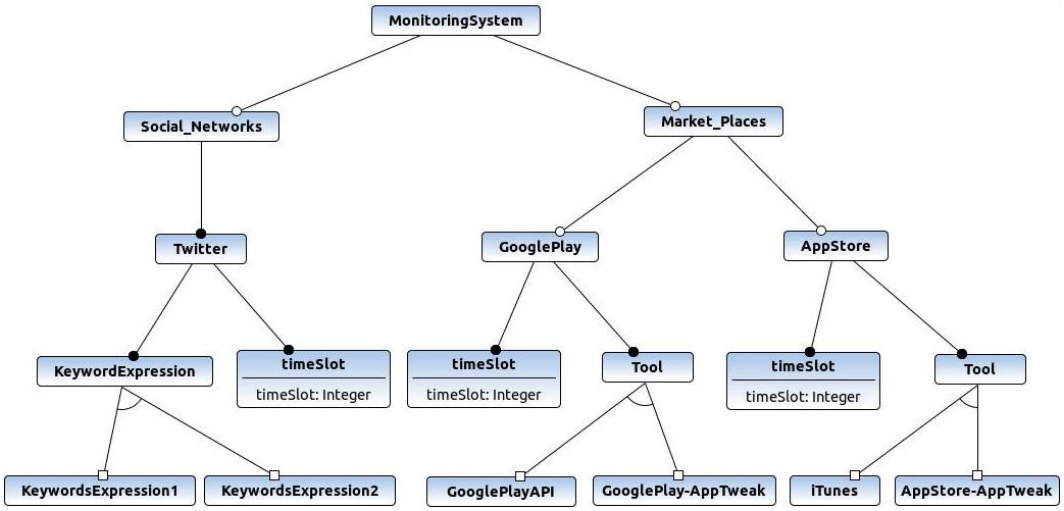

FIGURA 8.2: Exemple de Feature Model del sistema de monitoratge

diferents motius, el *kafkaEndpoint* actual estigui caigut i les dades rebotin. Davant aquests exemples, i un nombre indeterminat (d'acord amb les nostres necessitats), podem definir diferents casos o propostes que representin semànticament un canvi en el sistema.

Per modelar aquest aspecte, introduïm dos tipus de models addicionals: el ***Feature Model*** i les ***Feature Configurations*** [30].

Feature Model

Un *Feature Model* (FM), o model de característiques, en la seva definició genèrica, és un diagrama que permet gestionar el conjunt d'aspectes comuns i variables dins d'un sistema i els seus components. Representa, en definitiva, una modelització estandarditzada que defineix una jerarquia entre aquestes característiques i estableix les diferents opcionalitats de configuració dins un sistema.

Aplicat al nostre context, un *Feature Model* ens serveix per definir quines característiques del nostre sistema podem referenciar i modificar, i com s'estructuren aquestes des d'un punt de vista jeràrquic d'acord amb les implementacions de la classe abstracta de configuració de monitors. Un dels casos d'ús que contemplarem, ja que permet validar el nostre sistema alhora que mostrar fàcilment la repercussió dels canvis, és la reconfiguració de l'atribut *timeSlot* d'una instància de monitor. A la figura 8.2 podem observar un exemple de *Feature Model* del nostre sistema. Com podem veure, representa una modelació conceptual del sistema de monitoratge i la seva jerarquia: un sistema de monitoratge, amb el conjunt de tipus de monitors, que implementen un conjunt de monitors específics, on cadascun d'ells engloben una sèrie de *features*, és a dir, característiques que podem referenciar del nostre sistema. En aquest exemple, si ens fixem per exemple en el cas de Twitter, tenim per una banda la *feature timeSlot*, representada com un Integer de valor variable, i *keywordExpression*, que en comptes de tenir un valor (com podria ser un string) editable, defineix dues característiques opcionals, que representen dos possibilitats de valor associades a aquesta *feature*. De manera similar trobem el cas del monitor de Google Play i d'App Store, on la *feature toolName* és només configurable amb dos opcions en cada

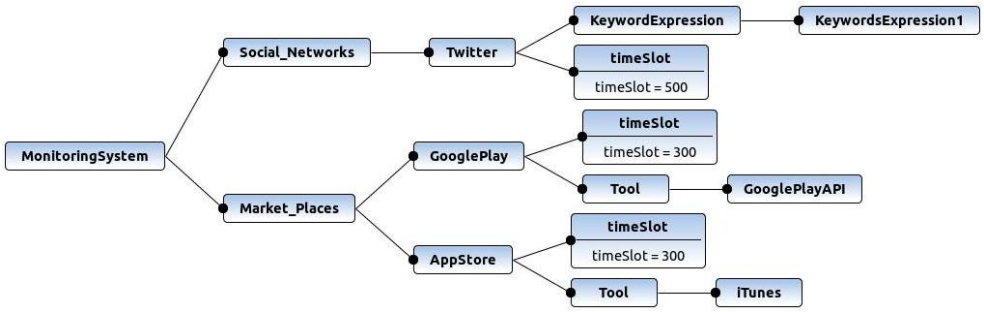

FIGURA 8.3: Exemple de Feature Configuration del sistema de monitoratge

cas, segons les *tools* que tenen implementades, i de nou el *timeSlot*. Cal destacar que aquest es tracta únicament d'un exemple de *Feature Model*: les diferents *features* es podrien ampliar i/o modificar, p.e. afegint noves *tools* conforme s'implementin, o bé afegint altres paràmetres a modificar (p.e. *kafkaEndpoint* o *kafkaTopic*).

Feature Configuration

Una *Feature Configuration* (FC) representa una configuració específica d'un *Feature Model*. És a dir: davant les diferents opcionalitats i variabilitats que un *Feature Model* defineix, com poden ser la personalització del valor d'una *feature*, o bé la selecció entre un conjunt d'opcions, la *Feature Configuration* és la modelació d'un cas específic d'aquestes *features*. A diferència del *Feature Model*, que presenta totes les opcions del sistema, la *Feature Configuration* presenta només una opció específica per aquells punts variables dins el *Feature Model*. Aquestes opcions específiques reben el nom de **selections** (seleccions).

Aplicat al nostre context, i seguint amb l'exemple anterior, l'objectiu d'una *Feature Configuration* es modelar una proposta de configuració del sistema de monitoratge. Així, definim uns valors específics pels atributs de les configuracions. A la figura 8.3 podem visualitzar un exemple basat en el *Feature Model* de la figura 8.2.

Si ens fixem en aquesta *Feature Configuration* i la comparem amb el *Feature Model*, veiem que tots els punts de variabilitat que aquesta segona definia s'especifiquen amb un valor o opcions específics. Per aquest cas, p.e., es proposa un valor de timeSlot específic per les configuracions de tots els tipus de monitors; paral·lelament, en el cas del monitor de Twitter s'especifica la *KeywordExpression1* com a opció a triar per aquest paràmetre, i les tools de *GooglePlayAPI* i *iTunes* per les *tools* dels monitors de Market Places. En definitiva, és la representació d'una proposta de configuració de les *features* definides. Orientat al nostre cas d'ús, aquesta proposta ens permet traduir-les al *Base Model* en una sèrie de modificacions d'atributs.

8.2.3 Advice Model

En alguns casos, i especialment considerant l'expansió i futur treball a partir de les bases establertes per aquest projecte, la modificació de models pot anar més enllà de la modificació o actualització d'un valor com el *timeslot* o algun altre paràmetre,

```
import "http://www.eclipse.org/uml2/5.0.0/UML"

pattern twitterTimeSlot(slot : Slot) {
    Slot.definingFeature.name(slot, name);
    Slot.owningInstance.classifier.name(slot, instanceName);
    check(name.equals("timeSlot"));
    check(instanceName.equals("SocialNetworks"));
}
```

FIGURA 8.4: Exemple de Pattern Model del sistema de monitoratge

com hem plantejat abans. De fet, per assolir una autonomia total del sistema de monitoratge, aquest hauria de ser capaç de modificar completament la seva activitat, mitjançant p.e. l'alta de nous processos de monitoratge o l'eliminació de processos existents.

Per aquest motiu, tot i que no serà el cas d'estudi d'aquest projecte, cal introduir el concepte d'*Advice Model*. Aquests models seran, dins el nostre context, diagrames de classes i instàncies en UML que modelen una secció del nostre sistema, com per exemple noves configuracions per a un monitor. La seva característica principal és que, a diferència dels *Base Model*, que representen una modelació d'un estat del sistema, aquests defineixen variacions, punts de variabilitat que es poden afegir, eliminar o substituir al *Base Model* original (més endavant veurem en què consisteixen aquestes modificacions). A la figura **??** podem veure un exemple aplicable al nostre context: una representació de dues propostes de noves configuracions de monitoratge que es podrien afegir al sistema.

8.2.4 Pattern Model

Aquesta modificació d'atributs, però, únicament ens està definint una acció genèrica. És a dir, la semàntica que podem desprendre d'una *Feature Configuration* és simplement la definició d'una proposta general per les configuracions dels monitors. El problema, però, està en que la *Feature Configuration* defineix tot el conjunt del sistema (en aquest cas, tots els atributs de tots els monitors), però no ens dona cap informació sobre quines instàncies del *Base Model* cal aplicar aquests canvis. És a dir: suposem que al *Base Model* de la figura 8.1 volem actualitzar el *timeSlot* de la instància *OlympicGamesTwitterMonitor*, però no de la instància *OlympicGamesMarketPlacesMonitor*.

Amb aquest objectiu, necessitem afegir informació addicional, una forma de modelar la identificació dels elements sobre els quals volem aplicar els canvis de reconfiguració (aplicant criteris variats, segons cada cas). Per satisfer aquesta necessitat, utilitzarem els *Pattern Models*. Aquests models defineixen, mitjançant un llenguatge específic de la llibreria UML2 (*Viatra Query Language*, VQL), patrons de cerca que permeten retornar elements (classes, instàncies, relacions, atributs, etc.) dins un diagrama UML[31].

A la figura 8.4 podem veure un exemple de *Pattern* aplicat al nostre cas d'estudi. Sense entrar en els detalls específics de la sintaxi, aquest patró anomenat *twitterTimeSlot* retorna tots els objectes de tipus *Slot*, que en l'especificació UML2 venen a ser els atributs d'una instància d'una classe, que compleixen dues característiques:

que reben per nom *timeSlot* (1) i que són atributs d'una instància de la classe amb nom *twitter* (2). Així, quan vulguem aplicar de forma dinàmica adaptacions sobre el sistema, i concretament sobre els *timeSlots* (1) de les instàncies de monitoratge del monitor de Twitter (2), podem referenciar aquests objectes amb la màxima eficiència, utilitzant les eines que la pròpia llibreria ens dona per treballar dinàmicament amb models UML.

De manera similar a les FC, necessitarem definir un *pattern* diferent per cada cas d'adaptació, d'acord amb els criteris específics. Tot i que això no exclou que alguns siguin reutilitzables entre diferents reconfiguracions. El potencial del VQL ens permet generar, amb una sintaxi relativament senzilla, patrons de cerca de tot tipus i que filtrin la cerca segons tot tipus de criteris: valors dels atributs de les instàncies, tipus de configuracions, relacions amb superclasses, etc.

8.2.5 Profile Model

A partir dels *patterns* definits, som capaços d'obtenir i referenciar aquests objectes UML sobre els quals aplicar adaptacions. Un cop obtinguts, a aquests elements se'ls aplica un rol o *stereotype* específic, que ens servirà bàsicament per distingir entre diferents tipus d'adaptacions dins d'una mateixa adaptació. És a dir: suposem que, dins una mateixa adaptació, volem actualitzar per una banda les configuracions dels processos de monitoratge del monitor de Twitter amb un valor x i, per altra banda, les del monitor de GooglePlay amb un valor y. Un cop l'Adapter computi les adaptacions i obtingui els elements a adaptar, es comunicarà amb el Model Adapter per sol·licitar-li l'aplicació d'una acció sobre els elements trobats identificats amb un rol específic. Aquesta adaptació composta queda fora del nostre abast, però és important introduïr per entendre el disseny genèric dels nostres models d'adaptació. Per la tasca d'etiquetar models, utilitzarem els anomenats *Profile Models*, que són models que defineixen una classificació de rols o *stereotypes* a aplicar sobre un cert grup d'elements UML segon el seu tipus i les seves característiques [32].

Pel nostre context, simplement identificarem un *stereotype* que anomenarem *jointpoint*, i ens servirà per etiquetar, dins el *Base Model*, aquells elements (bàsicament, *slots*) que requereixen ser actualitzats. A la figura 8.5 podem veure el *Profile Model* utilitzat a SUPERSEDE per l'adaptació de models. No ens centrarem en la seva sintaxi a fons; únicament ens centrarem en el rol *jointpoint*, que segons modela el *Profile Model*, pot ser aplicat a qualsevol objecte UML que implementi *Element* (que, segons la documentació de UML2, són tots aquells elements del diagram UML).

8.2.6 Aspect Model

Aquest tipus de model es tracta d'un tipus de document dissenyat i implementat per *partners* del projecte SUPERSEDE. Com a tal, el projecte inclou una llibreria capaç de processar i interpretar dinàmicament aquest model i les seves referències. Ens referirem a ell com a *Aspect Model* o *Adaptability Model*, ja que bàsicament es tracta d'un fitxer de format similar a un JSON que, des d'un punt de vista semàntic, descriu adaptacions específiques interrelacionant els elements que intervenen en una adaptació del sistema. Aquests aspectes inclouen:

- **FeatureId**. Referencia la *feature* d'un *Feature Model* sobre la qual descriu una adaptació específica.

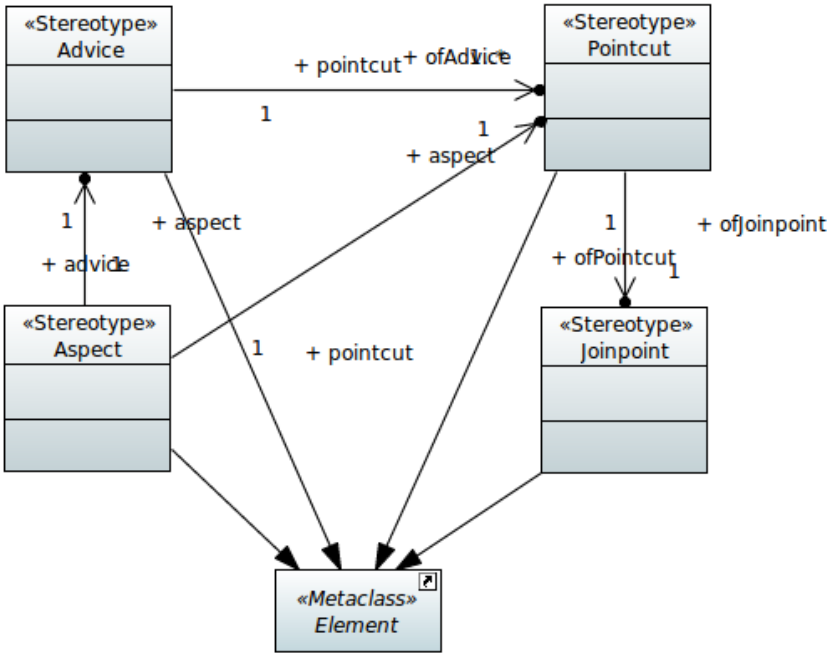

FIGURA 8.5: Exemple de Profile Model del sistema de monitoratge

- **Pointucts**. Llistat de punts on s'han d'aplicar canvis per aquella reconfiguració. Per cada *pointcut* que definim, s'especifica:
  - **Pattern**. El *pattern* utilitzat per buscar dinàmicament els elements al diagrama UML
  - **Role**. El rol o *stereotype* amb el qual s'etiquetaran aquests elements
- **Compositions**. Llistat de les adaptacions a realitzar sobre els punts o elements anteriorment identificats. Per cada *composition* que definim, s'especifica:
  - **Feature enabled**. Indica si s'ha d'aplicar quan la *Feature Configuration* indica que aquella *feature* passa a estar activada o desactivada
  - **Jointpoint role**. Indica el rol que han de tenir els elements sobre els quals s'han d'aplicar el canvi definit per aquesta *composition*. En aquest punt entraria en joc la composició d'adaptacions introduïda anteriorment al explicar els *Profile Models*.
  - **Action**. Indica el tipus d'acció a aplicar. En distingirem 2 tipus:
    - **Update**. Serà la operació en la qual ens centrarem per validar el nostre sistema. Consisteix en actualitzar el valor d'un atribut (*slot* en termes de modelació en UML2)
    - **Add/Delete/Replace**. Tot i que queden fora de l'abast de validació del nostre projecte, les afegirem al disseny i desenvolupament del projecte davant les necessitats del desenvolupament a SUPERSEDE. Es tracta d'operacions que, en referència a un *Advice Model* afegeixen, eliminen o substitueixen els elements UML presents a l'*Advice Model* al punt d'injecció definit com a *jointpoint* al *Base Model*.

```
aspect timeslottwitter {
    feature MonitoringSystem.MonitoringSystem.Social_Networks.Twitter.timeSlot,
    pointcuts{
        pointcut Twitter{
            pattern eu.supersede.dynadapt.usecases.patterns.twitterTimeSlot,
            role ADM.Joinpoint
        }
    },
    compositions{
        composition updateTimeSlot{
            feature_enabled true,
            jointpointRole ADM.Joinpoint,
            action update value '50'
        }
    }
}
```

FIGURA 8.6: Exemple de Pattern Model del sistema de monitoratge

A la figura 8.6 podem veure un exemple de model d'adaptabilitat amb els paràmetres anteriorment definits. En aquest cas, l'operació a aplicar és *Update*, ja que serà la que ens interessarà pel cas d'estudi.

Aquesta es tracta d'una aproximació genèrica inicial als models d'adaptabilitat. Per entendre el seu funcionament exacte i aprofundir en la sintaxi i les seves implicacions, en el següent capítol descriurem l'algorisme d'adaptabilitat i reconfiguració de monitors per entendre precisament com aquesta adaptació funciona amb els nostres models.

9 Sistema d'adaptabilitat

Els models UML prèviament descrits ens permeten dissenyar i modelar de forma estandarditzada el sistema de monitoratge. Proporciona, semànticament, tots els recursos necessaris per traduir propostes de configuracions de sistemes en accions reals sobre els monitors implementats.

El següent pas és el disseny i implementació d'un sistema que, a partir dels models anteriors, i de tot el domini que defineixen, sigui capaç de gestionar la persistència dels models, obtenir-ne els necessaris per aplicar reconfiguracions, actualitzar els models d'acord a aquestes modificacions, i traduir la informació implícita als models en accions de reconfiguració reals a enviar al sistema de monitoratge.

Procedirem a presentar els diferents components que componen aquest sistema.

9.1 Model Repository

El primer component que cal introduir per començar a entendre el *workflow* del sistema d'adaptabilitat és el **Model Repository**. Aquest component actua en termes genèrics com a un repositori que gestiona la persistència del conjunt de models UML que intervenen en el modelatge del sistema i la seva reconfiguració. Això, per tant, implica tot el conjunt de models descrits anteriorment: *Base Model*, *Feature Model*, *Feature Configuration*, *Pattern Model*, *Profile Model* i *Adaptability Model*.

9.1.1 Especificacions tècniques i funcionals

Principalment, les funcionalitats que ha de satisfer són les següents:

1. Gestionar la persistència dels models en disc
2. Mapejar i encapsular l'estructura de directoris definida per estructurar els models segons el seu tipus
3. Encapsular els mètodes CRUD bàsics per la gestió dels models, aïllant la lògica interna de la resta de components
4. Encapsular mètodes extensius que permetin obtenir models sota unes certes característiques

D'aquesta manera, podem concebre aquest component com una abstracció entre la naturalesa interna dels models i la lògica associada a la càrrega/descàrrega de fitxers amb el nostre sistema, assignant la responsabilitat d'aquests primers punts al Model Repository.

Una primera aproximació que ens podríem plantejar seria implementar un component unitari que definís un controlador (que després s'exposaria com a servei per

integrar a IF) amb tots els mètodes necessaris per la gestió dels models. Si al nostre sistema únicament resultés d'interès les operacions bàsiques de models, aquesta seria una bona alternativa, ja que estaríem simplificant l'arquitectura del sistema, i hi hauria poc marge per la millora. Però si anem un pas més enllà, i plantegem les necessitats de reconfiguració i adaptabilitat del nostre sistema, veiem que un controlador basat en operacions CRUD únicament ens permetria definir reconfiguracions on tots els models que intervenen en l'adaptació són triats estàticament, no dinàmicament. És a dir: el potencial que oferiria el Model Repository seria referenciar els models implicats en una adaptació mitjançant els seus identificadors, que haurien de ser introduïts manualment.

Això per tant implica que no podríem aplicar escenaris d'adaptació com per exemple:

Actualitzar el sistema de monitoratge amb la darrera Feature Configuration computada

Si tornem al cicle *MAPE-k*, suposant un sistema de ***P**lanificació* que produeix noves propostes de configuració (FC), aquest enviaria periòdicament aquests models al nostre sistema d'adaptabilitat. Si el nostre sistema és capaç de consultar les dates en què aquestes propostes es van afegint, és capaç de computar quina és la darrera afegida. I en definitiva, és capaç de computar, de forma automàtica i sense necessitat de donar-li cap mena d'informació, quins canvis aplicar al sistema de monitoratge.

Aquest escenari és clau pel nostre projecte. L'automatització del nostre sistema ve donada precisament gràcies a la persistència dels models UML i a l'actualització d'aquests, responsabilitat d'un altre sistema, a partir dels quals aquest és capaç de llegir, processar i traduir en modificacions reals sobre les activitats de monitoratge. Aquest escenari, però, és només un exemple de possible escenari d'adaptabilitat, basat en un criteri com la data, que serà el que nosaltres farem servir pel desenvolupament del projecte. Però el potencial està en veure que els criteris poden ser diversos, sempre i quan es treballin amb metadades que podem extreure a partir dels models, com és el cas de la data de generació de les propostes de reconfiguració.

Addicionalment a aquest problema, alguns models tenen dependències amb altres models UML. El cas més evident, l'*Adaptability Model*, té dependències amb *Feature Configurations*, que alhora tenen dependències amb *Feature Models*, i també amb *Pattern Models* i *Profile Models*. La resolució i gestió d'aquestes dependències pot arribar a ser extremadament complicada si, al carregar un d'aquests models, necessitem computar i resoldre quines són aquestes dependències cada vegada que volem carregar el Model Repository a memòria per executar una adaptació des del component encarregat d'aquest aspecte, l'Adapter, que satisfent el criteri de distribució del nostre sistema pot no tenir accés al mateix repositori en disc que el Model Repository.

Partint d'aquestes necessitats, és evident que necessitem estendre els mètodes de lectura de models a mètodes més complerts, on utilitzem dades auxiliars per fer cerques dins el nostre repositori. És en aquest punt quan aquesta primera aproximació resulta ineficient: si volem accedir a les metadades dels fitxers dels models, tals com la data de creació, l'autor, el sistema que l'ha computat, etc. (més endavant les veurem amb més detall), carregar tots els fitxers dinàmicament per accedir a aquestes dades per fer la cerca resulta molt ineficient. És per això que, alternativament, utilitzarem la següent arquitectura per gestionar la persistència de models:

- **Model Repository Manager.** Component que gestiona les metadades dels models, emmagatzemades en una base de dades relacionals, i que estén un controlador amb els mètodes de cerca per obtenir les metadades dels models desitjats.
- **Model Repository Client.** Component que es comunica amb el Model Repository Manager per obtenir les dades dels models, gestiona la persistència del repositori de models, i encapsula els mètodes i objectes per obtenir i treballar amb els models associats a les reconfiguracions.

D'aquesta manera, el primer assumeix les responsabilitats de cerca i gestió de models en base a les seves metadades, i el segon assumeix la responsabilitat principal d'encapsular programàticament l'accés als models, per tal que la resta de components del sistema d'adaptabilitat puguin aïllar-se de la lògica interna d'aquest punt.

9.1.2 Model Repository Manager

Satisfent les necessitats del Model Repository Manager, necessitem contemplar els següents punts:

1. Disseny i implementació d'una base de dades relacional que emmagatzemi les metadades per cada tipus de model.
2. Disseny i implementació d'un component que accedeixi a la base de dades, i que exposi a través d'un controlador els mètodes de consulta i modificació dels models.

Disseny de la base de dades

En primer lloc necessitem definir quines seran les metadades que considerarem per cada model. Generalment, podem considerar que tots els models definits tindran les següents dades:

- **Id.** Identificador d'aquell model, únic pel tipus de model (*Base Model*, *Feature Model*...) que representa.
- **Name.** Nom del fitxer del model (sense l'extensió).
- **AuthorId.** Identificador de l'autor del model.
- **CreationDate.** Data de creació del model.
- **LastModificationDate.** Data de la darrera modificació del fitxer del model.
- **FileExtension.** Extensió del fitxer (.uml, .vql, .yamfc ...)
- **SystemId.** Utilitzat dins el context SUPERSEDE per identificar els models que corresponen als diferents escenaris. En el nostre cas, aquest sempre serà *MonitoringReconfiguration*.
- **RelativePath.** Ruta relativa del directori *root* on s'emmagatzema el model.
- **Dependencies.** Llistat d'identificadors dels models dels quals depèn aquest model.

Tot i així, addicionalment existeix la possibilitat d'estendre atributs específics pels diferents models. Considerarem útils pel nostre context els següents:

- **Base Model**
  - **Status.** Indica si el model ha estat computat per un sistema extern (*Computed*), si és el resultat d'una adaptació dins el sistema d'adaptabilitat (*Enacted*), o bé si ha estat dissenyat manualment (*Designed*).
- **Feature Configuration**
  - **Status.** Indica si la *Feature Configuration* ha estat computada per un sistema extern (*Computed*), si s'ha aplicat com a adaptació dins el sistema (*Enacted*), o bé si ha estat dissenyada manualment (*Designed*).
- **Adaptability Model**
  - **Feature Id.** Identificador de la *feature* referenciada per l'Adaptability Model.

A partir d'aquestes dades podem procedir al disseny de la base de dades del repositori de metadades. Al existir atributs comuns i atributs diferenciats, hem de decidir quin tipus d'herència apliquem a la base de dades. Recordant les tres opcions, aplicades al nostre cas obtindríem el següent:

- ***Single table inheritance.*** Definir una única taula a la base de dades *Model* que inclogui tots els atributs possibles, inclosos els específics, i prengui valors nulls per aquells que no tenen aquell atribut.
- ***Class table inheritance.*** Definir una taula genèrica *Model* i N taules addicionals per cada tipus que referenciïn la primera, amb els atributs addicionals per cada cas.
- ***Concrete table inheritance.*** Definir una taula per cada tipus de model i replicar els atributs comuns.

En el nostre cas, optarem per l'opció *concrete table inheritance*. La raó principal d'aquesta opció és que, tot i compartir la major part de les dades, les entitats de models amb tipus diferents mai tindrà sentit contemplar-les conjuntament. És a dir: qualsevol lectura o modificació de models es farà sobre un model (o conjunt de models) d'un tipus específic, mai sobre models de forma genèrica (no tindrà sentit concebir l'entitat *Model* abstracta). Si considerem la segona opció, veiem que seria molt ineficient, ja que caldria fer *joins* internes per obtenir les dades els models, i per la mateixa raó que la ja esmentada sabem que seria un cost innecessari. Per tant, optarem per definir una taula per cada tipus, mantenint així la independència de les metadades entre models. A l'apèndix **??** podem veure el disseny proposat de la base de dades, d'acord als 6 tipus de models definits.

D'aquesta manera, la identificació d'un model queda definida per **id + ModelType**, el primer com a atribut explícit de metadades i el segon com a metadada implícita derivada de la taula en la qual emmagatzemem el model.

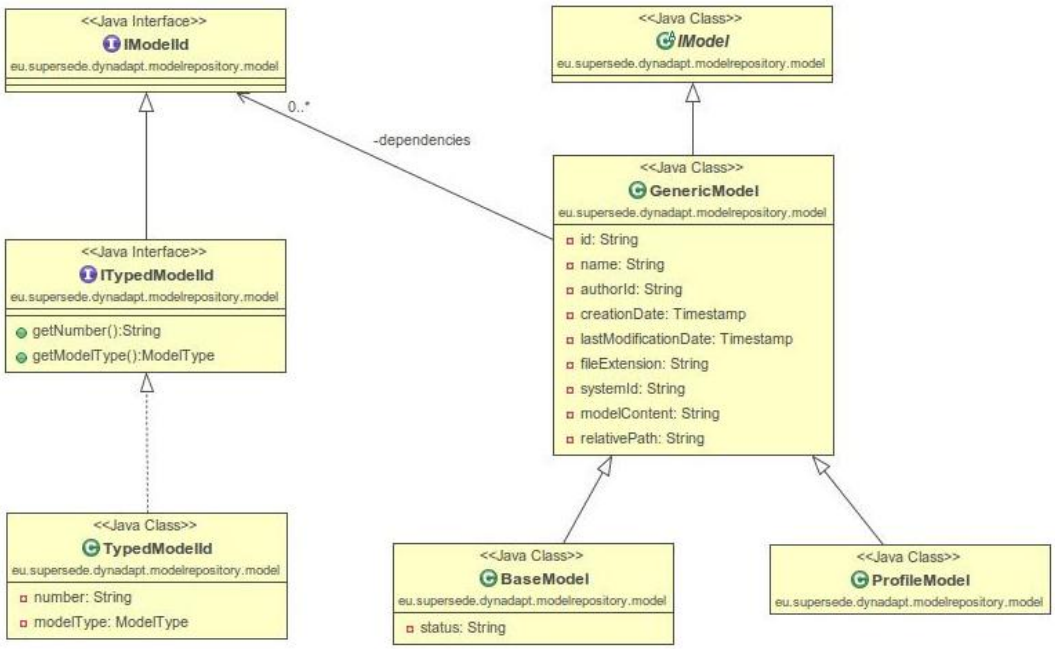

FIGURA 9.1: Disseny del domini del Model Repository Manager

Disseny i implementació

Modelat l'esquema de les metadades a la base de dades, podem procedir a implementar les classes de domini amb les quals treballarem per operar sobre aquestes. Partint del disseny anterior necessitem definir dos aspectes. En primer lloc, necessitem **modelar la jerarquia** de models, segons els atributs genèrics i els atributs específics de cada model. En segon lloc, definir com modelarem i identificarem el llistat de dependències de cada model.

A la figura 9.1 es presenta el diagrama simplificat de la proposta de disseny. Per la definició dels models, definim una classe genèrica *GenericModel* amb tots els atributs genèrics a tots els models. Addicionalment, s'implementa una classe per cada tipus de model específic (al diagrama apareixen *BaseModel* i *ProfileModel*, a mode d'exemple). Als atributs que apareixen al diagrama cal afegir els mètodes *getters* i *setters* tradicionals, no afegits per simplificació de l'esquema, així com els constructors. Per sobre de *GenericModel* definim una classe abstracta *IModel*, requisit establert per SUPERSEDE per futures extensions.

Paral·lelament es presenta una proposta de gestió de les dependències. Ja que els nostres models es troben identificats per la parella *id* + *ModelType*, es proposa la definició d'una interfície *IModelId* genèrica, totalment genèrica i sense cap mètode, oberta a possibles extensions i refactoritzacions necessàries pel context de SUPERSEDE. Per gestionar aquestes dependències pel nostre context, definim una nova interfície, *ITypedModelId*, que defineix els mètodes per extreure les dues dades necessàries per identificar un model, *getNumber* i *getModelType*. D'aquesta interfície definim una implementació, que obté aquests dos valors definits als mètodes anteriors mapejant directament aquests atributs. D'aquesta manera, el llistat de dependències dels models usats en la reconfiguració de monitors seran instàncies de la classe *TypedModelId* amb els dos atributs *id* i *ModelType*.

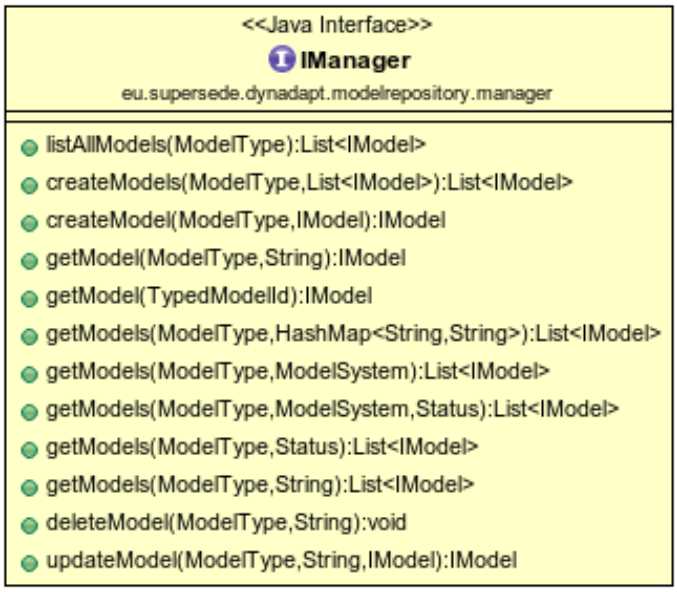

FIGURA 9.2: Disseny de la interfície del Model Repository Manager

Amb aquest domini podem procedir a implementar el controlador i, posteriorment, la seva exposició com a servei REST per la integració amb IF. Sense entrar en gaires detalls tècnics sobre aquesta part (molt similar a les anteriors exposicions com serveis), definirem les 5 operacions CRUD bàsiques, sempre considerant per *ModelType*. També serà responsabilitat del Model Repository Manager, associada a cadascuna d'aquestes operacions, la gestió del contingut dels models. Aquesta responsabilitat consistirà bàsicament en emmagatzemar en disc, segons on es trobi desplegat aquest component, els fitxers dels models, amb el contingut determinat. Aquesta persistència es gestionarà definint el contingut del model com un string, tant per indicar al Model Repository Manager el contingut a emmagatzemar, com per obtenir-lo quan es consultin les dades d'un mateix.

1. **Llista models d'un tipus.** Retorna les metadades de tots els models existents d'un tipus determinat.

2. **Obté un model.** Retorna les metadades i un string amb el contingut d'un model donat un identificador i un *ModelType*.

3. **Crea un nou model.** Emmagatzema un nou model amb les metadades passades al Model Repository Manager i guarda un fitxer a disc amb el contingut, el nom i l'extensió determinats a les metadades.

4. **Actualitza les metadades d'un model.** Donat un identificador i un *Model Type*, actualitza els valors de les metadades d'un model i el contingut, nom i/o extensió del fitxer (quan s'escaigui).

5. **Eliminar les metadades d'un model.** Donat un identificador i un *Model Type*, elimina la instància de metadades d'aquest model i elimina el fitxer del repositori.

A l'apèndix **??** es troba definida l'API utilitzada per la integració d'aquest component.

9.1.3 Model Repository Client

Aquest subcomponent del Model Repository s'encarrega d'actuar de pont entre el Model Repository Manager, que gestiona les dades/metadades dels models, i l'Adapter, component que utilitzarà el Model Repository per obtenir les instàncies dels models que necessita per gestionar les adaptacions de reconfiguracions. El disseny i la implementació d'aquest component han estat principalment realitzats per partners tercers del projecte, amb algunes col·laboracions i aportacions específiques per l'adaptabilitat dels models, especialment per la seva orientació a validar el cas d'ús de reconfiguració de monitors. Per tant, procedirem a explicar únicament aquells aspectes realitzats com a part del treball d'aquest TFG i els conceptes necessaris per entendre el seu funcionament.

Disseny i implementació

Des d'un punt de vista de **disseny**, aquest component defineix una interfície *IModelRepository* que defineix, per una banda, els mètodes CRUD per cadascun dels 6 tipus de models existents al sistema. Addicionalment, defineix mètodes de cerca més complexos en base a les metadades dels models. Aquests mètodes seran els que ens permetran executar les adaptacions de forma automatitzada, sense necessitat de definir models específics, a mode de l'exemple introduït anteriorment sobre la cerca de la darrera configuració computada. D'aquests mètodes, ens interessaran especialment els següents:

- **Obté el *Base Model* més actual** - Permet obtenir el darrer *Base Model* del sistema, que representa per tant l'estat actual del sistema de monitoratge, i ens servirà per agafar com a model per aplicar els canvis de reconfiguració
- **Obté la darrera *Feature Configuration* computada** - Obté la darrera *Feature Configuration* amb l'atribut *status* amb valor *Computed*. D'aquesta manera, podem obtenir aquella darrera proposta de configuració que no s'ha executat encara (i per tant, que no té l'atribut *status* amb valor *Enacted*)
- **Obté la darrera *Feature Configuration* executada** - Obté la darrera *Feature Configuration* amb l'atribut *status* amb valor *Enacted*. D'aquesta manera, podem obtenir la darrera configuració executada, i juntament amb l'anterior mètode, podem computar les diferències entre les dues.
- **Obté els *Adaptability Models* candidats** - Obté el llistat de *Adaptability Models* donat un *systemId*. En el nostre cas, ens interessar obtenir aquells amb sistema *MonitoringReconfiguration*, que representaran totes les adaptacions possibles a aplicar dins el nostre context.

La implementació d'aquesta interfície defineix la interacció amb el Model Repository Manager i la gestió de persistència dels models per tal de poder referenciar i treballar amb ells des del component Adapter. Podem veure-la definida a la figura 9.3. Degut a la seva extensió, i per major llegibilitat, fixem-nos que els primers mètodes es tracta dels 4 mètodes CRUD per cada *ModelType* dels quals n'hi ha 6 (i per tant, 24 mètodes amb les operacions bàsiques). A continuació trobem els mètodes anteriorment descrits, que utilitzarem per processar l'adaptabilitat de models i la posterior reconfiguració de monitors.

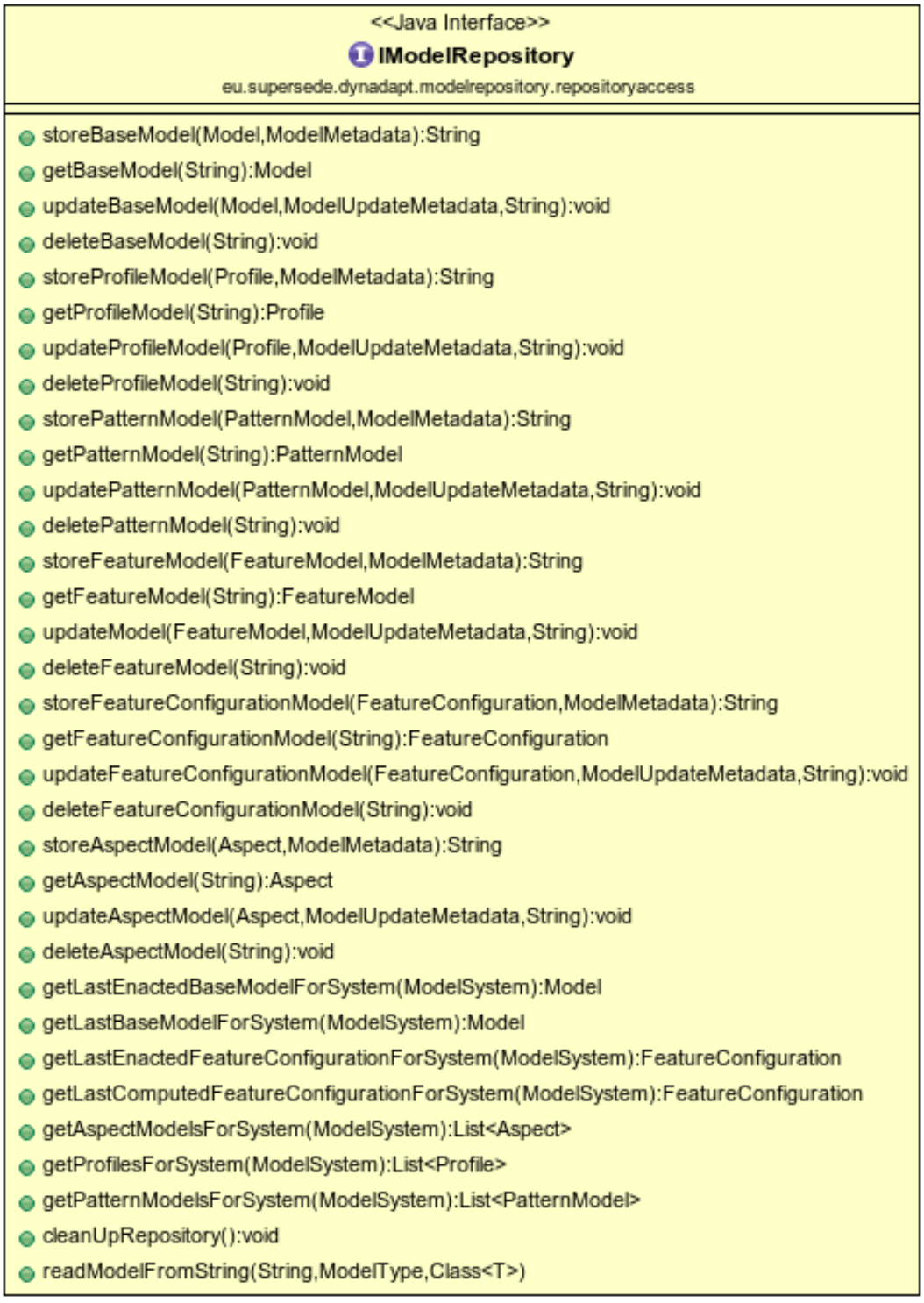

FIGURA 9.3: Disseny de la interfície del Model Repository Service

9.2 Model Adapter

L'adaptació dinàmica de models UML pot arribar a ser complexa, especialment si volem definir una adaptació de models que sigui prou genèrica com per a estendre-la a diferents tipus de models i diferents tipus d'operacions. No serà igual la modificació d'un diagrama de classes (com és el cas d'aquest projecte) que la modificació de, per exemple, un diagrama d'activitats. Per tal de permetre la seva integració a SUPERSEDE, per tant, cal que la lògica d'aquesta adaptabilitat segueixi una estructura prou abstracta que permeti estendre i implementar els diferents casos d'estudi. Per aquesta raó per desenvolupar el Model Adapter, component que assumirà la responsabilitat d'aquesta adaptació dinàmica, treballarem per una banda el disseny genèric del component, i per altra banda la implementació de la lògica necessària pel seu ús al cas de reconfiguració de monitors.

9.2.1 Especificacions tècniques i funcionals

El primer que necessitem és definir les diferents operacions que el Model Adapter pot suportar. D'acord amb allò ja definit al *Capítol 8. Modelatge de configuracions*, els *Adaptability Models*, que descriuen el comportament d'aquestes adaptacions, suporten 4 tipus d'operacions:

1. **ADD**. Donat un *BaseModel*, afegeix al punt d'injecció *jointpoint* els elements presents a *AdviceModel*.
2. **DELETE**. Donat un *BaseModel*, elimina del punt d'injecció *jointpoint* els elements presents a *AdviceModel*.
3. **REPLACE**. Donat un *BaseModel*, substitueix el punt d'injecció *jointpoint* amb els elements presents a *AdviceModel*.
4. **UPDATE**. Donat un *BaseModel*, actualitza amb el valor definit els elements (atributs, o *slots*) referenciats.

Aquestes operacions estan contemplades des de l'abstracció del tipus de model que volem adaptar. Independentment d'aquest, si analitzem les 4 operacions, veiem que per una banda les 3 primeres presenten un comportament similar: utilitzen la mateixa informació, i la única variació és quin és el tractament intern. La 4a en canvi (la que serà el nostre cas d'estudi) es diferencia tant per les dades que necessita com pel seu comportament intern, ja que es tracta de la modificació de valors d'atributs d'elements del model. Partint d'aquesta idea, per tant, podem discernir entre dos tipus d'operacions: una orientada a l'adaptació complexa de models (ADD, DELETE, REPLACE) i l'altra a l'actualització de valors (UPDATE).

9.2.2 Disseny i implementació

A la figura 9.4 podem veure el disseny de dues interfícies: una genèrica *IModelAdapter* que exposa els dos mètodes principals, i una *Composable* que defineix els 3 tipus d'adaptacions compostes segons el tipus d'element UML que estem modificant. Addicionalment, definim una *ComposableFactory* que actuarà com a factoria de les diferents implementacions de *Composable*. Aquest disseny queda justificat pel fet que l'Adapter, el component encarregat d'ordenar al Model Adapter l'adaptació de

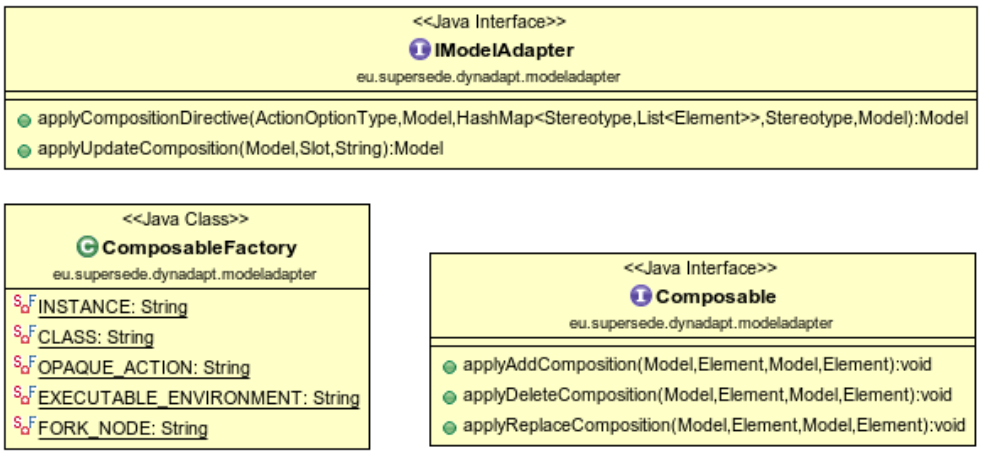

FIGURA 9.4: Disseny de les interfícies del Model Adapter

models, ha d'estar el màxim desacoblat de la lògica específica de l'adaptació de models. Ens veiem forçats a discernir entre UPDATE i la resta degut a la seva diferència des del punt de vista de requisits; però com que no és el cas amb les altres 3 accions, aquestes les encapsulem amb un sol mètode a la interfície *IModelAdapter*, que serà la que aquest Adapter invocarà.

Aquesta interfície *Composable* ens permet exposar els mètodes d'adaptabilitat de models segons cadascun dels 3 tipus d'operacions compostes. En tots casos, els paràmetres rebuts són els mateixos:

- *Base Model* on s'han d'aplicar els canvis
- Element etiquetat com a *jointpoint* (o amb el rol corresponent) al *Base Model*
- *Advice Model* que conté els canvis a aplicar
- Element etiquetat com a *jointpoint* (o amb el rol corresponent) al *Base Model*

Per comprendre millor el disseny i implementació de l'arquitectura genèrica del Model Adapter, la figura 9.5 mostra el diagrama de seqüència de la operació *applyCompositionDirective*, que rep per paràmetre:

- *actionType* - Instància del tipus d'acció (ADD, DELETE, REPLACE) a executar
- *baseModel* - Instància del *Base Model* on s'han d'aplicar els canvis
- *elements* - *Hashmap* identificat per *role* (amb el que s'han etiquetat) amb tots els elements que intervenen en l'adaptació
- *adviceRole* - *Role* amb el que està etiquetat l'element a l'*Advice Model*
- *variantModel* - Instància del *Variant Model* que conté els canvis a aplicar

Amb aquest disseny podem estendre els casos d'ús del Model Adapter segons la necessitat d'adaptació de model de cada cas d'estudi del sistema. Al definir amb interfícies les 4 operacions bàsiques, així com els paràmetres d'entrada i la lògica des que rep una petició fins que identifica i ressol la operació a aplicar, només necessitem estendre la implementació per personalitzar les operacions que aplica sobre el

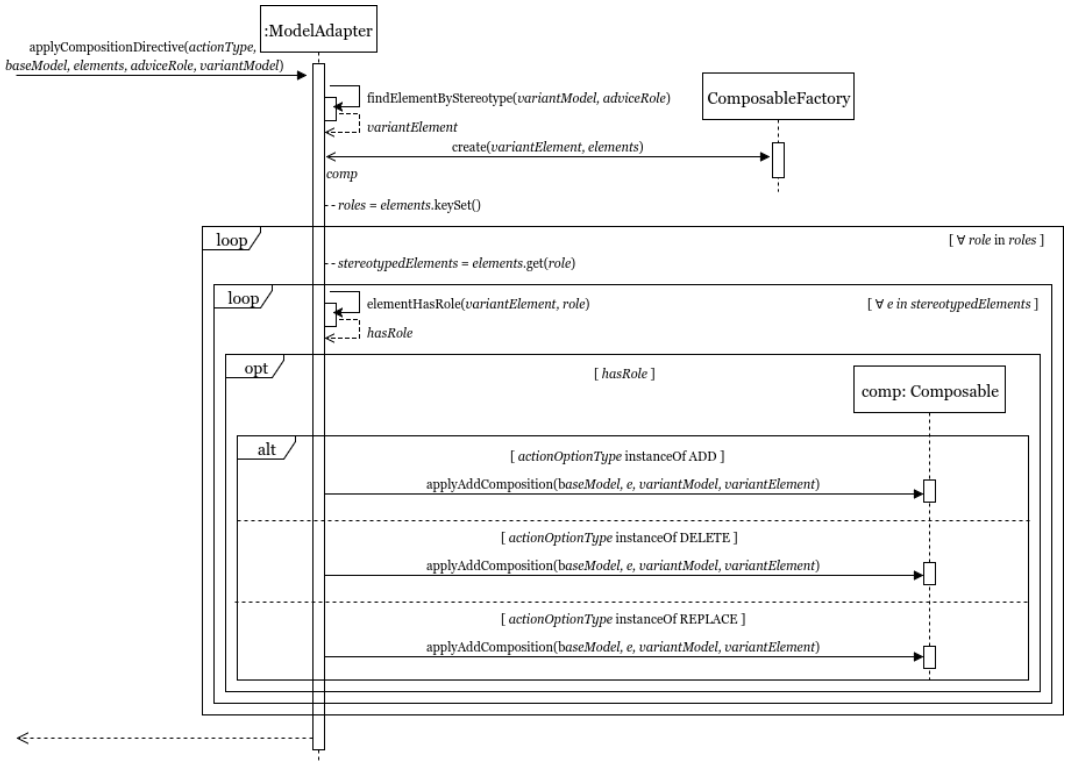

FIGURA 9.5: Diagrama de seqüència de l'adaptació composta del *Model Adapter*

model per aplicar les 3 operacions ADD, DELETE i REPLACE.

Utilitzant el disseny definit a la figura 9.4,podem estendre la implementació del Model Adapter per tal que treballi amb models UML de tipus diagrama de classes. Amb aquest objectiu, necessitem identificar els dos elements UML bàsics amb els quals treballarem: **classes** (p.e. la inserció d'un nou monitor) i **instàncies** (p.e. l'eliminació d'una configuració dins el sistema). Per tant, necessitem dues implementacions de la interfície *Composable*: *ComposableClass* i *ComposableInstanceSpecification*. A la figura 9.6 podem veure el disseny d'aquestes dues implementacions; bàsicament consisteix en la implementació dels 3 mètodes per cada tipus d'operació.

La implementació interna d'aquestes classes s'ha realitzat de forma conjunta amb altres *partners* del projecte, i en qualsevol cas, al no formar part del cas de validació d'aquest TFG, sinó d'una base a partir de la qual plantejar nous escenaris d'extensió i treball futur, no aprofundirem més en els detalls d'aquesta.

9.3 Components auxiliars

Per treballar dinàmicament amb models UML utilitzant llibreries i plugins d'Eclipse (com UML2 i Papyrus respectivament), necessitem algunes funcionalitats addicionals a les implementades al Model Adapter. Al tractar-se de components reutilitzats i a implementats al projecte SUPERSEDE per altres col·laboradors del projecte, els

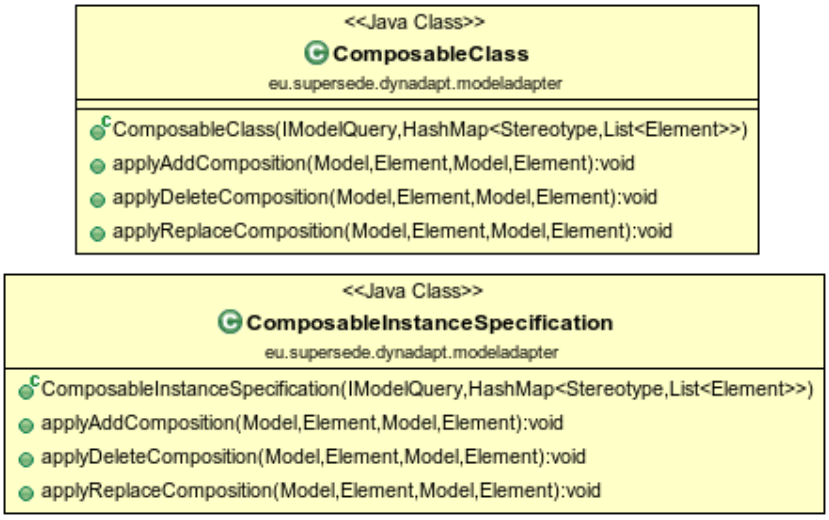

FIGURA 9.6: Implementacions de la interfície *Composable* per *Class* i *InstanceSpecification*

FIGURA 9.7: Disseny de la interfície del Model Query

presentarem per tal d'entendre el seu objectiu, les seves característiques i com encaixen dins el nostre projecte, però sense comptabilitzar-los com part del desenvolupament del treball. Aquests dos components seran el **Model Query** i l'**Enactor**.

9.3.1 Model Query

El **Model Query** consisteix en un component de gestió de models UML orientat a la cerca d'elements (classes, instàncies, relacions, atributs...) dins d'aquests models. Aquesta cerca s'utilitza combinant UML2, llibreria utilitzada per la càrrega dinàmica de models, amb VQL, utilitzat per definir patrons de cerca dins de models UML.

Aquest component ens serà útil a l'hora de buscar al *Base Model* que volem adaptar aquells elements que volem modificar (és a dir, p.e. els *slots* o atributs el valor dels quals hem de modificar-ne el valor). A la figura 9.7 podem veure la definició de la interfície que implementa Model Query, amb un mètode *query(Pattern)* que rep com a paràmetre el *pattern* amb el qual realitzarà la cerca. La nostra tasca serà, per tant, durant el procés d'adaptabilitat, obtenir el *pattern* que ens interessa (que, si recordem, es troba definit a l'*Adaptability Model*) i cridar a una instància d'aquest component inicialitzada amb el *Base Model* corresponent, que ens retornarà com a resultat una *Collection* amb elements UML trobats pel *pattern*. Al tractar-se d'un component ja implementat fora de l'abast del projecte, no entrarem en els detalls tècnics del seu funcionament.

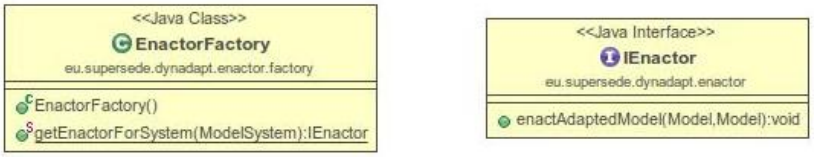

FIGURA 9.8: Disseny de la interfície de l'Enactor i EnactorFactory

9.3.2 Enactor

El component **Enactor** és un component orientat a la integració i comunicació entre el sistema d'adaptabilitat i el sistema de monitoratge. La seva tasca principal és rebre com a *input* dos *Base Model*, corresponents a l'actual del sistema abans de la reconfiguració i el nou *Adapted Model* computat de l'adaptació de models, i mitjançant la comparació d'aquests dos models, extreure i construïr peticions de reconfiguració que enviarà posteriorment al component d'entrada del sistema de monitoratge: l'Orchestrator.

L'Enactor es tracta d'un component genèric amb característiques pròpies de cada cas d'estudi. Dins el nostre context, l'Enactor ha de ser capaç de computar diferències entre diagrames de classes UML i traduïr-les a reconfiguracions de monitors. Però dins SUPERSEDE aquesta traducció pot ser amb models d'un altre tipus (p.e., diagrames d'activitat) a peticions de reconfiguració d'altres sistemes. Per aquesta raó, es defineixen dos components:

- **EnactorFactory.** Classe que encapsula els Enactors de cada *ModelSystem* definit a SUPERSEDE, inclòs *MonitoringReconfiguration*, i que a partir d'aquest paràmetre permet retornar una instància d'Enactor.
- **Enactor.** Component encarregat de la traducció de models a peticions de reconfiguració del sistema corresponent. La lògica interna d'aquest dependrà de cada *ModelSystem*.

Com en el cas anterior, ignorarem els detalls tècnics interns i ens centrarem únicament en la seva usabilitat. A la figura 9.8 trobem definida la interfície que implementa l'Enactor i la classe EnactorFactory.

9.4 Adapter

Satisfets els requisits tècnics i funcionals del Model Repository disposem d'un component que ens permet carregar dinàmicament tots els models i utilitzar els mètodes que la llibreria UML2 defineix amb els mateixos. Paral·lelament, el Model Adapter ens satisfà la necessitat d'adaptació dinàmica de diagrames de classe UML. El següent pas és definir el component que assumirà la responsabilitat de **gestionar i computar les reconfiguracions**. Aquesta responsabilitat serà assumida per l'Adapter, la responsabilitat principal del qual serà la interacció amb el Model Repository i el Model Adapter per aplicar l'algorisme que a continuació definim per computar reconfiguracions.

9.4.1 Especificacions tècniques i funcionals

L'**algorisme d'adaptabilitat** de models, o bé reconfiguració de monitors pel nostre cas d'estudi, consisteix en la implementació d'un algorisme que resolgui la següent problemàtica:

*Davant una petició de reconfiguració del sistema, calculem totes les diferències entre la **darrera Feature Configuration aplicada** i la **darrera Feature Configuration computada** i trobem totes les diferències de seleccions. Per cada diferència (és a dir, per cada **selection** present a una de les dues **Feature Configurations** però no a l'altra), apliquem les **accions definides als Adaptability Models** corresponents al **Base Model**, segons si aquesta s'activa a la nova configuració (apareix a la nova FC) o bé es desactiva (no apareix).*

Per entendre millor aquest plantejament, anem a analitzar pas per pas quins són els passos que s'han de realitzar per passar d'identificar una *feature* a modificar el *Base Model*:

1. Obtenim la **darrera *Feature Configuration*** amb *status = Computed*, que representa la última configuració del sistema proposada
2. Obtenim la **darrera *Feature Configuration*** amb *status = Enacted*, que representa la última configuració del sistema aplicada
3. Obtenim el **darrer *Base Model* del sistema**, que representa l'estat actual del sistema de monitoratge
4. Computem totes les diferències a nivell de **selections**, que apareixen només a una de les dues *Feature Configurations*.
5. Per cada *selection* diferent trobada:
   - 5.1. Obtenim els ***Adaptability Models* associats a la *feature*** que representa aquella *selection*
   - 5.2. Per cada *Adaptability Model* obtingut:
     - 5.2.1. Obté tots els ***pointcuts* definits per l'*Adaptability Model***
     - 5.2.2. Per cada *pointcut* obtingut:
       - 5.2.2.1. Obté el ***pattern* definit per aquell *pointcut***
       - 5.2.2.2. Utilitza el component *Model Query* per trobar tots els **elements al *Base Model*** segons el ***pattern* obtingut**
       - 5.2.2.3. Etiqueta amb el *role* definit pel *pointcut* tots els elements trobats pel *Model Query*
     - 5.2.3. Obté les ***compositions* definides per l'*Adaptability Model***
     - 5.2.4. Per cada *composition*:
       - 5.2.4.1. Comprova si aquella *composition* defineix l'**activació o desactivació** de la *feature*.
       - 5.2.4.2. En cas que coincideixi amb el valor de l'activació/desactivació de la *feature*, es comunica amb el Model Adapter per aplicar l'adaptació corresponent i obté el *Base Model* adaptat.
6. Actualitza el Model Repository amb el **nou *Base Model*** i la **nova *Feature Configuration***.

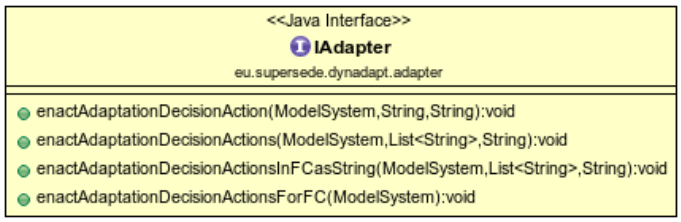

FIGURA 9.9: Disseny de la interfície de l'Adapter

9.4.2 Disseny i implementació

El disseny del component Adapter és relativament senzill en quant a arquitectura software, i el repte important és el disseny i la implementació de l'algorisme prèviament definit. Tot i així, la resta de components del sistema d'adaptabilitat han estat dissenyats i implementats per tal de fer que aquest disseny sigui el més senzill possible.

A la figura 9.9 podem veure la interfície definida per l'Adapter. Recordem que, la funció d'aquest component és realitzar l'adaptació de models de forma automàtica mitjançant un únic algorisme genèric. Aquest algorisme és el definit pel mètode *enactAdaptationDecisionActionsForFC(ModelSystem system)*. La resta de mètodes reben paràmetres addicionals ja que estan orientats a realitzar adaptacions no de forma totalment automatitzada, sinó definint paràmetres específics (p.e. definint una nova FC a computar, o bé unes *Actions* específiques). Per exemplificar i validar el nostre projecte, nosaltres ens centrarem en aplicar *Feature Configurations* específiques, per comprovar que els canvis definits satisfan allò especificat a la *Feature Configuration.*

A la figura 9.10 podem visualitzar el diagrama de seqüència que defineix l'algorisme prèviament explicat i la seva interacció amb els components que formen part del sistema d'adaptabilitat.

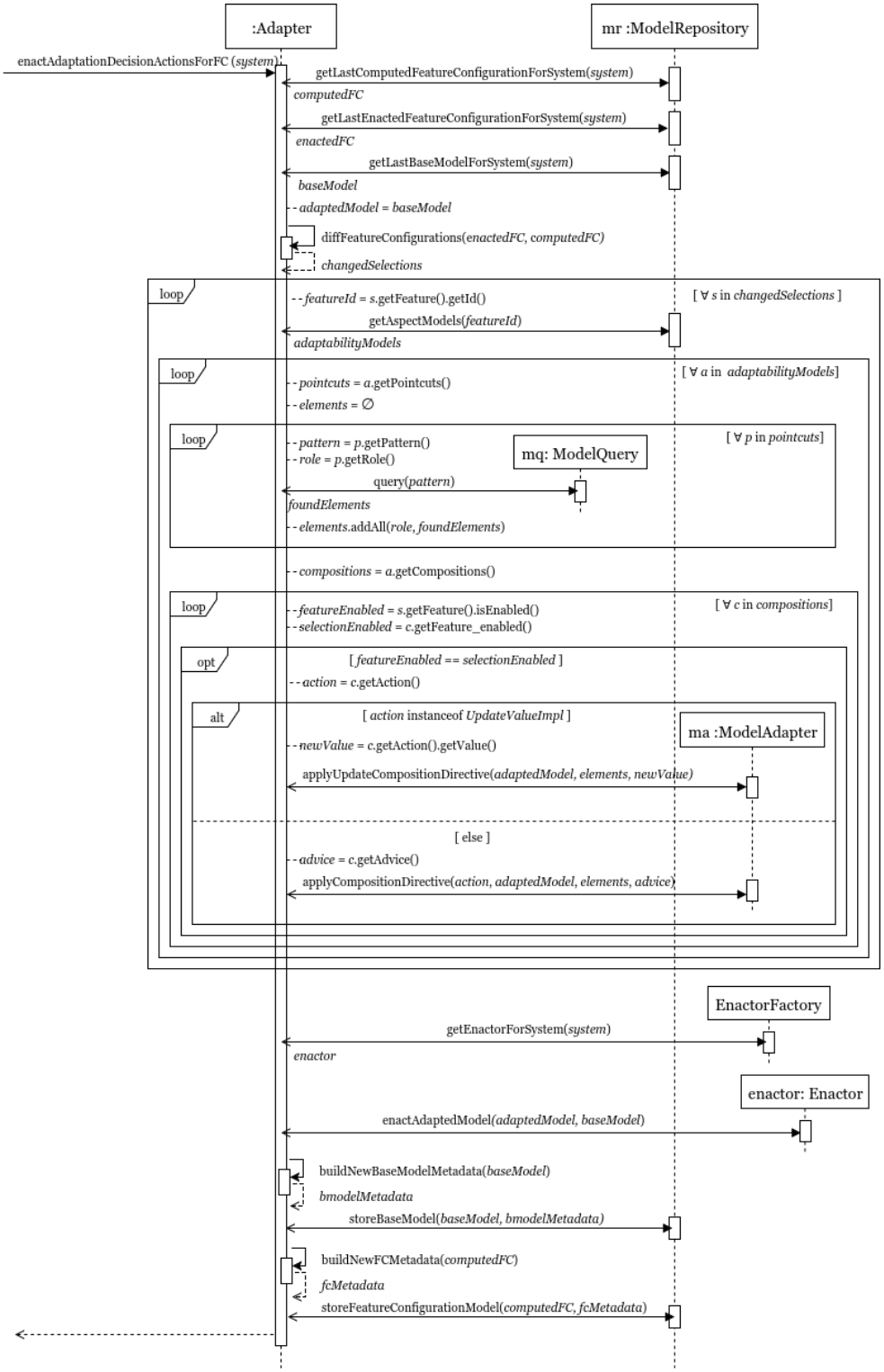

FIGURA 9.10: Diagrama de seqüència de la reconfiguració automàtica del sistema amb l'Adapter

10 Dashboard

Com a complement addicional al desenvolupament del projecte, es defineix la implementació d'un *dashboard* en mitjà d'aplicació web que ens permeti interactuar amb el nostre sistema per tal de, mitjançant una usabilitat molt bàsica i senzilla, poder executar reconfiguracions i visualitzar les diferents adaptacions que s'han realitzat. Buscarem, per tant, un disseny senzill que no requereixi una gran dedicació del projecte (ja que entenem que no és una prioritat), però si que ens permeti la interacció necessària per poder satisfer aquesta necessitat.

10.1 Anàlisi de requisits

Per satisfer aquest objectiu, volem que el nostre *dashboard* satisfaci les següents necessitats:

- **Visualitar propostes d'adaptacions.** A través dels models definits al Model Repository, permetre mostrar les dades principals de cada **Feature Configuration** emmagatzemada al nostre sistema.
- **Executar una adaptació.** A partir del llistat de *Feature Configurations* anterior, el *dashboard* ha de permetre a l'usuari seleccionar una proposta d'adaptació i demanar la seva execució al sistema corresponent. Aquesta serà la interacció principal, que ens permetrà executar els casos de validació de la reconfiguració del nostre sistema de monitoratge.
- **Visualitzar adaptacions executades.** Com en la 1a vista, permetre mostrar les dades principals de cada **Feature Configuration** amb *status* == *Enacted*, i per tant executada.

Des d'un punt de vista funcional, els requisits són relativament senzills, ja que traduït a necessitats de disseny i implementació únicament caldrà definir dues vistes i una interacció amb l'usuari, que es reprodueixi en una interacció amb l'Adapter. Aquesta interacció, com en la resta de casos, es realitza a través de l'eina IF.

10.2 Disseny de la interfície

Partint dels requisits anteriors, necessitem definir dues vistes a la interfície: la vista de **adaptacions suggerides** i **adaptacions executades**.

La vista d'adaptacions suggerides consisteix en una taula on cada fila representa una d'aquestes entitats (i, per tant, una *Feature Configuration* del sistema). Per cada instància, mostrarem les següents dades:

- **Adaptation id**. Identificador de la *Feature Configuration* que representa

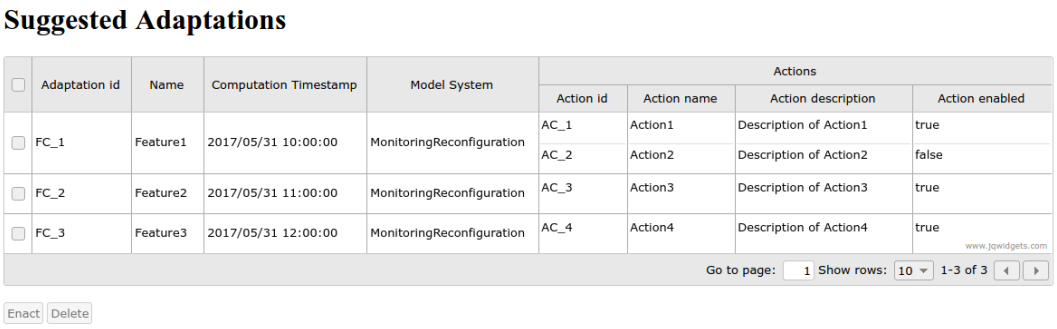

FIGURA 10.1: Disseny de la vista d'adaptacions suggerides del *dashboard*

- **Name**. Nom únic associat a aquella *Feature Configuration*, usualment associat semànticament a les seves característiques
- **Computation timestamp**. *Timestamp* de la creació de la *Feature Configuration*. Ens serveix per ubicar temporalment les diferents configuracions creades
- **Actions**. Cada *Feature Configuration* defineix una sèrie de *features*, o accions, que s'activen o desactiven en funció de les seves característiques. Per cada acció a aplicar, mostrarem:
  - **Action id**. Identificador de l'acció a realitzar
  - **Action name**. Nom únic per aquella *Feature Configuration* que rep l'acció o *feature* a modificar
  - **Action description**. Descripció textual que defineix la modificació i canvis implicats per aquella acció
  - **Action enabled**. Indica si l'acció indica l'activació d'una *feature* (true) o la seva desactivació (false).

Addicionalment, en aquesta vista cal contemplar el 2n requisit funcional d'**execució d'una adaptació**. Mitjançant la selecció d'una de les adaptacions suggerides al nostre sistema (és a dir, la selecció d'una fila a la taula), afegim un botó a la part inferior esquerra amb l'etiqueta *"Enact"*. Aquesta interacció activa la comunicació amb l'Adapter, i inicia l'algorisme d'adaptació de models (en aquest cas, no de forma completament automàtica, sinó donada una *Feature Configuration* específica). Com a resultat, una nova adaptació s'executa al nostre sistema; s'actualitzaran les dades al Model Repository, que podem consultar a través del *dashboard*, i es modificarà l'acció real dels components adaptats (monitors pel nostre cas), d'acord amb les modificacions anteriors.

En aquesta segona vista d'adaptacions executades, mostrarem de forma paral·lela al disseny anterior una taula on cada fila representa una *Feature Configuration*, però aquest cop que ja ha estat executada al nostre sistema. Mostrarem algunes de les dades bàsiques d'aquella *Feature Configuration*, com en el cas anterior, però afegirem les següents dades:

- **Enactment request time**. *Timestamp* que conté la data i hora en què s'ha realitzat la petició d'execució de l'adaptació

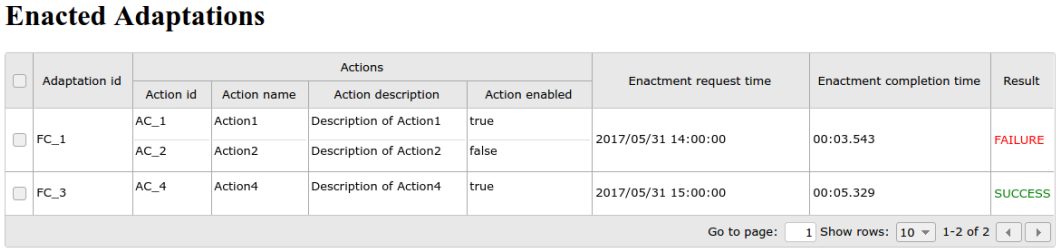

FIGURA 10.2: Disseny de la vista d'adaptacions executades del *dashboard*

- **Enactment completion time**. Mostra el temps total de l'execució d'aquesta reconfiguració, amb una precisió de mil·lisegons.
- **Result**. Indica si l'adaptació s'ha realitzat satisfactòriament (SUCCESS) o bé si s'ha produït algun error (FAILURE).

Complementàriament s'ha afegit l'opció d'eliminar una entitat (adaptació suggerida o executada) en cada vista, amb objectius orientats al desenvolupament i al testeig de l'aplicació.

Amb aquestes dades i amb la interacció definida, tenim un component bàsic usable que ens permetrà executar les reconfiguracions del sistema de monitoratge.

10.3 Implementació

La implementació del *dashboard* consisteix en l'extensió d'un projecte pare definit com el *front-end* del projecte SUPERSEDE. Aquest projecte consisteix en una aplicació web desenvolupada amb *Spring Framework*, que permet el desenvolupament d'aplicacions web i la gestió i *mapping* de persistència. Per desenvolupar el *dashboard* per la reconfiguració de models, implementarem un projecte que actuarà de submòdul dins aquest *front-end*.

Seguint l'arquitectura proposada pel desenvolupament d'aplicacions web amb Spring, a la figura 10.3 podem veure el disseny del domini disseny i implementat al *dashboard*. Primerament, cal definir les classes de domini que representen les diferents entitats presents al *dashboard*. Distingirem entre *Adaptation* com a entitat que representa una *Feature Configuration* suggerida, i *Enactment*, que representa una *Feature Configuration* executada. Addicionalment, i per millorar el disseny, definirem l'entitat *Action*, ja introduïda anteriorment, del qual una *Adaptation* en pot tenir vàries instàncies. Per cadascuna d'aquestes classes de domini (amb els atributs definits anteriorment), cal implementar una extensió de *JpaRepository*, que actuen com a controladors de la capa de domini [33]. Es tracta de classes que ofereix el framework Spring, i que defineix les operacions bàsiques per les classes de domini (operacions CRUD bàsicament).

Addicionalment, per implementar les vistes i l'aplicació web necessitem definir un controlador RESTful que s'utilitzarà per invocar els diferents mètodes del *dashboard* des de la pròpia aplicació. A la figura 10.4 podem veure els 3 controladors i les operacions implementades. Generalment aquestes estan orientades a test i validació,

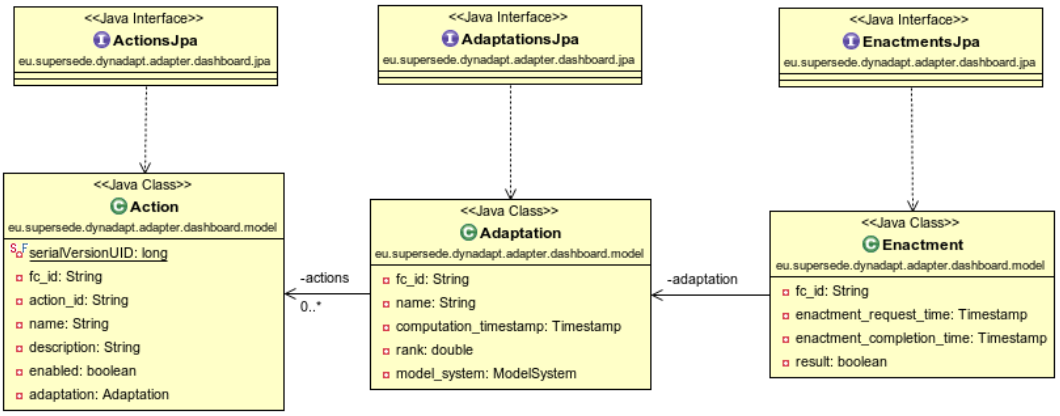

FIGURA 10.3: Disseny del domini del *dashboard*

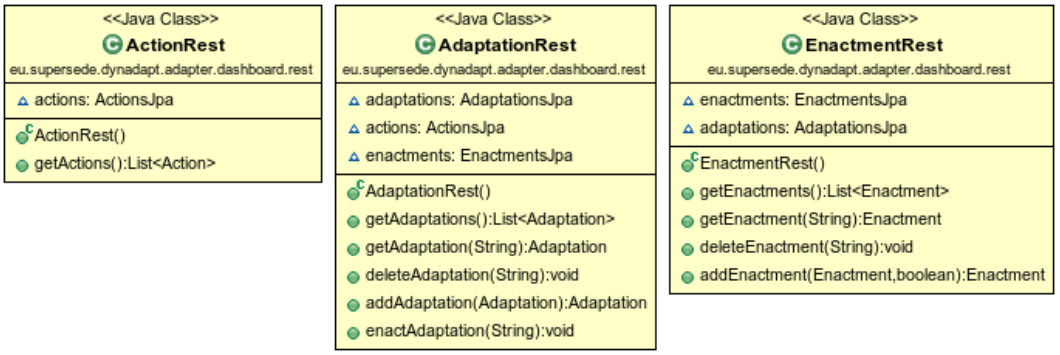

FIGURA 10.4: Disseny dels controladors REST del *dashboard*

i únicament 5 seran utilitzades dins el context de l'aplicació: *getAdaptations()*, *deleteAdaptation()*, *enactAdaptation()*, *getEnactments()* i *deleteEnactment()*. Cadascun d'aquests mètodes utilitza els repositoris implementats amb la lògica addicional necessària per satisfer la seva funcionalitat, com és el cas de l'operació *enactAdaptation*, que es comunica a través de IF amb l'Adapter.

11 Validació del sistema

Definits els components i detalls que componen el sistema de monitoratge i el sistema d'adaptabilitat, i un cop justificada des d'un punt de vista teòric la satisfacció dels nostres objectius, és el moment de presentar un exemple de cas d'ús i provar l'execució per validar el seu funcionament satisfactori i avaluar els resultats obtinguts.

11.1 Presentació de cas d'ús

Recordem la premissa genèrica del cas d'ús que volem que el nostre sistema satisfaci:

*Donada un **procés de monitoratge actiu** en un dels monitors del nostre sistema, i una proposta de **nova configuració** modelada, volem executar de forma automatitzada una **reconfiguració** d'aquell procés de monitoratge d'acord amb els **canvis computats respecte l'actual**.*

Per validar aquesta execució necessitem definir: un *Base Model* que modeli l'estat actual del sistema (figura 11.2, una *Feature Configuration* que modeli la darrera aplicada al sistema (figura 11.2, una *Feature Configuration* que modeli la nova proposta de configuració del sistema (figura 11.3) i un *Adaptability Model* que defineixi l'adaptació de models i la reconfiguració de monitors (figura 11.4).

Com podem veure, el *Base Model* defineix una configuració del sistema de monitoratge (ja vista anteriorment com a exemple en el *Capítol 8. Modelatge UML de les configuracions de monitors*) amb dues instàncies de processos de monitoratge corrent: una sobre el monitor de Twitter, i l'altra sobre el monitor de Google Play, ambdues configurades amb els seus propis paràmetres de configuració. Pel nostre cas d'estudi, suposarem que aquest és el darrer *Base Model* que defineix l'estat del sistema.

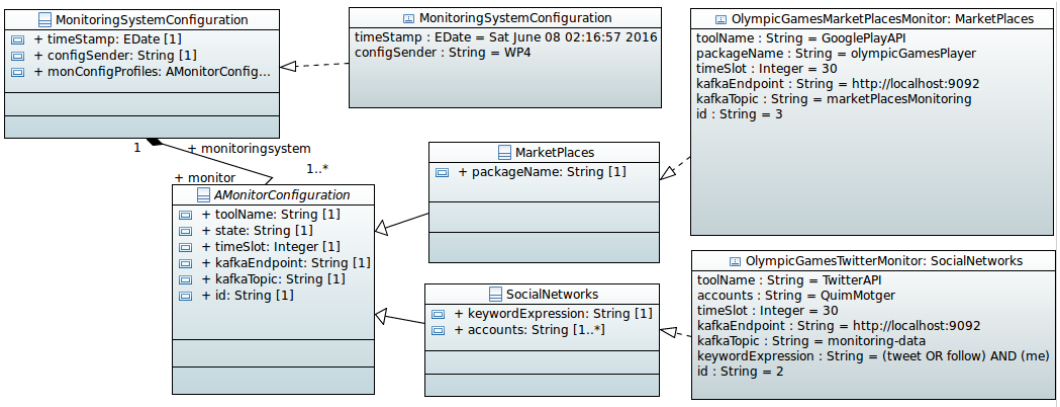

FIGURA 11.1: *Base Model* utilitzat per validar la reconfiguració del sistema

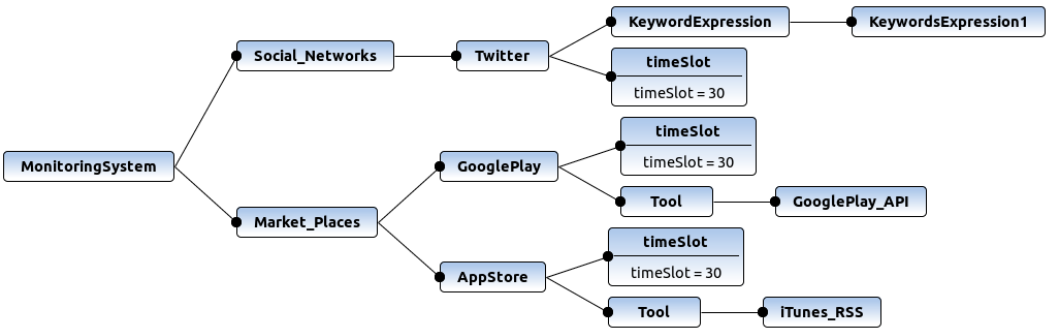

FIGURA 11.2: *Feature Configuration* que descriu la darrera configuració del sistema aplicada

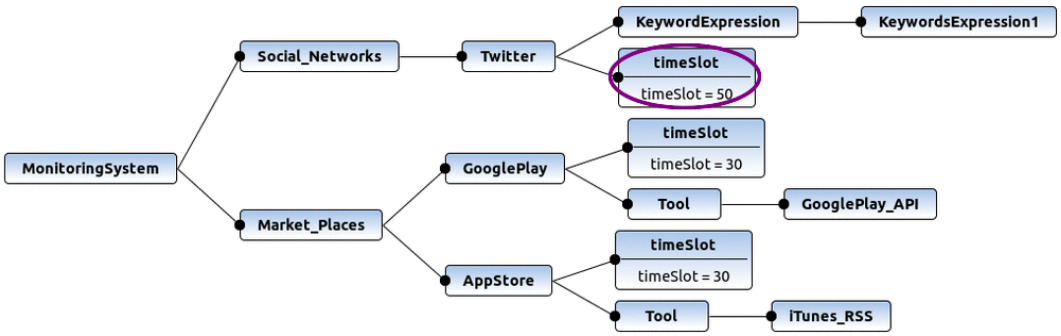

FIGURA 11.3: *Feature Configuration* que descriu la configuració a aplicar per la reconfiguració

Per la reconfiguració ens centrarem exclusivament en el procés de monitoratge de Twitter, el qual hem inicialitzat amb els paràmetres *keywordExpression* i *accounts* tal i com es mostra a la figura, per facilitar el *testing*.

Si ens fixem en les dues *Feature Configuration*, veiem que aquestes són pràcticament les mateixes, a excepció de l'atribut *timeSlot* per les instàncies del monitor de Twitter que, en la nova proposta de configuració, pren un valor més elevat (de 30 segons passa a valer 50). S'ha triat aquest cas d'ús (la modificació del *timeSlot*) per la validació del sistema per dos motius: primerament, per tractar-se d'un cas bàsic que permet observar amb detall la reconfiguració basant-nos en un únic punt de variabilitat, i segon perquè la modificació d'aquest *timeSlot* serà fàcilment visualitzable en el procés d'execució dels monitors.

Finalment, l'*Adaptability Model* descriu la *feature* timeSlot com a *feature* sobre la qual defineix una adaptació, així com els *patterns* i rols a aplicar sobre els models, i finalment l'acció d'actualització del *timeSlot* d'acord amb el valor definit.

```
aspect timeslottwitter {
    feature MonitoringSystem.MonitoringSystem.Social_Networks.Twitter.timeSlot,
    pointcuts{
        pointcut Twitter{
            pattern eu.supersede.dynadapt.usecases.patterns.twitterTimeSlot,
            role ADM.Joinpoint
        }
    },
    compositions{
        composition updateTimeSlot{
            feature_enabled true,
            jointpointRole ADM.Joinpoint,
            action update value '50'
        }
    }
}
```

FIGURA 11.4: *Adaptability Model* que descriu la reconfiguració a executar

11.2 Execució de la reconfiguració

El disparador de l'execució serà la sol·licitud a través del *dashboard* de la reconfiguració associada a la *Feature Configuration* anomenada *MonitoringSystemConfigHighTimeslot*. A la figura 11.5 podem visualitzar un exemple de visualització de la vista *Suggested Adaptations*, que en aquest cas mostra una única *Feature Configuratio* amb una sola acció: l'actualització del valor *timeslot* de les instàncies del monitor de Twitter a 50 segons. Quan l'usuari fa click al botó *"Enact"* amb la *Feature Configuration* seleccionada, el *Dashboard* construeix la sol·licitud amb les dades d'aquella nova configuració, i envia a l'Adapter la petició de reconfiguració. En aquest punt, s'executa l'algorisme de reconfiguració descrit al *Capítol 9. Sistema d'adaptabilitat*, amb la única diferència que la *Feature Configuration* ve donada per paràmetre d'entrada (utilitzem aquest mètode, descrit a la interfície de l'Adapter també present al mateix capítol, per facilitar la validació). Un cop l'Adapter finalitza la seva execució, obté una instància de l'Enactor, que a través dels models UML (l'original i l'adaptat) computa una petició a l'Orchestrator que representa l'actualització de la configuració amb els nous valors. A partir d'aquí, el *workflow* de la reconfiguració passa a estar en mans del sistema de monitoratge. L'Orchestrator envia la petició al Monitor Manager, i aquest la deriva al monitor corresponent. En aquest cas, corresponent al monitor de Twitter.

Suggested Adaptations

	Adaptation id	Name	Computation Timestamp	Model System	Actions			
					Action id	Action name	Action description	Action enabled
✓	2	MonitoringSystemConfigHighTimeslot	2017/06/16 15:34:29	MonitoringReconfiguration	1	UpdateTimesl	Updates the value of Tweeter timeslot instances to 50 (seconds)	true

Go to page: 1 Show rows: 10 1-1 of 1

Enact Delete

FIGURA 11.5: Sol·licitud d'execució de l'adaptació *MonitoringSystemConfigHighTimeslot*

Enacted Adaptations

Adaptation id	Actions				Enactment request time	Enactment completion time	Result
	Action id	Action name	Action description	Action enabled			
2	1	UpdateTimeslot	Updates the value of Tweeter timeslot instances to 50 (seconds)	true	2017/06/16 15:55:14	00:20.543	SUCCESS

Go to page: 1 Show rows: 10 1-1 of 1

FIGURA 11.6: Resultats de l'execució de l'adaptació *MonitoringSystemConfigHighTimeslot*

Cal recordar que tota la comunicació de components es realitza a través de IF (a excepció d'alguns components que es despleguen de forma integrada ja que actuen com a auxiliars, com és el cas del Model Adapter i l'Enactor respecte l'Adapter). Per tant, serà aquest el responsable de rebre i redireccionar les peticions, i conèixer en cada cas on es troben desplegats cadascun dels components amb els quals s'ha d'enviar la petició.

Un cop finalitzada aquesta reconfiguració, des del punt de vista del sistema d'adaptabilitat podem consultar la modificació des de la perspectiva teòrica. Si consultem la vista d'adaptacions executades, mostrada a la figura 11.6, podem veure com ens apareix una nova instància amb la *FeatureConfiguration MonitoringSystemConfigHighTimeslot*, i concretament veiem que s'ha executat amb èxit.

Per comprovar que efectivament ha estat així, necessitem veure què ha succeït a la pràctica al costat del sistema de monitoratge. Per fer-ho, hem desplegat un servidor Kafka en local per tal de poder visualitzar, inicialitzant un *consumer* per terminal (veure *Capítol 7. Sistema de monitoratge* per més detalls sobre aquest aspecte), les dades que el monitor ha generat durant el procés de monitoratge inicialitzat, abans i després de la seva reconfiguració.

A la figura 11.7 veiem el primer objecte JSON enviat a Kafka. Exactament 30 segons més tard, tal i com està definit a la configuració, el monitor envia una segona tanda de dades a Kafka. Això ho podem contrastar observant la figura 11.8 i comparant els camps *searchTimestamp* de cada objecte JSON, situat a l'objecte arrel, al final. Veiem que pel primer bloc de dades aquest té per valor *2017-06-16 17:54:34.424*, mentre que el segon té per valor *2017-06-16 17:55:04.424*. És a dir, exactament 30 segons després del primer enviament. Per tant, podem validar que l'execució d'actualització encara no ha tingut lloc.

```
{
    "SocialNetworksMonitoredData": {
        "confId": 2,
        "numDataItems": 2,
        "DataItems": [
            {
                "timeStamp": "Fri Jun 16 17:54:25 CEST 2017",
                "author": "@QuimMotger",
                "link": "https://twitter.com/QuimMotger/status/875743374728474626",
                "id": "875743374728474626",
                "message": "tweet me"
            },
            {
                "timeStamp": "Fri Jun 16 17:54:29 CEST 2017",
                "author": "@QuimMotger",
                "link": "https://twitter.com/QuimMotger/status/875743389475631105",
                "id": "875743389475631105",
                "message": "follow me"
            }
        ],
        "idOutput": 1,
        "searchTimeStamp": "2017-06-16 17:54:34.423"
    }
}
```


FIGURA 11.7: Primer enviament de dades del monitor de Twitter a Kafka


```
{
    "SocialNetworksMonitoredData": {
        "confId": 2,
        "numDataItems": 1,
        "DataItems": [
            {
                "timeStamp": "Fri Jun 16 17:54:41 CEST 2017",
                "author": "@QuimMotger",
                "link": "https://twitter.com/QuimMotger/status/875743442466459649",
                "id": "875743442466459649",
                "message": "tweet and follow me"
            }
        ],
        "idOutput": 2,
        "searchTimeStamp": "2017-06-16 17:55:04.424"
    }
}
```


FIGURA 11.8: Segon enviament de dades del monitor de Twitter a Kafka (abans de la reconfiguració)

En aquest punt saltem al 3r enviament de dades, representat a la figura 11.9. En aquest cas veiem que les dades s'han enviat exactament a *2017-06-16 17:56:05.489*. Si ens fixem, la diferència ha estat d'aproximadament 60 segons exacta respecte al darrer enviament. Aquest comportament és l'esperat i l'explicació és relativament senzilla: quan es produeix una reconfiguració del *timeslot*, aquesta es produeix durant un cicle de monitoratge del monitor. Això vol dir que el cicle actual es "trenca", i per tant al desenvolupar la *tool* del monitor hem de decidir com actuar. En aquest cas, s'ha optat per iniciar un nou temporitzador amb el nou *timeslot*, i un cop completat el nou cicle, s'envien totes les dades acumulades fins al moment. Podríem haver optat per altres opcions, com p.e. enviar totes les dades acumulades fins al moment i començar a partir d'allà un nou cicle; o simplement actualitzar la durada del cicle en procés i fer que aquesta es rebés amb el nou *timeslot* actualitzat. Al

tractar-se d'un punt que admet diverses consideracions, i ja que l'arquitectura dona llibertat al programador per decidir com actuar, no ho considerarem un aspecte a tenir en compte, i ens centrarem en la versió que s'ha implementat per aquest projecte.


```
{
    "SocialNetworksMonitoredData": {
        "confId": 2,
        "numDataItems": 1,
        "DataItems": [
            {
                "timeStamp": "Fri Jun 16 17:55:34 CEST 2017",
                "author": "@QuimMotger",
                "link": "https://twitter.com/QuimMotger/status/875743664152227841",
                "id": "875743664152227841",
                "message": "don't follow me"
            }
        ],
        "idOutput": 3,
        "searchTimeStamp": "2017-06-16 17:56:05.489"
    }
}
```


FIGURA 11.9: Tercer enviament de dades del monitor de Twitter a Kafka (després de la reconfiguració)

Per aquesta raó, tot i que podem observar una reconfiguració, veiem que encara no és estable, degut a aquest comportament. Però si observem el següent enviament de dades a la figura 11.10 podem veure com efectivament s'ha actualitzat el *timeslot* amb el valor 50, ja que si comparem el *timestamp* anterior amb el nou, *2017-06-16 17:56:55.489*, veiem que passen exactament 50 segons respecte el 3r enviament. En aquest cas, addicionalment, es mostra un exemple d'enviament de dades buit (és a dir, no s'han trobat tuits que satisfacin els paràmetres de cerca).


```
{
    "SocialNetworksMonitoredData": {
        "confId": 2,
        "numDataItems": 0,
        "DataItems": [],
        "idOutput": 4,
        "searchTimeStamp": "2017-06-16 17:56:55.489"
    }
}
```


FIGURA 11.10: Quart enviament de dades del monitor de Twitter a Kafka (configuració estable)

Per tant, tal i com hem pogut comprovar, la reconfiguració del monitor de Twitter es produeix satisfactòriament a partir del disparador implementat al *dashboard* que, gràcies al sistema de models, processa i executa a través del sistema d'adaptabilitat, i més tard del de monitoratge, la reconfiguració d'acord amb els criteris definits per aquest modelatge. Aquest, per suposat, es tracta d'un cas bàsic per validar la funcionalitat del nostre sistema. Amb el disseny i implementació d'aquests components, i amb aquest cas d'ús, queda demostrat que el sistema funciona i permet reconfigurar de forma automatitzada processos de monitoratge actius. El potencial de validació de nous casos d'ús, modificacions més complexes, etc., són modificacions que, tot i que queden fora de l'abast d'aquest projecte, són futures tasques sobre les quals aquest desenvolupament suposa una base teòrica i pràctica molt potent sobre la qual treballar.

12 Conclusions

Com a cloenda pel desenvolupament d'aquest projecte, aquest darrer capítol pretén recollir una avaluació general del producte generat, la feina desenvolupada, el seu potencial i en definitiva, resumir tots els aspectes que engloben el desenvolupament d'un TFG d'aquestes característiques.

12.1 Justificació de l'assoliment de competències

Com a part de la planificació del projecte i el curs de GEP, es van definir una sèrie de competències que es treballarien en aquest projecte en diferents graus. Un cop finalitzat el seu desenvolupament, hem d'assegurar no només la satisfacció dels objectius (plantejada al *Capítol 11. Validació del sistema*), sinó també d'aquestes competències i el seu treball amb el màxim rigor possible.

CES1.1. **Desenvolupar, mantenir i avaluar sistemes i serveis software complexos i/o crítics. [En profunditat]**

L'objectiu principal del projecte ha estat el desenvolupament de dos sistemes (de monitoratge i d'adaptabilitat) formats per un conjunt de components independents, integrats entre ells per satisfer un objectiu major. El desenvolupament de cadascun d'aquests components, així com el seu disseny, el seu *testing* i la seva validació han suposat un treball en profunditat del treball realitzat en un sistema software complex, amb un ús de tecnologies variat i amb requisits diferenciats.

CES1.2. **Donar solució a problemes d'integració en funció de les estratègies, dels estàndards i de les tecnologies disponibles. [En profunditat]**

La integració i comunicació d'aquests components s'ha dissenyat i desenvolupat al llarg del projecte. La necessitat de disposar d'un sistema distribuït ens ha presentat la necessitat de donar solució a la integració de components, d'una banda a través de la necessitat del component d'integració utilitzat (IF), i d'altra banda el disseny i implementació de la lògica necessària a cada component per ser exposat com a servei i poder ser així integrat a IF.

CES1.3. **Identificar, avaluar i gestionar els riscos potencials associats a la construcció de software que es poguessin presentar. [Bastant]**

Especialment treballada durant la planificació del projecte, i utilitzada durant el seu desenvolupament, la gestió i planificació temporal s'ha adaptat a les petites desviacions que s'han anat patint al llarg del desenvolupament.

CES1.5. **Especificar, dissenyar, implementar i avaluar bases de dades. [Bastant]**

Components com l'Orchestrator o el Model Repository Manager han necessitat l'especificació, el disseny, la implementació i l'avaluació de bases de dades relacionals. Aquestes tasques s'han realitzat d'acord a les necessitats de cada component.

CES1.7. **Controlar la qualitat i dissenyar proves en la producció de software. [En profunditat]**

Tots els components han estan validats independentment per satisfer la seva funcionalitat satisfactòriament; addicionalment, es dedica el *Capítol 11. Validació del sistema* per controlar i avaluar l'execució completa de tots els components dins el *workflow* d'adaptació del sistema de monitoratge.

CES1.8. **Desenvolupar, mantenir i avaluar sistemes de control i de temps real. [En profunditat]**

Els monitors implementats són components que interactuen en temps reals amb altres sistemes software. Aquests components s'han dissenyat i desenvolupat en el context d'aquest projecte, i ha calgut també la seva validació per permetre la seva integració.

CES1.9. **Demostrar comprensió en la gestió i govern dels sistemes software. [En profunditat]**

Al llarg de tot el projecte s'ha plantejat un fil conductor per descriure els diferents components a implementar i com els requisits del sistema s'han anat satisfent amb el desenvolupament d'aquests. Addicionalment s'han incorporat els raonaments i justificacions pertinents, tant per ajudar al lector a entendre el procés de desenvolupament, com per demostrar l'assimilació dels coneixements de gestió del sistema software generat.

CES2.1. **Definir i gestionar els requisits d'un sistema software. [En profunditat]**

Satisfet en la part de la planificació, des d'un punt de vista genèric per tot el sistema, i addicionalment durant el desenvolupament del projecte, conforme de cada component hem extret els requisits necessaris per procedir al seu desenvolupament.

CES2.2. **Dissenyar solucions apropiades en un o més dominis d'aplicació, utilitzant mètodes d'enginyeria del software que integrin aspectes ètics, socials, legals i econòmics. [Bastant]**

El *Capítol 4. Gestió i desenvolupament* presenta d'una banda un anàlisi detallat i numèric dels requisits econòmics del projecte, segons criteris com les hores necessàries, l'equip físic i humà, etc. Per altra banda, presenta l'anàlisi dels criteris de sostenibilitat des del punt de vista ètic, social i legal que al llarg de tot el projecte s'ha assegurat que es garanteixen

CES3.1. Desenvolupar serveis i aplicacions multimèdia. [En profunditat]

L'exposició de tots els components com a serveis RESTful per permetre la seva integració, o bé el desenvolupament del *dashboard* per l'adaptabilitat del sistema, ha suposat no només el treball d'aquests coneixements en aquest projecte, sinó també un aprenentatge en desenvolupament d'APIs i *front-end*.

12.2 Avaluació del potencial del producte generat

Els resultats generats i el potencial de cadascun dels components desenvolupats s'han anat tractant i presentant al llarg del projecte, però és convenient fer un resum sobre el potencial del producte generat de cara a desenvolupament futur i ampliació

de les seves funcionalitats.

Primerament, la **proposta d'una arquitectura genèrica** pel desenvolupament de monitors obre la possibilitat d'extensió del sistema de monitoratge amb relativa facilitat, partint de la documentació que aquesta mateixa memòria presenta. De fet, actualment altres col·laboradors del projecte han fet servir aquesta mateixa arquitectura per desenvolupar altres monitors que s'han integrat en el sistema SUPERSEDE per altres casos d'estudi. Això és un exemple de com aquesta proposta satisfà el nostre objectiu d'heterogeneïtat, i permet partint d'una arquitectura genèrica implementar un monitor reutilitzant el màxim dels components, i assumint com a única responsabilitat addicional del desenvolupador la lògica interna del monitor.

Respecte al sistema d'adaptabilitat, la validació d'aquest projecte es centra en la reconfiguració de processos de monitoratge actius, però els components desenvolupats i el model d'adaptació obre la porta a una **autonomia total del sistema** i una reconfiguració molt més completa. El sistema dona suport a modificacions més complexes de models UML, com per exemple afegir instàncies de configuracions (processos de monitoratge). Amb poques modificacions, podríem estendre i definir adaptacions molt més complexes: el sistema de monitoratge suporta completament qualsevol operació respecte els processos de monitoratge, i dins el sistema d'adaptabilitat, només caldria definir els models de configuració que defineixin aquests canvis, i adaptar el component Enactor per suportar la traducció a aquest tipus de peticions. Així, a mesura que es treballi en noves configuracions, podem disposar d'un sistema que no només sigui autoadaptable des del punt de vista de reconfiguració de processos de monitoratge, sino fins i tot en la posada en marxa i aturada.

Finalment, cal contemplar que aquest projecte ha estat desenvolupat centrant-se en el cas de reconfiguració de monitors, però sempre respectant la seva integració dins SUPERSEDE. Això ha permès que alguns dels components desenvolupats, com per exemple el Model Repository, l'Adapter o el Model Adapter són completament reutilitzables per altres casos d'estudi totalment independents a la reconfiguració de monitors. En cada cas, caldria plantejar les necessitats de models, i estendre el Model Adapter per suportar l'adaptació dels models definits. A partir d'aquí, des d'un punt de vista genèric, la reconfiguració de sistemes és totalment autònoma i abstracta a la necessitat del sistema.

En resum, podem satisfer que per una banda s'ha resolt una problemàtica específica, alhora que el producte generat permet el seu ús i reutilització per seguir treballant i desenvolupar en resoldre problemes dins la mateixa àrea i amb objectius similars.

12.3 Avaluació general

Davant els punts anteriorment exposats, es pot garantir no només la satisfacció dels objectius propis del projecte, sinó també els objectius des d'un punt de vista didàctic com a projecte que representa la cloenda dels estudis del Grau en Enginyeria Informàtica.

Aquest projecte ha suposat, a títol personal, una de les primeres experiències en el desenvolupament d'un projecte amb implicacions reals, entenent aquest desenvolupament com el transcurs en la seva totalitat: des del plantejament de les necessitats i requisits fins la seva validació, passant pel disseny software i la implementació dels components especificats. Com a futur professional de l'enginyeria del software, l'oportunitat de participar en aquestes fases d'un projecte i veure la seva evolució aporta un gran valor personal i professional, que serveix de perfecte tancament dels estudis mitjançant l'assoliment i validació final dels coneixements adquirits durant el transcurs del grau.

Addicionalment aquest projecte ha suposat no només una consolidació dels coneixements adquirits, sinó també un aprenentatge de tecnologies o conceptes que altrament és probable que no hagués tractat. Degut a la gran variabilitat de requisits tècnics dels diferents components del projecte, ha estat necessari afrontar reptes variats que en alguns casos han suposat dedicar hores d'aprenentatge (com, p.e., el disseny d'un *front-end* amb un *framework* específic). Aquest és un punt essencial que, a més, reflexa perfectament la realitat del món laboral dins el sector de l'enginyeria informàtica: la constant necessitat de renovació i aprenentatge en funció de les circumstàncies del moment.

En definitiva les sensacions al finalitzar aquest projecte són altament satisfactòries. Consolidant els coneixements interioritzats durant els 4 anys d'aquest grau, produeix una gran motivació veure d'una banda les habilitats i coneixements adquirits, i d'altra banda el potencial coneixement encara per adquirir i desenvolupar, ja sigui en altres entorns acadèmics o en el món laboral. És el moment de tancar el darrer capítol d'aquesta memòria i, amb sort, començar a escriure'n molts més allà on el futur professional em condueixi.

A Twitter Monitor API

Monitor [/configuration]

Create a new configuration for this monitor [POST]

+ Request (application/json)
"SocialNetworksMonitoringConfProf": "toolName": "TwitterAPI", "timeSlot": "30", "kafkaEndpoint": "http://localhost:9092", "kafkaTopic": "tweeterMonitoring", "keywordExpression": "(tweet OR follow) AND (me)", "accounts": ["QuimMotger"]
+ Response (application/json)
+ Body
"SocialNetworksMonitoringConfProfResult": ïdConf":"1", "status":"success

Configuration [/configuration/id]

Updates an existing configuration [PUT]

+ Request (application/json)
"SocialNetworksMonitoringConfProf": "toolName": "TwitterAPI", "timeSlot": "60", "kafkaEndpoint": "http://localhost:9092", "kafkaTopic": "tweeterMonitoring", "keywordExpression": "(tweet OR follow) AND (me)", "accounts": ["QuimMotger"]
+ Response (application/json)
+ Body
"SocialNetworksMonitoringConfProfResult": ïdConf":"1", "status":"success

Deletes a configuration instance [DELETE]

B Appendix Title Here

Write your Appendix content here.

C Appendix Title Here

Write your Appendix content here.

D Appendix Title Here

Write your Appendix content here.

E Appendix Title Here

Write your Appendix content here.

F Appendix Title Here

Write your Appendix content here.

G Appendix Title Here

Write your Appendix content here.

H Appendix Title Here

Write your Appendix content here.

Bibliografia

[1] Gabor Karsai et al. "An Approach to Self-adaptive Software Based on Supervisory Control". A: *Self-Adaptive Software: Applications (IWSAS)* (2001).

[2] Salvatore Venticinque. "Agent based services for negotiation, monitoring and reconfiguration of cloud resources." A: *European Research Activities in Cloud Computing* (2012).

[3] Ying Li et al. "Self-reconfiguration of service-based systems: A case study for service level agreements and resource optimization." A: *International Conference on Web Services* (2005).

[4] *SUPERSEDE - at a glance*. URL: https://www.supersede.eu/project/supersede-at-a-glance/ (cons. 30-4-2017).

[5] *Horizon 2020 EU Framework Programme Research and Innovation*. URL: https://ec.europa.eu/programmes/horizon2020/ (cons. 30-4-2017).

[6] Paolo Arcaini, Elvinia Riccobene i Patrizia Scandurra. "Modeling and Analyzing MAPE-K Feedback Loops for Self-adaptation". A: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Florence, Italy: IEEE Press, 2015, pàg. 13 - 23. URL: http://dl.acm.org/citation.cfm?id=2821357.2821362.

[7] Didac Gil De La Iglesia i Danny Weyns. "MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems". A: *ACM Trans. Auton. Adapt. Syst.* 10.3 (set. de 2015), 15:1 - 15:31. ISSN: 1556-4665. DOI: 10.1145/2724719. URL: http://doi.acm.org/10.1145/2724719.

[8] *La metodología Kanban*. URL: https://es.atlassian.com/agile/kanban (cons. 29-3-2017).

[9] *Kanban vs Scrum*. URL: http://www.agileweboperations.com/scrum-vs-kanban (cons. 29-3-2017).

[10] *Treball de Fi de Grau*. URL: https://www.fib.upc.edu/ca/estudis/graus/grau-en-enginyeria-informatica/treball-de-fi-de-grau (cons. 19-6-2017).

[11] *Los créditos europeos (ECTS)*. URL: http://portal.uned.es/portal/page?_pageid=355,3138322&_dad=portal (cons. 18-6-2017).

[12] STANDISH GROUP et al. "CHAOS manifesto 2013". A: *The Standish Group International. EUA* (2011).

[13] *GitHub*. URL: https://github.com/ (cons. 18-6-2017).

[14] *Java Platform Standard Edition 8 Documentation*. URL: http://docs.oracle.com/javase/8/docs/ (cons. 18-6-2017).

[15] *Eclipse IDE for Java Developers (Neon 4.6.2)*. URL: http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/neonr (cons. 18-6-2017).

[16] *Eclipse Modeling Tools (Neon 4.6.2)*. URL: http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/neon3 (cons. 18-6-2017).

[17] *Gradle Release Notes*. URL: https://docs.gradle.org/2.13/release-notes.html (cons. 18-6-2017).

[18] *Spring framework*. URL: https://projects.spring.io/spring-framework/ (cons. 18-6-2017).

[19] *UML2 EMF-based implementation*. URL: https://eclipse.org/modeling/mdt/?project=uml2 (cons. 18-6-2017).

[20] *Papyrus Modeling environment*. URL: https://eclipse.org/papyrus/ (cons. 18-6-2017).

[21] *MySQL 5.6 Reference Manual*. URL: https://dev.mysql.com/doc/refman/5.6/en/ (cons. 18-6-2017).

[22] *Apache Kafka streaming platform*. URL: https://kafka.apache.org/intro (cons. 26-3-2017).

[23] Ira R. Forman i Nate Forman. *Java Reflection in Action*. 2005.

[24] *Twitter4J*. URL: http://twitter4j.org/en/index.html (cons. 28-3-2017).

[25] *Twitter Development Documentation. Streaming APIs*. URL: https://dev.twitter.com/streaming/overview (cons. 28-3-2017).

[26] *Google Play Developer API - Reviews*. URL: https://developers.google.com/android-publisher/api-ref/reviews (cons. 28-3-2017).

[27] *apptweak - API Documentation or App Store and Google Play Store*. URL: https://apptweak.io/api (cons. 28-3-2017).

[28] *Monitoring Reviews on the App Store*. URL: https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/ImprovingCustomersExperience.html#//apple_ref/doc/uid/TP40011225-CH27-SW1 (cons. 28-3-2017).

[29] *UML2 Class Diagrams: an Agile Introduction*. URL: http://www.agilemodeling.com/artifacts/classDiagram.htm (cons. 18-4-2017).

[30] Antonio Ruiz-Cortes Sergio Segura Miguel Toro Pablo Trinidad David Benavides. "Explanations for Agile Feature Models". A: *Journal of Systems and Software* (2008).

[31] *Viatra Query User documentaiton - Query Language*. URL: https://wiki.eclipse.org/VIATRA/Query/UserDocumentation/QueryLanguage (cons. 3-4-2017).

[32] Lidia Fuentes i Antonio Vallecillo. "Una introducción a los perfiles UML". A: *Novática* 168 (2004), pàg. 6 - 11.

[33] *Developing a Spring Framework MVC application step-by-step*. URL: https://docs.spring.io/docs/Spring-MVC-step-by-step/ (cons. 21-5-2017).