LSim

Execució d'aplicacions en entorns realistes a gran escala

Títol: LSIM: execució d'aplicacions en entorns realistes a gran escala

Alumne: Esteve Verdura Gelmà

Director: Joan Manuel Marquès i Puig

Data: 16/01/2012

Índex

Agraïments

He gaudit molt d'aquest sis anys de carrera a la FIB (amb tot el que inclou), de moment ha estat la millor experiència que he tingut a la meva curta vida.

Primer m'agradaria donar les gràcies a en Joan Manuel Marqués, ja que em va donar l'oportunitat d'entrar a la UOC , realitzar aquest projecte i aguantar-me durant tot aquest temps.

Un agraïment també als meus companys de fatigues de la facultat, el grup del Fons a la Dreta and the New One's (Xevi Fuster, Àtia, Xevi Corrales, Imanol, Nistal (antinistalisme al pfc), Joan Roca, Roger Puig, l'Albert Casanovas, en Josep Maria, Jordi Casa, Miquel de Arcayne, etc) i en especial a en Xavi Azagra, que sempre estarà al nostre record. Amb tots vosaltres he passat penes i glòries no només a la facultat o al bar. Agrair-vos també als de la meva generació (Fibermunks: Geordina, Núria, Marc Sitges, Santi Boix, Faust, etc), amb els quals hem patit moltes hores d'estudi i feina de les assignatures i viatges per treure'ns estres de sobre.

Donar les gràcies a la meva família, els meus pares Joan i Mercè, ja que el seu esforç ha suposat que tingues l'oportunitat de poder fer carrera. També a la resta de la meva família (que no és poca), la meva germana Mercè, els meus avis Carmen i Esteve, el meu cosí Miquel, que m'ha dissenyat el logo de LSim, i tota la patuleia, que m'han hagut d'aguantar tots aquest anys els hi vingues explicant batalletes d'informàtica, cosa que entenc que és difícil.

Gràcies als meus amics de Calella, l'Adrià, Meritxell, Lluis, Oscar, Sandra i David, per totes les nits de discussions absurdes, cerveses per desconnectar de la feina i tot el suport (que conscient o inconscientment) m'heu anat donant tots aquest anys per seguir endavant.

Agrair també els meus companys de la UOC (Daniel, en Guillem, en Josep Maria, la Carmen, etc) que m'han donat suport durant el transcurs del projecte.

I finalment segurament em deixo a gent (com els companys d'Erasmus o els companys de l'equip de Handbol), ja que durant tot aquest temps he compartit moltes experiències amb els qui m'he anat creuant. Seria molt difícil poder-vos ficar a tots aquí, però això no vol dir que no us vulgui donar les gràcies a tots vosaltres també.

1 Introducció

1.1 Definició del projecte

La fase de proves en el procés de desenvolupament del programari té una problemàtica afegida quan l'aplicació és distribuïda i s'executa en xarxa. El desenvolupador ha de crear una forma de realitzar aquestes proves, cosa que és costosa i acaba comportant haver d'afegir tot un seguit de codi per posar en marxa, coordinar i recollir la informació de l'execució.

Aquest projecte introdueix LSim, una eina pel desplegament d'aplicacions en entorns distribuïts realistes. L'eina està formada per una llibreria, que automatitza l'execució, coordinació i recollida de resultats, i per un framework que desplega automàticament l'aplicació als recursos que realitzaran les proves.

L'objectiu de LSim és ajudar als desenvolupadors d'aplicacions en xarxa a provar-les en entorns distribuïts realistes en xarxa, havent de fer només petites modificacions al seu producte ja implementat.

LSim s'ha utilitzat al semestre de tardor del 2011 a l'assignatura d'arquitectura de sistemes distribuïts (ASD) de la UOC. Els alumnes implementen una part d'un protocol distribuït i, utilitzant LSim, el codi s'executa i avalua en un entorn realista.

1.2 Motivació

Tot programari ha de passar per un conjunt d'etapes abans de posar-se a disposició dels usuaris. El procés de desenvolupament del programari pot seguir diferents models (en cascada, incremental, cíclic, en espiral, evolutiu, etcètera), però cada un d'ells sempre segueix més o menys les mateixes etapes, que són: anàlisi de requisits, especificació, disseny del producte, implementació d'aquest, provar el correcte funcionament en diferents casos i entorns, i una última fase de manteniment.

Aquest projecte es centra en el desenvolupament d'aplicacions destinades a funcionar a la xarxa. Aquest tipus d'aplicacions tenen una problemàtica afegida a l'hora de validar el seu correcte funcionament, ja que han de ser executades en xarxa. S'ha d'obtenir aquest comportament de xarxa real per tal d'avaluar que el producte que s'esta desenvolupant funciona correctament en aquest entorn.

Hi ha varies formes de generar aquest comportament i avaluar el programari que s'està desenvolupant. Dos clars exemples són els simuladors i els emuladors.

Els simuladors, a partir d'un model del producte, permeten provar el seu comportament en múltiples situacions. Llavors a partir d'un model del programari a desenvolupar es pot avaluar el seu comportament.

Els emuladors permeten reproduir les funcions d'un primer sistema a un segon sistema. Llavors emulant un sistema en xarxa es pot avaluar el comportament de l'aplicació un cop ja realitzada.

En aquest projecte, però, interessa utilitzar un altre mètode per validar el programari, que és el desplegament de l'aplicació en un entorn distribuït realista en xarxa, és a dir, provar l'aplicació en una xarxa real, tal com és Internet.

A continuació a la taula 1 es mostra la comparació aquest tres mètodes:

Property	Fast network simulator	Emulated nodes and net	PlanetLab
System			
Scale	1,000s	~1,000	~500
Network topology and link characteristics	Flexible, latency only	Flexible, all effects	Hard-wired, all effects
Node effects	No	Yes	Yes
Stimulus			
Workload	Flexible	Flexible	Flexible and realistic
Operator actions	No	Flexible	Realistic
Faults	Net only	Flexible	Realistic
Measures			
Reproducibility	High	Medium	Low
Experiment management	Easy	Medium	Hard

Figura 1: Simulados vs Emulador vs Desplegament a PlanetLab [3]

PlanetLab és una xarxa mundial d'investigació que dóna suport al desenvolupament de nous serveis en xarxa. Té un conjunt de 1090 nodes distribuïts per tot el món que els desenvolupadors poden accedir per desplegar els seus serveis en xarxa [2].

El simulador permet fer proves a gran escala i reproduir-les fàcilment, cosa que és útil per a provar el model del programari en situacions difícils de reproduir en qualsevol altre entorn. És ideal per a realitzar experiments, ja que és fàcil de controlar-los i reproduir-los. L'inconvenient que tenen és que són difícils de configurar per a reproduir comportaments realistes en components individuals en xarxa.

L'emulador dóna una abstracció dels dispositius de la xarxa, fent que sigui més fàcil treballar amb ell. Permet reproduir el seu comportament i és més fàcil analitzar el comportament de cada un dels elements individualment, cosa que és difícil d'aconseguir amb un simulador. Per tant es poden realitzar experiments i reproduir-los. El problema que té és que no es tracta d'una xarxa real i és difícil aconseguir provar situacions extremes de tot el sistema que amb el simulador és més fàcil de provar.

Fer un desplegament de l'aplicació en un entorn distribuït realista com pot ser PlanetLab presentat en la taula 1, és ideal per provar el comportament dels components individuals del sistema i el comportament de tot sistema. Fer proves en un entorn distribuït realista és difícil, ja que cal fer-ne una abstracció dels recursos per poder-hi treballar de forma transparent i fàcil, tal com es fa en un emulador. Tampoc es poden provar els casos extrems, com si que es pot fer amb un simulador, i l'adaptació de l'aplicació és més complexa que en un emulador. Llavors la gestió d'experiments en aquest cas és difícil i treu molt de temps al desenvolupador de l'aplicació.

La utilització i intercalació d'aquests tres mètodes de proves durant el desenvolupament del programari seria ideal per obtenir un producte prou robust per a ser distribuït als usuaris, ja que cada un d'aquest mètodes és adequat en un punt diferent del procés de desenvolupament del programari.

Aquest projecte es centra en el desplegament d'aplicacions en un entorn distribuït realista en xarxa, per l'interès que té obtenir dades d'un entorn real.

1.3 Objectius

Els objectiu d'aquest projecte són els següents:

- Crear un framework que desplegui l'aplicació en un entorn distribuït realista
- Crear una llibreria per a adaptar l'aplicació a desplegar-se en l'entorn distribuït realista, coordinar l'execució del test i recollir-ne els resultats.
- Validar la llibreria i el framework.
- Desplegar una aplicació real en el framework utilitzant la llibreria.

1.4 Productes obtinguts

A partir d'aquest projecte s'obtindran dos producte:

1. LSim Framework que permet desplegar aplicacions.
2. LSim Library que ofereix funcions de coordinació i recollida de resultats a l'aplicació.

1.5 Planificació

A continuació es mostra la planificació inicial pel projecte:

	Nombre de tarea	Duración	Comienzo	Fin	Predec
1	Anàlisi de requisits	5 días	lun 13/06/11	vie 17/06/11	
2	Especificació	4 días	lun 20/06/11	jue 23/06/11	1
3	Disseny	25 días	lun 27/06/11	vie 29/07/11	2
4	**Implementació**	**40 días**	**lun 01/08/11**	**vie 23/09/11**	
5	Implementació de LSim	25 días	lun 01/08/11	vie 02/09/11	3
6	Implementació de LSim	25 días	lun 22/08/11	vie 23/09/11	
7	Proves	20 días	lun 26/09/11	vie 21/10/11	4
8	Adaptació a ASD	25 días	lun 24/10/11	vie 25/11/11	7
9	Elaboració de la memòria	20 días	lun 14/11/11	vie 09/12/11	
10	Elaboració de la presentac	5 días	lun 12/12/11	vie 16/12/11	9

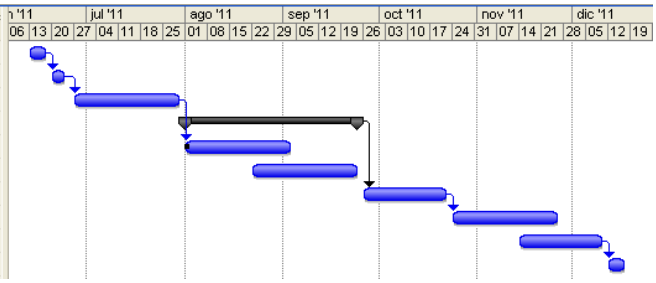


Figura 2: Planificació de LSim

La planificació inicial no s'ha pogut seguir fidelment, ja que la fase d'implementació, la de proves i la d'adaptació a ASD (pràctiques de l'assignatura d'arquitectura sistemes distribuïts) al final s'han acabat intercalant, ja que les proves en sistemes distribuïts són difícils de fer i ha portat més temps del previst solucionar alguns bugs.

1.6 Anàlisi econòmic

El primer pas de l'anàlisi econòmic és mirat el cost de les llicencies de les eines a utilitzar pel desenvolupament del projecte:

Producte	***Preu***
OpenOffice.org 2.0 (open source)	0€
ArgoUML (open source)	0€
Eclipse (open source)	0€
Diver (open source)	0€
Cost Total	0€

Taula 1: Cost de les llicencies utilitzades

Tal i com es veu a la taula 1, les eines utilitzades són d'open source, per tant el cost total de les llicencies és zero.

El següent pas serà veure el cost del desenvolupament del projecte. Per fer una estimació de les hores de durada del projecte i salari dels professionals que desenvoluparien el projecte, utilitzant els mètodes ensenyats a l'assignatura de PESBD (Projecte d'Enginyeria del Software i Bases de Dades) de la Facultat d'Informàtica de Barcelona.

Fases del projecte	*Perfil*	*Hores*	*Tarifa/hora*	*Cost*
Anàlisi previ i recollida de requeriments	Cap de projecte	48	60	2.880,00 €
Especificació	Analista	48	51	2.448,00 €
Disseny	Analista	238	51	12.138,00 €
Implementació	Analista/Programador	286	42	12.012,00 €
Proves	Analista	95	51	4.845,00 €
Adaptació a ASD	Programador	143	42	6.006,00 €
Documentació	Analista	95	51	4.845,00 €
TOTAL		953		45.174,00 €

Taula 2: Cost del desenvolupament del projecte

Tot i així el cost real del projecte està molt desajustat a les estimacions fetes, ja que s'ha dut a terme a cost d'un ajudant de recerca durant uns sis mesos.

1.7 Organització de la memòria

El primer apartat ofereix una visió general de LSim, juntament amb els objectius marcats, productes obtinguts, la motivació, la planificació i l'anàlisi econòmic del projecte.

A continuació es presenten altres eines existents que poden desplegar aplicacions en entorns distribuïts realistes per validar aplicacions distribuïdes, comparant-les amb l'eina que s'esta desenvolupant al projecte, LSim.

Al següent apartat es presenten els requeriments pel desenvolupament de l'eina, on primer es fa un anàlisi més exhaustiu del problema de desplegar aplicacions en entorns distribuïts realistes per validar-les, pel posterior anàlisi de requeriments.

Un cop definits els requeriments, es mostra l'especificació de l'eina, amb tots els casos d'ús que hi apareixen i les funcions de coordinació.

Després ve tota la fase de disseny, on es mostrarà el disseny de LSim Library, amb les funcions de coordinació, recollida de resultats i comunicació amb l'aplicació, i LSim Framework, amb els mètodes pel desplegament d'aplicacions.

Al següent apartat es parla de la implementació de la primera versió de LSim Framework i LSim Library, descrivint les funcionalitats implementades d'aquesta primera versió.

Un cop implementada la primera versió, es descriu un cas real on s'ha utilitzat LSim per validar el funcionament d'aplicacions, en aquest cas avaluar les pràctiques dels alumnes de l'assignatura d'arquitectura de sistemes distribuïts de la UOC.

Finalment es donen les conclusions de la realització del projecte, on s'exposen els objectius aconseguits, una opinió personal i el treball futur pel projecte.

2 Related work

2.1.1 OMF

OMF[12] és un frameworks per controlar, mesurar i gestionar entorns distribuïts realistes. És una eina molt completa que ofereix tota una plataforma on l'usuari gestiona el desplegament d'aplicacions en diferents entorns distribuït realistes per validar-ne el funcionament. Permet definir, desplegar i recollir resultats dels experiments especificats utilitzant el llenguatge OEDL, llenguatge propi d'OMF basat en Ruby. Es poden executar aplicacions desenvolupades en diferents llenguatges de programació, sempre i quan les màquines on s'executen tinguin el programari necessari per executar-les, només cal fer un descriptor de l'aplicació amb les dades necessàries per la seva execució. A partir del descriptor de l'experiment en OEDL, OMF pot gestionar els recursos on s'executarà l'experiment, definir els paràmetres de l'experiment, coordinar els nodes que hi participen i enviar paràmetres dinàmicament durant l'experiment [13]. Per recollir resultats ofereix una API a les aplicacions que permet guardar els resultats en una BD durant l'experiment, la qual es pot consultar posteriorment via web.

OMF ofereix més flexibilitat que LSim en quant a desplegar diferents aplicacions desenvolupades en diferents llenguatges de programació. En altres aspectes, però, és més semblant a LSim. Els dos necessiten un descriptor d'experiment, que requereix introduir canvis en el codi per a poder coordinar els diferents nodes o recollir-ne els resultats (a LSim s'han d'afegir les funcions de coordinació al codi i a OMF l'enviament de resultat a la BD). Una diferencia important és que a OMF s'han de definir els paràmetres de l'experiment de manera estàtica en el descriptor, fins i tot els que s'enviaran dinàmicament durant l'experiment. LSim en canvi, pot generar paràmetres durant l'experiment i enviar-los de manera dinàmica a través de les funcions de coordinació, donant més flexibilitat a l'hora de definir els comportaments dels nodes que realitzen l'experiment.

El fet que LSim sigui una llibreria restringeix més el tipus d'aplicacions que es poden validar (només podran aquelles que estiguin en el mateix llenguatge de programació que la llibreria). Pel mateix fet de ser una llibreria, la generació i comunicació de paràmetres durant l'experiment és més dinàmica, ja que es poden afegir aquestes funcions enmig del codi de l'aplicació.

2.1.2 SPLAY

Splay [14] [15] és una plataforma que simplifica la creació de prototips i el desenvolupament a gran escala d'aplicacions distribuïdes. Splay dóna suport a totes les fases de desenvolupament de programari: disseny, implementació i validació del prototipus. Permet als desenvolupadors especificar les seves aplicacions distribuïdes d'una manera concisa en un llenguatge especialitzat basat en Lua [5], un llenguatge de scripting d'alta eficiència. Executa les aplicacions en un entorn segur amb accés restringit als recursos locals (sistema d'arxius, xarxa, memòria) fent servir mètodes de sandboxing.

Splay té un control més ampli que LSim de les aplicacions que s'executen. Aquest cobreix un espectre més ampli que LSim pel que fa al desenvolupament d'aplicacions distribuïdes, ja que també hi entra la part de disseny i implementació. Un dels desavantatges és que s'han de desenvolupar les aplicacions usant Lua. Aquest desavantatge és comú a LSim, ja que només serà adaptable a les aplicacions implementades en el mateix llenguatge que la llibreria. Splay està pensat per fer tot el proces de desenvolupament del prototipus, agafant un comportament semblant a un simulador, on s'ha de fer un model de l'aplicació per validar-ne el funcionament. Al ser en un entorn distribuït realista, ofereix altres aspectes que són més difícils d'aconseguir en un simulador, tal com validar el comportament de components individuals del sistema que composa l'aplicació.

LSim està pensat per validar l'aplicació en la fase de proves en comptes de com a eina pel desenvolupament d'aplicacions. L'objectiu és que, a partir de la implementació de l'aplicació, i afegint els mètodes de coordinació de LSim, es pugui desplegar en un entorn distribuït realista per tal de validar-ne el funcionament. Per tant, és més flexible que Splay, ja que el desenvolupador pot utilitzar LSim si creu convenient a partir del que ja té desenvolupat. En canvi, en cas que vulgui utilitzar Splay, el desenvolupador ha de realitzar un model de l'aplicació usant el llenguatge especialitzat que incorpora.

2.1.3 Gush

Gush [16] és un framework controlador d'experiments que té com objectiu donar suport a la configuració del programari i l'execució en diversos tipus de recursos. Està pensat pel desenvolupament, desplegament i manteniment de programari executant-se en un conjunt de recursos distribuïts. Permet desplegar aplicacions fetes en diferents tipus de llenguatges de programació. S'especifica el comportament de l'experiment a partir d'un XML definint els diferents components de l'experiment i els recursos necessaris per aquests components. Permet afegir dinàmicament més recursos per l'experiment durant l'execució d'aquest. Ofereix tot un conjunt de comandes a les consoles de Linux per tal de gestionar els recursos on s'està executant l'aplicació.

Gush ofereix a l'usuari la gestió dels recursos per al desenvolupament de l'experiment. Pot desplegar de forma dinàmica nous elements de l'experiment durant el transcurs d'aquest. Per tant, a diferencia de LSim, pot controlar experiments de llarga durada. Gush pot definir quin tipus de recursos són necessaris per l'experiment (CPU, RAM, disc, etc), cosa que LSim, ja que l'obtenció de recursos és transparent a l'usuari. Gush no permet la gestió de paràmetres per l'experiment a través de l'especificació de l'aplicació, només permet definir el nombre de recursos i informació relativa a aquests. LSim permet fer la coordinació automàtica dels experiments, juntament amb la definició de paràmetres de l'experiment a l'especificació, cosa que utilitzant Gush cal fer manualment la coordinació de l'experiment a través de comandes a la consola de Linux, i definir de forma estàtica a l'aplicació els paràmetres de l'experiment.

3 Anàlisi de requeriments

A l'anàlisi de requeriments es detallarà el problema que es vol resoldre i s'ha abordat la solució a aquest problema.

3.1 Problemàtica d'un entorn realista per a realitzar proves

El desenvolupament de serveis, aplicacions o protocols en xarxa tenen una problemàtica afegida a l'hora de provar el seu funcionament. A la introducció s'ha parlat de varis mètodes que ajuden a realitzar proves d'un producte que es vol desenvolupar. Aquest projecte està centrat en crear una eina per ajudar al desplegament en un entorn realista de les proves, és a dir, utilitzar una xarxa real, tal com podria ser Internet, per a provar el funcionament del servei, aplicació o protocol en xarxa.

La realització de proves desplegant l'aplicació en un entorn realista és difícil, però els resultats que s'obtenen són reals. Per tant, la realització d'aquest tipus de proves un cop ja es té l'aplicació implementada és ideal per comprovar el correcta funcionament en un entorn real, i detectar problemes inesperats al sistema abans de distribuir-ho als usuaris.

El desplegament en un entorn realista és complex pels següents motius:

- El desplegament a la xarxa és una feina costosa, ja que s'ha de disposar d'un conjunt de recursos connectats en xarxa i tenir-hi l'aplicació a provar ja instal·lada a cada una de les màquines. Si aquest desplegament es fa en un conjunt petit de màquines és senzill de gestionar, però es torna una feina repetitiva i pesada quan el nombre de recursos on desplegar l'aplicació creix.
- Els entorns distribuïts realistes, tal com Internet, estan modelats com un conjunt de nodes independents. L'execució d'aplicacions sobre els nodes que formen l'entorn dificulta la realització de proves, ja que no hi ha cap mètode de coordinació establerta per defecte en un entorn distribuït realista. Per tant, el desenvolupador ha de buscar/crear un mètode per poder realitzar aquesta coordinació.
- L'aplicació a validar en l'entorn distribuït no té perquè tenir cap mètode de coordinació. Per tant, és necessari afegir la coordinació per poder realitzar proves en l'entorn distribuït. Implica doncs, que el desenvolupador ha de tenir coneixements de coordinació d'aplicacions en xarxa, cosa que haurà d'adquirir si no en té. A més s'han d'afegir els mètodes de coordinació a l'aplicació, cosa que implica s'ha de modificar el codi.
- El comportament d'un experiment vindrà definit segons el que es validi de l'aplicació i es pot recollir els resultats per tal d'analitzar-los. Així el desenvolupador pot crear i validar proves d'una forma fàcil. Un entorn distribuït no ofereix aquesta gestió. Per tant s'ha de buscar/crear una manera de poder fer aquesta gestió.

3.2 Requeriments de la solució

Vist el conjunt de problemes que suposa provar serveis, aplicacions o protocols en xarxa desplegant-los en un entorn distribuït realista, es vol veure quins requeriments ha de tenir el sistema que permeti realitzar aquestes proves:

- Desplegament automàtic i transparent de l'aplicació al conjunt de recursos: L'obtenció dels recursos, desplegament i execució de l'aplicació ha de ser completament transparent a l'usuari. Per tant, l'usuari ha de poder especificar quina aplicació vol desplegar i el nombre de màquines en que es vol que s'executi.

- Coordinació automàtica i transparent a l'usuari: Oferir mètodes de coordinació entre les diferents instàncies de l'aplicació un cop estigui desplegada a la xarxa. Aquesta coordinació s'ha de realitzar de forma automàtica, o sigui, que permeti definir punts de coordinació i que el sistema s'encarregui d'aconseguir aquesta coordinació de manera transparent a l'usuari.
- Mètodes poc intrusius en el codi de l'usuari: Es pretén que la diferència entre la versió de proves i la versió final de l'aplicació sigui la mínima i calgui realitzar pocs canvis.
- Permetre la gestió d'experiments: S'ha de poder especificar el comportament de l'experiment a partir de paràmetres.
- Recollida automàtica de resultats: El sistema ha de permetre recollir els resultats dels nodes distribuïts a l'entorn que s'estimin necessaris un cop ja s'ha realitzat la prova.
- Avaluació dels resultats: Ha de poder avaluar els resultats recollits dels diferents elements de l'entorn per tal de determinar el resultat de la prova.
- Adaptable a diferents entorns d'execució: L'usuari ha de poder elegir a quin entorn (conjunt d'ordinadors) es vol executar les proves. Per tant, cal poder adaptar-lo a diferents entorns de manera transparent a l'usuari.

4 Especificació

Aquest projecte està centrat en crear una eina per desplegar i coordinar de manera automàtica experiments. L'eina està composada per un frameworks i una llibreria.

Les funcionalitats que ha d'oferir el framework de cara a l'usuari són les següents:

1. Llançar aplicacions a ser provades en un conjunt de nodes distribuïts connectats per una xarxa real.
2. Recollir els resultats de l'execució de l'aplicació en aquest entorn.
3. Pujar l'aplicació que es vol executar al sistema.
4. Esborrar una aplicació que ja estigui en el sistema.

Aquests mètodes són els que han de solucionar la problemàtica de desplegar l'aplicació de l'usuari en un entorn distribuït.

A més cal especificar el conjunt de funcions per a coordinar les execucions que la llibreria ha d'oferir.

Es mostraran els models de casos d'ús per les funcionalitats que s'ofereixen a l'usuari i s'especificarà les funcions de coordinació que es volen utilitzar en aquest sistema.

4.1 Model de casos d'ús

A continuació es mostrarà el model de casos d'ús de les quatre funcions que ofereix el framework. Es dividiran en les referents a la gestió d'experiments i en la gestió d'aplicacions.

4.1.1 Gestió d'experiments

4.1.1.1 Diagrama de casos d'ús

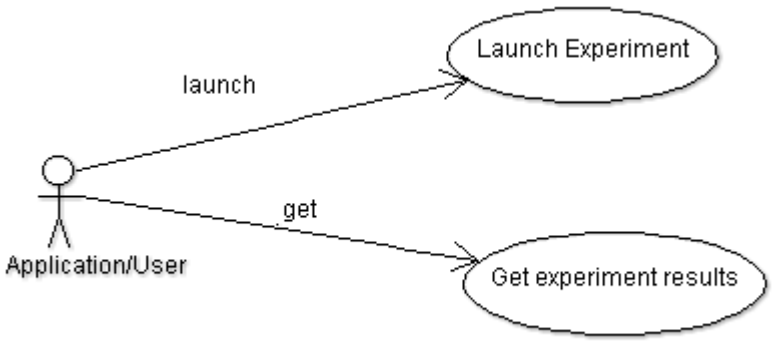


Diagrama 1: Diagrama de casos d'us de gestió d'experiments

4.1.1.2 Especificació dels casos d'ús

Cas d'ús *Llançar experiment*

CAS D'ÚS	*Llançar Experiment*
ACTORS	Aplicació (iniciada per l'usuari)
PROPÒSIT	Iniciar la prova d'una aplicació
RESUM	A partir d'una especificació l'usuari desplega la prova que es vol fer de l'aplicació en l'entorn distribuït realista

Curs típic d'esdeveniments

Aplicació	*LSim*
1. Introdueix l'especificació de l'experiment que es vol realitzar	
	2. Localitza els recursos on s'han de desplegar l'aplicació, desplega l'aplicació especificada en aquest recursos i inicialitza el test
3. Rep la confirmació de que s'ha llançat l'experiment	

Cas d'ús *Obtenció de resultats*

CAS D'ÚS	*Modificar servei*
ACTORS	Aplicació (iniciada per l'usuari)
PROPÒSIT	Obtenir el resultat de l'experiment realitzat
RESUM	L'aplicació demana al sistema si ja ha finalitzat l'experiment i aquest, un cop té els resultats, els retorna a l'aplicació

Curs típic d'esdeveniments

Aplicació	*LSim*
1. Introdueix l'identificador de l'experiment del que en vol obtenir els resultats	
	2. Comprova si l'experiment ha finalitzat. Si no ho ha fet manté l'aplicació esperant. Un cop té el resultat, el retorna a l'aplicació. Si aquesta ha fallat, ho notificaria a l'aplicació
3. Rep el resultat de l'experiment o la notificació de que l'experiment ha fallat durant l'execució si fos el cas	

4.1.2 Gestió d'aplicacions

4.1.2.1 Diagrama de casos d'ús

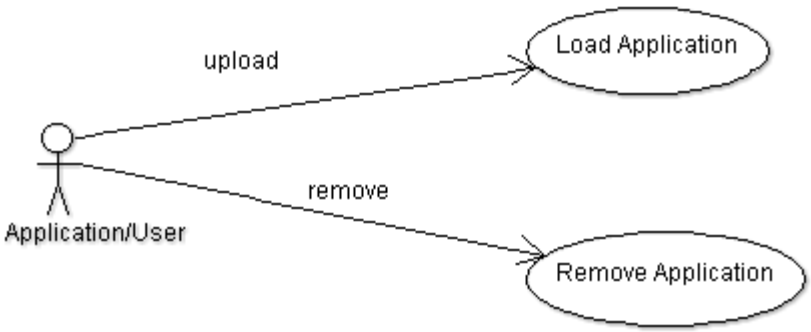


Diagrama 2: Diagrama de casos d'ús de la gestió d'aplicacions

4.1.2.2 Especificació dels casos d'ús

Cas d'ús *Pujar l'Aplicació*

CAS D'ÚS	***Pujar l'Aplicació***
ACTORS	Aplicació (iniciada per l'usuari)
PROPÒSIT	Pujar una aplicació per provar en el sistema
RESUM	L'aplicació facilita els fitxers necessaris per executar l'experiment

Curs típic d'esdeveniments

Aplicació	***LSim***
1. Introdueix l'URL de l'aplicació que es vol pujar al sistema	
	2. Comprova si l'URL és accesible. Si és així, copia els fitxers de l'aplicació al sistema
3. Rep la confirmació que s'ha pujat correctament l'aplicació	

Cas d'ús *Esborrat d'aplicació*

CAS D'ÚS	***Esborrat d'aplicació***
ACTORS	Aplicació (iniciada per l'usuari)
PROPÒSIT	Esborrar l'aplicació del sistema
RESUM	S'esborra una aplicació prèviament pujada al sistema

Curs típic d'esdeveniments

Aplicació	*LSim*
1. Introdueix l'identificador de l'aplicació a ser esborrada	
	2. Comprova si l'aplicació està en el sistema. En cas de ser-hi l'esborra i ho notifica a l'aplicació. En cas de no ser-hi o si hi ha algun error en l'esborrat també se li ho notifica.
3. Rep la notificació d'esborrat en cas positiu, o el motiu pel que no s'ha pogut realitzar	

4.2 Funcions de coordinació

Cal de definir un conjunt de funcions de coordinació que la llibreria ofereix a l'usuari per tal de controlar i coordinar les proves de l'aplicació un cop desplegada.

En aquest apartat s'introdueix quines són les funcions que es volen oferir a l'usuari, no el mètode per coordinar-les, aquests mètodes es presenten a l'apartat de disseny.

El conjunt de funcions és el següent:

Nom	Funcionalitat	Rep	Envia
Inicialització	Obtenir la configuració i paràmetres per iniciar el test	Configuració del test	Possible paràmetre de resposta d'inicialització
Start	Sincronitza el començament de l'experiment amb els altres nodes	Informació resultant de la inicialització	
Sincronització	Crea un punt on es sincronitzen tots els nodes que intervenen en el test	Informació per prosseguir el test	Informació de l'estat del test
Stop	Atura l'execució del test fins a rebre una altra senyal de start		Resultat parcial del test
Enviament de resultats	Envia el resultat de l'execució del test		Resultat del test

Taula 3: Especificacions de les funcions de coordinació

La taula 3 hi ha representat el nom de la funció, una descripció de la seva funcionalitat, la informació que pot rebre i la informació que pot enviar.

5 Disseny

LSim està dividit en dues parts relacionades amb dues de les grans problemàtiques a resoldre, el desplegament i la coordinació de les proves.

La primera part es tracta del framework LSim Framework. Aquest framework el que solucionarà és tota la problemàtica de desplegar i executar d'una forma transparent les aplicacions en els diferents recursos disponibles per a les proves. A més permet la gestió d'experiments, on a partir de diferents especificacions d'experiments els llança aquest en els diferents recursos existents.

La segona part es tracta de la llibreria LSim Library. La llibreria s'encarrega de donar diferents elements i funcions per coordinar les proves i recollir-ne resultats de forma transparent l'aplicació. Es vol que aquest mètodes siguin poc intrusius al codi de l'usuari.

5.1 LSim Library

La LSim Library és la que conté les funcions i elements de coordinació de l'experiment. La intenció d'aquesta llibreria és que, d'una forma poc intrusiva al codi, permeti avaluar l'aplicació en el framework. Per tant amb la implementació de l'aplicació i afegint funcions de coordinació, que es pugui adaptar d'una forma fàcil a l'entorn d'execució i provar l'aplicació en un entorn distribuït realista, i tot de forma transparent a l'usuari.

5.1.1 Elements de LSim Library

LSim Library té elements per a facilitar la coordinació i recollida de resultats de les proves. Aquest dos elements són rols que poden adaptar les aplicacions i funcions de coordinació entre aquest rols.

L'objectiu és que la LSim Library sigui poc intrusiva en el codi de l'aplicació a provar. Per això s'ofereix un conjunt de funcions que fan transparent la coordinació, on simplement bloqueja l'execució de l'aplicació fins a coordinar-se amb el conjunt d'elements en la prova.

Per facilitar la coordinació i fer-la escalable a un conjunt més gran de màquines a provar, LSim té un conjunt de rols que pot adquirir per tal de coordinar l'execució de les proves.

5.1.1.1 Rols de LSim Library

Per a poder coordinar les aplicacions, LSim Library ofereix diferents rols que pot adoptar la implementació de l'aplicació. Aquest rols són el Coordinator, el Worker i l'Evaluator.

1. El Coordinator és el rol que permet a l'aplicació coordinar les diferents instàncies de Workers que s'estiguin executant. És el que porta el control de tot l'experiment, decideix en quin punt s'executa cada fase i s'encarrega de distribuir la informació necessària per a cada prova. És el que gestiona la configuració de l'experiment i s'encarrega de fer-la arribar a cada un dels elements de la prova. De moment està pensat que només n'hi hagi un sol per

cada execució, però està pensat per fer-lo escalable i tolerar fallades, afegint més elements de Coordinator més endavant.

2. El Worker és el rol que executa els elements a provar de l'aplicació en si. Cada una de les instàncies de Worker és el que s'executaria de forma independent en cada un dels recursos, a més de que es poden parametritzar per adquirir el comportament que es vulgui. Dins d'un experiment poden haver-hi diferents Workers, fent que es puguin posar a provar diferents elements en un mateix experiment si aquest ho requereix. Els Workers es bloquejaran en les fases fins a rebre la senyal del Coordinator per poder seguir executant-se, juntament amb informació per les proves depenent de la fase.
3. Finalment l'Evaluator és el rol que permetrà la recollida dels resultats de tots els Workers que es vulguin avaluar. També el coordina el Coordinator, que li especificarà els Workers a avaluar i tota la resta d'informació necessària.

Cada un d'aquest elements s'han d'implementar per a realitzar les proves satisfactòriament, on la realització del Coordinator i l'Evaluator no ha de suposar un gran esforç per l'usuari.

Inicialment s'ha separat l'element de coordinació del d'avaluació perquè el procés d'avaluació pot ser costós i consumir recursos destinats a l'execució de l'experiment, especialment en el cas que s'hagin de fer avaluacions parcials durant l'execució de l'experiment.

5.1.1.2 Fases de LSim Library

Tal i com s'ha comentat, LSim ofereix diferents funcions de coordinació per tal d'executar proves en entorns remots. A continuació es parlarà de les principals funcions, que anomenarem fases de LSim. Les fases adquireixen un comportament diferent depenent del rol que les estigui executant, així establint una comunicació entre rols depenent de la fase en que es trobi. Per tant, es detallarà cada un d'aquests comportaments per cada un dels rols. Cada una d'aquestes fases el que fa és bloquejar l'execució de l'aplicació fins a rebre una senyal, ja sigui externa o interna d'aquesta, tot això transparent a l'usuari.

Les fases de coordinació que inclourem a LSim són:

- Fase d'inicialització: Serveix per a rebre paràmetres pel test i configurar-lo.
- Fase de start: Atura l'execució del codi fins a rebre el senyal que pot començar a executar-se la prova.
- Fase de sincronització: Serveix per generar punts de control dins l'execució del codi. Quedara bloquejat fins a rebre una indicació de que pot seguir des de el punt on s'ha cridat la fase. Sincronitza tots els elements que estiguin en la fase.
- Fase d'enviament de resultats: Envia els resultats obtinguts fins llavors a un Evaluator i aquest els avalua. Pot suposar la finalització de l'experiment o simplement l'avaluació d'una part.

S'ha presidit de la funció de stop ja que encara no s'havia definit del tot el seu funcionament.

5.1.2 Comportament de LSim Library

Cada un dels rols de LSim té un comportament diferent segons la fase de LSim que estigui executant. Aquest comportament estableix la comunicació necessària per a coordinar les proves utilitzant LSim. A continuació és mostrarà quin comportament adquireix cada rol i la comunicació entre ells en cada una de les fases.

Cada una de les fases disposa d'un temps per realitzar-se, a part d'un temps total d'execució del test. Si s'esgota aquest temps, cada un dels rols posa en marxa la mesura necessària davant d'aquest esdeveniment, ja que és indici que hi ha alguna fallada en algun element. Es pren aquesta mesura per evitar deixar elements consumint recursos en cas de fallada d'algun dels elements de l'experiment.

Per a explicar el comportament dels components de LSim es fa servir unes gràfiques a més d'un llenguatge propi per expressar la comunicació d'una forma més formal. A continuació es descriurà com s'han d'interpretar els gràfics i aquest llenguatge:

1- Gràfic: La LSim Library està representada com a un rectangle amb LSim, adjuntant al costat quin comportament adapta la llibreria depenent del rol. El rol que adquireix l'aplicació ve representat en una el·lipse amb el nom del rol que adquireix. En cas de ser un Worker, s'especificarà de quin numero Worker es tracta, de 1...n. Finalment les transaccions i comunicacions entre elements estaran representats amb fletxes.

2- Llenguatge: Es descriuran tres part, que són: l'estat inicial, les transaccions i finalment el cas de timeout. A continuació es descriurà els diferents elements del llenguatge que s'utilitza.

- Elements: Els elements que es comuniquen estan representats pel rol que tenen. Està dividit en dues parts, la part de l'aplicació i la part de la LSim Library. A continuació es mostren les seves representacions. El que hi ha dins la clau és l'estat que s'explica en el següent punt.

  -Aplicació: Element[Estat]

  -LSim Library: LSim.Element[Estat]

  On per exemple si es tracta d'un Worker tindrem: Worker[Estat] i LSim.Worker[Estat]

- Estat: Els elements tenen dos estats: run i blocked. L'estat de run és quan l'aplicació s'executa de forma normal. L'estat de blocked és quan l'aplicació no pot seguir executant-se perquè està esperant algun missatge. En cada transacció els elements poden passar de run a blocked o de blocked a run.

- Transaccions: A partir de les transaccions els elements es comuniquen i canvien d'estat. Les transaccions es representen com a una fletxa entre element i element, on entre mig de la fletxa s'especifica el missatge que s'envien entre els elements. Aquesta és la seva representació:

  --[missatge]-->

Exemples de transaccions:

Worker[run] –[missatge]--> LSim.Worker[blocked]

LSim.Worker[blocked] –[missatge]--> LSim.Coordinator[run]

- Missatges: Els missatges han de comunicar els elements, per tant s'han de poder especificar els paràmetres que comuniquen. Llavors els missatges tenen un identificador per missatge i un conjunt de paràmetres que s'envien. Així anirà representat:

  Identificador(paràmetre1, paràmetre2, ..., paràmetreK)

  Exemples:

  Coordinator[run] --[init(configuration, timeout)]--> LSim.Coordinator[blocked]

- Elements de control: Els elements de control serveixen per definir comportaments en les transaccions. Hi ha dos tipus d'elements: el *for each* i el *upon event*. El *for each* permet realitzar una transaccióper cadascun dels elements de la condició et deixi i el *upon event* realitzarà la transacció un cop es compleixi certa condició. Es representen de la següent manera:

  for each(condició) **do**

  transacció 1

  transacció 2

  ....

  transacció N

  upon event(condició d'inicialització) **do**

  transacció 1

  transacció 2

  ....

  transacció N

  En l'apartat de les variables veurem un exemple de cadascun.

- Variables: Les variables serveixen per representar les condicions dels elements de control. Per exemple:

  Experiment_Workers = conjunt de Workers de l'experiment.

  num_workers_message = enter amb el nombre de missatges que rep el Coordinator dels Workers.

  for each (Worker_i € Experiment_Workers) **do**

  LSim.Coordinator[run] –[status(parameter_i)]--> Worker_i[blocked]

  upon event (|Experiment_Workers| == num_workers_message) **do**

  k = num_workers_message

LSim.Coordinator[run] –[status(U<i=1 to k> stat_i)]--> Coordinator[run]

5.1.2.1 Fase d'inicialització

La fase d'inicialització és per a rebre els paràmetres de la prova i configurar-ne els diferents elements per tal d'executar el que s'hagi especificat en l'experiment.

En aquesta fase apareix un element relacionat amb LSim que encara no s'ha presentat, el Launcher, que apareix a la secció on està dissenyat el LSim Framework. Per ara només apuntar que el Launcher llança l'experiment i és el que dóna la senyal d'inicialització al Coordinator juntament amb la seva configuració.

El Coordinator s'encarrega de rebre la configuració per aquesta prova i aïllar per cada un dels elements quina és la seva configuració. Un cop té la configuració de cada part, simplement els envia una senyal d'inicialització *init* amb la configuració de cada element per aquella prova. Finalment es bloqueja fins a rebre la resposta de tots els elements iniciats per tal de poder iniciar la prova.

Els Workers es bloquegen fins a rebre la senyal *init* amb la configuració per a la prova. Seguidament fan tot el procés d'inicialització, per a que quan rebin la senyal de començar ja estigui tot preparat. Un cop configurats, aquest responen al Coordinator que ja han fet el procés d'inicialització.

En aquest cas l'Evaluator té el mateix comportament que un Worker.

A continuació es mostra una gràfica amb la comunicació entre Coordinador i Workers en aquesta fase. Aquí es formalitza una mica més el comportament dels elements.

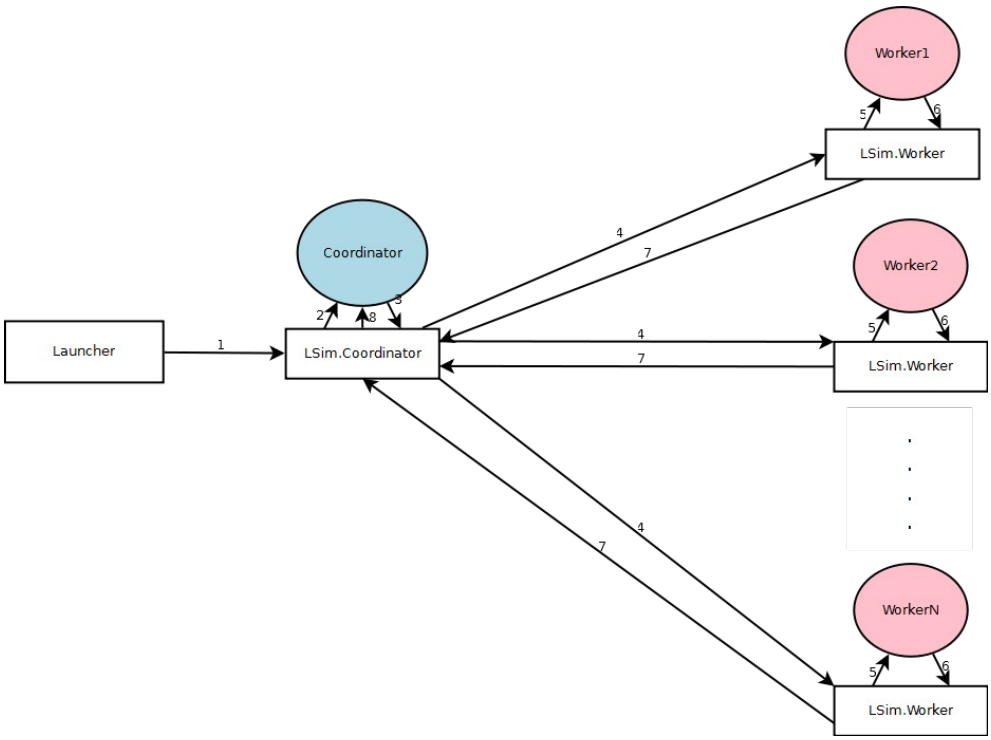


Figura 3: Fase d'inicialització

Estat inicial:
-Coordinator[blocked] es troba bloquejat a causa de que ja està esperant la senyal d'inicialització del Launcher, juntament amb la configuració del test a executar.
-Worker[blocked] es troba bloquejat esperant la senyal d'inicialització del Coordinator, juntament amb la configuració del test a executar.
-Launcher[run]
-Experiment_Workers = conjunt de Workers en l'experiment
-workers_reply = nombre de missatges de Worker a Coordinator

Transaccions:
1- Launcher[run] --[init(configuration)]--> LSim.Coordinator[blocked]
2- LSim.Coordinator[blocked] --[init(configuration)]--> Coordinator[run]
3- Coordinator[run] --[init(init_elements)] --> LSim.Coordinator[blocked]
4- **for each** (Worker_i € Experiment_Workers) **do**
LSim.Coordinator[blocked] –[init(w_configuration_i**)]--> -->**LSim.Worker_i[blocked]
5- LSim.Worker_i[blocked] –[init(configuration_i)]--> Worker_i[run]
6- Worker_i[run] --[init(feedback_i)]--> LSim.Worker_i[run]
7- LSim.Workers[run] --[init(parameter)]--> LSim.Coordinator[blocked]
8- **upon event** (|Experiment_Workers| == workers_reply) **do**
n = workers_reply
LSim.Coordinator[blocked] –[init(U<i=1 to n> parameter_i]--> Coordinator[run]

Time out:
-El Coordinator iniciarà l'aturada del test enviant a tots els elements que deixin d'executar-se.
-El Worker aturarà l'execució.

5.1.2.2 Fase de start

La fase de start indica quan pot començar la prova.

El Coordinator indica quan a cada Worker quan ha de començar. En aquesta fase de start també pot enviar als Workers informació com a paràmetre, en cas de que fos necessari.

El Worker està bloquejat fins a rebre la senyal de start del Coordinator. Un cop rep la senyal, agafa els paràmetres(si n'hi ha) i es desbloqueja per a iniciar la prova.

A continuació es mostra una gràfica amb la comunicació entre Coordinador i Workers en aquesta fase. Aquí es formalitza una mica més el comportament dels elements.

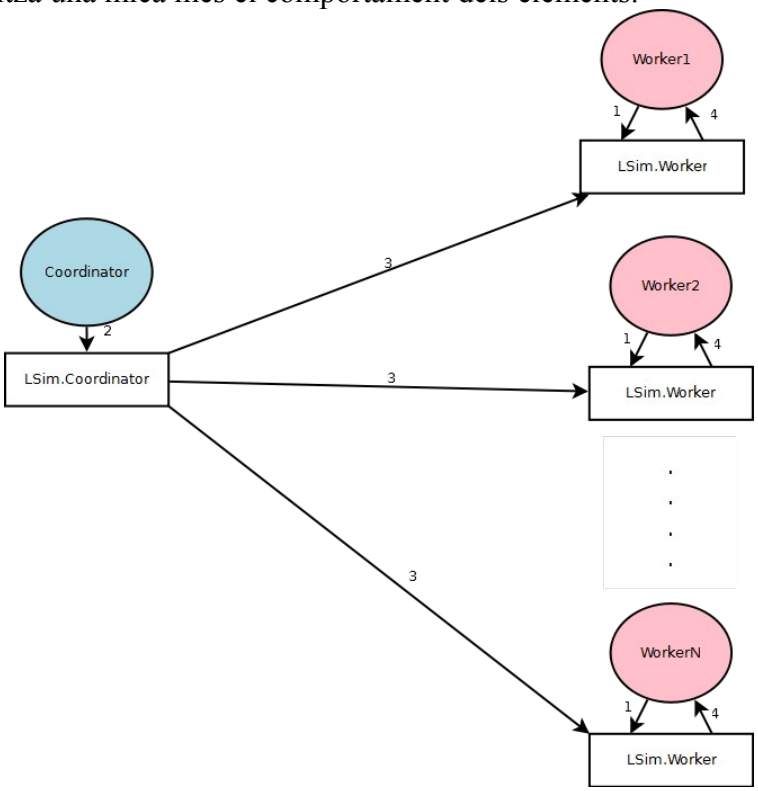


Figura 4: Fase de start

Estat inicial:
-Coordinator[run]
-Worker[run]
-Experiment_Workers = conjunt de Workers en l'experiment

Transaccions:
1-Worker[run] --[start()]--> LSim.Worker[blocked]
2-Coordinator[run] --[start()]-->LSim.Coordinator[run]
3- **for each** (Worker_i € Experiment_Workers) **do**
LSim.Coordinator[run] --[start(parameter1, parameter2, ... , parameterk)]-->
-->LSim.Workers_i[blocked]
4-LSim.Worker[blocked] --[start(parameter1, parameter2, ... , parameterk)]--> Worker[run]

Time out:
-El Coordinator no es bloqueja, per tant no té time out.
-El Worker atura la seva execució.

5.1.2.3 *Fase de sincronització*

Aquesta fase és per a coordinar tots els elements en un punt en concret de l'execució.

El Coordinator es bloqueja fins a rebre la senyal dels Workers que estan en aquella fase. Un cop ha rebut la comunicació dels Workers, LSim.Coordinator ho notifica a la part de l'aplicació del coordinador, esperant que aquest li indiqui quan pot iniciar la fase. Un cop s'ha indicat l'inici de la fase, aquest comunica als Workers dins la fase que ja poden començar la fase, amb aquesta senyal es pot enviar algun paràmetre als Workers.

El Worker indica al Coordinator que està esperant per a l'inici de la fase. Amb aquesta senyal pot enviar algun paràmetre al Coordinator. Un cop rep la senyal d'inici de fase, agafa el possible paràmetre del Coordinator i segueix executant el test.

A continuació es mostra una gràfica amb la comunicació entre Coordinador i Workers en aquesta fase. Aquí es formalitza una mica més el comportament dels elements.

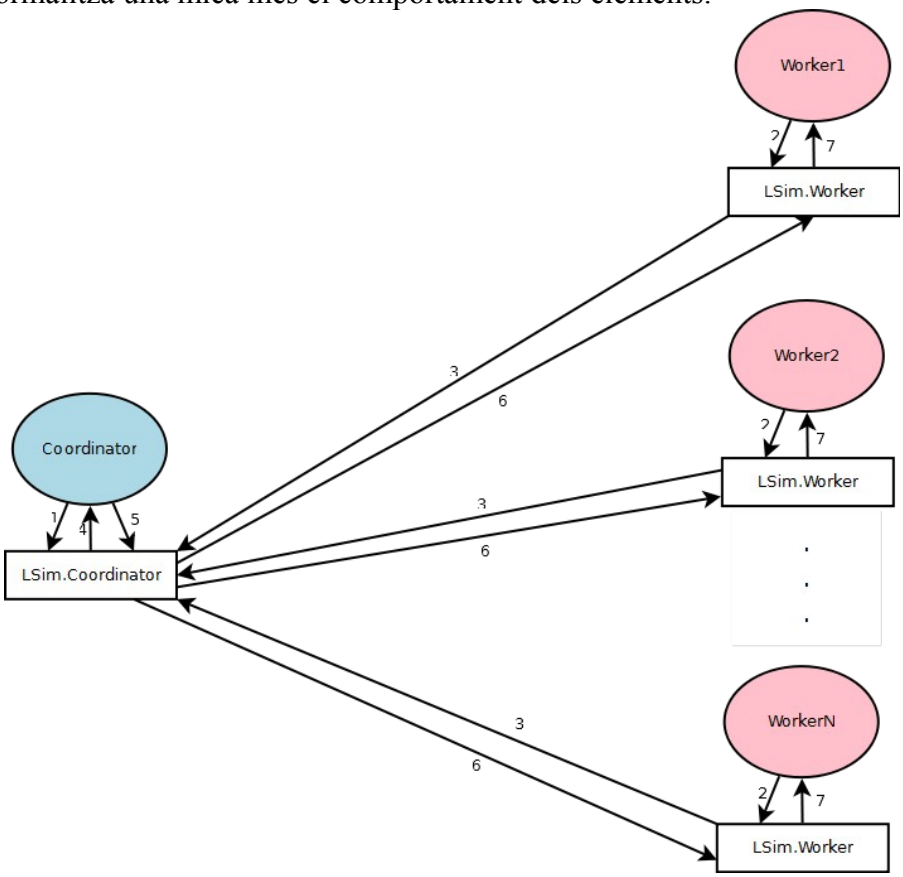


Figura 5: Punt de sincronització

Estat inicial:
-Coordinator[run]
-Worker[run]
- Workers_in_phase = conjunt de Workers que participen en el punt de sincronització
- workers= nombre de missatges de Worker a Coordinator

Transaccions:
1- Coordinator[run] --[phase()]--> LSim.Coordinator[blocked]
2- Worker[run] --[phase(parameter)]--> LSim.Worker[blocked]
3- LSim.Worker[blocked] --[phase(parameter)]--> LSim.Coordinator[blocked]
4- **upon event** (|Workers_in_phase| == workers) **do**
m = workers
LSim.Coordinator[blocked] --[phase(U <i=1 to m> parameter_i)]-->
-->LSim.Coordinator[run]
5- LSim.Coordinator[run] --[phase(parameter1, parameter2, ... ,parameter_k)]-->
LSim.Coordinator[run]
6- **for each**(Worker_i € Workers_in_phase) **do**
LSim.Coordinator[run] --[phase(parameter1, parameter2, ... ,parameter_k)]-->
-->LSim.Workers[blocked]
7- LSim.Worker[blocked] --[phase(parameter1, parameter2, ... ,parameter_k)]--> Worker[run]

Time out:
-El Coordinator comprova si el nombre de Workers que han enviat la senyal és igual o més gran al conjunt de Workers a avaluar. En cas de que sigui cert, envia la senyal de començar la fase als Workers que hagin contestat i als altres els hi envia que poden finalitzar l'execució, ja que si no han respost es suposa que han caigut.

-El Worker atura la seva execució.

5.1.2.4 Fase d'enviament de resultats.

La fase d'enviament de resultat serveix per avaluar les proves realitzades en els Workers. Per fer-ho, l'avaluador s'encarrega de recollir i avaluar tots els resultats dels Workers que s'han d'avaluar. A l'especificació es pot especificar el nombre de Workers que al final s'han d'avaluar per tal de treure resultats de la prova realitzada.

El Coordinator en aquesta fase espera a rebre les peticions de resultat dels Workers. Un cop rep la primera, aquest s'encarrega de buscar un Evaluator i configurar-lo per a avaluar el resultat del test. Després espera a rebre totes les peticions dels Workers i els indica l'adreça de l'Evaluator que avaluarà el test. En aquesta fase el Worker no es bloqueja, fent que pugui continuar executant-se, tot i que està pensat per acabar la seva execució després de l'enviament de resultats.

El Worker pregunta al Coordinator quin és l'Evaluator, un cop rep la contestació, li envia el resultat.

L'Evaluator espera que el Coordinator l'inicialitzi. Un cop configurat espera a rebre el nombre de Workers necessaris per l'avaluació. Finalment avalua els resultats i els guarda on se li hagi indicat.

A continuació es mostra una gràfica amb la comunicació entre Coordinador i Workers en aquesta fase. Aquí es formalitza una mica més el comportament dels elements.

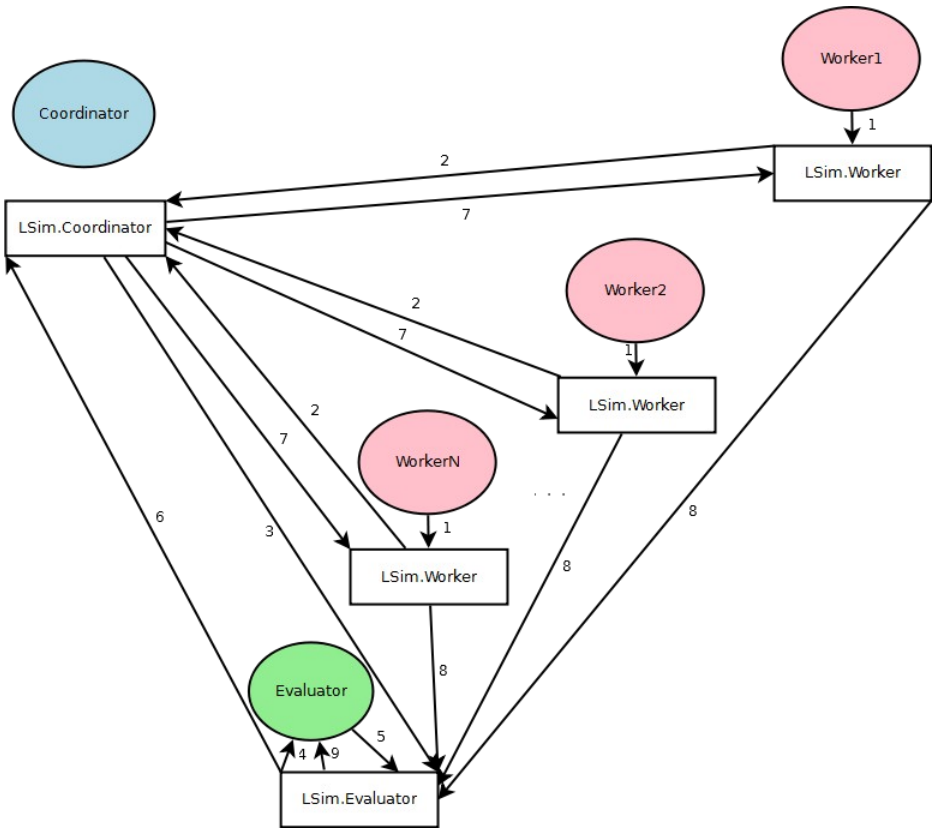


Figura 6: Fase d'enviament de resultats

Estat Inicial:
-Coordinator[blocked], l'aplicació ha instanciat la funció d'enviament de resultats.
-Worker[run]
-Evaluator[blocked], l'aplicació ha instanciat la funció d'enviament de resultats.
-Workers_to_Evaluate = conjunt de Workers a avaluar
- workers_results = nombre de missatges de Worker a Evaluator

Transaccions:
1-Worker[run] --[result(result)]-->LSim.Worker[run]
2-LSim.Worker[run] --[result()]--> LSim.Coordinator[blocked]
3-LSim.Coordinator[blocked] --[init(configuration_e)]--> LSim.Evaluator[blocked]
4-LSim.Evaluator[blocked] --[init(configuration_e)]--> Evaluator[run]
5-Evaluator[run] --[init()]--> LSim.Evaluator[blocked]
6-LSim.Evaluator[blocked] --[init()]--> LSim.Coordinator[blocked]
7- **for each** (Worker_i € Workers_to_Evaluate) **do**
LSim.Coordinator[run] --[result(evaluator_address)]--> LSim.Worker_i[run]
8-LSim.Worker[run] --[result(result_i)]--> LSim.Evaluator[blocked]
9- **upon event** (|Workers_to_Evaluate| == workers_results) **do**
m = workers_results
LSim.Evaluator[blocked] --[result(U <i=1 to m> result_i)]--> Evaluator[run]

Time out:

-El Coordinator quan espera a l'Evaluator, si aquest no contesta, en buscarà un altre. Si rep la contestació del primer en cas de que n'hagi començat un altre, començarà l'experiment amb el primer que contesti i descartarà els altres.

-L'Evaluator si no rep tots els Workers a avaluar, dóna el test com fallat.

5.1.3 Arquitectura de LSim Library

A continuació es mostra l'arquitectura proposada per LSim Library amb un diagrama de classes, juntament amb un petita descripció de cada classe.

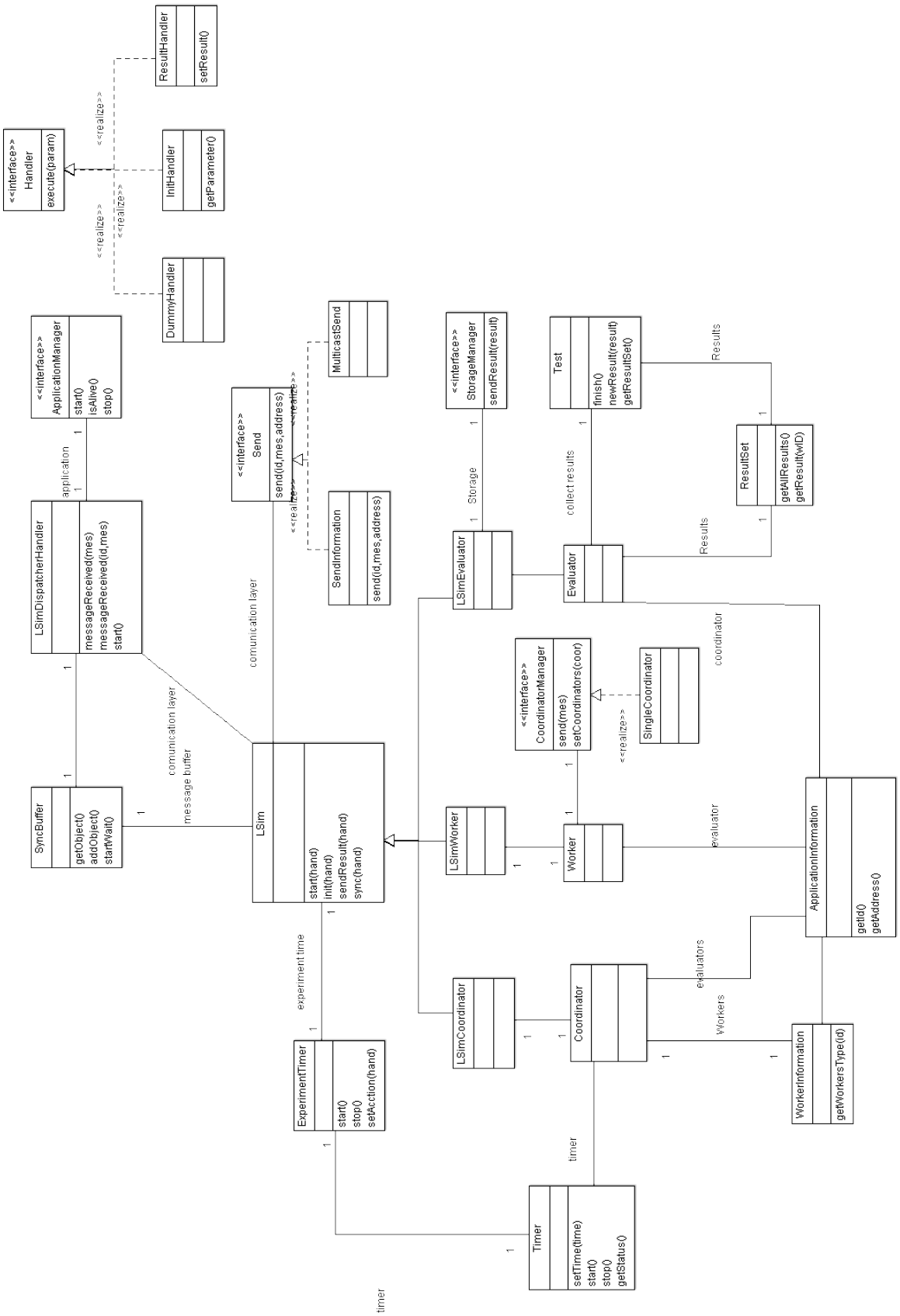


Diagrama 3: Diagrama de classes de LSim Library

-LSim: classe abstracte que implementa els mètodes generals de tots els possibles rols i mètodes que d'implementar cada rol.

-LSimCoordinator: dóna les diferents funcions del Coordinator en les diferents fases. L'usuari utilitza aquesta classe quan l'aplicació actua com a Coordinator.

-LSimWorker: dóna les diferents funcions del Worker en les diferents fases. L'usuari utilitza aquesta classe quan l'aplicació actua com a Worker.

-LSimEvaluator: dóna les diferents funcions de l'Evaluator en les diferents fases. L'usuari utilitza aquesta classe quan l'aplicació actua com a Evaluator.

-Coordinator: implementa el comportament del Coordinator. L'implementació del comportament està separat del LSimCoordinator per si en un futur és vol implementar un altre comportament de Coordinator, tal com utilitzar més d'un Coordinator. D'aquesta forma es pot millorar la llibreria i canviar la lògica del Coordinator de forma transparent a l'usuari.

-Worker: implementa el comportament del Worker. Està separat del LSimWorker pel mateix que en el cas del Coordinator.

-Evaluator: implementa el comportament de l'Evaluator. Està separat del LSimEvaluator pel mateix que en el cas del Coordinator.

-ApplicationInformation: té les dades d'un element de LSim, tal com el seu identificador i la seva adreça.

-WorkerInformation: conté tots els Workers que participen en l'experiment. Els divideix per tipus de Workers i li serveix al Coordinator per poder accedir a ells. Per tant cada tipus de Workers tindrà un llistat de Workers que formen.

-CoordinatorManager: conté els mètodes de comunicació del Worker cap al Coordinator. Està separat del Worker per fer transparent la comunicació amb els Coordinators. En un futur està pensat tenir més d'un Coordinator, per tant la manera de comunicar-se variarà en funció del nombre de Coordinators. Segueix el patró Strategy per adaptar-se a aquest futur canvi. Actualment hi ha la classe *SingleCoordinator* que permet comunicar-se amb un sol Coordinator.

-StorageManager: interfície que l'usuari ha d'implementar per especificar que ha de realitzar l'Evaluator al finalitzar l'avaluació del test.

-Test: controla l'arribada de resultats dels Workers. El control del test està separat de l'Evaluator per fer transparent el control del test de la recollida de resultats que efectua l'Evaluator.

-ResultSet: conjunt de resultats dels Workers. Al fer l'avaluació l'usuari haurà d'agafar els resultats a partir d'aquesta classe.

-Handler: interfície que permet interactuar l'usuari amb les fases de inicialització, start, sincronització i enviament de resultats de LSim Library. Executa la part de l'aplicació a dins d'aquestes fases. Permet a l'aplicació obtenir paràmetres de l'experiment, enviar resultats o avaluar l'experiment. Gràcies al Handler les funcions de LSim Library són poc intrusives al codi de l'aplicació, ja que permet executar la part referent a l'experiment de l'aplicació a dins de les pròpies fases de LSim Library. El Handler segueix el patró Strategy, on l'aplicació indica quin comportament tindrà en cada fase.

-ExperimentTimer: temporitzador que fixa la durada de l'experiment. L'avorta en cas de superar aquest temps.

-Timer: implementació d'un temporitzador.

-SyncBuffer: buffer que serveix per a rebre missatge de la part de comunicació. Com que està pensat que s'executí en paral·lel l'aplicació de la part de comunicació pròpia de LSim Library, aquest buffer permet comunicar les dues parts sense problemes.

-LSimDispatcherHandler: part que comunica el framework amb LSim.

-ApplicationManager: interfície que ha d'implementar l'usuari per executar la seva aplicació en el framework. Simplement implementant la *function start()*, tal com si fos un main, ja pot funcionar l'aplicació en el framework que s'utilitzi.

-Send: interfície que conté les primitives per enviar missatges als elements de LSim. Està en forma d'interfície per a poder escollir la millor implementació segons l'escala de l'experiment.

5.1.4 Disseny de les fases

Per a realitzar els diagrames de seqüència s'ha utilitzat un plugin d'Eclipse anomenat Diver, ajuda a fer l'anàlisi del codi d'una aplicació tot generant diagrames de seqüència. Aquesta eina genera diagrames més agradables a la vista que altres eines com l'ArgoUML, l'inconvenient que té és que afegeix totes les classes que intervenen en el codi, per tant les classes de Java en algun cas també sortirà en algun diagrama. A la fase de disseny del projecte aquest diagrames es varen fer sobre paper, per tant s'ha decidit utilitzar Diver per generar els diagrames de seqüència, ja que en la

majoria dels casos quedaven més complerts que utilitzar altres eines. En la fase de disseny surten totes les funcionalitats inicialment pensades, cosa que després en el prototipus implementat s'ha deixat de banda alguna d'aquestes.

Les fases de LSim segueixen la següent estructura:

- Part de bloqueig o espera. Es bloqueja l'execució fins a rebre la senyal de poder continuar. Retorna informació referent a l'experiment. La *function wait[Phase_Name]() return Object*, representa aquesta part.

- Part d'execució de l'aplicació. S'executa el Handler especificat a la fase per l'aplicació. Permet la interacció de l'aplicació amb les fases de LSim Library, on pot obtenir paràmetres referents a l'experiment, retornar resultats o avaluar l'experiment. La funció del Handler, *function execute(Object) return Object*, és la que s'executa en aquesta part.

- Part de comunicació. Es la part on els diferents rols de LSim Library es comuniquen els resultats de la fase. Normalment es comunica el que retorna la funció del Handler. La funció que representa aquesta part és *function end[Phase_Name](Object)*.

Aquestes funcions permeten definir els diferents comportaments dels rols de LSim Library. L'implementació de la primera i útilma part és la que permetrà a cada rol definir el seu comportament. A continuació es mostra els diagrames de seqüència de cada una de les fases de LSim Library:

1-Fase d'inicialització

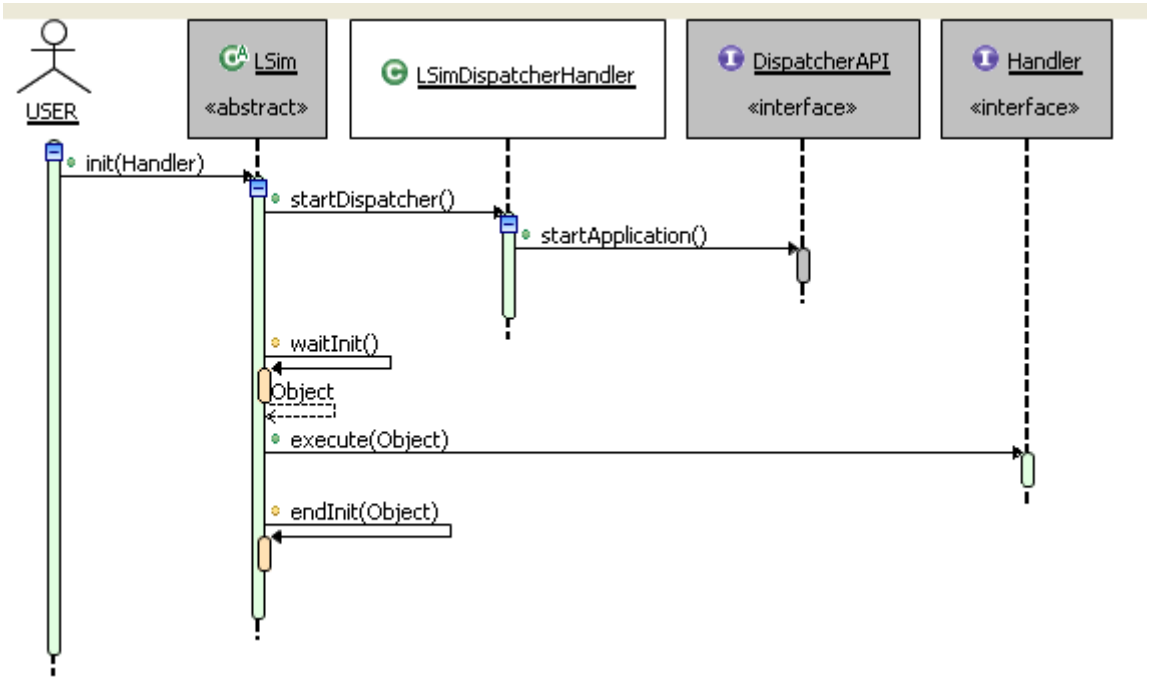


Diagrama 4: Diagrama de seqüència de function init(Handler);

A la fase d'inicialització és quan és crea l'enllaç amb la part de comunicació.

2-Fase de start

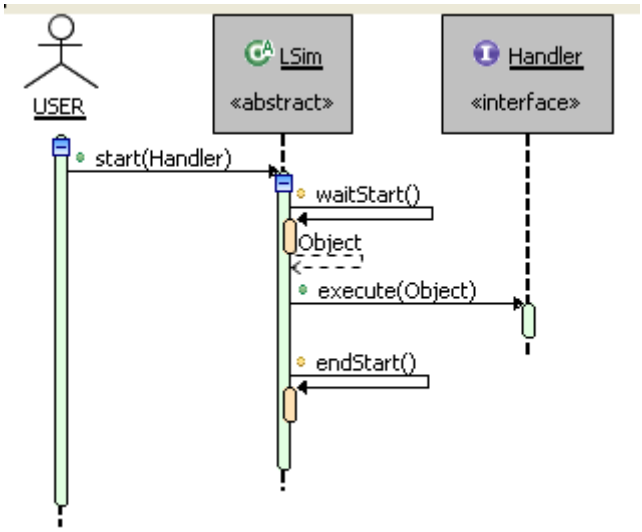


Diagrama 5: Diagrama de seqüència de function start(Handler)

3-Fase de sincronització

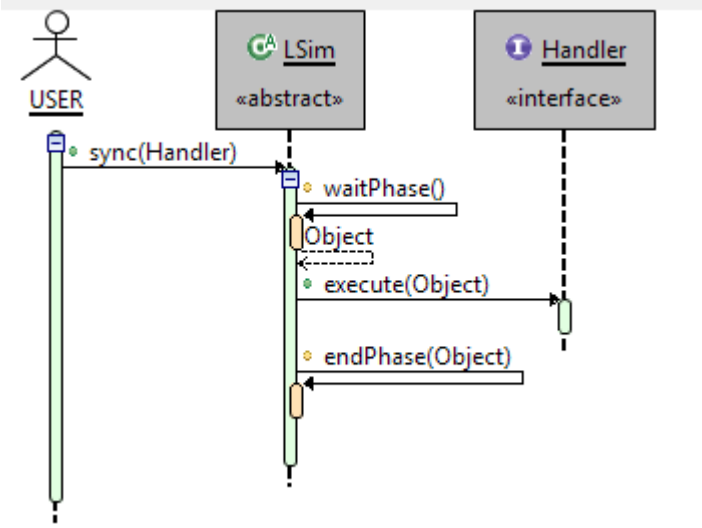


Diagrama 6: Diagrama de seqüència de function sync(Handler)

4-Fase d'enviament de resultats

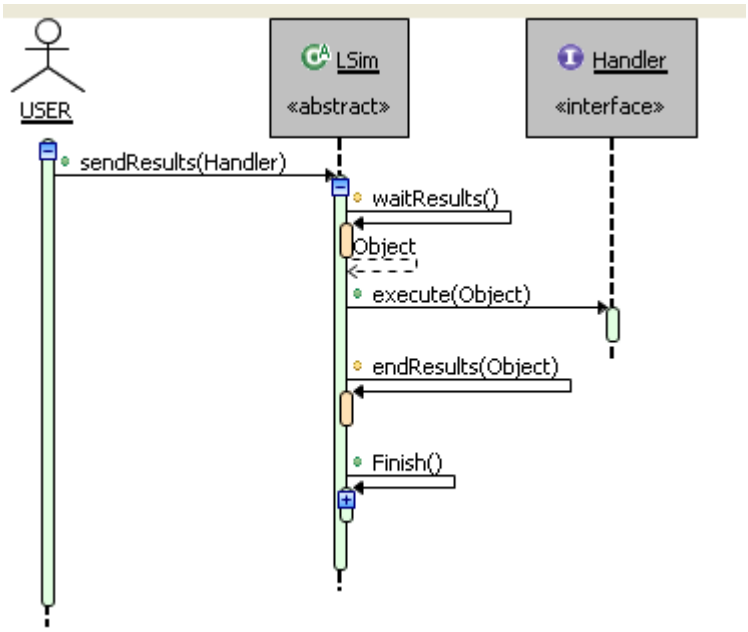


Diagrama 7: Diagrama de seqüència de function sendResult(Handler)

A continuació es veurà els diferents dissenys dels comportaments de les fases en els diferents rols. Primer s'aclarirà les funcions de les següents classes per facilitar la lectura dels diagrames.

-S'utilitza el *SyncBuffer* com a mitjà de comunicació amb la capa de comunicació. El mètode *function startWait()* i *function startWait(Timer time)* el que fa és bloquejar l'aplicació fins que rep un missatge evitant així que consumeixi recursos de la màquina.

-El *ExperimentTimer* és el que controla el temps de l'experiment i n'atura l'execució en cas d'haver esgotat aquest temps. Durant l'experiment s'assignen dos temps en el temporitzador. El primer s'assigna a la fase d'inicialització abans de rebre la configuració de l'experiment. Aquest ha de ser més petit en comparació amb el temps de l'experiment. La seva utilitat és aturar l'execució en cas de no rebre la configuració de l'experiment en el temps assignat. El següent se li assigna durant la fase de start. Aquest temps es pot donar com a paràmetre i assignar-li a partir de la *function setExperimentTime(Integer)* de la classe LSim, sinó LSim ficarà un temps per defecte. Aquest ha de ser una estimació de la durada de l'experiment.

5.1.4.1 Disseny de les fases en el Coordinator

A continuació és veurà el comportament del Coordinator en cada una de les fases.

5.1.4.1.1 Fase d'inicialització

En aquesta fase el Coordinator rep els paràmetres del Launcher, configura el test i envia la configuració als diferents Workers.

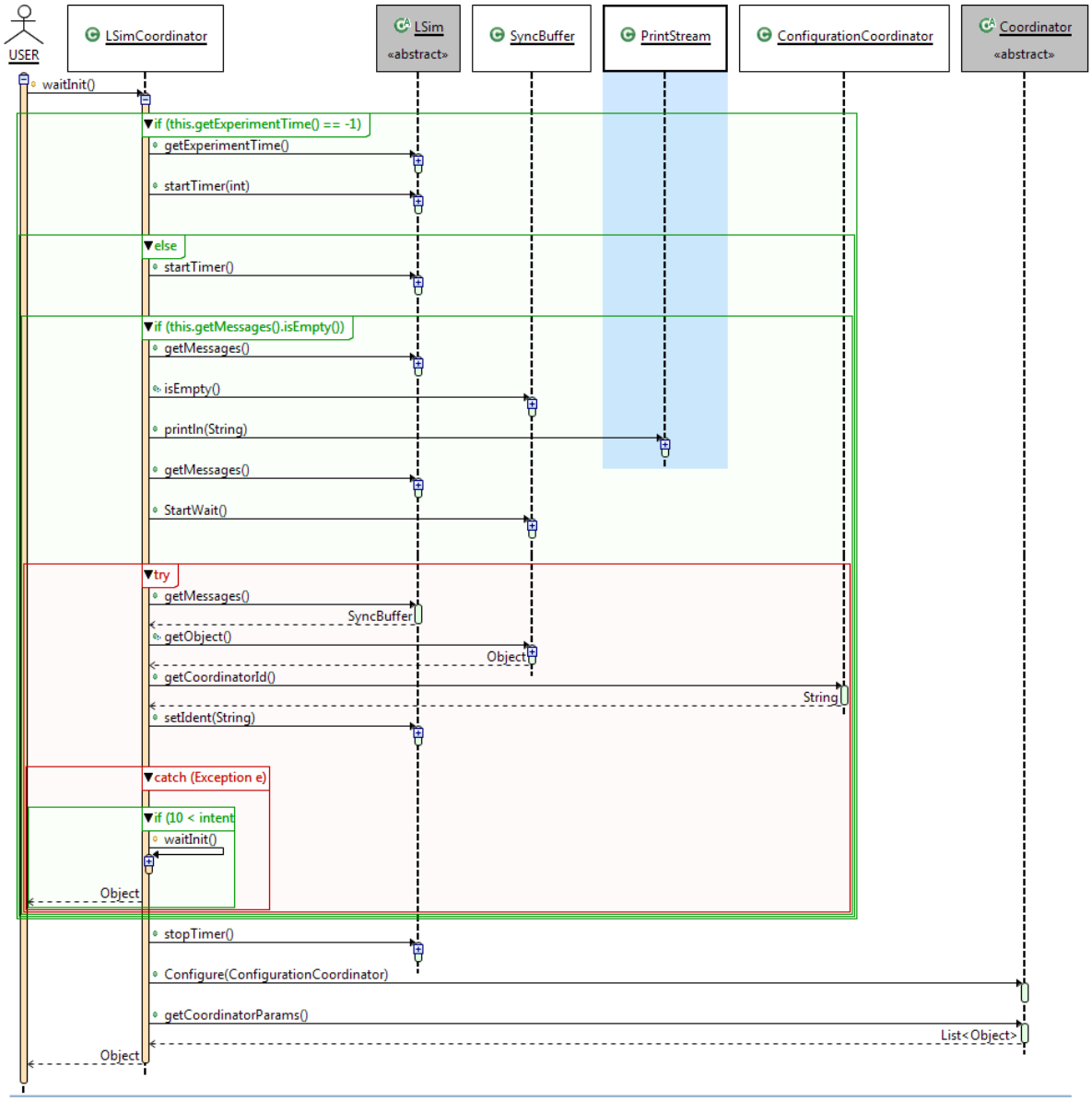


Diagrama 8: Diagrama de seqüència de function waitInit() return Object del Coordinator

A continuació es comentarà el diagrama 8 que conté la part de bloqueig de la fase d'inicialització.

S'inicialitza el temporitzador abans de rebré la configuració a través de *LSim* amb *function startTimer(Integer)*.

A continuació inicia el bloqueig a través del del SyncBuffer amb la *function StartWait()*. L'execució quedarà bloquejada fins a rebre la configuració del Launcher. Comprova que la configuració enviada sigui correcte, en cas de ser-ho segueix amb l'execució, en cas contrari es torna a bloquejar. Seguidament atura el temporitzador a través de la *function stopTimer()*, configura el Coordinator i retorna els paràmetres de l'experiment per l'aplicació.

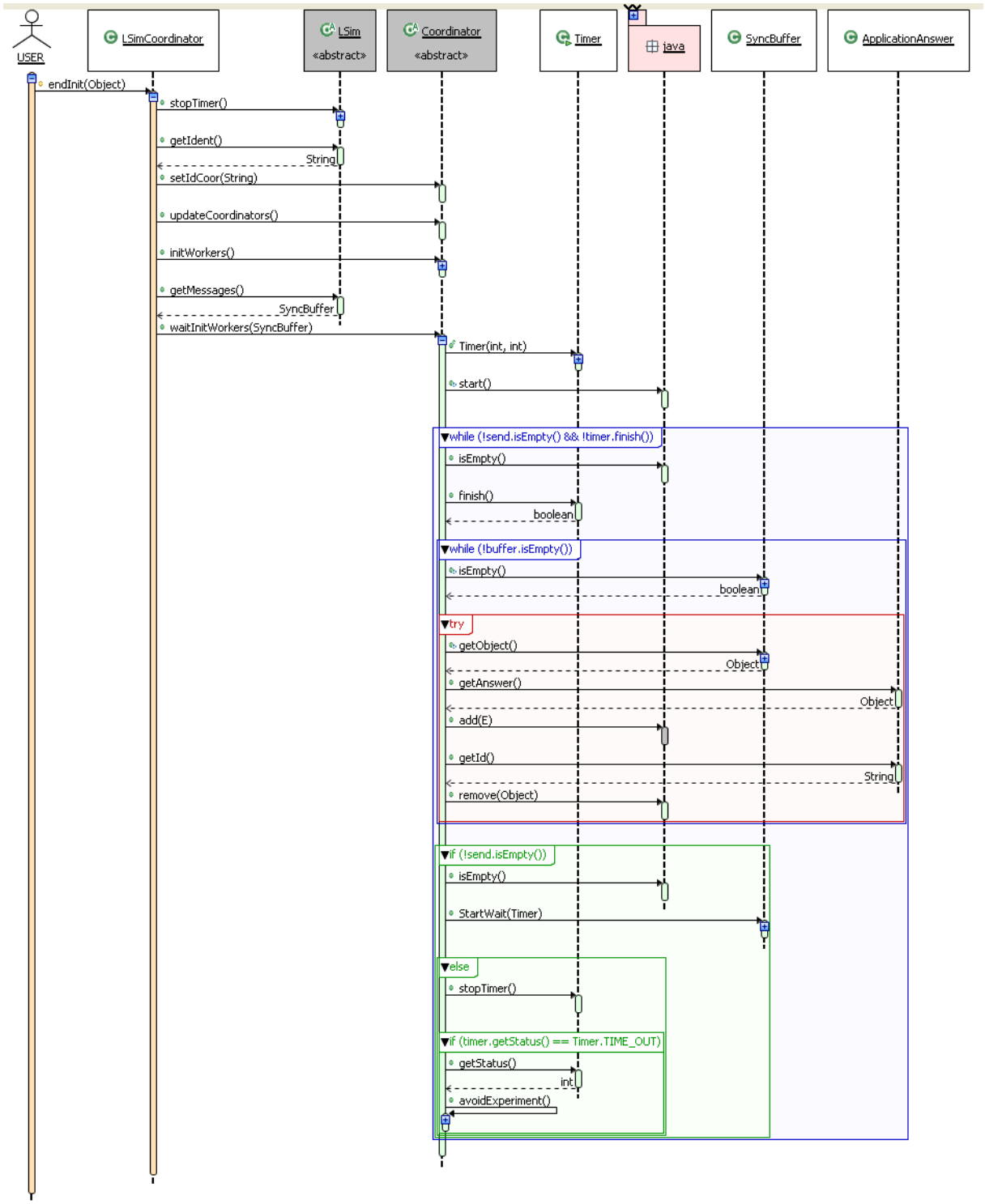


Diagrama 9: Diagrama de seqüència de function endInit(Object) del Coordinator

En la *function endInit(Object p)* el Coordinator representada al diagrama 9 inicialitza tots els Workers de l'experiment.

La funció del Coordinator *function initWorkers()* s'encarrega d'enviar el missatge d'inicialització i configuració als Workers. Aquest crea la variable *send* amb el conjunt de Workers que s'han inicialitzat.

El primer bucle comprova que s'hagin rebut les respostes dels Workers inicialitzats a través de la

variable *send*. Cada cop que es rep una resposta, s'extreu el Worker que ha contestat de *send*. Aquest té un temporitzador propi, ja que el Coordinator es bloqueja, on si esgota el temps i no han contestat tots els Workers s'avorta d'experiment.

El segon bucle bloqueja el Coordinator fins a rebre alguna resposta. Les respostes són instàncies de la classe *ApplicationAnswer*, les qual poden contenir paràmetres dels Workers cap al Coordinator. Aquest es guardaran en una estructura interna per a ser retornats a la següent fase a l'aplicació.

5.1.4.1.2 Fase de start

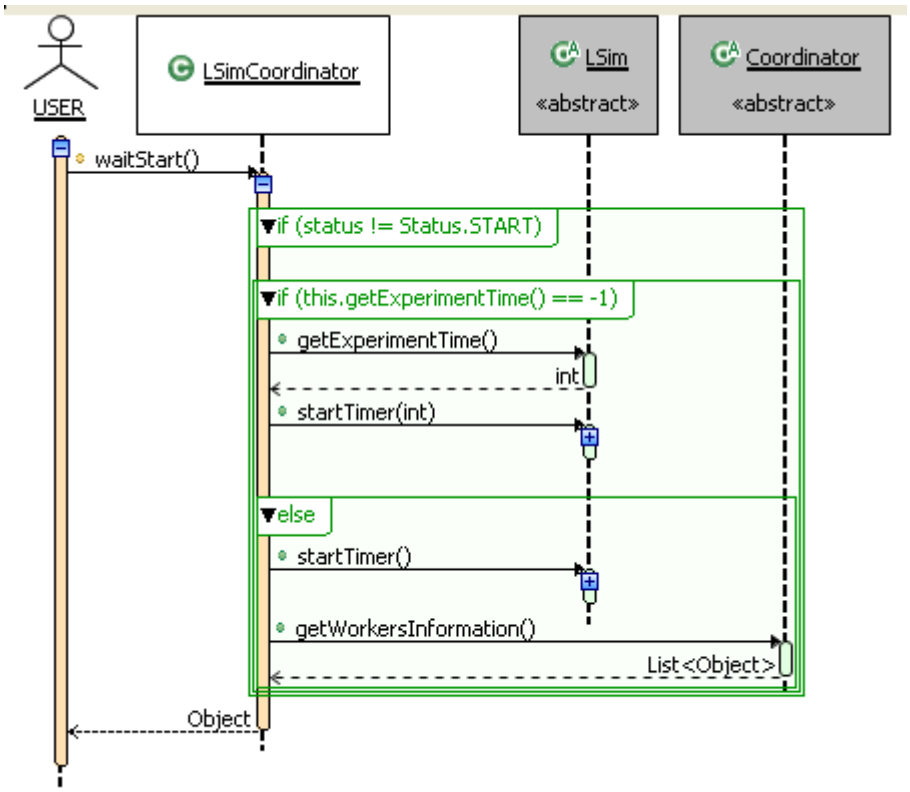


Diagrama 10: Diagrama de seqüència de function waitStart() return Object del Coordinator

El diagrama 10 mostra la part de bloqueig del Coordinator. En aquest cas no és necessari el bloqueig, ja que no ha d'esperar cap senyal. El Coordinator inicialitza el segon temporitzador. Retorna el conjunt de respostes dels Workers. Aquest conjunt són el paràmetre que envia de resposta els Workers a la fase d'inicialització.

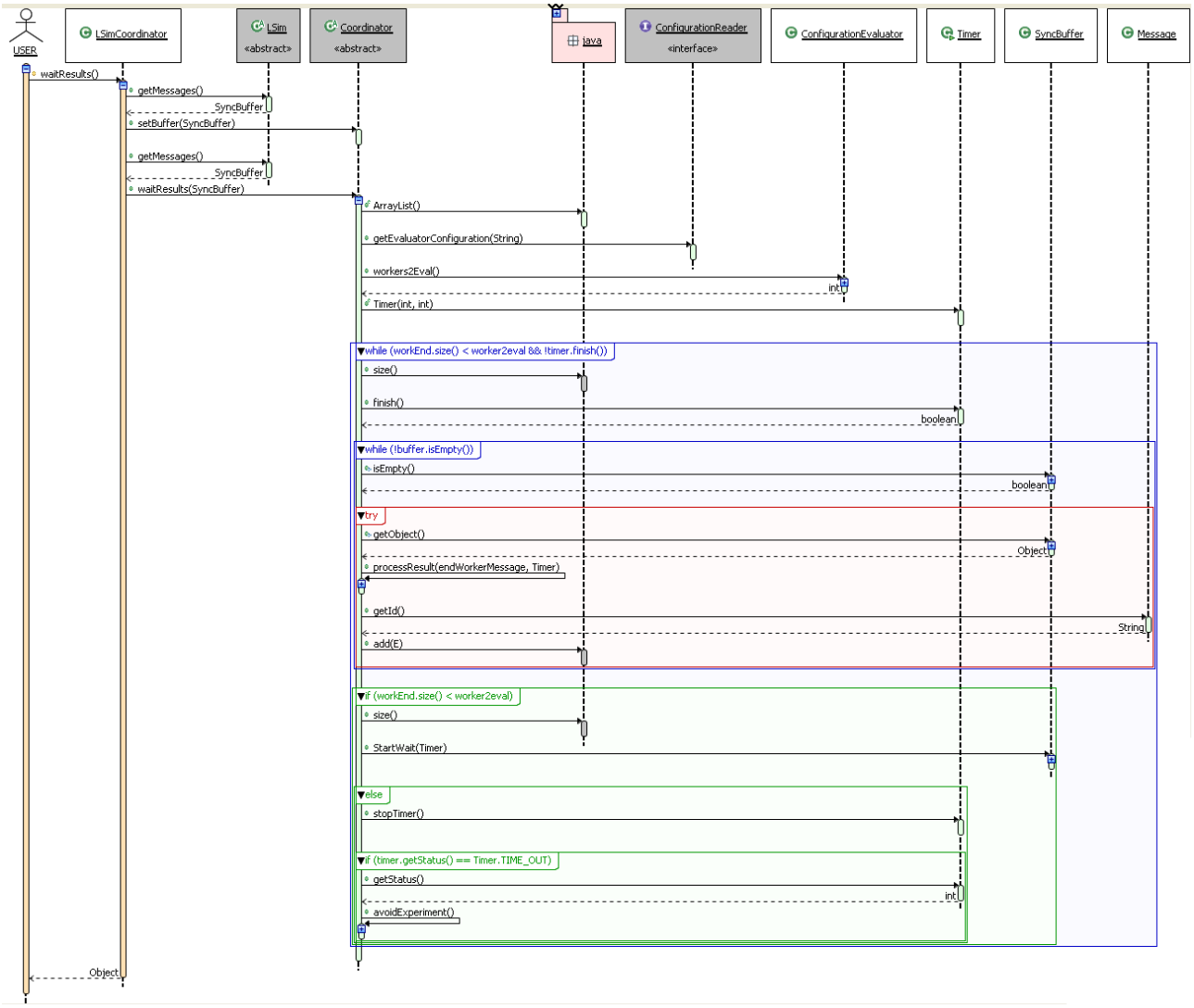


Diagrama 11: Diagrama de seqüència de function endStart(Object) del Coordinator

A la *function endStart(Object p)* el Coordinator envia la senyal de start a tots els Workers. Al diagrama 11 es pot veure en el bucle com s'accedeixen als Workers a través de la classe *WorkerInformation*. Aquest envia en el missatge als Workers el paràmetre de la funció, que és el que retorna el Handler de la fase.

5.1.4.1.3 Fase de sincronització

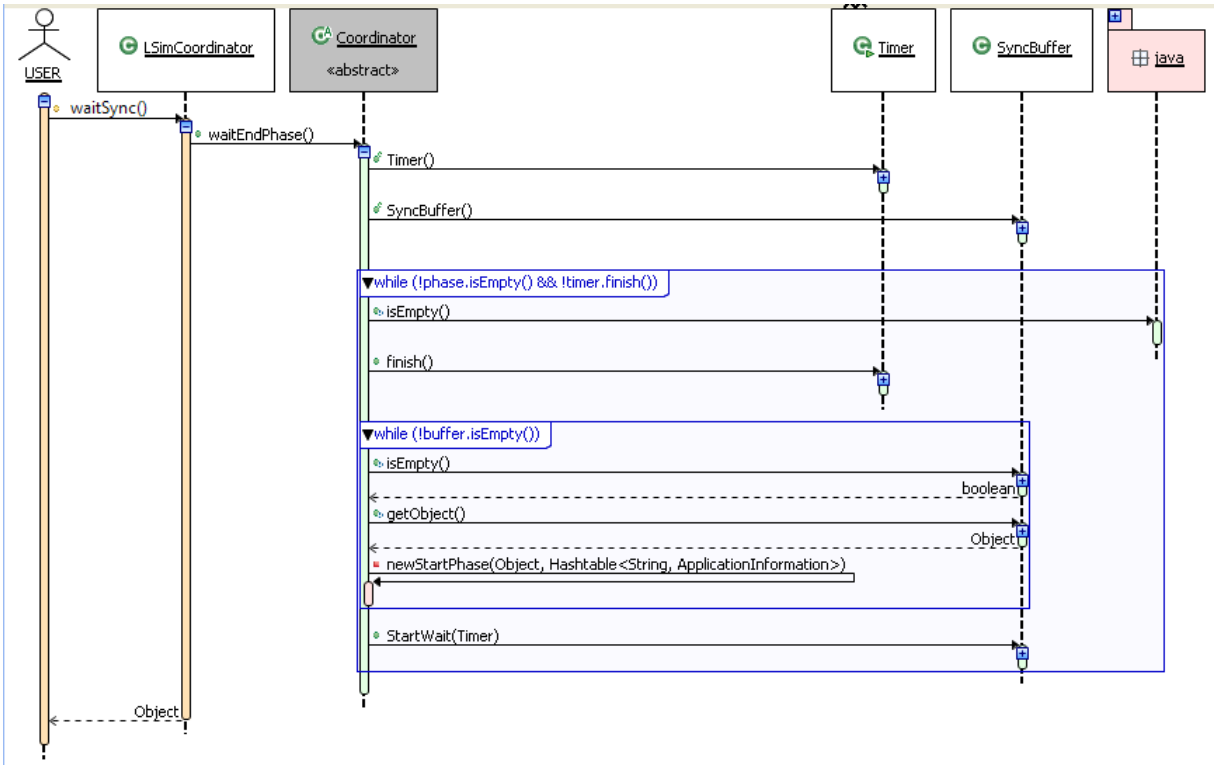


Diagrama 12: Diagrama de seqüència de function waitSync()return Object del Coordinator

El diagrama 12 mostra el comportament de la part de bloqueig del Coordinator en la fase de sincronització. El la variable *phase* conté el conjunt de Workers que participen en aquest punt. El primer bucle mostra la condició de bloqueig del Coordinator. La variable *phase* treu del conjunt el Worker que ha enviat el missatge de participació en la sincronització. Té un temporitzador assignat a aquesta fase, ja que el Coordinator es bloqueja. La funció *funcion newStartPhase(Objet, Object)*, és la que gestiona els missatges dels Workers que participen a la fase. Es surt del bucle en cas de rebre el missatge de tots els Workers que participen al punt o esgotar el temps. Retorna al Handler el conjunt de missatges que envien els Workers.

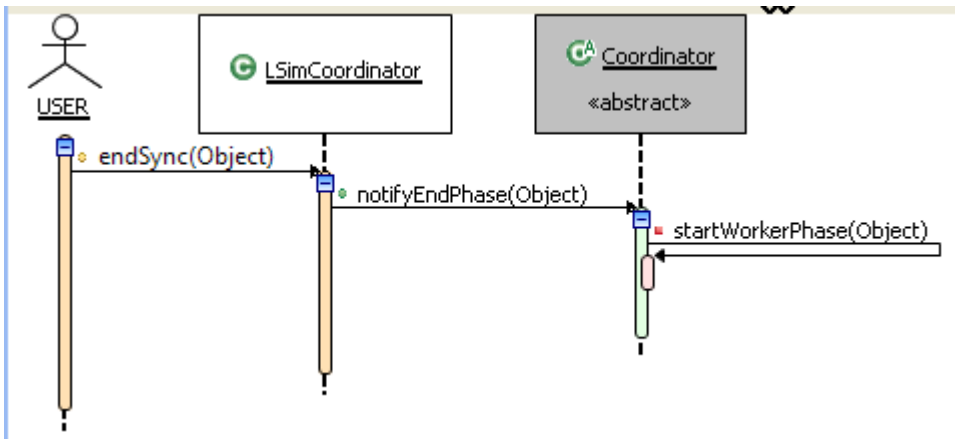


Diagrama 13: Diagrama de seqüència de function endSync(Object) del Coordinator

En el diagrama 13 es veu com el Coordinator comunica a tots els Workers que han participat en el punt de sincronització que poden iniciar l'execució. La *function startWorkerPhase(Object)* té el mateix comportament que el mostrat al diagrama 11. El paràmetre de la *function endSync(Object)*,

és el que el que s'envia cap als Workers en el missatge.

5.1.4.1.4 Fase de resultats

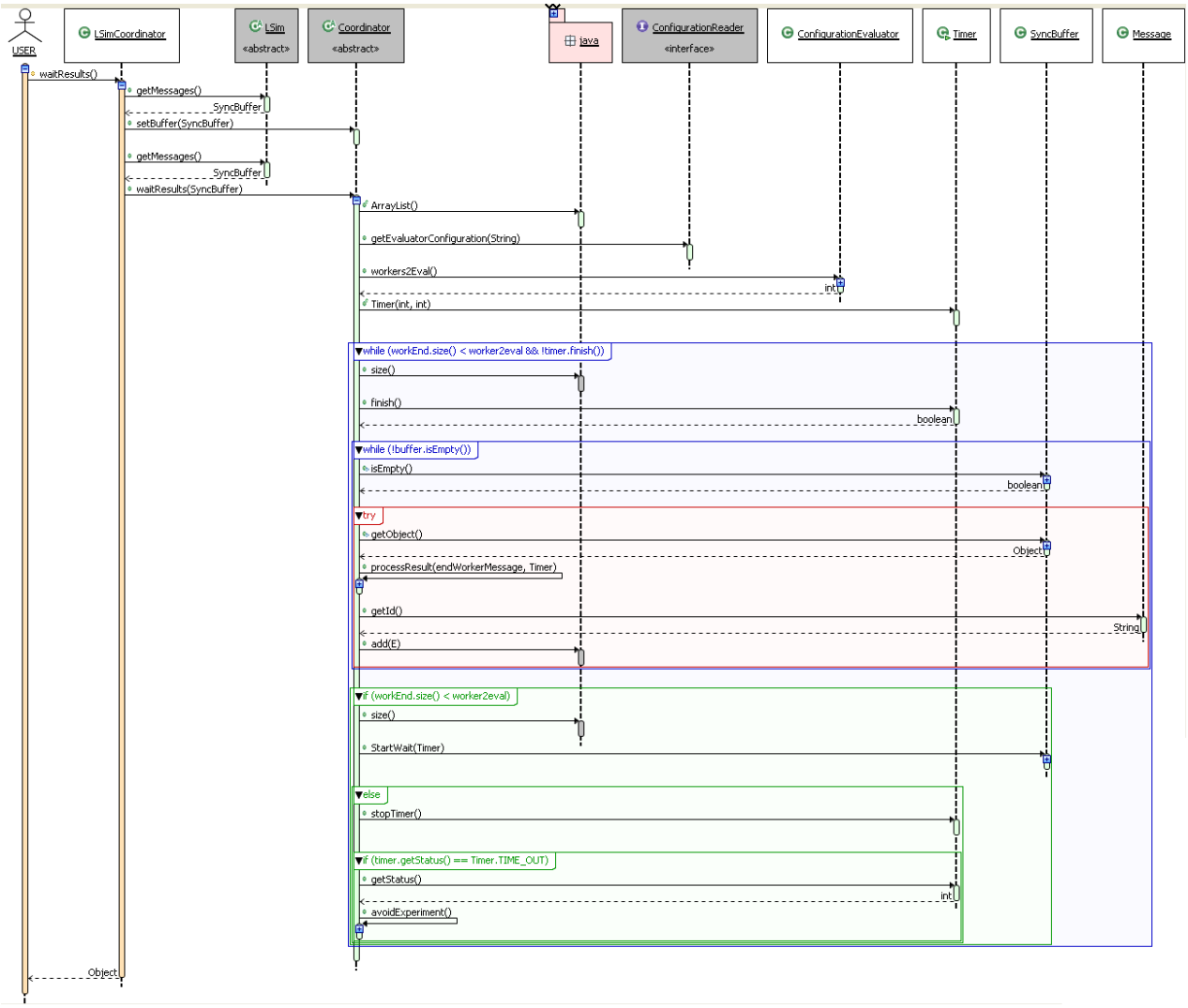


Diagrama 14: Diagrama de seqüència de function endResult(Object) del Coordinator

La *function endResult(Object p)* implementa tot el comportament de la fase d'enviament de resultats del Coordinator, la qual es mostra al diagrama 14. El Coordinator ha d'assignar un Evaluator per avaluar l'experiment. El procés d'avaluació està separat del Coordinator per si el procés d'avaluació és costos en temps i recursos, fent-lo així independent del curs de l'experiment.

El bucle del diagrama controla els Workers que s'han d'avaluar en aquesta fase. La variable *workers2Eval* conté el nombre de Workers a avaluar i la variable *workEnd* el conjunt de Workers que han fet la petició d'avaluació. Disposa d'un temporitzador propi per aquesta fase. Es surt del bucle quan rep el missatge de tots els Workers a avaluar o s'esgota el temps assignat. En cas d'esgotar el temps, s'avorta l'experiment.

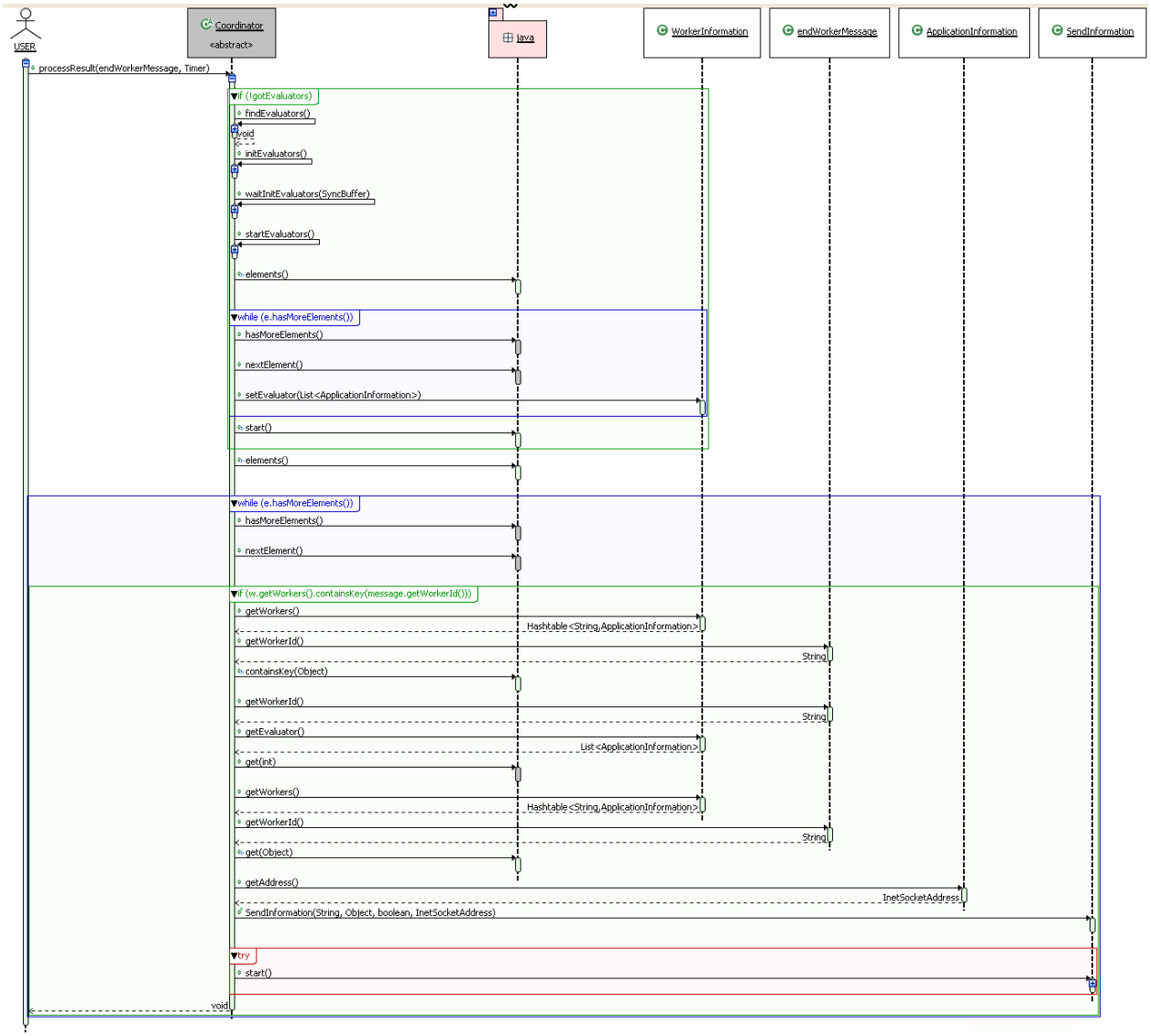


Diagrama 15: Diagrama de seqüencia de function processResult(Object, Timer) del Coordinator

El procés de creació i assignació d'Evaluators el fa la funció del Coordinator *function processResult(Object endWmessage, Timer timer)* que es mostra al diagrama 15. Aquesta s'executarà per cada missatge que envia un Worker preguntar per l'Evaluator, tal i com es mostra al diagrama 14.

Si l'experiment no té un Evaluator assignat, aquest s'encarrega de buscar-ne un per avaluar l'experiment. La *function findEvaluators()*, busca un Evaluator disponible per avaluar l'experiment. La *function initEvaluators()*, configura l'Evaluator per avaluar l'experiment. La *function startEvaluators()* inicia l'execució d'aquest.

El primer bucle següents s'encarreguen d'actualitzar l'Evaluator assignat en cada un dels Workers que s'han d'avaluar. Està preparat per a la possibilitat d'haver-hi més d'un Evaluator per experiment, per tant les assignacions d'Evaluators es fan per tipus de Workers.

El segon bucle s'encarrega d'informar, als Workers que ho han demanat, la ubicació de l'Evaluator.

5.1.4.2 Disseny de les fases en el Worker

El Worker s'ha de bloquejar en la majoria d'aquestes fases esperant a rebre la senyal de continuar del Coordinator. Així es coordina amb els altres Workers per executar els testos. Com s'ha comentat préviament, pot haver-hi diferents tipus de Workers, que es diferencien en el codi que han d'executar o els paràmetres que se'ls hi ha d'enviar entre d'altres coses.

El que identificarà un tipus de Worker és primer el codi que s'ha d'executar, segon el nombre d'instàncies que se'n volen fer, tercer els paràmetres que se li passarà i un identificador pel grup de Workers. A l'apartat 5.2.1.1 es parlarà de l'especificació dels Workers i que els diferencia amb més detall.

5.1.4.2.1 Fase d'inicialització

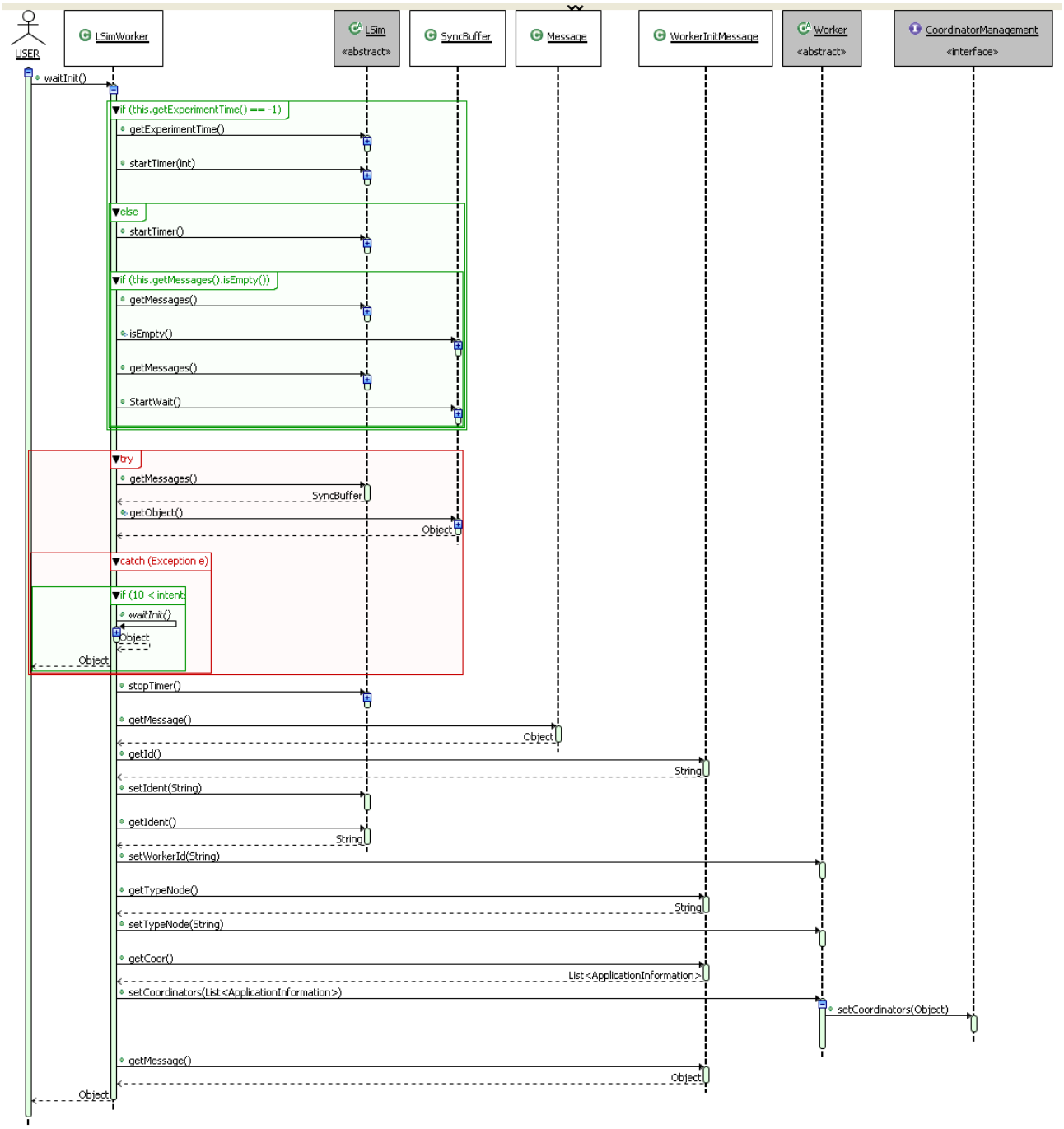


Diagrama 16: Diagrama de seqüència de function waitInit()return Object del Worker

A la *function waitInit() return Object* del diagrama 16, el Worker espera a rebre la configuració per l'experiment actual juntament amb els paràmetres per l'aplicació. Un cop rebut, crea l'instància de *Worker* i *CoordinatorManagment* per al test, i retorna els paràmetres per a l'aplicació. Aquests paràmetres els rebrà com a paràmetre la *function execute(Object param) return Object* del Handler que hagi implementat l'usuari i el que retorni és el que s'enviarà al Coordinator per després comunicar-ho a la fase de start als altres Workers. A dins aquest Handler és on haurà d'anar tot el procés d'inicialització del test del Worker.
En el primer condicional s'inicia el primer temporitzador, tal i com s'ha vist prèviament amb el Coordinator.

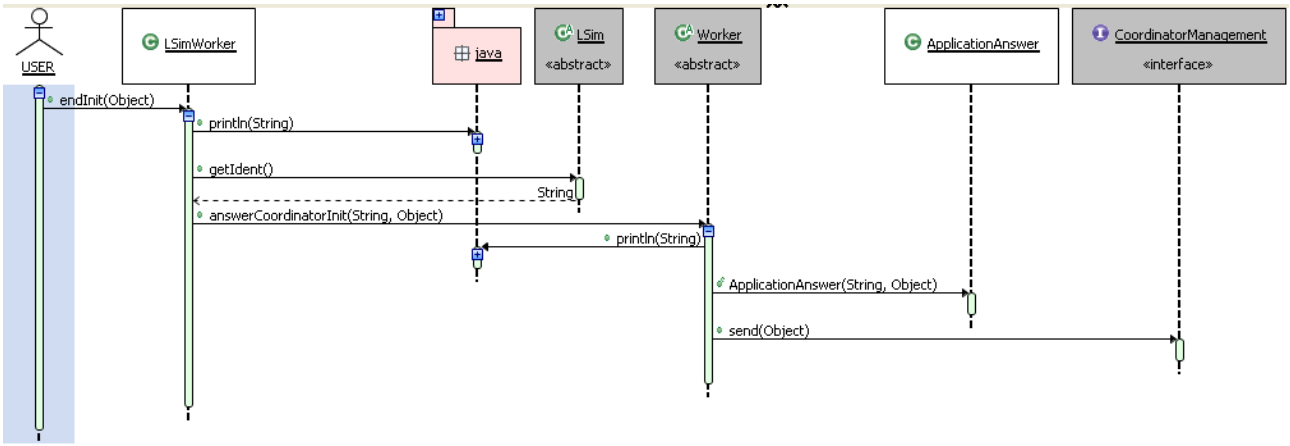


Diagrama 17: Diagrama de seqüència de function endInit(Object) del Worker

La *function endInit(Object)* del diagrama 17 envia el missatge de que s'ha finalitzat la inicialització del Worker. Aquest envia el paràmetre de la funció cap al Coordinator. Aquest és el que retorna el Handler d'aquesta fase.

5.1.4.2.2 Fase de start

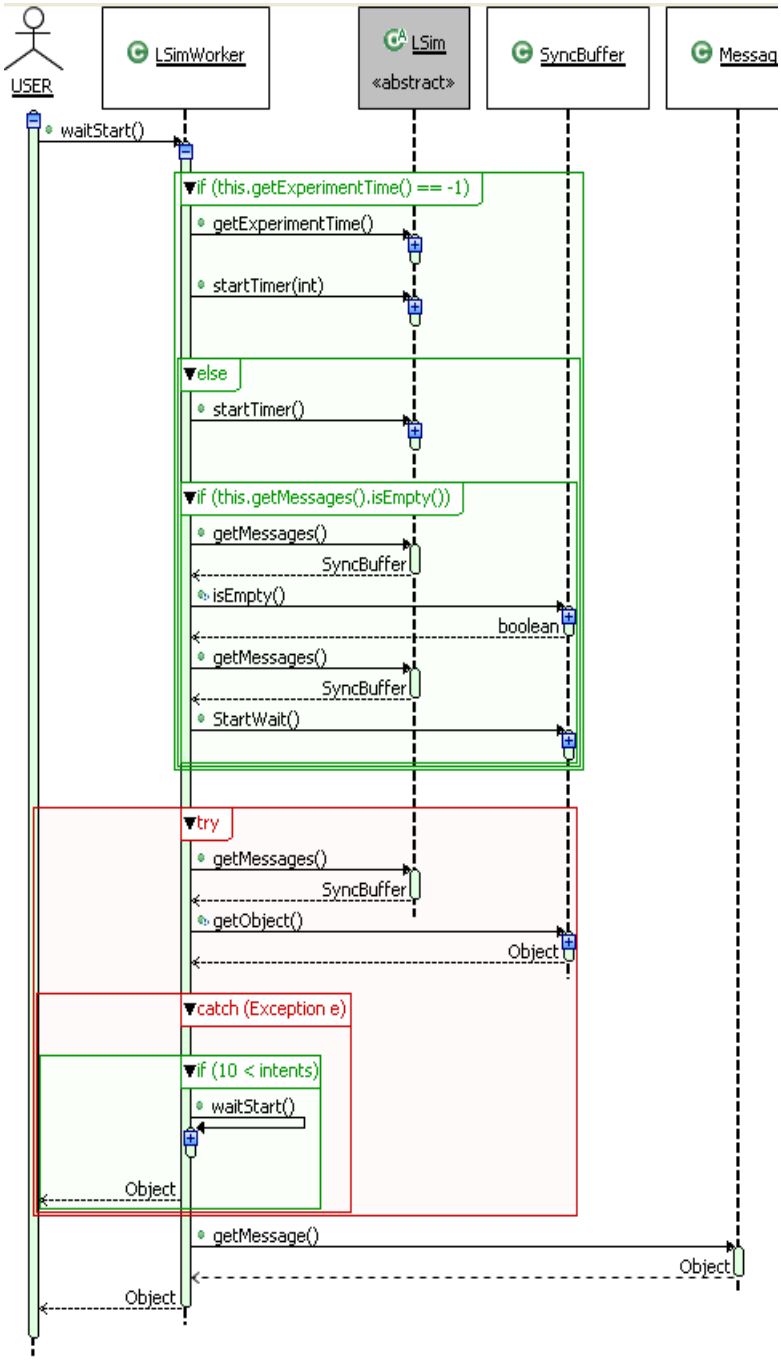


Diagrama 18: Diagrama de seqüència de function waitStart() return Object del Worker

La *function waitStart() return Object,* mostrada al diagrama 18, bloqueja el Worker fins a rebre la senyal que pot iniciar l'execució. En el primer condicional s'inicia el segon temporitzador.

Es comprova que el missatge enviat sigui correcte. En cas de no ser-ho, el Worker es torna a bloquejar fins a rebre el missatge que toca.

El missatge conté el paràmetre de la *function execute(Object) return Object* del Handler de la fase de start.

5.1.4.2.3 Fase de sincronització

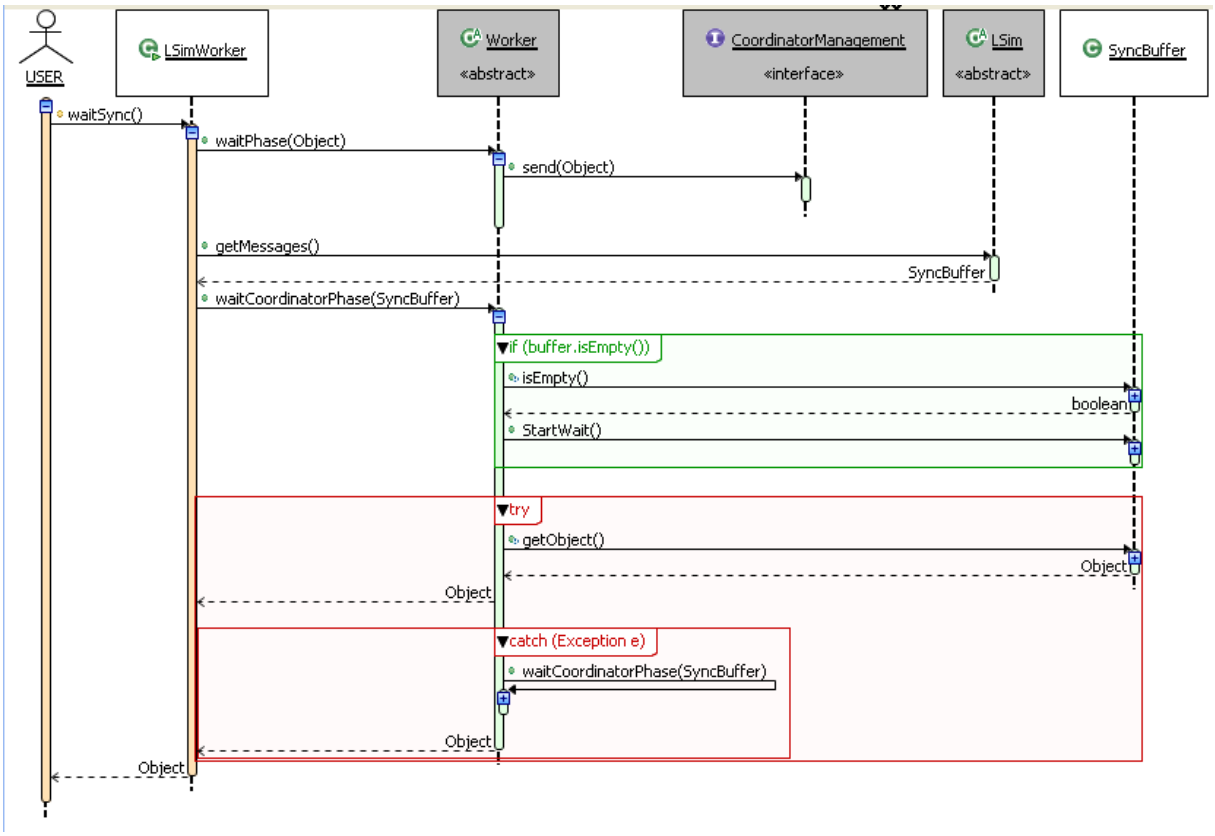


Diagrama 19: Diagrama de seqüència de function waitSync() return Object del Worker

La *function waitSync() return Object*, que es mostra al diagrama 19, el Worker envia al Coordinator el missatge informant que es troba en el punt de sincronització. Amb el missatge es pot enviar informació del Worker com a paràmetre.

A dins el condicional el Worker es bloqueja fins a rebre la resposta del Coordinar. Retorna el missatge de resposta del Coordinator al Handler de l'aplicació.

La *function endSync(Object)* no té cap funcionalitat en el rol de Worker.

5.1.4.2.4 Fase de resultats

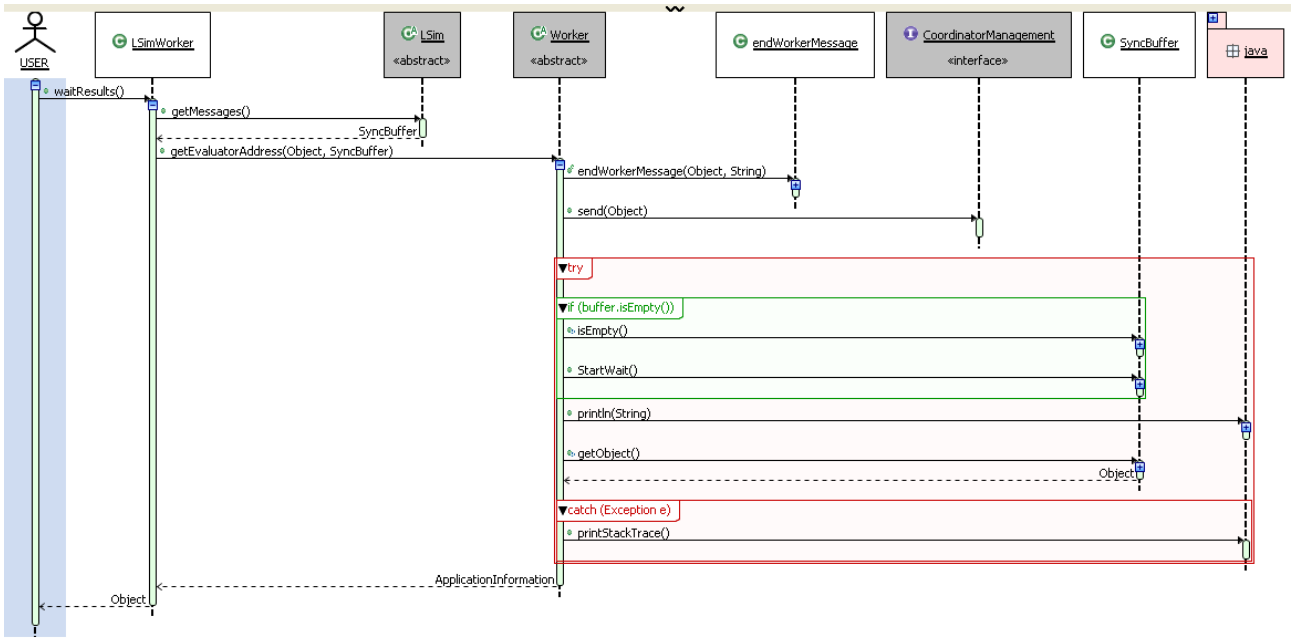


Diagrama 20: Diagrama de seqüència de function waitResult() return Object del Worker

En la *function waitResult() return Object,* que es mostra al diagrama 20, el Worker obté l'adreça del Evaluator a través del Coordinator. Segueix el mateix comportament que el diagrama 19.

Diagrama 21: Diagrama de seqüència de function endResult(Object) del Worker

En la *function endResult(Object res)*, que es mostra en el diagrama 21, envia el resultat a l'Evaluator. El resultat s'obté a partir del Handler de la fase. L'aplicació ha de retornar en el Handler el resultat del Worker.

S'ha separat d'aquesta forma per si en un futur es vol canviar la manera d'avaluar, com per exemple ajuntar el Coordinator amb l'Evaluator en casos en que l'avaluació sigui de poc pes. Així només canviant l'implementació del Worker, s'adaptarà fàcilment al nou mètode.

5.1.4.3 Disseny de les fases en l'Evaluator

L'Evaluator és la part que s'encarrega de recollir i avaluar els resultats dels Workers. És crearà quan ho necessiti el test per evitar que es consumeixin recursos de forma innecessària, i separat del Coordinator per independitzar l'avaluació de la coordinació del test, ja que pot haver-hi avaluacions que siguin costoses en temps o en recursos.

5.1.4.3.1 Fase d'inicialització

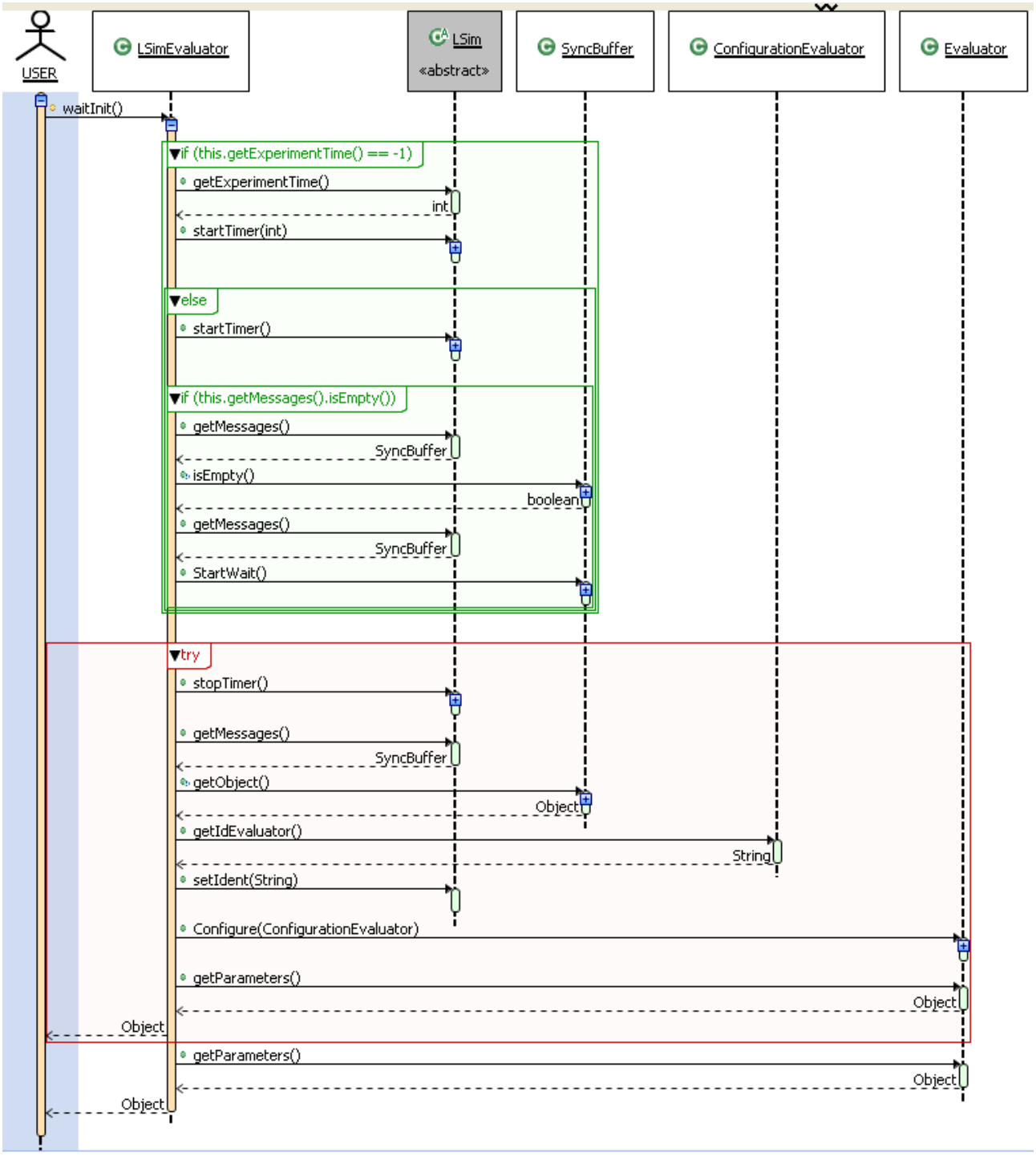


Diagrama 22: Diagrama de seqüència de function waitInit()return Object de l'Evaluator

La *function waitInit() return Object,* que es mostra al diagrama 22, segueix un comportament semblant a les altres fases d'inicialització d'altres rols. Inicialitza el primer temporitzador, es bloqueja fins a rebré la configuració de l'experiment del Coordinator i retorna els paràmetres de l'experiment al Handler de l'aplicació.

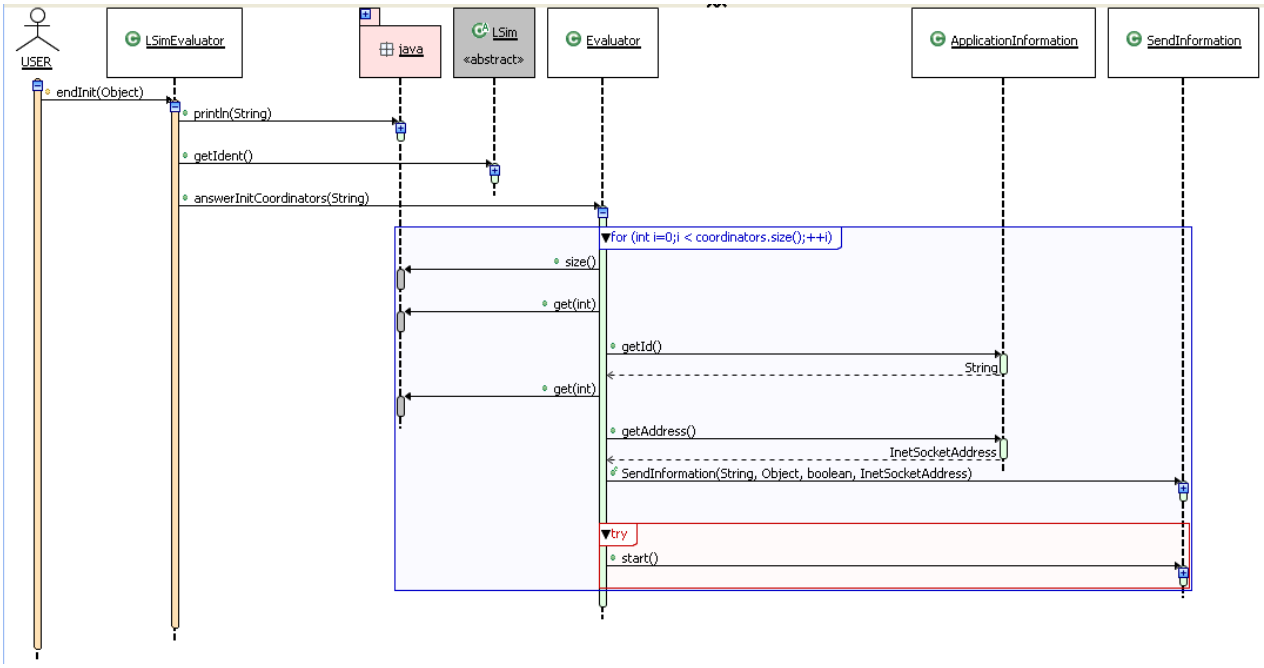


Diagrama 23: Diagrama de seqüència de function endInit(Object) de l'Evaluator

La *function endInit(Object)*, que es mostra al diagrama 23, envia el missatge de que l'Evaluator ha finalitzat la inicialització. En el bucle envia aquest missatge a un conjunt de Coordinators. Actualment aquest conjunt és d'un sol Coordinator, però ja està preparat si hi hagués més d'un Coordinator en l'experiment. Aquest punt és l'únic on l'Evaluator enviarà algun missatge al Coordinator.

5.1.4.3.2 Fase de resultats

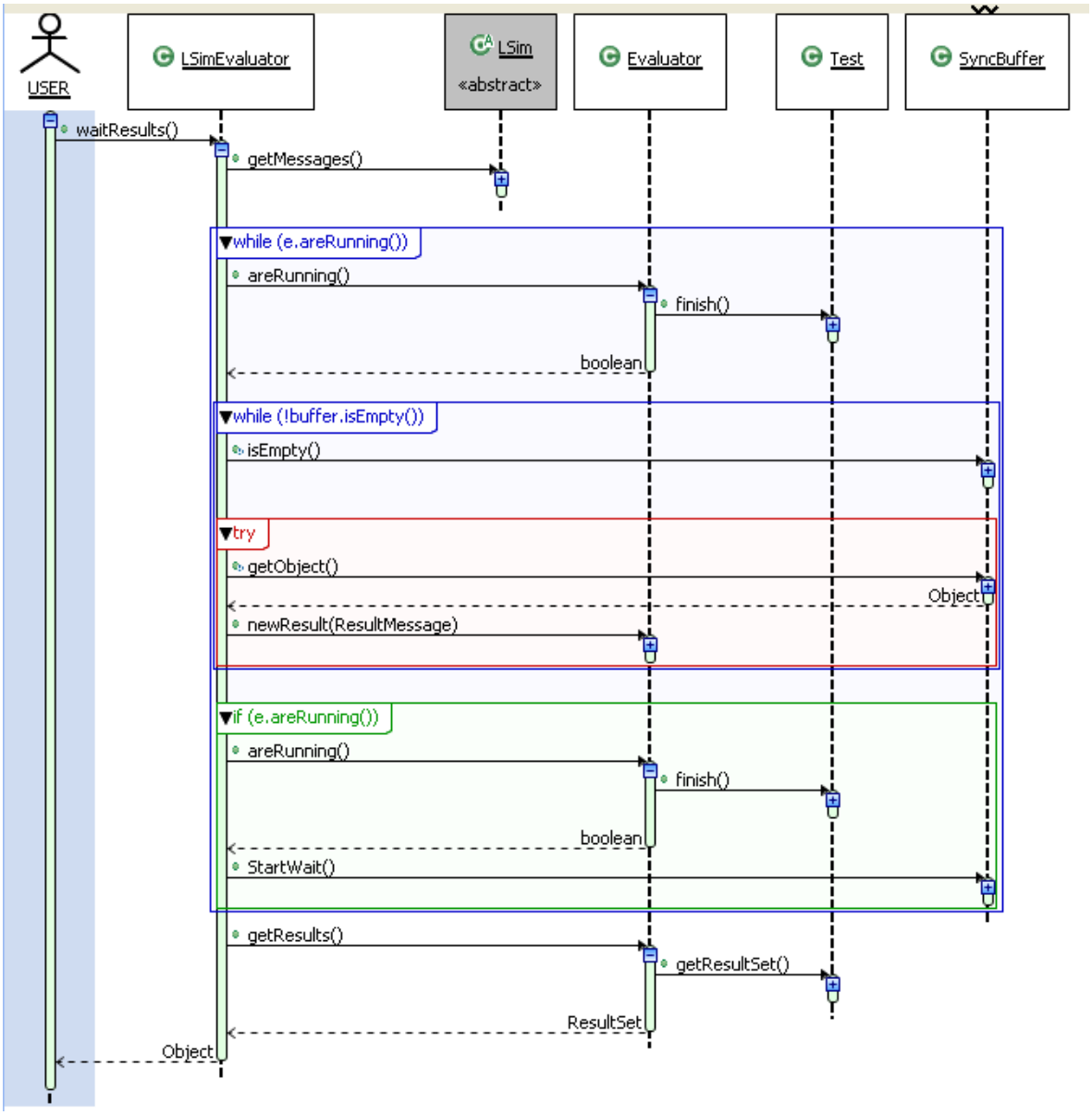


Diagrama 24: Diagrama de seqüència de function waitResult()return Object de l'Evaluator

La *function waitResults() return Object*, que es mostra al diagrama 24, espera a rebre els resultats dels Workers que participin al test. La condició de sortida del primer bucle ve donada per la classe *Test*, que és la que té el control de l'experiment. Se li assignen els resultats que va rebent l'Evaluator a través de *function newResult(ResultMessage)*. Així el control de l'experiment és transparent als mètodes de recollida de resultats, fent-lo modulable per a possibles variacions de recollida de resultats.

Anteriorment s'ha iniciat el segon temporitzador de la mateixa manera que és fa en els diagrames 9 i 17. Quan aquest temps s'esgota, l'Evaluator deixa d'executar-se.

Retorna el conjunt de resultats dels Workers a avaluar. El retorna en una instància de la classe *ResultSet*. Aquest és el paràmetre de la *function execute(Object) return Object* del Handler assignat a la fase d'enviament de resultats. L'aplicació ha de fer l'avaluació dels resultats dels Workers en aquesta funció.

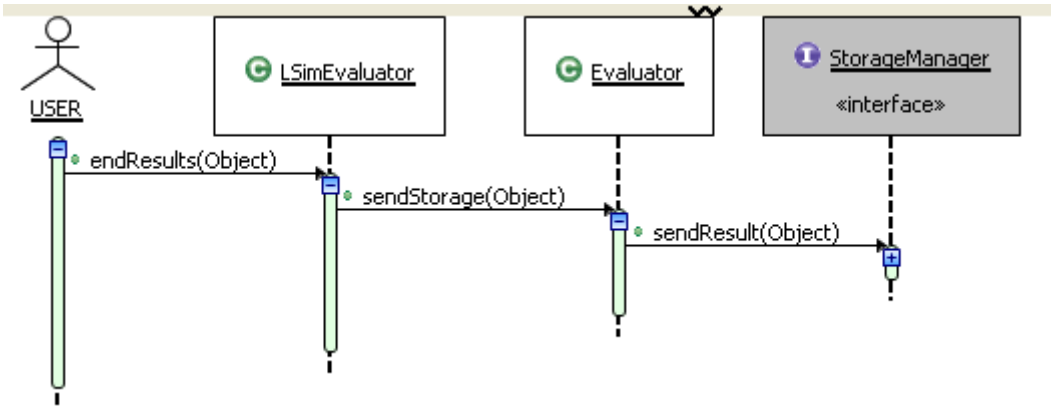


Diagrama 25: Diagrama de seqüència de function endResults(Object) de l'Evaluator

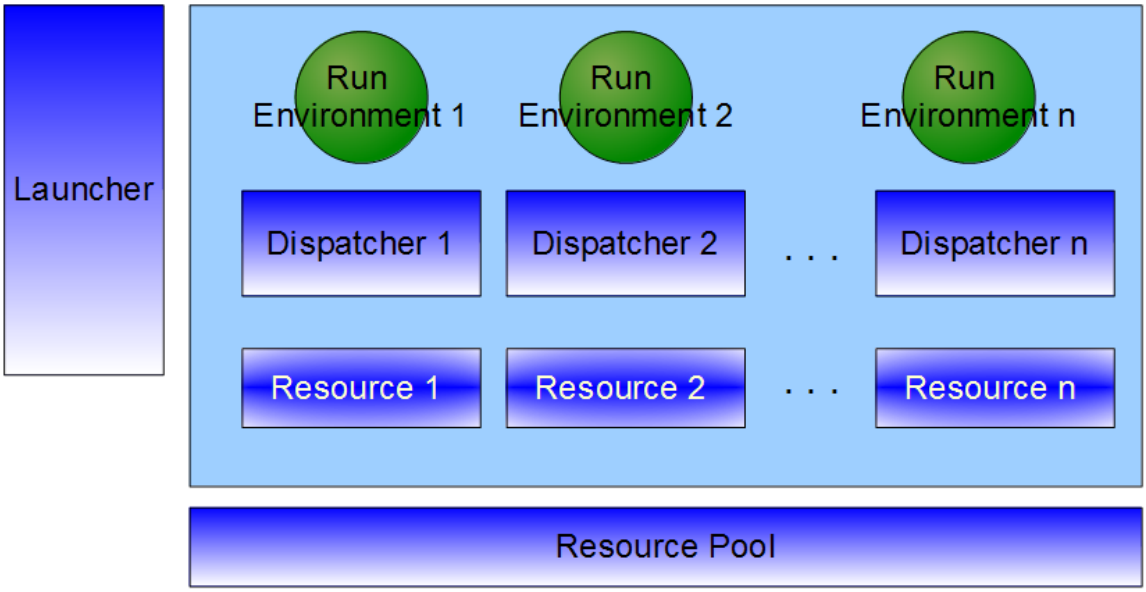
La *function endResults(Object)*, que es mostra al diagrama 25, rep com a paràmetre el resultat de l'avaluació de l'experiment. Aquest paràmetre és el que ha retornat la funció del Handler de l'aplicació assignat a aquesta fase. Aquest és passat com a paràmetre a la *function sendResult(Object)* del *StorageManager* per guardar-ne el resultat.

5.2 LSim Framework

LSim Framework és l'encarregat d'executar i controlar les diferents aplicacions en el conjunt de recursos disponibles.

5.2.1 Arquitectura de LSim Framework

A continuació es presenta l'arquitectura de LSim Framework:



-Resouces: Són les diferents màquines disponibles i connectades en xarxa on es té permís per executar el que es vulgui.

-Resource pool: Permet saber la ubicació dels recursos disponibles, o sigui, la ubicació dels Dispatcher que estiguin executant-se i funcionant en els recursos disponibles.

-Launcher: Permet preparar l'entorn on s'executarà l'experiment. A partir d'una especificació, obté (a partir del resource pool) els recursos necessaris per l'experiment i hi desplega els diferents programes que conformen l'experiment a executar.

-Dispatcher: Permet executar una aplicacion en una entorn d'execució propi i tenir-ne un control (o sigui si s'esta executant o no, aturar-ne l'execució). Un Dispatcher pot executar més d'una aplicació simultàneament. En aquest cas, cada aplicació s'executarà en un entorn pròpi independent de les altres execucions. També obre un port de comunicació on pot rebre missatges per a les diferents aplicacions que té executant-se.

De LSim Framework es desenvoluparà el Launcher i el Dispatcher. Pel resource pool s'ha decidit utilitzar un gestor de recursos distribuïts, en aquest cas CoDeS. S'ha decidit utilitzar CoDeS ja que ofereix una API que permet fer-ne la implementació, cosa que ofereix modularitat al resourse pool. A l'apartat 6.1.4 s'explica més detalladament que és CoDeS i que ofereix i a l'apartat 6.2.3

l'adaptació a LSim Framework.

5.2.2 Launcher

El Launcher és el que ens permet desplegar experiments en l'entorn on estiguin executant el servei del Dispatcher. En el disseny es mostra com a un servei que aten peticions per a desplegar serveis i també com una llibreria per adaptar a aplicacions ja existents a desplegar serveis.

El Launcher permet:

- Gestionar diferents formats d'especificacions.
- Adaptar-se a diferents entorns distribuïts realistes.
- Servei per a desplegar aplicacions en un entorn distribuït realista.
- Funcions per a desplegar aplicacions.

5.2.2.1 Arquitectura del Launcher

A continuació es mostra el diagrama de classes del Launcher amb una breu descripció de cada classe i a quina part dóna solució:

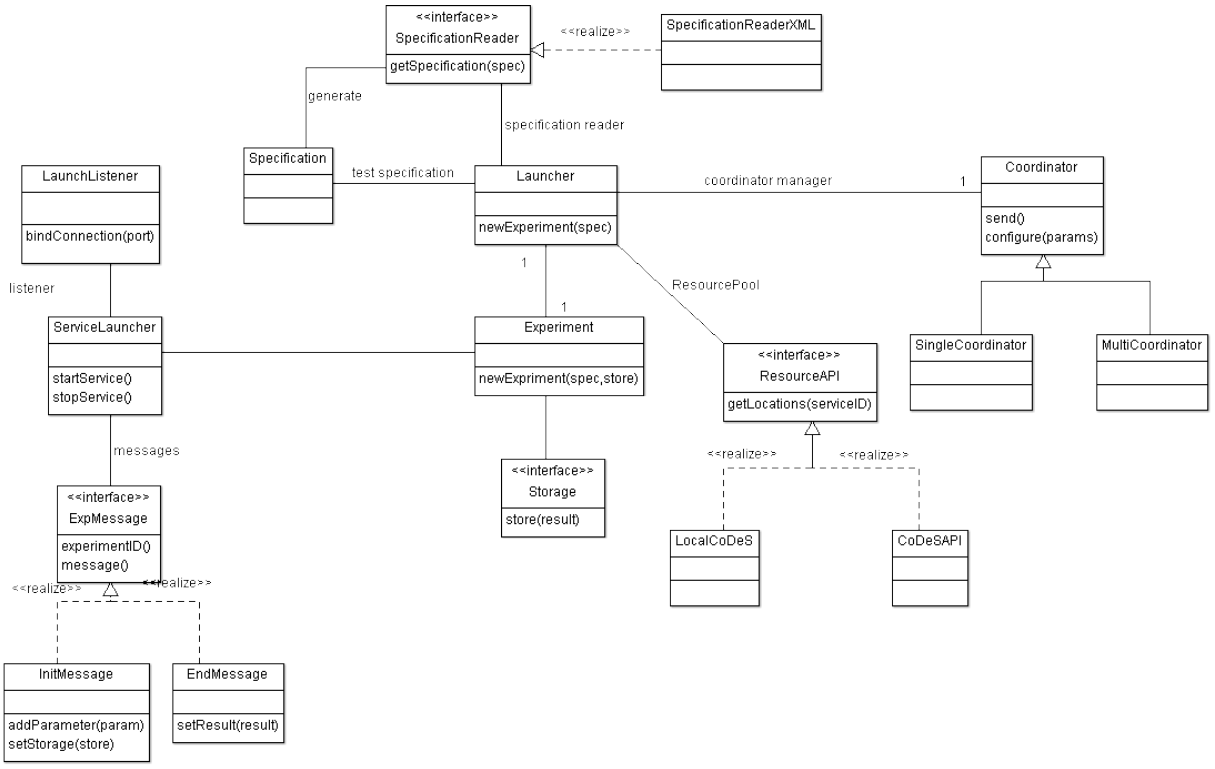


Diagrama 26: Diagrama de classes del Launcher

- LauncherService: crea i gestiona el servei del Launcher. Permet rebre peticions d'experiments a l'adreça on s'estigui executant.

- LaunchListener: fa l'abstracció dels components de xarxa, tal com obrir un socket on rebre peticions o utilitzar qualsevol altre tecnologia que permeti la comunicació per la xarxa.
- Experiment: gestiona l'execució d'un experiment.
- Storage: interfície que ha d'implementar l'usuari. És on s'enviaran els resultats de l'experiment. S'assigna al missatge d'inici.
- ExpMessage: interfície que proporciona les funcions bàsiques pels missatges que entendrà el servei del Launcher.
- InitMessage: missatge d'inicialització d'experiment. Conté l'especificació de l'experiment, paràmetres afegits de forma dinàmica (és a dir, que no estan a l'especificació) i el Storage que s'executarà al finalitzar l'experiment (on es pot indicar on es vol enviar el resultat).
- EndMessage: missatge de finalització d'experiment. Conté el resultat de l'experiment.
- Launcher: conté tot el necessari per a la gestió de l'experiment, tal com la lectura de l'especificació, l'accés al gestor de recursos i la distribució dels components de l'experiment sobre els recursos.
- ResourceAPI: interfície que conté els mètodes d'obtenció d'adreces dels recursos. D'aquesta forma ens permet fàcilment adaptar diferents formes d'obtenció de recursos amb les implementacions de la inteficie, tal i com surt representat en el diagrama.
- Specification: conté l'especificació de l'experiment.
- SpecificationReader: interfície que conté els mètodes per obtenir l'especificació de l'experiment. Aquesta especificació s'obté a partir del paràmetre de la funció, que pot ser diferents coses depenent de l'implementació de l'interfície.
- SpecificationReaderXML: implementació de SpecificationReader que obté l'especificació de l'experiment a través d'un fitxer o string en format XML.
- Coordinator: fa la distribució de l'experiment sobre l'entorn on es trobin els recursos. Està pensada per fer arribar la senyal d'inicialització als coordinadors, per això es tracta d'una classe abstracte on s'ha de crear la implementació de les funcions d'enviament depenent del nombre de coordinadors de l'experiment.

5.2.2.2 Disseny d'inici d'experiment

El Launcher està pensat per a llançar experiment, per a continuació es mostra el diagrama de seqüència de la *function newExperiment(String specification)*, que es mostra al diagrama 27, on es veurà el disseny del llançament d'experiments sobre el framework. Aquí és mostra tot el que ha de realitzar el Launcher abans de llançar qualsevol experiment.

Diagrama 27: Diagrama de seqüència de function newExperiment(String)

En el diagrama 27 es mostra com s'obté l'especificació de l'experiment a través del SpecificationReader, a partir de la *function newSpecification(String) return Specification*.

La *function getServiceInstances(Integer, String) return List<Address>* del diagrama 27, obté les adreces dels serveis en l'entorn. Amb aquesta funció es localitza les ubicacions del Coordinator i els Workers.

La *function initServiceApplication(String, String, List<Address>)*, del diagrama 27, desplega l'aplicació a l'entorn. Com a paràmetres té la referencia de l'aplicació a executar, l'identificador d'aquesta i el conjunt d'adreces on s'ha de desplegar l'aplicació. Permet desplegar el Coordinator i els Workers a l'entorn distribuït.

En el bucle és on es despleguen tots els diferents tipus de Workers. La variable *workers* conté el conjunt dels diferents tipus de Workers.

5.2.2.3 Gestió d'experiments al servei de Launcher.

El Launcher admet l'execució d'experiments de forma remota. Enviant un missatge d'inicialització d'experiment aquest el desplega a l'entorn d'execució al que està connectat el Launcher.

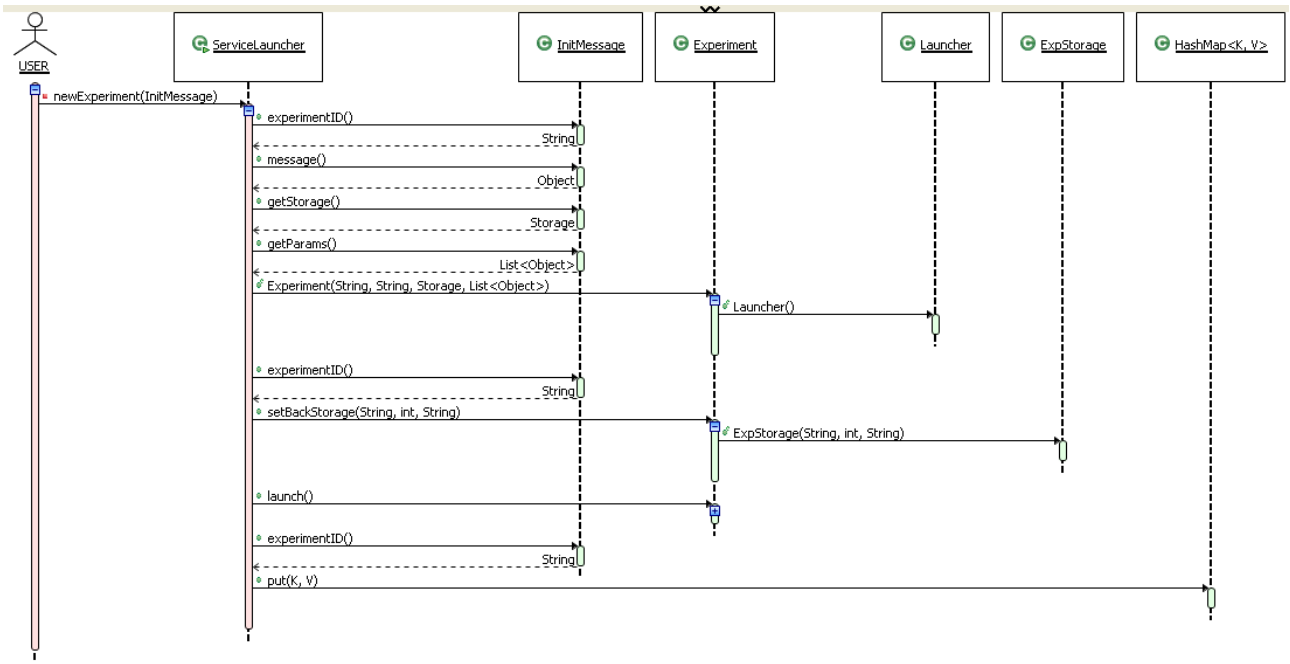


Diagrama 28: Diagrama de seqüència de function newExperiment(InitMessage)

Al diagrama 28 es mostra la *function newExperiment(InitMessage)*, la qual gestiona el missatge d'inicialització d'un experiment al servei del Launcher. La classe *InitMessage* conté les dades necessàries per a desplegar l'experiment. Per cada experiment es crea una instància de la classe *Experiment*, la qual tindrà el control de l'experiment desplegat.

La *function launch()* crida la *function newExperiment(String specification)* del Launcher i desplega l'experiment a l'entorn.

Un cop s'ha desplegat i iniciat l'experiment, el Launcher restarà a l'espera dels resultats de l'experiment. La classe *EndResult* conté el resultat de l'experiment. El resultat el gestionarà el *Storage* que s'ha assignat al iniciar l'experiment.

5.2.2.4 Especificació de l'experiment

A l'especificació s'ha de poder indicar els elements variables per a cada un dels experiments a desplegar sobre el framework. Per cada un dels rols de LSim s'ha de poder especificar i variar segons el test:

- Identificador: Per definir diferents tipus de Workers, i per tant, poder especificar diferents paràmetres per cadascun d'aquest tipus.
- Referencia del codi a executar: D'on s'obté el codi de l'aplicació i com s'ha d'executar.
- Nombre d'instàncies: Nombre de vegades que hi ha d'haver de cada tipus de Workers en l'experiment.
- Paràmetres del test: Cadascun dels rols ha de poder rebre una llista de paràmetres que s'especifiquen en aquest punt.

A més pels Evaluators s'ha de poder especificar dues coses més:

- Workers a avaluar: Els tipus de Workers que s'han d'avaluar al final, i per tant, dels que vol

recollir-ne els resultats.

- Nombre de Workers: Per cada tipus de Worker a avaluar, de quants Workers n'ha de recollir resultats.

D'aquesta forma es poden realitzar diferents testos i definir-ne el seu comportament un cop ja estiguin implementats, ja que pots variar tots els aspectes del test, des de a que executarà fins a quins són els paràmetres per aquella aplicació.

Aquesta parametrització permet fer execucions sota diferents condicions o passar diferents tipus de testos de l'aplicació.

5.2.3 Dispatcher

El Dispatcher és un servei que permet executar cada aplicació en un entorn d'execució independent. A més dóna l'opció d'enviar missatges a les aplicacions que s'estan executant a partir de l'adreça on està ubicat el Dispatcher. Per tant et permet tenir varies aplicacions executant-se al mateix temps i comunicades entre altres nodes de l'entorn distribuït a traves del Dispatcher. Té un control sobre els entorns que s'estan executant, poden fer-los desaparèixer quan calgui.

Per a LSim ens servirà per a executar les aplicacions en el LSim Framework i comunicar la capa de la LSim Library entre si, per això en el diagrama de classes de la llibreria apareix la interfície *LSimDispatcherHandler* que és la que comunica amb l'exterior.

Per facilitar l'adaptació a LSim Library i el debugat de les aplicacions, el Dispatcher també pot executar les aplicacions en el seu mateix entorn d'execució. En funció de la referencia al codi de l'especificació, es crearà un entorn d'execució nou per l'aplicació o s'executarà en el mateix entorn que el Dispatcher. També ofereix un sistema de manipulació de fitxers, tal que pot obtenir els fitxers de l'aplicació i executar-los.

A continuació es detalla el diagrama de seqüència del Dispatcher:

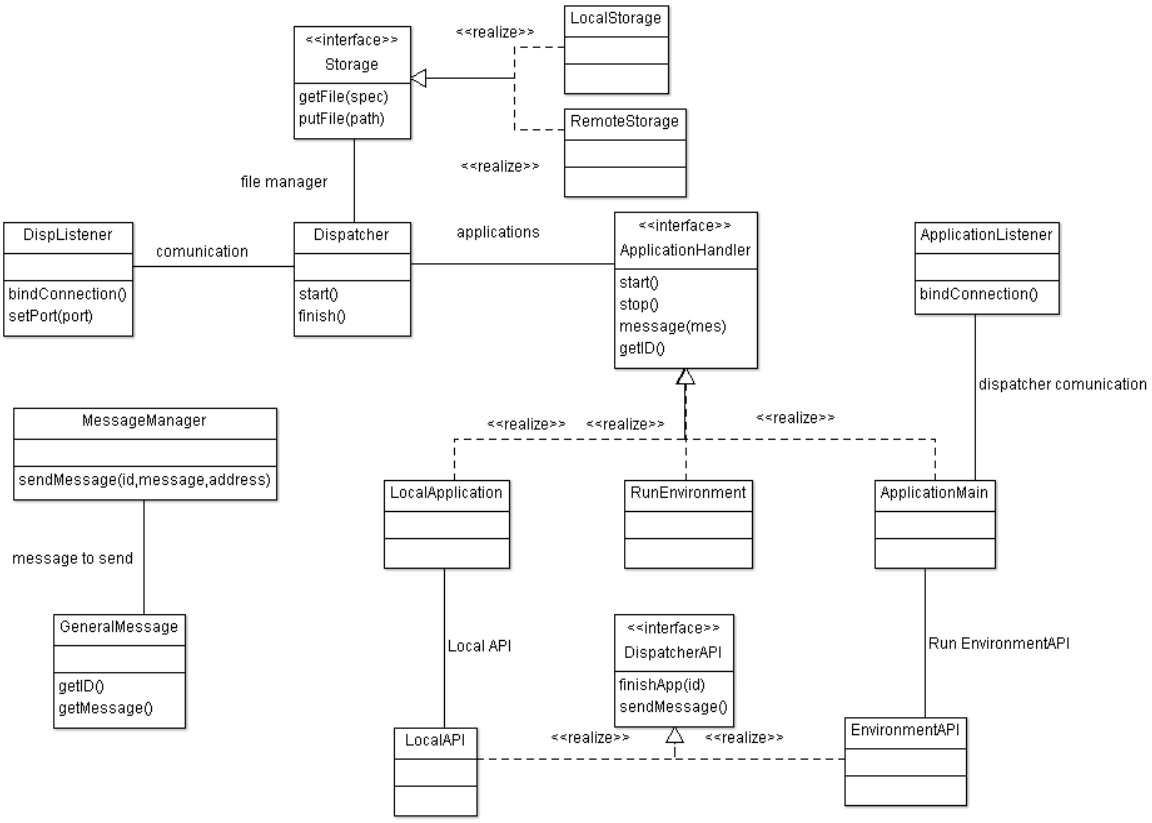


Diagrama 29: Diagrama de classes del Dispatcher

- ApplicationHandler: interfície que ha d'implementar l'aplicació que es vol executar al Dispatcher si vol comunicar-se amb aquest. Segueix el patró Strategy per poder canviar la forma d'executar l'aplicació pel Dispatcher.
- LocalApplication: instància d'una aplicació que s'executa en el mateix entorn que el Dispatcher.
- RunEnvironment: instància que es crea cada cop que s'instancia una nova aplicació en un entorn d'execució. Per a cada aplicació d'aquest tipus, el Dispatcher tindrà una instància nova d'aquesta classe.
- ApplicationMain: instància que permetrà executar l'aplicació en un entorn d'execució independent. Manté la comunicació a traves de sockets amb el *RunEnvironment* que controla aquesta instància de l'aplicació.
- DispatcherAPI: té els mètodes de comunicació de l'aplicació cap al Dispatcher.
- LocalAPI: instància que permet comunicar les aplicacions que s'estan executant al mateix entorn que el Dispatcher.
- EnvironmentAPI: instància que permet comunicar una aplicació que s'esta executant en un entorn d'execució diferent del Dispatcher amb aquest.
- ApplicationListener: permet la comunicació del Dispatcher amb l'aplicació.
- Dispatcher: posa el servei en marxa, controla les execucions de les aplicacions i les comunica amb l'exterior.
- Storage: interfície que té els mètodes d'obtenció de fitxers per a executar les aplicacions.
- LocalStorage: instància que permet fer la manipulació de fitxers de forma local.

- RemoteStorage: instància que permet obtenir un fitxer remot.
- DispListener: permet la comunicació d'elements externs amb el Dispatcher. Es comporta com un daemon a l'espera de missatges pel Dispatcher.
- MessageManager: fa transparent l'enviament de missatges de l'aplicació cap al Dispatcher.
- GeneralMessage: classe que conté les dades bàsiques dels missatges.

6 Implementació

En aquest apartat es parlarà de com s'ha implementat el LSim Framework i la llibreria LSim. En tots dos es veurà quina tecnologia s'ha utilitzat per crear-la i com s'ha decidit finalment implementar-los.

6.1 Tecnologies

A continuació es farà un repas de les tecnologies que utilitzarà el projecte i les decisions que s'han pres per a la utilització d'aquestes i no d'altres.

6.1.1 Java com a llenguatge de programació

Java és un llenguatge orientat a objectes que ens permet adaptar molt bé el disseny que s'ha proposat, i donant-nos modularitat per poder crear noves versions sobre el primer prototipus i adaptar-lo a altres entorns. Al ser un llenguatge interpretat, un programa escrit en Java es podrà executar a qualsevol màquina que disposi d'una màquina virtual Java, donant portabilitat desitjable a diferents màquines. A més és fàcilment adaptable a un entorn real en xarxa.

Hi han altres llenguatges de programació que també podrien ser útils per a programar LSim, però s'ha elegit Java per adapta les pràctiques de l'assignatura d'arquitectura de sistemes distribuïts de la UOC a LSim.

6.1.2 NIOSockets de Mina

Per a la comunicació a través de la xarxa interessa tenir una eina que ens permeti dues coses: La primera és que sigui escalable a molts nodes i l'altre que pugui atendre múltiples peticions d'accés al servei, ja que a LSim cal que es coordinin múltiples nodes.

En aquest primer prototipus s'ha decidit utilitzar NIOSockets de Mina. Són Sockets que no es bloquegen quan estan llegint o escrivint dades, per tant pot estar llegint o escrivint en un socket en un mateix instant de temps. A més permet mantenir sessions entre els nodes, fent més eficient l'enviament de dades un cop s'ha establert la primera connexió.

Per utilitzar els NIOSockets es fa servir una llibreria que s'havia implementat anteriorment per a altres projectes a la UOC, anomenada ConnectorLayer. Per tant, el cost d'adaptació de Mina està ja feta, cosa que facilita la seva utilització. En les proves d'aquesta llibreria es va comprovar que Mina ens deixa mantenir un total de 243 sessions obertes, cosa que serà més que suficient per aquesta primera versió de LSim.

Més endavant està pensat en utilitzar altres mètodes per a fer més escalable la comunicació i, per tant, tenir més nodes en els experiment. Aquest mètodes són sistemes de multicast o busos de dades.

6.1.3 JDom i XML

Per a l'especificació de l'experiment és desitjable utilitzar un format que permeti transmetes a traves de la xarxa fàcilment. XML és el format ideal per aquesta tasca, ja que és un metallenguatge que ens permet definir la gramàtica d'un altre llenguatge i transmetre instàncies d'aquest llenguatge a través de la xarxa. Per tant és una bona forma de definir les especificacions.

JDom és la llibreria Java que permetrà treballar amb XML a qualsevol aplicació en Java. Permet d'una forma senzilla i esquemàtica treballar amb aquest metallenguatge.

6.1.4 CoDeS

CoDeS és un middleware per a la construcció de comunitats contributives, és a dir,una comunitat formada per un conjunt d'equips (recursos) i els seus propietaris. Permet als usuaris a contribuir amb una part dels seus recursos de computació a la comunitat. Aquests recursos s'utilitzen per allotjar els serveis en benefici de la comunitat. Cada propietari aporta una o més màquines a la comunitat, i pot desconnectar-se de la comunitat en qualsevol moment. Membres de la comunitat també es poden desplegar serveis i especificar els requisits d'execució (una certa quantitat de CPU, RAM, disc, etc, necessaris per a una sola rèplica del servei), els arxius necessaris per a la seva execució (tant arxius executables i fitxers de dades) i el nombre de rèpliques que han d'estar disponibles[1].

Permet desplegar cada component com a un servei en els ordinadors que formen la comunitat CoDeS. Per cada servei es pot especificar el nombre de repliques que se'n vol tenir. CoDeS fa el desplegament automàtic del servei.

CoDeS garanteix un nombre de rèpliques demanat per cada servei desplegat. Per tant, desplegant el nombre de Dispatcher que siguin necessaris sobre CoDeS, aquest en fa la gestió necessària per poder accedir sempre que es necessiti al Dispatcher.

L'API de CoDeS pot instanciar-se amb diferents implementacions. Per tant permet crear i adaptar altres formes d'accedir als recursos. S'accedeix a aquesta API a través de RMI, cosa que independitza la localització i desplegament de recursos i canviar-lo sense la necessitat d'aturar les execucions que estiguin en marxa.

6.2 Implementació de LSim Framework.

A continuació es parlarà de la implementació de LSim Framework. Tal i com s'ha vist, LSim Framework està composat per tres parts, les quals s'ha implementat només el Launcher, el Dispatcher i una versió per a proves de l'API de CoDeS.

6.2.1 Implementació del Launcher

La implementació d'aquest ha estat fidel al seu disseny original.

Launcher ha d'entendre el format de LSim, d'aquesta manera té LSim Library configurada en el classpath.

El Launcher està dividit en tres parts. La primera t'ofereix les funcions per gestionar experiments a la classe *Launcher*. La segona que et permet oferir un servei que llança experiments a la classe *ServiceLauncher*. La tercera que et permet especificar els experiments a llançar. A continuació es

detallarà la implementació d'aquestes tres parts.

6.2.1.1 Especificació

L'especificació de l'experiment reflecteix tot el que varia d'un a experiment a un altre. El format que s'utilitzarà per fer les especificacions és XML, ja que permet definir un llenguatge o format propi per a l'especificació, validar-la de forma senzilla i està preparat per funcionar en entorns distribuïts.

A continuació es mostra l'esquema d'una especificació:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Specification>

        <experimentId>Experiment ID</experimentId>

        <Coordinators>
                <num>1</num>
                <codeRef>Code reference</codeRef>
                <timeInit>1</timeInit>
                <timeEval>1</timeEval>
                <param>parameter 1</param>
                <param>parameter 2</param>
                <param>parameter 3</param>
        </Coordinators>
        <Evaluators>
                <Evaluator>
                        <codeRef>Code reference</codeRef>
                        <idTest>1</idTest>
                        <Node>
                                <idNode>Node ID1</idNode>
                                <nMess>1</nMess>
                        </Node>
                        <Node>
                                <idNode>Node ID2</idNode>
                                <nMess>1</nMess>
                        </Node>
                        <param>parameter 1</param>
                        <param>parameter 2</param>
                        <param>parameter 3</param>
                </Evaluator>
        </Evaluators>
        <Workers>
                <Worker>
                        <num>1</num>
                        <idWorker>Node ID1</idWorker>
                        <codeRef>Code reference</codeRef>
                        <param>parameter 1</param>
                        <param>parameter 2</param>
                        <param>parameter 3</param>
                </Worker>
                <Worker>
                        <num>1</num>
                        <idWorker>Node ID2</idWorker>
                        <codeRef>Code reference</codeRef>
                        <param>parameter 1</param>
                        <param>parameter 2</param>
                        <param>parameter 3</param>
                </Worker>
        </Workers>

</Specification>
```

Text 1: exemple d'especificació

A continuació es detallaran els camps més importants de l'especificació:

1. Camps genèrics:
    - experimentID: Identificador de l'experiment.

- num: Nombre de vegades que hi ha d'haver desplegat a l'entorn de l'aplicació.
- codeRef: Referencia del codi de l'aplicació. Aquesta pot variar segons l'entorn on s'executi.
- param: Paràmetre per l'aplicació. Es poden especificar tants paràmetres com es vulgui.

2. Específics del Coordinator:
   - timeInit: Temps d'espera per rebre els missatges de resposta dels Workers a la fase d'inicialització.
   - timeEval: Temps d'espera per rebre missatges dels Workers preguntant l'adreça del Evaluator a la fase d'enviament de resultats.
3. Específics de l'Evaluator:
   - Node: Conté la informació d'un tipus de Worker a avaluar.
   - idNode: Identificador del tipus de Worker a avaluar.
   - nMess: Nombre de Workers que s'ha de recollir del tipus especificat.
4. Específics del Worker:
   - idWorker: Identificador del tipus de Worker.

6.2.1.2 Llançament d'experiments

La classe *Launcher* ofereix la funció *public void newExperiment(String spec)*, mostrada en el diagrama 27, que llança els experiment. Segueix la següent seqüència de passos:

1. A partir del SpecificationReader s'obté l'especificació de l'experiment en el format que entén del Launcher.
2. La funció *private List<InetSocketAddress> getServiceInstances(int num, String Service);* on obté la ubicació dels serveis de Dispatcher en l'entorn s'executen les proves. En cas de no haver-hi suficient, se n'encarregarà de crear-ne de nous a l'entorn. En aquest punt obté els recursos necessaris pel Coordinator i Workers.
3. La funció *private List<ApplicationInformation> initServiceApplication(String codeRef, String service, List<InetSocketAddress> address);* inicia l'execució del Coordinator i Workers en els Dispatchers. En aquest punt es genera els identificadors interns per a LSim dels elements que s'explicaran més endavant en aquest capítol.
4. Genera la configuració del Coordinator i l'especificació que obtindrà de l'experiment.
5. Envia la senyal d'inicialització al Coordinator amb la configuració i l'especificació.

Els identificadors dels elements de LSim han de ser únics per cada un dels experiments, per això s'ha decidit adquirir aquest format:

experimentID[timestamp]ElementNumType

-experimentID: Identificador de l'experiment.

-timestamp: Marca de temps en format d'any, mes, dia, hora, minut i segon (yyyymmddhhmmss)

-Element: Inicial del element, C per Coordinator, W per Worker i E per Evaluator.

-Num: Numero d'instància d'aquest element. Per exemple si hi ha tres Coordinators i aquest

és l'identificador pel segon, el numero serà el 2.

-Type: Identificador del tipus de Worker juntament amb el numero d'instància d'aquest tipus es tracta.

Un exemple de cada:

- → Coordinator: testingDHT[20121221000156]C0
- → Tipus de Worker: testingDHT[20121221000156]W0SleepyNode0
- → Altre tipus de Worker: testingDHT[20121221000156]W0ActiveNode12
- → Evaluator: testingDHT[20121221000156]E0

6.2.1.3 Servei Launcher

El servei de Launcher es troba a la classe *ServiceLauncher*. Aquest permet rebre especificacions a través de la xarxa i desplegar l'experiment. Pot gestionar la finalització de l'experiment amb la interfície Storage, que permet a l'usuari donar el comportament que ell vulgui amb el resultat que rep al final de l'experiment.

Tot el sistema de comunicació es fa a través de la classe *LaunchListener* utilitzant NIOSockets de Mina amb la llibreria ConnectorLayer.

Per iniciar un experiment s'ha de crear una instància de la classe *InitMessage*, que Té la creadora i funcions següents:

-public InitMessage(String ID, String spec, Storage store);

-public void addParameter(Object parameter);

Cada missatge té un identificador per l'experiment, l'especificació i el Storage a executar al final d'aquest. També es poden afegir paràmetres pel Coordinator que no estiguin a l'especificació, paràmetres que es poden generar a partir del codi, tals com una cadena de bytes o qualsevol altre cosa que no es pugui formalitzar en un XML.

La classe *EndMessage* permet enviar al servei Launcher la finalització d'un experiment. L'usuari decideix com gestionar els seus resultats, a través del Storage del Launcher o a través del StorageManager de LSim Library.

6.2.2 Implementació del Dispatcher

Durant la implementació del Dispatcher, s'ha hagut de retallar varies funcionalitats que s'havien pensat en un principi per manca de temps i poder tenir una primera versió del tot funcionant. Aquestes funcions són les de gestió de fitxers remots i l'execució d'aplicacions implementades en qualsevol llenguatge. De fet es pot executar qualsevol tipus d'aplicació, sempre i quan es pugui executar el script de l'aplicació en la màquina, però només permet fer la comunicació entre el Dispatcher i les aplicacions fetes en Java, ja que el Dispatcher està implementat en Java.

A continuació es detalla com s'ha implementat el Dispatcher.

6.2.2.1 Llibreria de RunEnvironment

S'ha afegit una altra tecnologia per poder fer funcionar el Dispatcher i crear entorns d'execució. El RunEnvironment és una llibreria feta a la UOC que permet executar aplicacions a partir dels fitxers amb el codi i un script a executar.

Per a cada aplicació executant-se, el RunEnvironment en controla la seva execució. Té assignada

una carpeta on té copiats tots els fitxers de l'aplicació.

Aquesta és la API que ofereix el RunEnvironment per a crear i controlar les execucions de les aplicacions:

```
package runenvironment;

public interface RunEnvironmentAPI {

/**
 * Creates the environment. It relates all application work space, the files
that it needs, etc.
 * @param Environment specification
 * @return 0 if it success, error code otherwise.
 */
public int createEnvironment(Object spec);

/**
 * Starts the new application.
 * @return 0 if it success, error code otherwise.
 */
public int startApplication();

/**
 * Stops the application and frees the environment. Erases all related with the
run environment, like work space, files, etc
 * @return 0 if it success, error code otherwise.
 */
public int stopApplication();

/**
 * Reset the application. Stop the application and start again.
 * @return 0 if it success or error code
 */
public int resetApplication();

/**
 * Returns if the application is still running
 * @return True if the application is running, false otherwise
 */
public boolean isAlive();

/**
 * Returns the state of application.
 * @return Specific object to knows the environment state
 */
public Object stateOfApplication();

/**
 * Returns the application output log.
 * @return Application output log.
 */
public String outputLog();

/**
 * Returns the error log in case that the application had some errors while it
was running
 * @return Application error log.
 */
public String errorLog();

}
```

Text 2: Interfície de RunEnvironment

A partir de les primitives mostrades al text 2, el Dispatcher pot controlar els diferents entorns d'execució i obtenir-ne els logs de l'execució en cas de ser necessari.

Per crear aquest entorn, actualment no hi ha implementat una forma d'obtenir els fitxers de forma remota, així que una de les retallades en el Dispatcher és que els fitxers han d'estar localment en la màquina per poder executar l'aplicació. No obstant, està preparat tot per poder afegir aquest nous mètodes en un futur pròxim.

6.2.2.2 Mètodes de comunicació entre entorns

El Dispatcher n'obria una via de comunicació entre les aplicacions que controla i l'exterior. El fet de crear un entorn d'execució independent per cada aplicació complica aquesta tasca, ja que no es poden comunicar estructures de dades entre diferents entorns, que és com es fa quan s'executa tot en un mateix entorn.

S'utilitzen Sockets per comunicar els entorns de les aplicacions amb el Dispatcher, en concret NIOSockets, ja que es disposa de la llibreria ConnectorLayer per establir comunicacions.

Les classes *DispListener* i *ApplicationListener* són les que permeten tenir NIOSockets escoltant pel Dispatcher i per l'aplicació que es executada. La classe *MessageManager* és la que permet enviar missatges a aquest elements.

Per a poder garantir que cada aplicació pot rebre missatges de forma genèrica hi ha l'interfície *ApplicationHandler*, que té diferents implementacions depenent de qui l'estigui utilitzant. A continuació es mostra les diferents implementacions i utilitats:

1- Control de les aplicacions des del Dispatcher. El Dispatcher té un conjunt de *ApplicationHandler* que permet controlar les aplicacions que hi ha executant-se.

2- Rebre missatges a les aplicacions des del Dispatcher.

3- La implementació del *RunEnvironment* d'aquesta interfície és la que està assignada al Dispatcher en cas d'executar l'aplicació en un entorn diferent del seu, i per tant, permet enviar missatges a l'aplicació a través de Sockets.

4-La implementació *ApplicationMain* és la que permet rebre missatges a les aplicacions des de el Dispatcher quan s'executen en entorns diferents. Aquesta classe és la que rep el *ApplicationHandler* de l'aplicació i la gestiona.

A continuació es mostra de forma gràfica tots aquest elements i com es fa la comunicació:

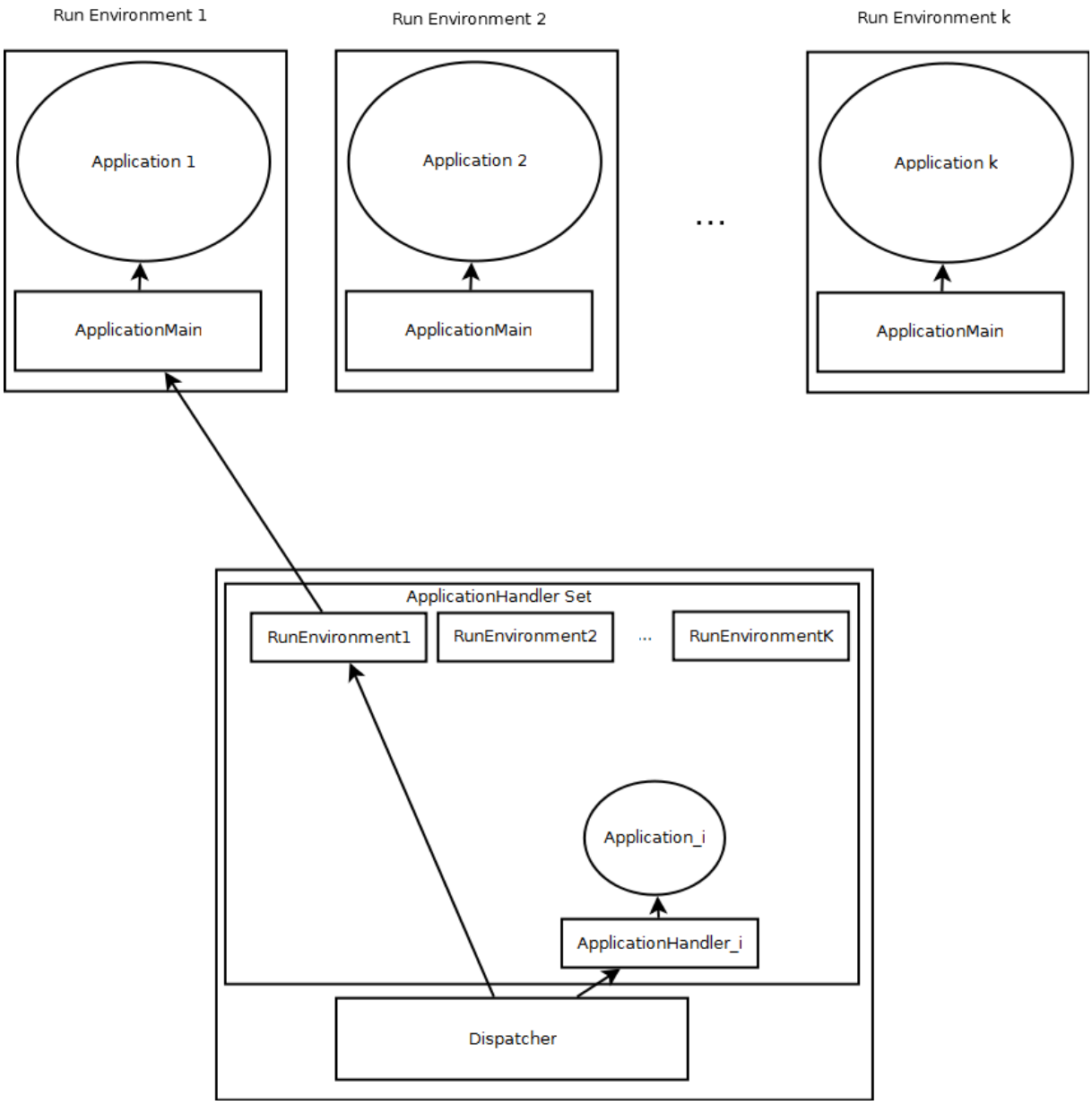


Figura 7: Exemple de ApplicationHandler del Dispatcher

6.2.2.3 Interfícies de comunicació entre aplicacions

El Dispatcher ofereix dues interfícies per comunicar-se amb l'aplicació.

La primer és la que s'ha comentat en l'apartat anterior, l'*ApplicationHandler*:

```
package dispatcher.application;

/**
 * Interface to communicate with a dispatcher
 * @author Esteve Verdura Gelmà
 *
 */
public interface ApplicationHandler{

/**
 * Message received without a session
 * @param message
 */
public void messageRecived(Object mes);

/**
 * Message received with a session
 * @param Session ID
 * @param message
 */
public void messageRecived(String session,Object mes);

/**
 * Sets Dispatcher API to communicate with the dispatcher
 * @param DispatcherAPI
 */
public void setDispatcher(DispatcherAPI disp);

/**
 * Returns if the current application is running
 * @return True if it runs, false otherwise
 */
public boolean alive();

/**
 * Starts the application
 * @param Object -- parameter if it need it
 */
public void start(Object obj);

/**
 * Stops the current application
 */
public void stopApp();
}
```

Text 3: Interfície ApplicationHandler

Cada aplicació s'execut-hi i comuniqui amb el Dispatcher ha d'implementar aquest mètodes. A més hi ha les implementacions que s'han comentat en l'apartat anterior per tal de poder-se comunicar entre diferents entorns.

La segona interfície ofereix l'API de comunicació de l'aplicació cap al Dispatcher:

```
package dispatcher.application;

/**
 * API to connect the application with the dispatcher
 * @author Esteve Verdura Gelmà
 *
 */
public interface DispatcherAPI {

/**
 * Sends a message to the Dispatcher
 * @param ID of application who you want to send the message
 * @param Message
 * @param Connect mode (true if you want to keep the session, false otherwise)
 * @param Host name
 * @param Port
 * @return Reply of the other application in case that connect is true
 */
public Object sendMessage(String id,Object mes,boolean connect, String host,
int port);

/**
 * Sends a message to a session that is connected
 * @param Session ID
 * @param Message
 */
public void answerMessage(String session,Object mes);

/**
 * Sets own application ID
 * @param ID
 */
public void setApplicationID(String ID);

/**
 * This function makes know to the dispatcher that the new application is able
to receive messages
 */
public void startApplication(String id);
public void startApplication();

/**
 * Finish the application on the Dispatcher
 * @param ID of the application that is about to finish
 */
public void finishApplication(String id);
public void finishApplication();
}
```

Text 4: Interfície DispatcherAPI

L'API del text 4 té dues implementacions. La primera per les aplicacions que s'executen en el mateix entorn, anomenada *DispatcherLocalAPI*. La segona per les aplicacions en entorns diferents, anomenada *DispatcherRunAPI*. La utilització d'una o altre és transparent a l'usuari.

6.2.3 Adaptació a CoDeS

Per a les proves s'ha implementat l'API de CoDeS per a gestionar els recursos de manera estàtica. La gràcia d'utilitzar aquesta API és que per l'aplicació sigui transparent utilitzar CoDeS o la implementació de proves, ja que fa servir RMI per oferir les funcions per obtenir i desplegar l'aplicació. Per tant canviar d'una implementació a l'altre només caldrà canviar el fitxer configAPI.properties. Això fa que l'obtenció de serveis de Dispatchers pugui canviar dinàmicament durant l'execució del Launcher, podent canviar l'entorn on s'executen els testos.

Llavors sempre el Launcher farà crides a les mateixes funcions, el que variarà és la implementació del que les serveis. Per no mostrar tota la API, s'ensenyarà només les funcions que utilitza el Launcher i s'implementen a la versió de proves en el text 5:

```
/**
* Returns a list of ServiceHandle, containing the locations of the service
* @param id - id of the service to locate
* @return
*/
public List<ServiceHandle> getLocations(String id) throws RemoteException;

/**
* Deploys a service in the community.
* The Service Specification must have been previously registered.
* @param id - id of the service that will be deployed.
*/
public void deployService(String id) throws RemoteException;
```

Text 5: Funcions de l'interfície codes.api.CoDeSAPI

La implementació de proves obté l'adreça dels serveis de manera estàtica a partir del fitxer sources.txt. Els serveis han d'estar desplegats prèviament als recursos especificats al fitxer.

6.3 Implementació de la LSim Library

S'ha implementat una primera versió de la LSim Library que permeti executar experiments sobre LSim Framework i que sigui fàcilment modificable.

S'ha estat fidel al diagrama de classes del disseny, per tant les classes proposades en la fase de disseny es troben també a la implementació. S'han afegit d'altres classes per tal de que funciones correctament la llibreria.

6.3.1 Funcionalitats a implementar

S'ha decidit implementar cada un dels diferents comportament dels diferents rols de LSim Library: El Coordinator, l'Evaluator i el Worker.

Aquest primer prototipus s'ha decidit no implementar la fase de sincronització ja que no és essencial per a l'execució d'un test. S'ha optat per crear i validar un prototipus que funcionés amb les funcionalitats bàsiques. En una segona fase ja s'implementarà la resta de funcionalitats. Per tant s'ha decidit implementar per cada un dels elements:

- function void init(Handler hand);
- function void start(Handler hand);
- function void sendResult(Handler hand);

I conseqüentment s'implementa les funcions especifiques per a cadascun dels elements:

- function Object waitInit();
- function void endInit(Object param);
- function Object waitStart();
- function void endStart(Object param);
- function Object waitResult();
- function void enResult(Object param);

LSim té un temporitzador global per tot l'experiment que es pot configurar fàcilment a través de la funció de la classe LSim *public void setExperimentTime(int time)*. Per cadascuna de les funcions es dirà si té un temporitzador propi o no.

A continuació es donaran detalls de la implementació per a cada un dels rols. El comportament de cadascuna d'aquestes fases es veu als diagrames de seqüència de la fase de disseny (apartat 4.1.4), per tant apuntarem el que realment s'ha implementat. Es vol mostrar les decisions que s'han pres a l'hora d'implementar cada una de les funcions.

6.3.1.1 Implementació del Coordinator

En aquest apartat es parlarà de la classe *LSimCoordinator* i *Coordinator*, ja que són les classes principals que implementen el Coordinator. El Coordinator està implementat en aquestes dos classes, ja que podem tenir diferents implementacions de Coordinators depenent del comportament ha de tenir la coordinació, com tenir múltiples Coordinators o tenir el Coordinator i Evaluator en un sol element. Actualment segueix el comportament especificat a la fase de disseny.

A diferencia d'altres rols, el Coordinator té dos temporitzadors assignats a través de l'especificació, concretament un a la fase d'inicialització i l'altre a la fase d'enviament de resultats. Els dos temporitzadors s'indiquen a l'especificació perquè l'usuari pugui variar el temps de control de l'experiment d'una forma més senzilla. Només s'indiquen al Coordinator ja que té el control de l'experiment. Tot i així és poden configurar els temporitzadors sempre que es vulgui a LSim a través de la funció *public void setExperimentTime(int time)*.

S'ha afegit la funció *public void addParameter(String ID, Object p);* per poder afegir paràmetres al elements de LSim de forma dinàmica. Es podran afegir per tipus de Worker o per l'Evaluator.

En aquest apartat primerament es parlarà de les estructures de dades més importants, i seguidament de la implementació de cada una de les fases.

6.3.1.1.1 Fase d'inicialització

A continuació es veurà detalls de la implementació de la funció d'inicialització del Coordinator.

public Object waitInit():

El Coordinator es queda bloquejat fins a rebre l'especificació que li envia el Launcher.

Aquest rep la configuració i especificació de l'experiment. A partir de la configuració es configura el Coordinator. A través de l'especificació s'extrau la configuració dels Workers. L'especificació es rep com a XML. El Coordinator disposa de la classe *ConfigurationReader* que el permet obtenir l'especificació dels Workers. Aquesta no apareix al diagrama de classes ja que s'ha afegit durant la implementació.

De la configuració del Coordinator s'extrauen els paràmetres per l'aplicació.

public void endInit(Object p):

El Coordinator envia la senyal d'inicialització a tots els Workers del test juntament amb la seva configuració. Un cop enviat, es bloqueja esperant la resposta de tots els Workers que ha enviat a inicialitzar. Els Workers pot enviar juntament amb el missatge, un paràmetre que el Coordinator guardarà per la següent fase. Per tant tindrà una llista de paràmetres de les contestacions dels Workers.

La funció té un temporitzador per esperar les respostes dels Workers. Aquest temps s'indica a l'especificació del Coordinator. Si s'esgota el temps, el Coordinator avorta l'experiment, comunicant-ho a tots els elements d'aquest.

6.3.1.1.2 Fase de start

El Coordinator ofereix més varietat de funcions de start, en concret ofereix les següents:

- function void start(Handler hand): Envia la senyal de start a tots els Workers.
- function void start(Handler hand, String workersID): Envia la senyal de start només als tipus de Workers de workersID.
- function void start(Handler hand, String workersID, int num): Envia la senyal de start només als tipus de Workers del workersID i només al nombre de Workers num.

S'ha donat aquesta utilitat extra al Coordinator per fer més flexibles les Coordinacions d'experiments.

A continuació es veurà detalls de la implementació de la funció de start del Coordinator.

public Object waitStart():

Configura el temps de l'experiment a través de la funció que ofereix LSim i retorna el llistat de paràmetres dels Workers de la fase d'inicialització. Aquest es passat com a paràmetre al Handler de l'aplicació.

public void endStart(Object p):

El Coordinator envia la senyal de start al conjunt de Workers que pertoqui, juntament amb el paràmetre de la funció. Per defecte envia el conjunt de respostes dels Workers de la fase d'inicialització en cas de que el Handler de l'aplicació retorni un paràmetre nul.

6.3.1.1.3 Fase d'enviament de resultats

A continuació es veurà detalls de la implementació de la funció d'enviament de resultats del Coordinator.

public Object waitResult():

A partir de l'especificació el Coordinator obté el conjunt de Workers que s'han d'avaluar. El Coordinator es queda bloquejat fins a rebre el primer missatge d'un Worker preguntant per l'Evaluator.

El Coordinator s'encarrega de crear un Evaluator per avaluar el test un cop ha rebut el primer missatge. A partir de l'especificació obtinguda a la fase d'inicialització i del *ConfigurationReader* el Coordinator obté la configuració de l'Evaluator. Obté i crea l'Evaluator a l'entorn i ho realitza de la mateixa manera que el Launcher al llançar l'experiment (obtenint els recursos a través de CoDeS). Seguidament l'hi envia la senyal d'inicialització juntament amb la configuració de l'avaluació del test.

El Coordinator assigna als Workers l'Evaluator un cop ja s'ha inicialitzat. Per controlar el procés d'avaluació, s'inicia el segon temporitzador que hi havia a l'especificació.

Quan ja hagi rebut el missatge de tots els Workers a avaluar-se, el Coordinator sortirà de la funció. Avortarà l'experiment en si s'esgota el temps del temporitzador.

6.3.1.2 Implementació del Worker

El Worker també és implementat per dos classes on *LSimWorker* ofereix les funcionalitats a l'aplicació i la classe *Worker* les implementa.

A continuació es comentaran cada una de les funcions i com finalment s'ha implementat en el Worker.

6.3.1.2.1 Fase d'inicialització

A continuació es mostrarà detalls de la implementació de la inicialització del Worker.

public Object waitInit():

El Worker està bloquejat fins a rebre la configuració de l'experiment. Aquest retorna els paràmetres per a l'aplicació en forma de llista d'objectes.

En aquest punt hi ha un temporitzador executant-se, en cas de superar el temps establert, el Worker deixarà d'executar-se. S'atura quan rep el missatge del Coordinator. Té un temps assignat de base, tot i que l'usuari la pot modificar amb el mètode de LSim ja esmentat.

public void endInit(Object p):

Envia al Coordinator un missatge per indicar que ha finalitzat la inicialització. Com a paràmetre del missatge s'enviarà el paràmetre de la funció.

6.3.1.2.2 Fase de start

A continuació es mostrarà detalls de la implementació de la fase de start del Worker. La funció *public void endStart(Object)* no realitza res.

public Object waitInit():

El Worker inicia el temporitzador per a l'execució de l'experiment de la mateixa forma que es fa a la fase d'inicialització.

El Worker es bloqueja fins a rebre el missatge que pot començar l'experiment. El missatge té com a paràmetre el que ha retornat el Handler del Coordinator de la fase de start. Per defecte és la llista dels paràmetres dels Workers de la fase d'inicialització. En tot cas la funció retorna

aquest paràmetre.

6.3.1.2.3 Fase d'enviament de resultats

A continuació es mostrarà detalls de la implementació de la fase d'enviament de resultats del Worker.

public Object waitResult():

El Worker contacta amb el Coordinator per obtenir l'adreça de l'Evaluator. Es bloqueja fins a rebre la resposta.

En aquesta fase LSim no re-configura el temporitzador, utilitza el temporitzador que s'ha iniciat a la fase anterior. No s'ha posat un temporitzador especific en aquest punt ja que té funcionant el de la fase anterior. En tot cas l'usuari pot tornar a iniciar en aquest fase el temporitzador.

Retorna l'adreça de l'Evaluator. L'aplicació no té la necessitat de saber aquesta adreça, però es facilita per si vol tenir control sobre quin element està avaluant el resultat.

public void endResult(Object p):

Aquest rep com a paràmetre el resultat a enviar a l'Evaluator, que és el que retorna el Handler de l'aplicació. Per això LSim disposa ja d'un Handler especial per l'enviament de resultat que es descriurà en la secció d'interfícies (apartat 5.3.2.2).

Finalment envia el resultat a l'Evaluator.

6.3.1.3 Implementació del Evaluator

La implementació de l'Evaluator segueix el mateix esquema que els altres elements, on *LSimEvaluator* dóna les funcionalitats d'aquest i la classe *Evaluator* és qui les implementa.

La classe *Test* és la que controlarà que ha rebut tots els resultats a avaluar. Es segueix la mateixa idea que en el diagrama de classes, on s'ha separat el que és el control de testos al mètode de recollida de resultats. Així el mètode el control de resultats és modulable i el mètode la recollida és transparent al control.

La classe *ResultSet* és el que rebrà l'aplicació com a conjunt de resultats dels Workers. Aquesta ofereix els següents mètodes per a la gestió de resultats:

- public List<Object> getAllResults(): Retorna una llista amb tots els resultats dels Workers.
- public List<Object> getResults(String workerID): Retorna una llista amb tots els resultats del tipus de Workers que s'especifica per paràmetre.

S'ofereixen aquestes dos funcions per obtenir els resultats perquè hi hagi coherència amb l'idea de diferents tipus de Workers. D'aquesta manera l'usuari podrà elegir com vol obtenir els resultats en funció de la dificultat de l'avaluació.

A continuació es detallarà part de la implementació en cada una de les fases de l'Evaluator. En aquest cas s'ha omès la fase de start ja que, tot i estar implementada, no és necessària per l'Evaluator.

6.3.1.3.1 Fase d'inicialització

A continuació es mostrarà detalls de la implementació de la fase d'inicialització de l'Evaluator.

public Object waitInit():

L'Evaluator es bloqueja fins a rebre la senyal d'inicialització des de Coordinator, juntament amb la configuració del test.

S'inicia el temporitzador de la fase de la mateixa forma que ho fa el Worker. En cas de superar aquest temps l'Evaluator deixarà d'executar-se.

Un cop rep la configuració del test, aquest crea una instància de la classe *Test* amb la configuració de l'experiment. Retorna el llistat de paràmetres per aquest test al Handler de l'aplicació.

public void endInit(Object p):

Aquest envia al Coordinator que ja ha inicialitzat el test. L'aplicació no pot enviar cap tipus de missatge al Coordinator, ja que aquest no té un Handler assignat per a la gestió del missatge. La inicialització de l'Evaluator, el Coordinator la fa de forma transparent a l'usuari.

6.3.1.3.2 Fase d'enviament de resultats

A continuació es mostrarà detalls de la implementació de la fase d'enviament de resultats de l'Evaluator.

public Object waitResult():

Anteriorment a aquesta fase l'Evaluator ha iniciat un temporitzador per controlar l'enviament de resultats. Aquest és pot modificar sempre des de LSim tal i com és fa amb la majoria de temporitzadors.

L'Evaluator espera a rebre tots els resultats dels Workers que s'han d'avaluar. Cada cop que rep un missatge aquest l'hi ho passa a la classe *Test*, que és la que controla l'experiment. Un cop el *Test* té tots els resultats necessaris per a l'avaluació dels Workers, l'Evaluator deixa de rebre missatge i retorna a l'aplicació una instància de la classe *ResultSet* amb el conjunt de resultats a avaluar. El Handler que rep aquest paràmetre serà el que s'encarregarà de fer l'avaluació de l'experiment.

public void endResult(Object p):

El Handler de l'aplicació passa per paràmetre el resultat de l'avaluació del test. Aquest és passat a la instància de la interfície *StorageManager* que tingui aquest experiment. Aquest és el que s'encarregarà de guardar els resultats on l'usuari especifiqui. En la secció següent hi ha una explicació més detallada del *StorageManager* (apartat 5.3.2.3).

6.3.2 Interfícies per l'usuari

LSim Librari no només ofereix una sèrie de funcions, també ofereix un conjunt d'interfícies per a comunicar l'aplicació amb la part de LSim Library i separar la part del experiment del codi original de l'aplicació.

A continuació es parlarà de la utilització d'aquestes interfícies.

6.3.2.1 Interfície ApplicationManager

La interfície permet a l'usuari executar l'aplicació utilitzant LSim Library en el LSim Framework. Està pensada perquè el Dispatcher actual l'executi. A continuació es mostra la interfície:

```
package lsim.application;

import lsim.LSimDispatcherHandler;

/**
 * Interface that needs to be implemented to run an application on LSim
environment.
 * @author Esteve Verdura Gelmà
 *
 */
public interface ApplicationManager {

	/**
	 * It is where user has to put all your code as a main.
	 * @param disp
	 */
	public void start(LSimDispatcherHandler disp);

	/**
	 * Stops the application by the dispatcher when it is needed.
	 * @see Implementation on lsim.worker.WorkerTemplate
	 */
	public void stop();

	/**
	 * Make to know to the dispatcher is the application still alive.
	 * @return true by default, false in case that you want to finish the
application, or it is over.
	 */
	public boolean isAlive();

	/**
	 * @deprecated This version does not work in the current dispatcher
implementation.
	 */
	void start();

}
```

Text 6: Interfície ApplicationManager

Com es pot veure en el text 6, ofereix els mètodes bàsic per a executar l'aplicació:

- start: Executa el codi de l'aplicació, és el que fa de main. En cas d'estar en LSim Framework utilitza la primera funció de start, d'altre forma s'utilitzaria la segona.
- stop: Ha d'aturar l'execució de l'aplicació. El mètode s'ofereix per poder alliberar recursos i que l'aplicació pugui gestionar el seu acabament. Així pot guardar o realitzar el que necessiti.
- is alive: Comprova si l'aplicació segueix executant-se, per en cas negatiu alliberar els rescursos.

6.3.2.2 Interfície Handler

El Handler és una la interfície més importants de LSim Library. Permet la interacció de l'usuari amb les fases de LSim. S'executa al mig de cada una de les fases, on depenent de la fase i el rol, rebrà paràmetres o haurà de retornar algun resultat, depenent de com l'usuari vulgui fer anar el test. El fet de permetre aquesta interacció al mig de les fases permet que LSim Library sigui poc intrusiva al codi de l'aplicació.

S'ofereixen implementacions genèriques per a casos trivials.

A continuació al text 7, es mostra la interfície Handler:

```
package lsim.application.handler;

/**
 * Handler to implement in every phase on LSim. It will execute during the
phase.
 * It is better make your own implementation for every handler, but you can use
the handlers that are implement.
 * @author Esteve Verdura Gelmà
 * @see lsim.application.handler.InitHandler for an example of initial handler
 * @see lsim.application.handler.StartHandler for an example of start handler
 * @see lsim.application.handler.ResultHandler for an example of result handler
 * @see lsim.application.handler.DummyHandler for a useful dummy handler
 */
public interface Handler {

          /**
           * Executes the code that you want during the phase. To know how
works every phase in every element, take a look to the documentation.
           * @param Parameter of this phase
           * @return Result/reply that need to send in LSim layer depending on
phase and element.
           */
          public Object execute(Object obj);

}
```

Text 7: Interfície Handler

La funció *execute* serà la que l'usuari haurà de definir el comportament de l'aplicació en cada una de les fases. A continuació es defineix per a cada rol i fase, quin paràmetre té i que pot retornar el Handler. En cas de ser un paràmetre es definirà el tipus d'aquest:

- Coordinor init:

  Paràmetre: Paràmetres del test -> *List<Object>*

  Retorna: null

- Coordinator start:

  Paràmetre: Conjunt de respostes en la fase d'inicialització dels Workers -> *List<Object>*

  Retorna: null

- Coordinator sendResult:

  Paràmetre: null

Retorna: null

- Worker init:

  Paràmetre: Paràmetres del test -> *List<Object>*

  Retorna: Paràmetre de resposta que es comunicarà al Coordinator.

- Worker start:

  Paràmetre: Conjunt de respostes en la fase d'inicialització dels Workers -> *List<Object>*

  Retorna: null

- Worker sendResult:

  Paràmetre: Adreça del Evaluator -> *ApplicationInformation*

  Retorna: Resultat del Worker

- Evaluator init:

  Paràmetre: Paràmetres del test -> *List<Object>*

  Retorna: null

- Evaluator start:

  Paràmetre: null

  Retorna: null

- Evaluator sendResult:

  Paràmetre: Conjunt de resultats dels Workers -> *ResultSet*

  Retorna: Resultat de l'avaluació

Tal i com s'ha comentat, s'ofereixen implementacions ja fetes de Handler. A continuació es mostren aquestes implementacions de forma esquemàtica i la seva funcionalitat.

1-InitHandler: Adquireix de manera automàtica per obtenir-los després d'executar-se la fase d'inicialització. Es pot assignar el temps del primer temporitzador a parir de la funció creadora de la classe.

- public InitHandler(LSim lsim, int time);
- public List<Object> getParameters();

2-DummyHandler: Handler que no fa res. Serveix per aquelles fases que no és necessari que l'aplicació realitzi res.

3-ResultHandler: Pensat per la fase d'enviament de resultats del Worker. Retorna el resultat que se l'hi ha assignat a l'instantització de la classe.

- public ResultHandler(Object result);

Els Handlers es poden assignar als temporitzadors. S'executaran quan el temporitzador esgoti el temps que té assignat, d'aquesta forma l'usuari pot intervenir en els cassos de time out en cada una de les fases o en l'experiment. La funció de la classe *LSim* que permet assignar un Handler al temporitzador és la següent:

- public void setTimeOutAction(Handler hand);

6.3.2.3 Interfície StorageManager

El StorageManager s'encarregarà de guardar el resultat del test. Permet a l'usuari fer la gestió del resultat de l'experiment. S'ha separat d'aquesta forma perquè està pensat oferir un servei per a guardar resultats més endavant.

A continuació es mostra l'interfície:

```
package lsim.storage;

import lsim.application.ApplicationManager;

/**
 * Stores the test result
 * @author Esteve Verdura Gelmà
 *
 */
public interface StorageManager {

          /**
           * Stores the result where the current implementation idicate it
           * @param result
           */
          public void sendResult(Object result);
}
```

Text 8: Interfície Storage Manager

La funció *sendResult* rep com a paràmetre el retorn el Handler de la fase d'avaluació de l'Evaluator. Així l'usuari pot gestionar el format dels resultats de la manera que més li convingui.

6.3.3 Implementació de la comunicació i bloqueig de LSim

Com a mètode de comunicació i bloqueig hi ha la classe *SyncBuffer*, la qual ens permet aturar el Thread on s'esta executant l'aplicació, sempre i quan no tingui cap missatge que atendre. Fa d'enllaç entre LSim i qualsevol mètode de comunicació.

El *SyncBuffer* és una cua FIFO. Ofereix les següents funcions:

- public void startWait(): bloqueja l'execució fins que té un missatge per atendre.
- public void startWait(Timer timer): bloqueja l'execució fins que té un missatge o salta el temporitzador.
- public void addObject(Object p): afegeix un paràmetre a la cua i el desbloqueja.
- public Object getObject(): retorna el primer paràmetre que ha entrat a la cua.
- public boolean isEmpty(): retorna cert si la cua està buida, fals d'altre forma.

En aquesta primera versió de LSim, aquest es comunica amb el Dispatcher. La classe *LSimDispatcherHandler* fa l'abstracció de comunicació entre el *SyncBuffer* i el Dispatcher.

Per enviar missatges LSim Library fa servir l'implementació *SendInformation* de la interfície *Send*. Aquesta fa la comunicació a través de NIOSockets a partir de la llibreria ConnectorLayer que s'ha comentat anteriorment.

6.3.4 Gestió de paquets

LSim Librari ofereix totes les funcionalitats d'una forma intuïtiva i organitzada. Per això les classes s'han anat repartint en diferents paquets, de tal forma que l'usuari pugui accedir a les funcionalitats d'una forma fàcil.

S'ha decidit dividir en paquets segons el tipus de funcionalitat que s'ofereix. A continuació es mostraran els paquets principals del projecte amb una descripció del que contenen.

- lsim: Conté totes les classes generals de LSim Library, tals com *LSim* o *ExperimentTimer*. Ofereix les funcionalitats que utilitzaran tots els elements de LSim Lybrary (Worker, Coordinator i Evaluator).
- lsim.application: Ofereix les interfícies que ha d'implementar l'usuari i classes relacionades amb l'aplicació.
- lsim.application.handler: Conté la interfície Handler juntament amb implementacions predeterminades per a cada una de les fases per a facilitar la feina a l'usuari.
- lsim.coordinator: Conté tot relacionat amb el Coordinator. L'usuari pot obtenir el *LSimCoordinator* des de aquest paquet i la classe *Coordinator*, per a futures modificacions en el comportament de la coordinació. Ofereix una implementació de *ApplicationManager* anomenada *StandartCoordinator*, la qual conté la implementació d'un Coordinator senzill, vàlid per a testos simples.
- lsim.evaluator: Conté tot relacionat amb l'Evaluator, tal com el *LSimEvaluator* i la classe Evaluator. Ofereix el *StandartEvaluator* semblant al que ofereix el Coordinator, el qual només se l'hi ha d'indicar quin StorageManager té el test i el Handler que avalua els resultats. D'aquesta forma l'usuari no s'ha de preocupar d'implementar un Evaluator.
- lsim.messages: Conté tots els missatges que utilitza LSim internament. El paquet està dividit per Evaluator, Coordinator i Worker en funció d'on vagin dirigits els missatges.
- lsim.net: Conté l'interfície *Send* juntament amb l'implementació *SendInformation* per a fer funcionar LSim utilitzant NIOSockets de Mina. Aquí és on hi hauran els mètodes de comunicació que s'hagin d'implementar per tal de fer LSim més escalable.
- lsim.result: Conté tot el que estigui relacionat amb l'avaluació i obtenció de resultats del test. Es troba la classe *ResultSet*, que és la que conté el conjunt de resultats dels Workers.
- lsim.storage: Conté l'interfície *StorageManager* amb la seva implementació per defecte. S'ha separat del paquet lsim.application perquè en un futur es podrien oferir un mètode d'emmagatzemament automàtic, amb gestió de fitxers remots, fent que deixi de ser una interfície que hagi d'implementar l'usuari a passar a ser una nova funcionalitat de LSim.
- lsim.worker: Conté tot lo relacionat amb el Worker, tal com la classe *LSimWorker* i *Worker*. Ofereix també una plantilla amb la implementació d'un Worker, per tal de facilitar la implementació d'aquest a l'usuari.

6.4 Validació

Per validar LSim s'han fet un conjunt d'aplicacions dedicades a fer funcionar les diferents funcionalitats de LSim. No es detallarà el com s'ha realitzat aquestes validacions, però si que s'ha

validat en aquesta fase.

S'ha validat les següents funcionalitats:

- Desplegament correcte de tots els rols de l'especificació.

- Assignació correcte d'identificadors als rols.

- Es comprova que els missatges s'enviïn correctament des de totes les fases i amb els paràmetres corresponents.

- Es comprova que l'Evaluator rep tots els resultats a avaluar dels Workers.

- Es fa fallar diferents nodes en cada una de les fases i es comprova que s'alliberen els recursos automàticament. Així es validen els temporitzadors i gestions de time out.

- Es valida que LSim Library és fàcilment adaptable a les aplicacions comprovant que les línies de codi afegides són poques.

7 Adaptació de LSim a les pràctiques d'ASD

A l'assignatura d'arquitectura de sistemes distribuïts (ASD) de la UOC s'ha utilitzat LSim per avaluar les practiques dels alumnes al semestre de tardor del curs 2011-2012.

7.1 Descripció de la pràctica

Els alumnes han d'implementar un protocol d'encaminament basat en claus en un entorn distribuït, en aquest cas Internet. Durant la implementació de la seva solució, els estudiants a nivell local poden posar a prova la seva aplicació mitjançant un script per a l'execució local proporcionat pel professorat. Un cop finalitzada, l'alumne ha provar la seva solució en un entorn distribuït realista. La implementació feta per l'alumne s'executa en l'entorn realista i se li passa un seguit de testos preparats pel professors de l'assignatura. Els resultats d'aquestes proves es recullen i avaluen de manera automàtica i cada alumne pot veure el resultat de les seves execucions a través d'una aplicació web.

LSim s'encarrega de desplegar la solució de l'alumne, passar-hi els jocs de proves i validar-ne el correcte funcionament.

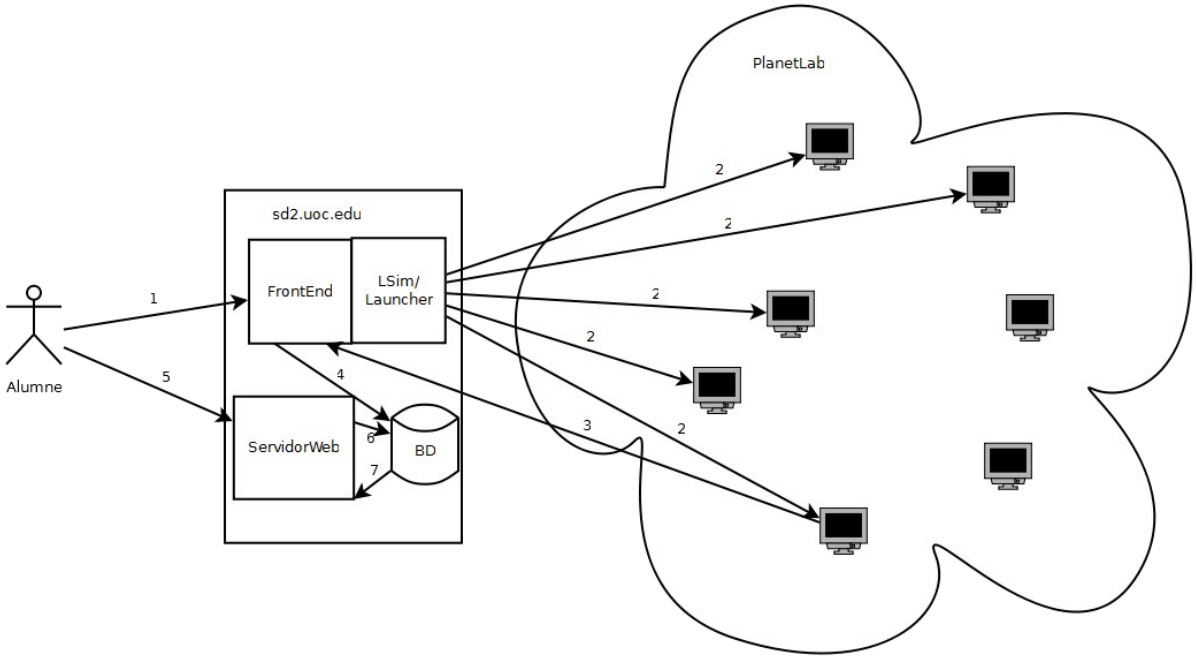


Figura 8: Representació de l'execució d'un alumne

A la figura 8 es representa el sistema que permet realitzar les proves als alumnes i el flux del llançament d'una prova a l'entorn d'execució:

1. L'alumne envia la implementació de la seva solució al servidor sd2.uoc.edu.
2. El servidor FrontEnd rep la petició de l'alumne. Prepara l'especificació de l'experiment incloent la implementació de la pràctica feta per l'alumne i ho desplega utilitzant el Launcher.
3. Un cop finalitzada l'execució, l'Evalutor retorna l'avaluació de l'experiment al servidor FrontEnd de sd2.uoc.edu.

4. El FrontEnd guarda el resultat a la BD de l'assignatura.
5. L'alumne consulta el resultat de la prova al web de l'assignatura. Els passos 6 i 7 del servidor web consulten els resultats a la BD i els mostren a l'alumne.

7.2 Resultats

La utilització de LSim per avaluar les pràctiques d'ASD va servir per validar el funcionament de LSim Framework i LSim Library en una situació real, cosa que ha estat de gran ajuda per crear una versió robusta. Alguns exemples de comportaments dels usuaris que ens hem trobat i hem hagut de tractar són: entrar a dins d'un bucle infinit, errors produïts durant l'execució de la solució de l'alumne o no complir amb l'especificació de l'enunciat (per com estaven definides les proves no s'admetia la de declaració de classes a dins de classes o compilació del codi en una versió de Java diferent a la de les proves).

Es va validar el següent:

- Pas de paràmetres dinàmicament a través del Launcher i Coordinator. Es generaven els executables de l'alumne al FrontEnd i s'afegien dinàmicament els paràmetres de l'experiment.
- Robustesa del Dispatcher. Hi havia aplicacions d'alumnes que fallaven, cosa que el Dispatcher detectava i n'aturava l'execució, tot alliberant els recursos.
- Desplegament de l'experiment. Es varen realitzar molts experiments (1257 execucions) sobre el LSim Framework, validant-ne així el seu funcionament.
- Recuperació davant fallades de l'experiment. Si algun node es penjava, els altres elements de LSim deixaven d'executar-se un cop esgotat el temps assignat a l'experiment.

Per limitacions imposades pels jocs de proves, les execucions fetes amb LSim només utilitzaven un sol node, que executava l'avaluació del test amb la implementació de l'alumne i en retornava el resultat a FrontEnd. Tot i només executar un node, intervenien tres elements a l'experiment: el Coordinator, el Worker (és el que executa el codi de l'alumne) i l'Evaluator.

Actualment s'està implementant els jocs de proves de manera que es puguin executar de manera totalment distribuïda i, per tant, utilitzar tota la potencialitat que ofereix LSIM. La idea és tenir-ho tot apunt per les execucions del semestre de primavera del curs 2011-2012.

A partir dels enviaments dels alumnes s'han extret les següents dades:

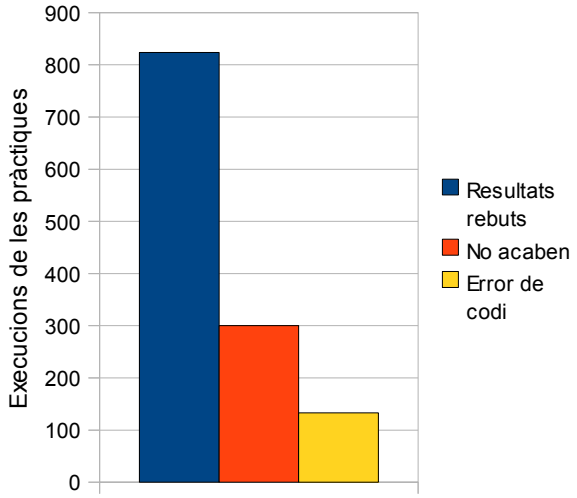


Figura 9: Resultats d'ASD

La figura 9 mostra com s'ha comportat cadascuna de les 1257 execucions de la pràctica que els alumnes han enviat a executar utilitzant LSim. 824 execucions de les 1257 enviades (un 65,55%) han finalitzat l'execució dels testos que es passaven. Poden no haver superat cap test, algun test o tots. El que ens interessa aquí és saber que LSim les ha pogut executar. 133 (un 10,58%) execucions va generar un error d'execució que va aturar el transcurs de l'experiment. LSim va capturar aquests errors i va avortar l'execució de l'experiment. Les 300 execucions restants (un 23,87%) el codi enviat a executar no funcionava correctament i feia que l'execució no acabés (per exemple, el codi de l'alumne entrava en un bucle infinit) o sobrepassés el temps màxim assignat per a que el joc de proves acabés. En aquests casos, LSim va aturar i alliberar tots els recursos agafats un cop finalitzat el temps màxim assignat a l'experiment.

8 Conclusions

8.1 Objectius assolits

S'han assolit tots els objectius que s'han plantejat a la introducció (apartat 1.3):

- S'ha implementat una primera versió de LSim Framework que permet desplegar automàticament aplicacions en entorns distribuïts realistes. Falta la gestió de fitxers remots, que s'afegirà a la segona versió del Dispatcher.
- La LSim Library permet adaptar l'aplicació al framework, coordinar-ne els diferents rols i, recollir i avaluar els resultats dels nodes. A la primera versió de LSim Library li falta la funció de sincronització que s'afegirà en la segona versió.
- S'ha validat el correcte funcionament de LSim Framework i LSim Library desplegant varies aplicacions realitzades expressament per validar les funcionalitats de LSim.
- S'ha utilitzat LSim per desplegar les pràctiques de l'assignatura d'arquitectura de sistemes distribuït de la UOC. Ha servit per validar d'una forma més exhaustiva LSim.

Utilitzar LSim en un cas real com el de les practiques d'arquitectura de sistemes distribuïts de la UOC, ha suposat que s'havia de crear una primera versió robusta de LSim.

8.2 Conclusions personals

Durant el desenvolupament del projecte he pogut veure les dificultats reals del desenvolupament d'aplicacions distribuïdes i les avantatges de tenir una eina per poder realitzar proves en entorns distribuïts realistes.

Hem desenvolupat una eina que encara li queda molt per treballar. Durant aquest temps s'ha pogut desenvolupar una primera versió robusta i que ja es pot utilitzar, però que encara es pot millorar molt. Personalment és un proposta engrescadora poder seguir treballant en aquest projecte i fer-ne una eina més completa.

8.3 Treball futur

Una de les conclusions que s'han tret és que tot just es presenta una eina bàsica per al desplegament d'aplicacions en un entorn distribuït realista. A continuació es parlarà del treball que queda per fer per a tenir una eina bona:

- Fer més escalable el sistema. Afegir mètodes de multi-difusió per a propagar missatges a més nodes eficientment en funció de la mida de l'experiment.
- Afegir la fase de sincronització per a donar més llibertat a l'hora de coordinar experiments.
- Afegir monitorització per saber en quin punt es troba de l'experiment.
- Afegir la gestió remota de fitxers al Dispatcher. Així es podrà obtenir els fitxers de l'aplicació per executar en el Dispatcher remotament.
- Tolerància a fallades. Afegir repliques del Coordinator els que controlin els nodes que han caigut i puguin recuperar nodes caiguts. Una opció seria el utilitzar mètodes de checkpointing per recuperar l'estat dels nodes caiguts en cas de que es necessiti.
- Avaluació més eficient. Adaptar l'Evaluator depenent de la complexitat d'avaluació que hi hagi en el experiment. Utilitzar diferents Evaluators en cas de que sigui complexa i necessiti

de recursos, on cada Evaluator s'encarrega d'una part de l'avaluació.

- Afegir mètodes de sandbox per controlar l'execució de les aplicacions.
- Desplegar LSim en altres frameworks, tals com OMF o GUSH.
- Afegir mètodes per poder planificar experiments de llarga durada en funció dels recursos disponibles a l'entorn.
- Millorar el sistema d'avaluació de pràctiques de sistemes distribuïts de la UOC.

9 Bibliografia

[1] CoDeS pàgina principal: http://www.dpcs.uoc.edu (3/01/2012)
[2] PlanetLab pàgina principal: http://planet-lab.org (13/01/2012)
[3] Marquès, J.M., Lázaro, D., Juan, A., Vilajosana, X., Domingo, M., Jorba, J. (2010). PlanetLab@UOC: A Real Lab over the Internet to experiment with Distributed Systems. Computer Applications in Engineering Education
[4] Marquès, J.M., Juan, A., Perez, A., Daradoumis, T., Lázaro, D., Mondejar, R. (2012). Using real labs over the Internet in distance-learning courses on distributed systems: students' behavior and feedback. International Journal of Electrical Engineering Education
[5] Lua pàgina principal: http://www.lua.org/ (15/01/2012)
[6] Apache Mina pàgina principal: http://mina.apache.org/ (13/01/2012)
[7] JDom pàgina principal: http://www.jdom.org/ (13/01/2012)
[8] Diver pàgina principal: http://diver.sourceforge.net/ (13/01/2012)
[9]ArgoUML pàgina principal: http://argouml.tigris.org/ (13/01/2012)
[10] Codeploy pàgina principal: http://codeen.cs.princeton.edu/codeploy/ (13/01/2012)
[11] Andrea Omicini, Franco Zambonelli (1999) Coordination for Internet application development
[12] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. 2010. OMF: a control and management framework for networking testbeds. SIGOPS Oper. Syst. Rev. 43, 4 (January 2010), 54-59.
[13] OMF pàgina principal: http://omf.mytestbed.net (13/01/2012)
[14] Lorenzo Leonini, Étienne Rivière, and Pascal Felber. 2009. SPLAY: distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze). In Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI'09). USENIX Association, Berkeley, CA, USA, 185-198.
[15] SPLAY pàgina principal: http://www.splay-project.org/ (13/01/2012)
[16] Jeannie Albrecht and Danny Yuxing Huang. Managing Distributed Applications using Gush. Proceedings of the Sixth International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, Testbeds Practices Session (TridentCom), May 2010.
[17] Gush pàgina principal: http://gush.cs.williams.edu/trac/gush (13/01/2012)
[18] Jeannie Albrecht. Bringing Big Systems to Small Schools: Distributed Systems for Undergraduates. Proceedings of the Fortieth ACM Technical Symposium on Computer Science Education (SIGCSE), March 2009.

10 Annex

10.1 Tutorial

10.1.1 Que es necessita tenir abans de fer res?

El primer que és necessita és tenir una aplicació feta en Java que és vulgui provar en un entorn distribuït realista.

Com a exemple al tutorial es fa servir una aplicació que l'únic que fa és comunicar-se entre els diferents nodes. En concret cada un dels nodes d'aquesta té un numero que vol comunicar als altres nodes que coneix, i espera a rebre el mateix d'aquest nodes. Un cop té els números, simplement els suma. En tots els nodes repartits per la xarxa, al final haurien de tenir el mateix número, ja que es coneixen entre tots ells, i per tant interessa que hi hagi aquesta comunicació entre ells. Aquesta aplicació es dirà NodeSum. Servirà per il·lustrar com realitzar aquest procés d'adaptació.

En aquest cas interessa crear un experiment que et permeti passar per paràmetre quin numero té cada node, que sincronitzi l'inici de l'experiment i que reculli els resultats de cada node per comprovar que han sumat bé i per tant la comunicació entre ells ha estat correcte.

Per que es fa servir aquest exemple? Doncs perquè toca tots els aspectes que ofereix LSim, que són l'especificació de l'experiment, el pas de paràmetres, la configuració del test a la inicialització i la recollida de resultats.

Doncs un cop es té l'aplicació que volem provar, s'ha d'obtenir la llibreria LSim i totes les seves dependències per a poder crear aquesta prova. Actualment s'esta construint una pàgina Web on es pugui descarregar LSim (http://dpcs.uoc.edu/projects/lsim) i obtenir informació, però en tot cas sempre podeu contactar a traves de esteve.verdura@est.fib.upc.edu per obtenir la llibreria.

10.1.2 Com s'utilitza LSim amb l'aplicació?

El primer que s'ha de fer és pensar com ha d'anar l'experiment. A partir del codi, podem identificar diferents parts que aniran en l'experiment, les quals són: inicialització de l'aplicació, inicialització del test, execució del test i recollida de resultats.

En la inicialització de l'aplicació el que es fa és crear totes les estructures i dades necessàries que no siguin dependents del test. Tot seguit en la inicialització del test es rebran els paràmetres de l'especificació de l'experiment, i per tant l'aplicació s'adaptarà segons aquest paràmetres. A continuació executa el test de forma normal. Finalment s'envien els resultats del test per a ser avaluats.

Seguint aquesta lògica, es crea la implementació de la interfície *lsim.application.ApplicationManager* per crear cada un dels rols a LSim, en concret el que interessa és posar el que s'executarà a la funció *public void start(LSimDispatcherHandler dips)*. Per a assignar el mètode de comunicació per a LSim, en tots els cases es recomana que les dues primeres línies de codi siguin les següents:

```
1        LSimElement lsim = new LSimElement();
```

```
2          lsim.setDispatcher(disp);
```

On Element es refereix a si és un Worker, Coordinator o Evaluator.
Per a cada una de les fases LSim té una funció assignada. Llavors és recomanable que l'usuari implementi la interfície lsim.application.handler.Handler per a interactuar amb cada una d'aquestes fases. Tot i així, LSim t'ofereix una sèrie d'implementacions de Handlers útils per a cada una de les fases:

-Per la fase d'inicialització es dóna el *lsim.application.handler.InitHandler* que té el mètode *public List<Object> getParameters()* el qual et retorna un llistat d'objectes que són els paràmetres de l'especificació.

-Per a la fase de start, té el *lsim.application.handler.DummyHandler*, el qual no fa res. Aquest també es pot utilitzar per aquelles fases que l'aplicació no hagi d'interactuar amb LSim.

-Per a la fase de resultats ofereix el *lsim.application.handler.ResultHandler*, el qual al crear una instància és necessari assignar-li un resultat que és el que s'enviarà a la fase d'enviament de resultats. És molt útil per als Workers.

-S'ha proporcionat el *lsim.application.handler.EvaluationHandler* per ajudar al mètode d'avaluació del Evaluator. Es parlarà amb més detall en aquest apartat de l'utilitzat d'aquest Handler.

10.1.2.1 Worker

Per veure com implementar un Worker s'exemplificarà amb l'aplicació NodeSum.

Com ja s'ha comentat, el NodeSum és una aplicació que es comunica amb les altres instàncies d'aquesta en la xarxa. Aquí hi ha el que és el seu codi separat per les part que s'han explicat anteriorment:

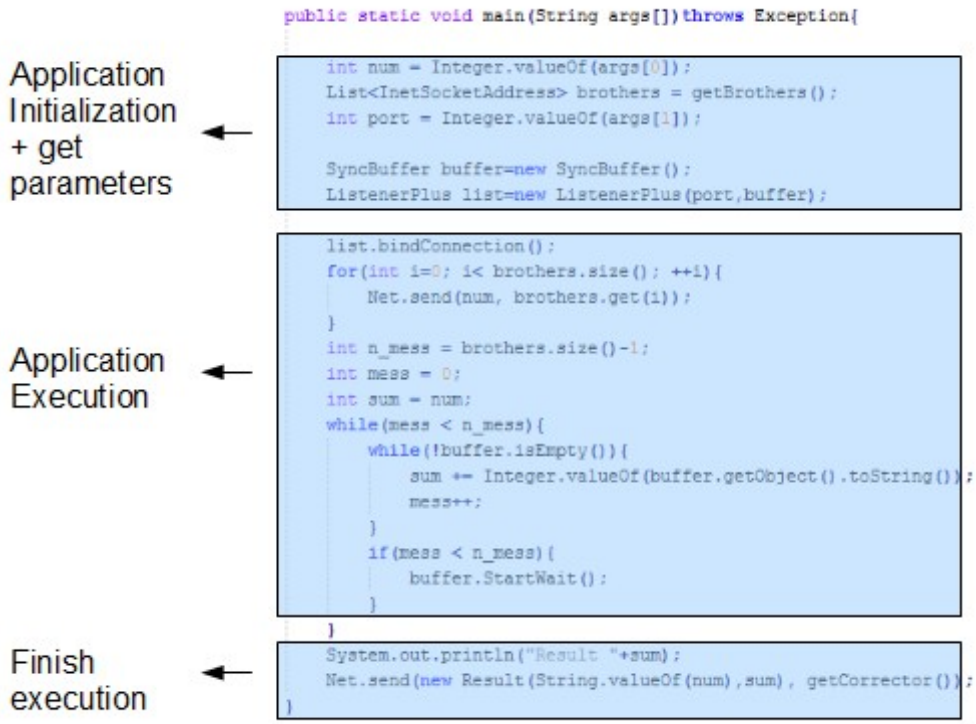


Figura 10: Exemple NodeSum

Com es pot veure, tenim definides les diferents parts en quadrats. La primera part d'inicialització i obtenció de paràmetres, on s'obté el numero per aquest node, el llistat de nodes on comunicar-se i el port on ha d'obrir la comunicació. La segona d'execució de l'aplicació i una tercera de finalització de l'aplicació. Els mètodes per obtenir els paràmetres en aquest cas són estàtics, per tant si es vulgues canviar-los s'hauria d'anar a la màquina que executa l'aplicació i canviar-los.

A continuació es veurà que s'ha afegit per adaptar-lo a LSim:

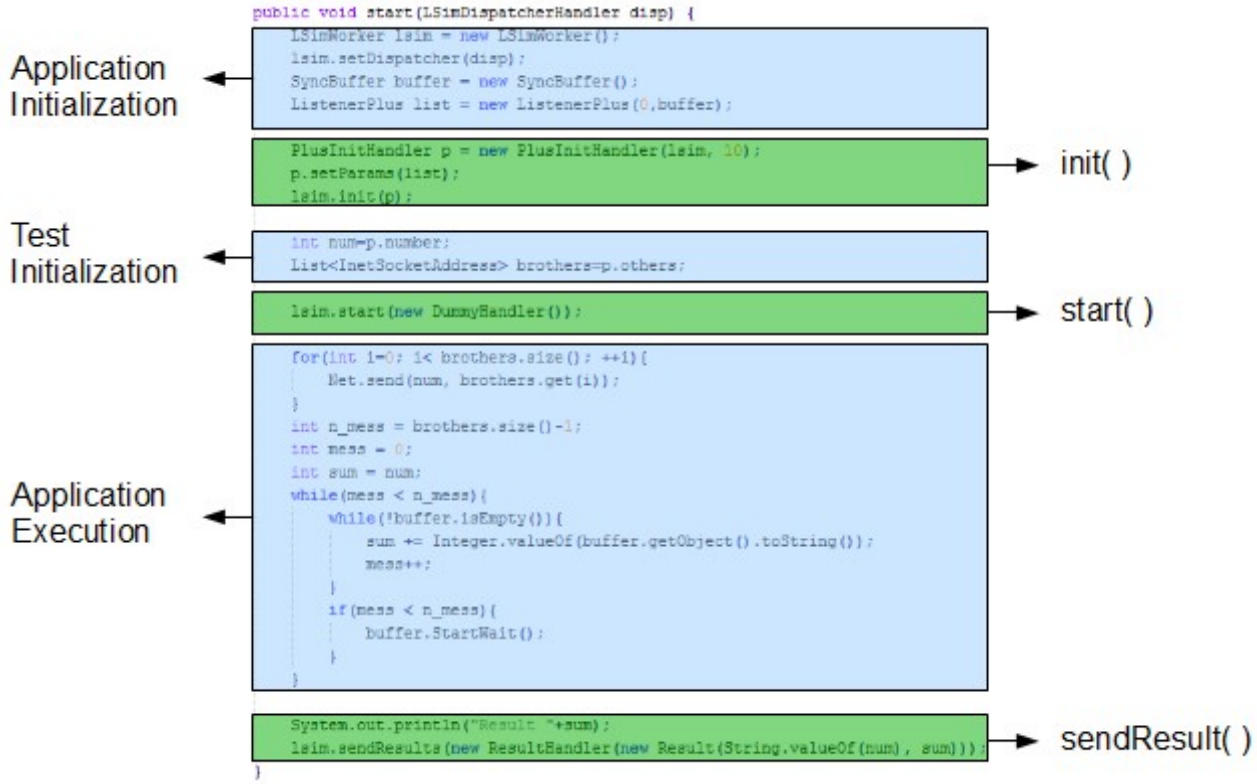


Figura 11: Exemple NodeSum amb LSim

Com es pot veure simplement s'ha afegit les funcions de coordinació de LSim indicant en cadascun quin Handler s'ha d'utilitzar.

A continuació es facilita una plantilla per a l'implementació del Worker, la qual es pot trobar amb més detall a la classe *lsim.worker.WorkerTemplate*:

```java
/**
 * It is a template that how you have to use the LSim library to coordinate
your experiments.
 * In this case, we will explain all phases and how you can use it.
 * We will use ours handlers implementations, but you can implement yours to do
your work in every phase.
 * You can copy this code and use it for your implementations, it is useless,
it's only a template
 * @author Esteve Verdura Gelmà
 * @see lsim.application.ApplicationManager for more information about
application implementations
 */
public class WorkerTemplate implements ApplicationManager{

private boolean alive;

public WorkerTemplate(){
        alive=true;
}

/**
 * In this function is where you have to put your code as a main.
 * @param Dispatcher handler, needed to communicate with dispatcher
 * @see lsim.applicaiton.handler.Handler for more informationabout handlers
 * @see lsim.application.handler.InitHandler for more information about initial
handler
 * @see lsim.application.handler.StartHandler for more information about start
handler
 * @see lsim.application.handler.ResultHandler for more information about
result handler
 */
        public void start(LSimDispatcherHandler disp) {
        //This two code lines are to create a worker instance and set the
dispatcher to LSim. This two lines will be always needed in every LSim element.
        LSimWorker lsim=new LSimWorker();
        lsim.setDispatcher(disp);

        //This is our initial handler implementation. The parameters are LSim
instance and experiment time. It is so important make your own initial handler,
to assure that all initialization process is doing there.
        InitHandler init=new InitHandler(lsim,30);
        //Starts initial phase, wait till coordinator will send the
parameters for worker
        lsim.init(init);

        //Parameters are in a list of objects, where you need to cast every
parameter to the type that you know that they are.
        List<Object> param=init.getParameters();

        //In this kind of behind comments we will put like how  it has to
work. Our implementation will call him as Node.
        //We recommend do all this initialization process in the initial
handler, because in the return statement you can send information to other
workers that it will receive it on start phase. It is more useful for
coordinator or evaluator elements.
        /*
         * 1  int param1 = Integer.valueOf(param.get(0));
         * 2  String param2 = param.get(1).toString();
         * 3  String param3 = param.get(2).toString();
         * 4  InternalClass param4 = (InternalClass)param.get(3);
         * 5  Node node = new Node(param1, param2, param3, param4);
```

```
           * 6  node.initProcess();
           */

          //The StartHandler is a typical implementation of start handler
process. You can use it for a start phase on your implementation.
          StartHandler start = new StartHandler();

          //Starts start phase. Here will wait till coordinator will send start
message.
          lsim.start(start);
          //You can receive workers initial replies in this phase as a
List<Object> where you know what is on this object.
          List<Object> wreplies=start.getWorkersReplies();
          //In this part is where you have to execute your code as you thing at
first time. Here an example where function processNode has all node's logic.
          /*
           * 7  node.setNodesInformation(wreplies);
           * 8  node.processNode();
           */

          //It will be the result of this worker, it can be any class that you
know that your evaluator will know.
          Object result=null;
          /*
           * 9  result = node.getResult();
           */
          //Starts send result phase, and then it will finish the worker
execution.
          //ResultHandler is a special handler for the workers to send results.
You can use it.
          lsim.sendResults(new ResultHandler(result));

          alive = false;

          }

}
```

Text 9: Plantilla de Worker

10.1.2.2 Coordinator

La LSim Library ofereix un Coordinator que serveix pels casos que estiguin tots els paràmetres a l'especificació i no s'hagi de parametritzar res en aquest. A continuació es mostra la classe *lsim.coordinator.StarndartCoordinator* la qual implementa aquest Coordinator:

```java
/**
 * Standard coordinator implementation. In case that you specified all in XML
and only have a typical coordination phases (init, start and sendResult), it is
the coordinator implementation.
 * It is a typical coordinator implementations, you can use it as a template
for your next implementations.
 * It also is useful to check how can implements an LSim element.
 * @author Esteve Verdura Gelmà
 *
 */
public class StandardCoordinator implements ApplicationManager{

    private boolean alive;

          public StandardCoordinator(){
                    alive=true;
          }

          /**
           * In this function is where you have to put you code, as a main.
           * @param Dispatcher Handler, needed to communicate with the
dispatcher in LSim process
           */
          public void start(LSimDispatcherHandler disp) {
                    LSimCoordinator lsim=new LSimCoordinator();
                    lsim.setDispatcher(disp);

                    InitHandler init=new InitHandler(lsim,30);
                    lsim.init(init);

                    DummyHandler hand=new DummyHandler();
                    lsim.start(hand);

                    lsim.sendResults(hand);
                    alive=false;

}

}
```

Text 10: Exemple d'implementació de Coordinator

Si es vol implementar un mateix el seu Coordinator, ja que és necessari tenir més aspectes en compte, es pot seguir aquest esquema per obtenir el seu resultat.

10.1.2.3 Evaluator

La LSim Library ofereix un Evaluator ja implementat, el qual només s'ha d'especificar al fitxer de propietats quin Handler realitzarà l'avaluació dels resultats. En aquest fitxer també s'ha d'especificar l'implementació de *StorageManager* que ha d'utilitzar l'Evaluator.

```
/**
 * Evaluator implementation where the user only needs to specify the Handler
that will evaluate the results
 * @author Esteve Verdura Gelmà
 *
 */
public class StandartEvaluator implements ApplicationManager {

          public StandartEvaluator(){

          }

          public void start(LSimDispatcherHandler disp) {

          LSimEvaluator lsim=new LSimEvaluator();

Properties prop=PropertiesReader.getProperties("application.properties");
          Handler result=null;
          try{

result=(Handler)Class.forName(prop.getProperty("handler")).newInstance();
          }catch(Exception e){
                    result=new EvaluatorHandler();
          }
          InitHandler init=new InitHandler(lsim,60);
          DummyHandler start=new DummyHandler();

          lsim.init(init);
          lsim.start(start);
          lsim.sendResults(result);
}

}
```

Text 11: Implementació d'un Evaluator generic

En cas de tenir una avaluació més complexa, sempre és pot utilitzar aquesta classe com a base per a crear un Evaluator.

A continuació es mostra el Handler d'avaluació del NodeSum. El que fa en aquest cas és comprovar que el valor que ha retornat cada node és el mateix. En cas de que tots siguin iguals retorna que el test és vàlid, d'altre manera retorna test invàlid:

```
public Object execute(Object obj) {
          //Get results from ResultSet
          ResultSet results = (ResultSet)obj;
          List<Object> sums = results.getAllResults();

          //Evaluate results
          int r=((Result)sums.get(0)).getVal();
          for(int i=1;i<sums.size();++i){
                    if(((Result)sums.get(i)).getVal()!=r){
                              return new TestResult(false);
                    }
          }
          return new TestResult(true);
}
```

Text 12: Funció execute del Handler d'avaluació de NodeSum

Per més informació dels mètodes per obtenir els resultats a partir del *ResultSet,* consultar l'API de LSim que s'oferirà a la web.

10.1.3 Preparar l'experiment

Per preparar l'experiment primer s'ha de preparar els elements a executar i després preparar l'especificació.

10.1.3.1 Preparació del elements

S'ha de preparar cada un dels elements per a ser executats des de el Dispatcher. El primer que s'ha de fer és preparar en un directori tot el necessari per executar l'element. Aquest directori anirà distribuït per l'entorn on s'hagi de fer les proves. El directori ha de contenir:

- Fitxers binaris de l'aplicació
- Dependències pròpies de l'aplicació
- LSim Library juntament amb les seves dependències
- Llibreria del Dispatcher juntament amb les seves dependències (es facilita en el CD)

Aquest fitxers es poden gestionar de la forma que es vulgui a dins el directori, sempre i quant després totes les referencies siguin relatives i coherents amb l'ubicació que tenen a dins el fitxer.

Actualment depenent de l'entorn d'execució s'ha de distribuir aquest directori a mà. En el cas d'utilitzar PlanetLab, aquest té la comanda multicopy que ofereix el CodePloy per a Linux. Aquesta et permet fer la distribució del directori de forma automàtica a totes les màquines de PlanetLab disponibles per a fer les proves. Més endavant s'oferirà la gestió de fitxers remotament, però actualment LSim no disposa d'aquesta opció.

El Dispatcher permet executar scripts i controlar el procés que ha executat en el script. En el cas de LSim, tots els scripts seguiran el mateix esquema:

```
-java -cp path_sources/*:path_dispatcher/dispatcher.jar
dispatcher.application.ApplicationMain
```

On el path_sources és el directori on tens les dependències de l'aplicació (llibreries, fitxers binaris, LSim Library, etc), el path_dispatcher on hi ha la llibreria del Dispatcher i sempre s'executa la classe *ApplicationMain* del Dispatcher, ja que aquesta és la que permet la comunicació entre el Dispatcher i l'entorn d'execució de l'aplicació.

A continuació es mostra el script per a NodeSum:

```
java -cp lib/*:dlib/*:dispatcher.jar dispatcher.application.ApplicationMain
```

Aquest script ha d'anar a dins el directori.

Finalment s'ha d'especificar al fitxer application.properties quina és la classe que implementa l'interfície *ApplicationManager* de la LSim Library. A continuació es mostra el fitxer del NodeSum per a un Worker:

```
applicationCode=sum.network.node.NodeSum
storage=sum.network.evaluation.Storage
handler=sum.network.evaluation.SumEvaluation
```

Text 13: application.properties del Worker de NodeSum

Al camp applicationCode és on s'ha assignat la classe que executarà el rol de Worker.

Els camps de storage i handler són necessaris només en el cas de l'Evaluator, per indicar quin és el *StorageManager* per a guardar els resultats i el Handler que avalua el test en cas d'utilitzar l'Evaluator que ofereix LSim.

10.1.3.2 Especificació

Per saber quins camps hi ha a l'especificació es pot mirar l'apartat 5.2.1.1 Especificació on es descriuen cadascun dels camps. El que s'explicarà en aquest apartat és com referenciar el codi de l'aplicació i es mostrarà l'exemple de NodeSum.

La referencia del codi té el següent format:

lsim.LSimDispatcherHandler::directory_path::script_name.sh

- El lsim.LSimDispatcherHandler és la part que es comunica amb el Dispatcher, i per tant s'ha de deixar sempre igual.
- El directory_path s'ha d'indicar el directori, pensant en l'ubicació en la màquina remota, que conté tot el referent a l'aplicació.
- El script_name.sh ha de contenir el nom del script que s'ha generat per executar l'aplicació. S'ha d'especificar el path relatiu del directori on es troba si aquest no està a l'arrel d'aquest.

A continuació es mostra l'especificació per a executar una prova de NodeSum sobre LSim:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Specification>

         <experimentId>NodeSumNetwork</experimentId>

         <Coordinators>
                  <num>1</num>


<codeRef>lsim.LSimDispatcherHandler::~/Elements/coordinatorElements::nodeSum.sh
</codeRef>
                  <timeInit>2</timeInit>
                  <timeEval>5</timeEval>
         </Coordinators>
         <Evaluators>
                  <Evaluator>


<codeRef>lsim.LSimDispatcherHandler::~/Elements/evaluatorElements::nodeSum.sh</
codeRef>
                           <idTest>1</idTest>
                           <Node>
                                    <idNode>SumNonFail</idNode>
                                    <nMess>20</nMess>
                           </Node>
                           <Node>
                                    <idNode>SumFail</idNode>
                                    <nMess>20</nMess>
                           </Node>
                           <!-- Address to send results -->
                           <param>sd2.uoc.edu</param>
                           <param>9102</param>
                  </Evaluator>
         </Evaluators>
         <Workers>
                  <Worker>
                           <num>20</num>
                           <idWorker>SumNonFail</idWorker>


<codeRef>lsim.LSimDispatcherHandler::~/Elements/SumNonFailW::ackNodes.sh</codeR
ef>
                           <!-- Seed to generate numbers -->
                           <param>30</param>
                           <!-- Address to get brothers -->
                           <param>sd2.uoc.edu</param>
                           <param>9101</param>
                  </Worker>
                  <Worker>
                           <num>20</num>
                           <idWorker>SumFail</idWorker>


<codeRef>lsim.LSimDispatcherHandler::~/Elements/SumFailW::ackNodes.sh</codeRef>
                           <!-- Seed to generate numbers -->
                           <param>30</param>
                           <!-- Address to get brothers -->
                           <param>sd2.uoc.edu</param>
                           <param>9101</param>
                  </Worker>
         </Workers>
```

```
</Specification>
```

Text 14: Especificació de NodeSum

10.1.4 Llançar l'experiment

Per a llançar experiments el que s'ha de fer és enviar un missatge a qualsevol Launcher que estigui funcionant. Per tant a priori s'ha de conèixer l'adreça del Launcher on es vulgui llançar l'experiment. Cada Launcher tindrà assignat un entorn diferent on executar les proves, per tant simplement canviant l'adreça del Launcher es pot canviar l'entorn on provar l'aplicació.

El Launcher ofereix la classe *InitMessage* la qual et permet llançar experiments. Aquest és un missatge d'inicialització d'experiment que s'envia al Launcher el qual es tingui l'adreça. A continuació es mostra un exemple d'inicialització d'experiment, on es poden afegir paràmetres fora de l'especificació:

```
/**
 * Example how to launch an experiment on Launcher
 * @author Esteve Verdura Gelmà
 *
 */
public class LaunchExperiment {

public static void main(String args[])throws Exception{

          InetSocketAddress address = getLauncherAddress(args[0]);
          String specification = readSpecificationFile(args[1]);

          /**
           * @param Experiment ID
           * @param Experiment specification
           * @param Launcher Storage, in this case doesn't need it, because
LSim Storage will store result
           */
          InitMessage init=new InitMessage("SumNode",specification,null);

          //Add dinamic parameters example
          init.addParam(getNumberList(args[2]));
          init.addParam("LSim Store");

          //Launch the experiment to the specified Launcher
          new Net().sendMessage(address, init);

}


}
```

Text 15: Exemple de llançador d'experiments

Com es pot veure, implementar-se un mateix un llançador d'experiments com aquest no té molt secret.