UOC Universitat Oberta de Catalunya

NexTReT

Solució de monitorització d'experiència d'usuari en entorns web amb Selenium

PROJECTE FINAL DE CARRERA

Estudiant: Xavier Font Erruz
Enginyeria en informàtica
Consultor: Oscar Escudero Sánchez

1 Resum executiu

L'objectiu d'aquest PFC és desenvolupar una solució complerta per tal de monitoritzar l'experiència d'usuari en entorns Web mitjançant transacció sintètica. Per tal d'aconseguir aquest objectiu s'utilitzaran tècniques d'automatització de navegadors. Es vol que aquesta solució pugui ser explotada com un SaaS (Software As A Service), per tant ha de permetre als clients de la solució:

- Obtenir un sistema fiable de monitorització d'experiència d'usuari en entorn Web amb transacció sintètica
- La solució ha de poder executar test web en diferents navegadors, principalment amb els més utilitzats (Chrome, FireFox i Internet Explorer)
- Els tests han de poder-se executar en diferents sistemes operatius
- La solució ha d'oferir algun tipus d'eina per a l'autogestió del servei. Almenys que els clients puguin realitzar les tasques bàsiques, deixant al suport expert associat les tasques més complexes
- Ha de recollir mètriques útils per al client, com poden ser el temps de resposta de cada una de les accions i la disponibilitat
- Proporcionar dades per a diagnosticar la font de desviacions en el temps de resposta així com enfront d'errors
- Totes les mètriques recollides han de poder ser explotades pel client, mitjançant algun tipus d'aplicació web

La solució que presentem compleix tots aquests objectius.

1 Resum executiu 2
2 Introducció 9
2.1 Descripció del projecte 9
2.2 Descripció del PFC 10
2.3 Pla de treball 13
3 Estudis previs 14
3.1 Patrons 14
3.1.1 Que són? 14
3.1.2 Patró de disseny Factory Method 15
3.1.3 Patró de disseny Singleton (instancia única) 15
3.1.4 Patró de disseny Wrapping (Adapter) 16
3.1.5 Patró Model-Vista-Controlador (MVC) 16
3.1.6 Patró Page Object 17
3.2 Comparativa de solucions d'automatització de navegadors 18
3.2.1 Selenium2/WebDriver 18
3.2.2 WatiN 20
3.2.3 Sahi 21
3.2.4 Telerik Testing Framework 22
3.2.5 Resum 0
3.2.7 Elecció final 0
3.3 Frameworks i APIs utilitzats 1
3.3.1 Selenium 1
3.3.2 Hibernate 1
3.3.3 Bootstrap 3
3.3.4 REST 3
3.3.4.1 REST vs SOAP 5
4 Anàlisis i Disseny de l'Agent MEU 6
4.1 Requeriments 6
4.2 Dependències 7
4.3 Estructura i jerarquia de paquets 8
4.4 Configuració 9
4.5 Disseny de l'aplicació 9
4.5.1 Patró singleton 10
4.5.2 ClassLoader 11
4.5.3 Classes customitzades 13

4.5.4 Fiddler 15
4.5.5 Internacionalització 17
4.5.6 Diagrama de classes de l'Agent MEU 17
4.5.7 Diagrama de seqüencies de l'Agent MEU 19
4.5.8 Diagrama de flux de l'Agent MEU 20
5 Anàlisis i disseny del WebService 21
5.1 Requeriments 21
5.2 Dependències 21
5.3 Estructura i jerarquia de paquets 22
5.4 Configuració 23
5.5 Disseny 24
5.5.1 Diagrama de classes del WebService 24
5.5.2 Diagrama de seqüencia WebService 26
5.5.3 Hibernate 27
5.5.4 REST WebService 27
6 Anàlisis i disseny de la consola d'administració 29
6.1 Requeriments 29
6.2 Dependències 29
6.3 Estructura i jerarquia de paquets 30
6.4 Configuració 31
6.5 Disseny 32
6.5.1 Disseny responsive 32
6.5.2 Hibernate 32
6.5.3 Diagrama de classes 32
6.5.4 Diagrama de casos d'ús 34
6.5.5 Diagrama de seqüències 35
6.5.5.1 Login 35
6.5.6 Nou Robot 36
6.5.6.1 Esborrar robot 37
6.5.6.2 Modificar robot 38
6.5.7 Nou calendari d'un test 39
6.5.8 Editar calendari d'un test 40
6.5.9 Esborrar calendari de test 41
6.5.10 Nou test 42
6.5.11 Editar test 43
6.5.12 Esborrar test 44

6.5.13 Nou usuari 45
6.5.14 Editar usuari 46
6.5.15 Esborrar usuari 47
7 Base de dades 48
7.1 Disseny de les taules 49
8 Conclusions 50
9 Bibliografia 51
11 Annexos 52
11.1 Manual d'instal·lació 52
11.1.1 Agent 52
11.1.1.1 Instal·lació 52
11.1.1.2 Configuració 54
11.1.2 WebService 55
11.1.3 Portal d'administració 56
11.1.4 Base de dades 56
11.2 Manual d'usuari 59
11.2.1 Agent 59
11.2.2 Portal d'administració 60
11.2.2.1 Robots 61
11.2.2.2 Test 65
11.2.3.2 Usuaris 72

Índex de figures

FIGURA 1 - DIAGRAMA DE GANTT 13
FIGURA 2 - FUNCIONAMENT PATRÓ DE DISSENY FACTORY METHOD 15
FIGURA 3 - PATRÓ DE DISSENY WRAPPING 16
FIGURA 4 - PATRÓ MVC 16
FIGURA 5 - ARQUITECTURA WEBDRIVER 18
FIGURA 6 - INTERACCIÓ ENTRE ELS MÒDULS DE WEBDRIVER 19
FIGURA 7 - WATIN INTERACCIÓ AMB DRIVERS 20
FIGURA 8 - SAHI INJECCIÓ DE JAVASCRIPT 21
FIGURA 9 - TELERIK TESTING FRAMEWORK I TEST STUDIO 22
FIGURA 10 - ARQUITECTURA GENERAL HIBERNATE 1
FIGURA 11 - ARQUITECTURA HIBERNATE (DETALL) 2
FIGURA 12 - FUNCIONAMENT ARQUITECTURA REST 4
FIGURA 13 - DIAGRAMA DE DEPENDÈNCIES AGENT MEU 7
FIGURA 14 - DIAGRAMA DE PAQUETS AGENT 8
FIGURA 15 - FITXER DE CONFIGURACIÓ AGENT 9
FIGURA 16 - ESTRUCTURA PATRÓ SINGLETON 10
FIGURA 17 - RUNTEST JUNIT CORE 11
FIGURA 18 - ESTRUCTURA DE CARREGADORS DE CLASSE 11
FIGURA 19 RELACIÓ ENTRE CLASSLOADERS 12
FIGURA 20 - DIAGRAMA DE CLASSES CUSTOMITZADES 14
FIGURA 21 - HAR FILE 15
FIGURA 22 - CLASSE FIDDLER 16
FIGURA 23 - DIAGRAMA DE CLASSES AGENT MEU 18
FIGURA 24 - DIAGRAMA DE SEQÜENCIES AGENT MEU 19
FIGURA 25 - DIAGRAMA DE FLUX AGENT 20
FIGURA 26 - DIAGRAMA DE DEPENDÈNCIES WEBSERVICE 21
FIGURA 27 - DIAGRAMA DE PAQUETS WEBSERVICE 22
FIGURA 28 - DIAGRAMA DE CLASSES WEBSERVICE 25
FIGURA 29 - DIAGRAMA DE SEQÜENCIA WEBSERVICE 26
FIGURA 30 - DIAGRAMA DE DEPENDÈNCIES CONSOLA D'ADMINISTRACIÓ 29
FIGURA 31 - DIAGRAMA DE PAQUETS ADMINISTRACIÓ 30
FIGURA 32 - DIAGRAMA DE CLASSES CONSOLA D'ADMINISTRACIÓ 33
FIGURA 33 - DIAGRAMA DE CASOS D'ÚS CONSOLA D'ADMINISTRACIÓ 34
FIGURA 34 - DIAGRAMA DE SEQÜENCIES LOGIN 35
FIGURA 35 - DIAGRAMA DE SEQÜENCIES NOU ROBOT 36
FIGURA 36 - DIAGRAMA DE SEQÜENCIES ESBORRAR ROBOT 37
FIGURA 37 - DIAGRAMA DE SEQÜENCIA MODIFICAR ROBOT 38
FIGURA 38 - DIAGRAMA DE SEQÜENCIA NOU CALENDARI D'UN TEST 39
FIGURA 39 DIAGRAMA DE SEQÜENCIA EDITAR CALENDARI DE TEST 40
FIGURA 40 - DIAGRAMA DE SEQÜENCIA ESBORRAR CALENDARI DE TEST 41
FIGURA 41 - DIAGRAMA DE SEQÜENCIA NOU TEST 42
FIGURA 42 - DIAGRAMA DE SEQÜENCIA EDITAR TEST 43
FIGURA 43 - DIAGRAMA DE SEQÜENCIA ESBORRAR TEST 44
FIGURA 44 DIAGRAMA DE SEQÜENCIA NOU USUARI 45
FIGURA 45 - DIAGRAMA DE SEQÜENCIA EDITAR USUARI 46
FIGURA 46 - DIAGRAMA DE SEQÜENCIA ESBORRAR USUARI 47
FIGURA 47 - DIAGRAMA DE BASE DE DADES 49
FIGURA 48 AGENT - INSTAL·LACIÓ 1 52
FIGURA 49 AGENT - INSTAL·LACIÓ 2 52
FIGURA 50 FIDDLER - INSTAL·LACIÓ 53
FIGURA 51 INSTAL·LACIÓ BBDD 1 56

FIGURA 52 INSTAL·LACIÓ BBDD 2 57
FIGURA 53 INSTAL·LACIÓ BBDD 3 57
FIGURA 54 INSTAL·LACIÓ BBDD 4 57
FIGURA 55 INSTAL·LACIÓ BBDD 5 57
FIGURA 56 INSTAL·LACIÓ BBDD 6 58
FIGURA 57 INSTAL·LACIÓ BBDD 7 58
FIGURA 58 INTERFÍCIE GRÀFICA AGENT 59
FIGURA 59 HOME PORTAL D'ADMINISTRACIÓ 60
FIGURA 60 MENÚ PORTAL D'ADMINISTRACIÓ 60
FIGURA 61 NOU ROBOT 61
FIGURA 62 ESBORRAR ROBOT 61
FIGURA 63 SELECCIÓ DE ROBOT 62
FIGURA 64 EDICIÓ DE ROBOT 62
FIGURA 65 PANTALLA D'ADMINISTRACIÓ DE ROBOTS 63
FIGURA 66 ASSIGNACIÓ DE TEST A ROBOT 63
FIGURA 67 DETALL D'UN ROBOT 64
FIGURA 68 OCUPACIÓ D'UN ROBOT 64
FIGURA 69 ADMINISTRACIÓ D'USUARIS ASSOCIATS A UN ROBOT 65
FIGURA 70 FORMULARI D'ALTA D'UN TEST 66
FIGURA 71 PANTALLA DE PUJADA DE TEST 66
FIGURA 72 PANTALLA PER ESBORRAR TESTS 67
FIGURA 73 SELECCIÓ DE TEST A EDITAR 67
FIGURA 74 SELECCIÓ DE TEST A ADMINISTRAR 68
FIGURA 75 DETALL D'UN TEST 68
FIGURA 76 DETALL DEL CALENDARI D'UN TEST 69
FIGURA 77 PANTALLA DE CREACIÓ DE CALENDARI 69
FIGURA 78 DETALL D'UN CALENDARI ASSOCIAT A UN TEST 70
FIGURA 79 EDICIÓ D'UN CALENDARI 70
FIGURA 80 VISTA D'USUARIS ASSIGNATS A UN TEST 71
FIGURA 81 FORMULARI D'ALTA D'USUARI 72
FIGURA 82 PANTALLA D'ESBORRAR USUARI 72
FIGURA 83 SELECCIÓ D'USUARI 73
FIGURA 84 EDICIÓ D'USUARI 73

Índex de taules

TAULA 1 - RELACIÓ DELS MÈTODES HTTP AMB LES OPERACIONS QUE REALITZA 4
TAULA 2 - PAQUETS DE L'AGENT MEU 8
TAULA 3 - ESTRUCTURA DE PAQUETS WEBSERVICE 22
TAULA 4 - WEBSERVICE: HIBERNATE.CFG.XML 23
TAULA 5 - ESTRUCTURA DE PAQUETS ADMINISTRACIÓ 30
TAULA 6 - CONSOLA D'ADMINISTRACIÓ HIBERNATE.CFG.XML 31

2 Introducció

2.1 Descripció del projecte

Fins fa poc l'ús de webs per a realitzar gran quantitat d'accions quotidianes: compres, tràmits amb administració pública, reserves d'hotels, administració de comptes bancaris, etc. S'ha convertit en quelcom habitual a la societat. Amb la massificació de l'ús de dispositius mòbils i tabletes, ja no depenem d'un ordinador per a realitzar totes aquestes accions sinó que es poden realitzar des de qualsevol lloc. Les empreses fa temps que són conscients d'aquest nou gran mercat per a vendre els seus productes, i cada cop la competència és més gran. Per això, totes aquestes empreses que ofereixen als seus clients productes o serveis, han d'oferir un servei perfecte, ja que si el client no queda satisfet, no ja amb el producte en si, sinó amb la plataforma web on s'ofereix (lentitud, indisponiblitat, etc.), aquest marxarà a utilitzar el servei ofert per la competència. Per aquest motiu sorgeix la necessitat, per part de les empreses, de conèixer en tot moment l'estat del seus serveis. Fins ara era suficient en monitoritzar l'estat dels servidors, però amb una competència tant acarnissada, s'ha d'anar més enllà, ja que tot i que el servidor estigui disponible, pot l'empresa saber el rendiment que s'està oferint als seus clients? Pot l'aplicació web tenir una indisponiblitat tot i que els servidors no tinguin cap problema?

Per a resoldre aquest problema, i poder conèixer des del punt de vista del client, el funcionament dels serveis oferts existeixen dues solucions:

- Real user monitoring: s'insereix un petit codi a l'aplicació web del client (equivalent a google analytics) que s'executarà al navegador del client, recollint informació de temps de resposta i disponibilitat que experimenta el client. Tota aquesta informació es recollida i enviada a un servidor on s'emmagatzema i és explotada posteriorment. Aquesta solució permet recollir informació de tots els clients que tenim. El problema principal és que si no hi ha clients, no podrem detectar possibles problemes al no tenir mostres, per tant ja no ens podrem anticipar als problemes i minimitzar-ne l'afectació.
- Monitorització amb transacció sintètica: es tracta d'una mesura proactiva que simula usuaris reals, realitzant accions predefinides que simulen les operatives més habituals dels clients o bé aquelles més critiques. Aquestes mesures són transparents i no intrusives. En aquest cas al no dependre d'elements externs per a generar mostres, podem assegurar-nos que en tot moment podrem conèixer el rendiment i disponibilitat de les nostres aplicacions de forma objectiva.

L'objectiu d'aquest PFC és el de desenvolupar una solució que permeti realitzar monitorització d'experiència d'usuari en entorn web mitjançat transacció sintètica multi navegador i multi plataforma.

2.2 Descripció del PFC

Com comentàvem anteriorment l'objectiu d'aquest PFC és el de desenvolupar una solució que permeti realitzar monitorització d'experiència d'usuari en entorn web mitjançat transacció sintètica multinavegador i multiplataforma. Al tractar-se d'un projecte d'empresa aquesta solució ha de poder ser explotada com una solució Software As A Service (SaaS a partir d'ara), als diferents clients objectiu, com poden ser: administració pública, banca, assegurances, comerç electrònic, bàsicament tot aquella empresa que una part important o la totalitat de les seves fons d'ingrés provinguin del món web.

Per tal de poder desenvolupar aquest servei, necessitarem diferents mòduls, on cada un d'ells realitzarà una tasca concreta. A continuació detallarem cada un d'aquests mòduls:

- **Agent:** la seva principal tasca és la de realitzar l'execució dels circuits associats a un robot. Entenem per robot aquell sistema on correra el software de monitorització (un PC tant físic com virtual). A cada robot s'associa un o més circuits/tests. A l'iniciar-se l'agent el primer que farà és llegir el seu fitxer de configuració on se li indicarà quin robot és, mitjançant una petició al WebService preguntarà quines són els circuits que ha d'executar aquell robot. Un cop tingui aquesta informació l'agent començarà a executar els circuits assignats segons la planificació especificada, recollint per a cada execució un seguit de mètriques (temps de resposta, disponibilitat, etc.) Aquesta informació, un cop recollida, serà enviada al WebService per a que aquest s'encarregui d'emmagatzemar-ho a la Base de dades central. L'objectiu del projecte no és crear un motor de creació de de circuits, sinó utilitzant eina de tercers que realitzi aquesta tasca. Per aquest motiu s'ha decidit utilitzar el framework Selenium. Selenium és una eina d'automatització d'aplicacions web per a testing. És una eina molt coneguda i utilitzada entre els equips de QA. La idea és utilitzar la potencia de Selenium per a automatitzar accions sobre navegadors, no per realitzar testing d'aplicacions web, sinó per a realitzar monitorització d'aplicacions web. Per tant amb Selenium té un plugin per a firefox per a realitzar la gravació dels circuits de forma senzilla. Un cop gravat aquests poden ser exportats en J Unit. Aquests fitxers en J Unit són el circuits que executarà l'agent. L'agent estarà desenvolupat en J2EE tindrà una petita interfície gràfica que es desenvoluparà amb SWT. Com hem comentat l'agent s'ha de comunicar amb un WebService, per tant haurà de tenir una part client per a comunicar-se amb ell. Aquest client es desenvoluparà utilitzant JAX-RS (Java API for RESTful Web Services) amb el framework Jersey, ja que el WebService estarà desenvolupat amb aquesta tecnologia

- **<u>WebService:</u>** La funció del Web Service (WS a partir d'ara) és la de realitzar d'enllaç de comunicació entre l'agent i la Base de dades. Per a que l'agent es comuniqui amb la Base de dades tenim dos opcions o bé aquest accedia directament a la base de dades on s'inclou un component que faci d'intermediari entre els dos. Es descarta la opció de connexió directa amb la base de dades per temes de seguretat. L'agent pot córrer en qualsevol ordinador, això vol dir que aquests poden estar tant dins com fora de la xarxa de la base de dades, realitzar connexions directes a la base de dades des qualsevol localització implicaria publicar la base de dades a internet, això en termes de seguretat és inacceptable. Per tant, al voler poder tenir robots a qualsevol punt geogràfic necessitem un component que sigui accessible des d'internet. S'elegeix fer el WS amb RESTful, ja que WS i client parlen per HTTP, que és accessible des de qualsevol lloc, podem tenir un robot a una empresa per a monitoritzar la seves aplicacions internet, si la comunicació es fes per un protocol diferent al HTTP s'hauria de demanar als encarregats de seguretat informàtica que obrissin accés cap al nostre servidor, com volem que la implantació sigui el més senzilla possible el millor és utilitzar protocol HTTP. Com comentem s'ha escollit implementar el WS amb JAX-RS i el framework Jersey. La comunicació entre client i servidor pot realitzar-se amb diferents hipermitjans, normalment HTML, XML o JSON. Per al desenvolupament del nostre WS hem escollit JSON, al creure que és el que s'ajusta més a les nostres necessitats. Les peticions que realitzarà el client típicament provocaran accessos a la base de dades, ja sigui per a realitzar consultes i retornar informació o bé per emmagatzemar la informació que ens envia el client. Es decideix utilitzar el framework Hibernate per a fer el mapeig objecte-relacional i centralitzar en aquest framework la capa d'accés a la base de dades

- **<u>Base de dades i Data warehouse:</u>** La base de dades amb la que treballarem és SQL Server 2008 R2, el principal motiu és per coneixement d'aquest producte, al ser amb el que treballem diàriament dins del meu món professional, i estar certificat per Microsoft en aquesta solució. Com hem comentat, l'agent generarà mètriques de disponibilitat i temps de resposta de les execucions dels circuits, aquestes dades seran emmagatzemades a la base de dades. Com tenim previsió de tenir un nombre alt de dades, hem de fer un tractament previ de les dades abans de presentar-les, tant per evitar treballar amb gran volum de dades (taules amb molts registres) com per a extreure la informació que realment servirà a l'usuari final. Per això les dades que s'aniran recollint seran historificades periòdicament mitjançant "stored procedures" per tal de crear informació útil per a l'usuari.

- **Reporting:** Per tal d'explotar totes les dades recollides es crearà una aplicació web on el client podrà consultar reports de l'estat i rendiment dels seus serveis. Aquesta eina ha de permetre poder crear reports personalitzats, consultar els errors detectats, etc de forma molt visual i senzilla. Tota aquesta informació que es presentarà a l'eina de reporting es nodrirà amb les dades tractades i emmagatzemades a la base de dades mitjançant la historificació periòdica. Aquesta aplicació ja existeix actualment, la tasca principal serà la d'integrar les noves dades recollides per la nova aplicació en el reporting actual, Aquesta aplicació web ha estat desenvolupada en J2EE, seguint el patró model-vista-controlador utilitzant el framework Spring. La capa de presentació s'ajuda de tecnologies com JQuery i AJAX (per les crides asíncrones). Per a desenvolupar la capa de persistència i accés a dades es va utilitzar el framework Hibernate.

- **Panell d'administració:** per a poder gestionar tot el servei ofert al client necessitem d'una eina que ens permeti, de forma centralitzada, controlar l'estat dels diferents robots, així com realitzar configuracions concretes sobre els diferents elements. Per exemple podrem crear nous robots, pujar nous circuits, crear planificacions d'execució, etc. Aquesta eina ha de permetre tant a client gestionar i configurar els seus recursos assignats, com als administradors del servei poder: crear nous clients, assignar recursos a clients, etc. Resumint fer una administració global del servei. Aquesta serà una aplicació web desenvolupada en J2EE. La capa de presentació es desenvoluparà amb l'ajut del framework Bootstrap i complementant-se amb tecnologies con JQuery i AJAX, HTML5 i CSS3. Per a desenvolupar la capa de persistència i accés a dades es desenvoluparà amb el framework Hibernate.

- **Sistema d'alertes:** Aquest sistema, ja existent actualment, està desenvolupat en .NET, te com a tasca principal analitzar les dades recollides per als diferents robots, en cas de detectar que un circuit ha fallat, enviarà alertes a les persones assignades com a receptores informant de la caiguda del servei. Un cop el sistema detecti que el servei s'ha restablert enviarà un altre notificació a les persones assignades informant del restabliment del servei. Les alertes s'envien mitjançant correu electrònic i/o SMS. També és possible integrar aquest sistema amb el sistema de monitorització del client, s'ha integrat satisfactòriament amb sistemes com HP OpenView o Nagios. La tasca a realitzar serà integrar aquesta eina amb les noves dades generades.

2.3 Pla de treball

A continuació presento el pla de treball a seguir per a la realització del PFC.

Es poden diferenciar quatre fites en aquesta planificació:

1. Fita 1 – Pla de treball: en aquesta primera fase definirem quin és l'objectiu del projecte, així marcar el pla de treball a seguir, presentat en un diagrama de Gantt.
2. Fita 2 – Anàlisis i disseny: en aquesta fase crearem l'esquelet de la nostre solució. Haurem d'identificar clarament les funcionalitats de cada un dels mòduls a desenvolupar per a poder fer un disseny òptim de com ho farem. Com part d'aquest anàlisis i disseny ja s'ha començat a realitzar ho solaparem la fase d'implementació. Hi ha 26 dies hàbils per a aquesta fase, per tal de no sortir-nos de calendari, la part d'implementació que es realitzarà durant aquesta fase es realitzarà en dies festius.
3. Fita 3 – Implementació: entrega d'una solució de monitorització d'experiència d'usuari. S'han comptabilitzat 55 dies hàbils per a la fase d'implementació, que s'hauran de reforçar en dies festius per tal de poder assolir tots els objectius.
4. Fita 4 – Documentació: entrega de la memòria, la presentació. La confecció de la memòria es realitzarà des del primer dia.

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesora
1		**PFC**	**75 días**	**mar 04/03/14**	**mar 17/06/14**	
2		**Pla de treball**	**7 días**	**mar 04/03/14**	**mié 12/03/14**	
3		Documentació prèvia	4 días	mar 04/03/14	vie 07/03/14	
4		Introducció i propòsits	1 día	lun 10/03/14	lun 10/03/14	3
5		Objectius	1 día	mar 11/03/14	mar 11/03/14	4
6		Planificació	1 día	mié 12/03/14	mié 12/03/14	5
7		PAC1	0 días	mié 12/03/14	mié 12/03/14	6
8		**Anàlisis i disseny**	**26 días**	**jue 13/03/14**	**jue 17/04/14**	**2**
9		**Anàlisis**	**9 días**	**jue 13/03/14**	**mar 25/03/14**	
10		Agent	3 días	jue 13/03/14	lun 17/03/14	6
11		Base de dades	1 día	mar 18/03/14	mar 18/03/14	10
12		Consola d'administració	3 días	mié 19/03/14	vie 21/03/14	11
13		WebService	2 días	lun 24/03/14	mar 25/03/14	12
14		**Disseny**	**17 días**	**mié 26/03/14**	**jue 17/04/14**	
15		Agent	5 días	mié 26/03/14	mar 01/04/14	13
16		Base de dades	3 días	mié 02/04/14	vie 04/04/14	15
17		Consola d'administració	5 días	lun 07/04/14	vie 11/04/14	16
18		WebService	4 días	lun 14/04/14	jue 17/04/14	17
19		PAC2	0 días	jue 17/04/14	jue 17/04/14	18
20		**Implementació**	**55 días**	**mar 18/03/14**	**lun 02/06/14**	
21		Creació d'agent	16 días	mar 18/03/14	mar 08/04/14	10
22		WebService	6 días	mié 09/04/14	mié 16/04/14	21
23		Consola d'administració	15 días	jue 17/04/14	mié 07/05/14	22
24		Base de dades	5 días	jue 08/05/14	mié 14/05/14	23
25		Integracions	7 días	jue 15/05/14	vie 23/05/14	24
26		Proves	6 días	lun 26/05/14	lun 02/06/14	25
26		Proves	6 días	lun 26/05/14	lun 02/06/14	25
27		PAC3	0 días	lun 02/06/14	lun 02/06/14	26
28		**Documentació**	**75 días**	**mar 04/03/14**	**mar 17/06/14**	
29		Memòria	75 días	mar 04/03/14	lun 16/06/14	
30		Presentació	10 días	mar 03/06/14	lun 16/06/14	27
31		Entrega final	0 días	mar 17/06/14	mar 17/06/14	30

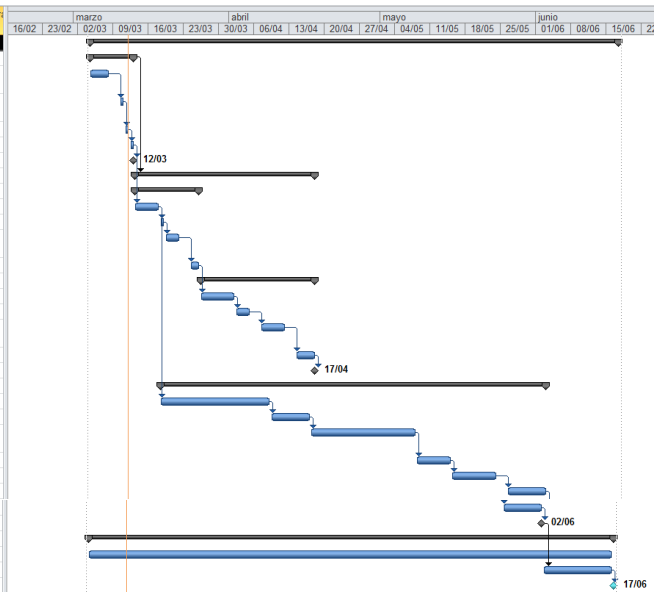


Figura 1 - Diagrama de Gantt

3 Estudis previs

3.1 Patrons

3.1.1 Que són?

En el mon de la enginyeria, concretament en la enginyeria del software, al dissenyar una aplicació es parla dels patrons de disseny a utilitzar, quins seran més òptims per a cada situació, però que són exactament aquests patrons?

Els patrons de disseny és una base de cerca de solucions per a problemes comuns en el desenvolupament de software. És una solució **reutilitzable** a un problema de disseny. Cal remarcar que tot i ser una solució no vol dir que ens generi automàticament el codi font per a l'aplicació que volem dissenyar, sinó que ens proporciona les eines necessàries per a resoldre el problema.

L'objectiu dels patrons de disseny són:

- Proporcionar catàlegs d'elements reutilitzables en el disseny
- Evitar la reiteració en la cerca de solucions a problemes coneguts i resolts anteriorment
- Estandarditzar la forma en que és realitza el disseny
- Crear un vocabulari comú entre els dissenyadors

En aquest PFC, a l'estar desenvolupat amb J2EE, seguirem els patrons de disseny enunciats per Oracle , i que estan dividits en tres capes: presentació, negoci i integració.

3.1.2 Patró de disseny Factory Method

El problema que vol resoldre aquest patró de disseny és quan l'objecte amb el que treballem ha de crear altres objectes i no pot anticipar la classe a la que aquests pertanyen. Es vol delegar la responsabilitat d'especificar la classe dels objectes que s'han de crear a les nostres subclasses.

Per tal de solucionar-ho, el que fa indica aquest patró de disseny, és definir a la classe abstracta Creador (la que crea els objectes), un mètode de crear encarregat de la creació i que anomenarem mètode factoria (Factory method). Proporcionarem a cada subclasse de la classe abstracta Creador, una implementació del mètode factoria que creï un tipus concret d'objecte.

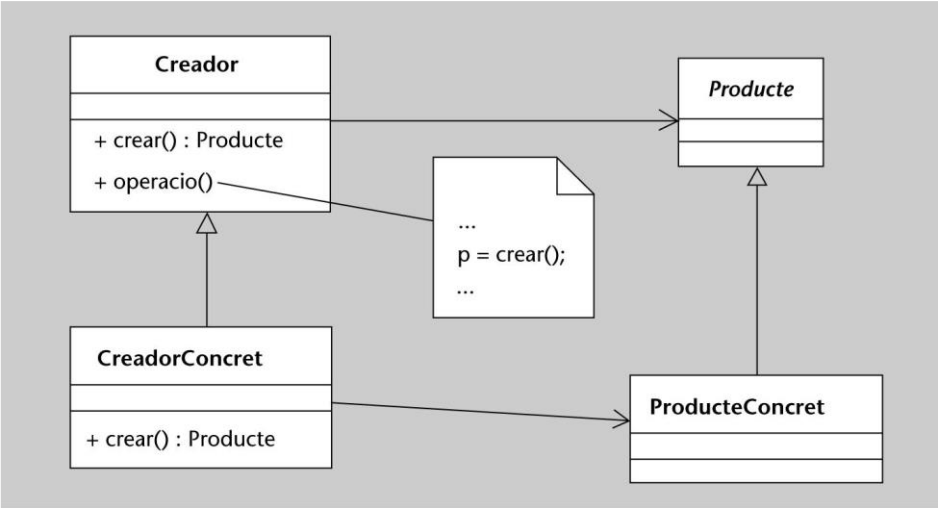


Figura 2 - Funcionament patró de disseny Factory Method

3.1.3 Patró de disseny Singleton (instancia única)

Permet assegura que, d'una determinada classe, només n'hi ha una instància en tot el sistema. La implementació d'una instància singleton, no tindrà operacions de construcció accessibles (públiques) des de fora de la mateixa classe, aconseguint que només pugui ser instanciada per ella mateixa. A fora publicarem una funció per a que ens retorni la instancia de la classe.

3.1.4 Patró de disseny Wrapping (Adapter)

S'utilitza per a transformar una interfície en un altre, de tal forma que una classe que no pot utilitzar la primera, faci ús d'ella a través de la segona.

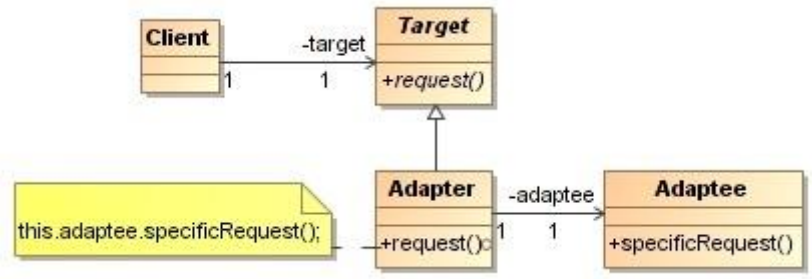


Figura 3 - Patró de disseny Wrapping

3.1.5 Patró Model-Vista-Controlador (MVC)

El patró MVC serveix per a desacoblar la interfície gràfica del nostre sistema de la resta del sistema, el que és el mateix separar les dades de la lògica de negoci de la interfície d'usuari i el mòdul encarregat de gestionar els esdeveniments i les comunicacions.

Per tal de dur-ho a terme MVC defineix la construcció de tres elements diferents:

- **Model:** representació de la informació amb la que opera el sistema, per tant gestiona tots els accessos a aquesta informació, tant les consultes com les actualitzacions. Conté el nucli de la funcionalitat de l'aplicació. No sap res (és independent) del controlador i la vista.
- **Vista:** presenta les dades als usuaris i en recullen les interaccions, per enviar-les als controladors. Pot accedir al model però mai podrà canviar-ne l'estat. Pot ser notificada quan hi ha un canvi a l'estat del model
- **Controlador:** estableixen la correspondència entre les accions de l'usuari i els esdeveniments del sistema. També decideixen quines vistes es mostren als usuaris

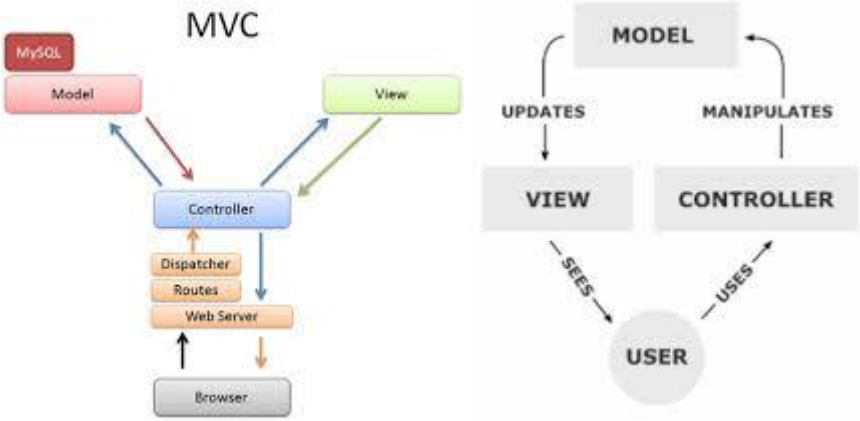


Figura 4 - Patró MVC

3.1.6 Patró Page Object

Es tracta d'un patró de disseny utilitzat per a l'automatització de proves de UI end-to-end. Aquest patró de disseny encapsula un component de la UI en una classe, augmentant entre altres aspectes el manteniment, llegibilitat i centralitzant l'acoblament en un únic punt.

Entrant en més detall, consisteix en crear un objecte per cada un dels elements significatius de la interfície amb la que interactuem. El nom "Page object" ens pot portar a pensar que cada objecte creat ha de representar una pàgina de la nostre aplicació, però si dins d'una pàgina tenim elements visuals que es reutilitzen en d'altres parts, es pot construir un Page Object d'aquest element per a poder reutilitzar-lo. Per exemple el menú de navegació de la pàgina segurament serà comú en totes les pàgines de l'aplicació, es crearà un Page Object per aquest objecte que serà reaprofitat.

3.2 Comparativa de solucions d'automatització de navegadors

3.2.1 Selenium2/WebDriver

Concretament parlarem de Selenium 2, que no és més que la fusió de dos projectes, Selenium Core (conegut inicialment com Selenium 1) i Selenium WebDriver. Es tracta d'un framework per a automatitzar el testing d'aplicacions web, molt conegut i utilitzat pels QA's per realitzar proves unitàries, d'integració i end-to-end, a grans trets realitza proves funcionals des de la perspectiva de l'usuari.

Per a realitzar això es necessiten dos components:

- Selenium IDE: eina per a gravar i executar tests. Concretament és un plugin de Firefox. Permet exportar els test en diferents formats/llenguatges, com pot ser .NET, JUnit, etc.
- WebDriver: també conegut com Selenium2, és una millora de l'API anterior, que utilitza mecanismes d'automatització de navegadors per a executar comandes als navegadors.

El que ens interessa conèixer és com funciona el WebDriver. L'arquitectura del WebDriver està dividida en tres parts:

- Enllaços a nivell de llenguatge: llenguatges en els que podem implementar el codi de Selenium webdriver. Selenium WebDriver ens dona les eines (framework) per interactuar amb el WebDriver.
- API selenium Webdriver: tenim el framework que ens permet "parlar" amb el webdriver, aquest serà l'encarregat de recollir les funcions que s'executen des del llenguatge de programació, processar-les i realitzar les interaccions necessàries amb el navegador, concretament amb el driver de cada navegador.
- Drivers: divers específics per cada un dels navegadors suportats

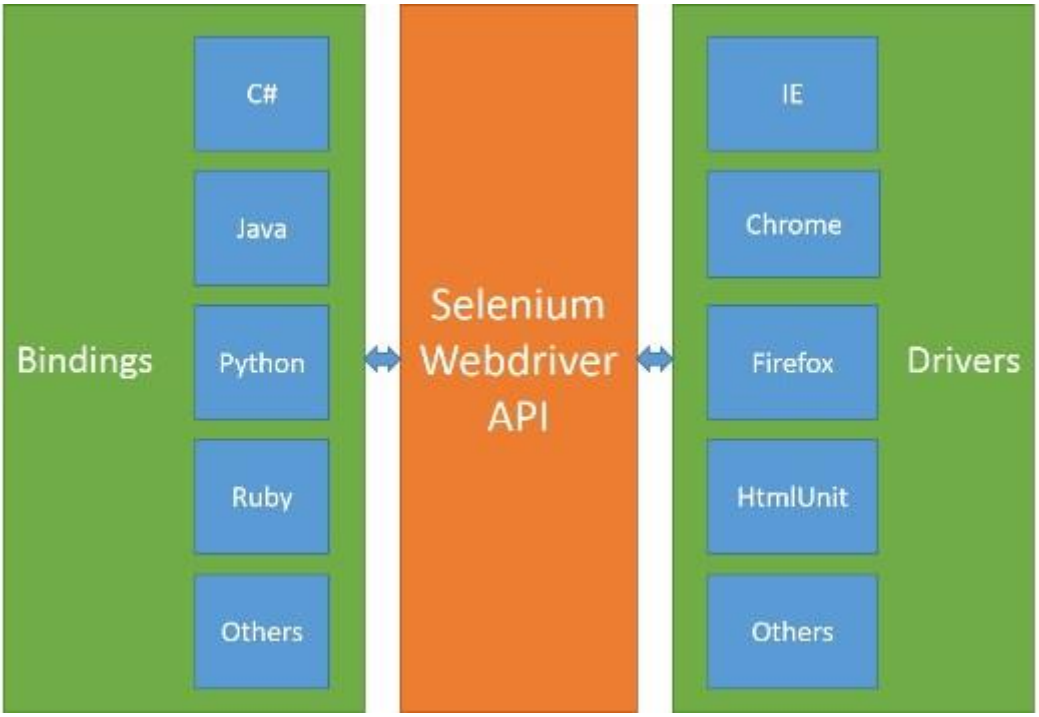


Figura 5 - Arquitectura WebDriver

A grans trets, el funcionament de la relació entre els tres mòduls és: tenim el nostre llenguatge de programació, per exemple Java, que enviarà comandes a través del WebDriver API. A l'altre banda tenim el driver del navegador, que està escoltant les comandes que enviem, les interpretarà i les executarà sobre el navegador, i finalment retornarà el resultat utilitzant el WebDriver API al nostre codi, per tal que puguem processar-lo.

Figura 6 - Interacció entre els mòduls de WebDriver

Selenium utilitza el patró de disseny Page Object.

WebDriver de Selenium2 s'ha presentat es va presentar com a candidat al W3C per a que sigui un estàndard d'internet.

3.2.2 WatiN

Watin o WebApplication Testing in .NET és una eina que s'inspira en el projecte WatiR (Web Application Testing in Ruby). L'objectiu de WatiR es crear una solució de testing d'aplicacions web per a entorns .NET.

WatiR permet:

- Automatitzar la major part dels element HTML fàcilment
- Trobar elements web mitjançant diversos elements (Id, nom, etc)
- Suport natiu pels models "Page" i "Control"
- Suport per testing de pàgina AJAX
- Suport per crear captures de pantalla de les pàgines
- Suport per frames (multi domini) i iframes
- Control sobre diàlegs popup com poden ser alerts, missatges de confirmació ,autenticació, etc.
- Suport per diàlegs HTML (modals i no modals)
- Fàcil d'integrar amb la teva eina de testing
- Suport per IE 6 a 9 i firefox 2 i 3
- Pot ser desenvolupat en qualsevol llenguatge .NET

S'ha inclòs també compatibilitat amb WebDriver (de selenium). Com podem veure a continuació la API de WatiN, utilitza els seus propis divers per a comunicar-se amb els diferents navegadors, o bé pot utilitzar WebDriver, que ja té el seu propi mètode d'automatització de navegadors.

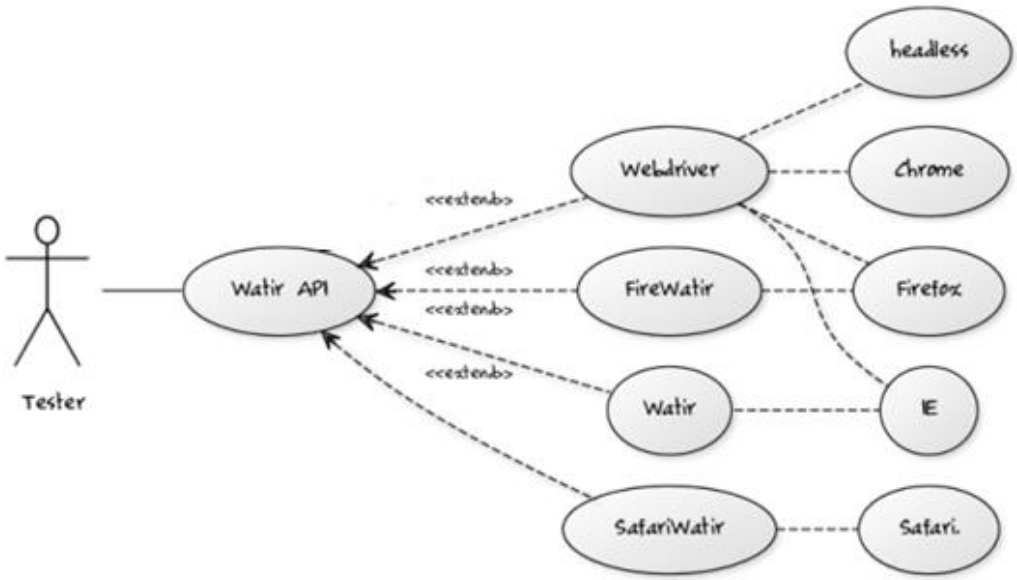


Figura 7 - WatiN interacció amb drivers

3.2.3 Sahi

Sahi és una solució d'automatització de test web. Segons el seu fabricant els seus trets diferencials són:

- Comptar amb un excel·lent gravador i identificador d'objectes web que funciona amb tots els navegadors moderns (IE, Firefox, Chrome, Safari, Opera)
- Reproducció de test en qualsevol navegador d'escriptori i fins i tot en navegadors de terminals mòbils.
- Mecanisme d'identificació d'objectes simple i robust que permet identificar el mateix element en diferents navegadors.
- Espera automàtica a que la pàgina està carregada i a que les accions Ajax han finalitzat. No s'han d'incloure en el 95% dels casos.
- No te necessitat de tenir el focus del navegador
- Pot reproduir múltiples tests simultàniament, reduint el temps de reproducció
- Creació automàtica de reports sense necessitat d'incloure codi extra.

Sahi està desenvolupat en Java, per tant pot funcionar en diferents Sistemes Operatius

Per a funcionar Sahi crea un Proxy i modifica (mentre està funcionant) la configuració dels navegadors, afegint-hi aquest Proxy. Aquest Proxy s'utilitza per a injectar codi JavaScript a les pàgines:

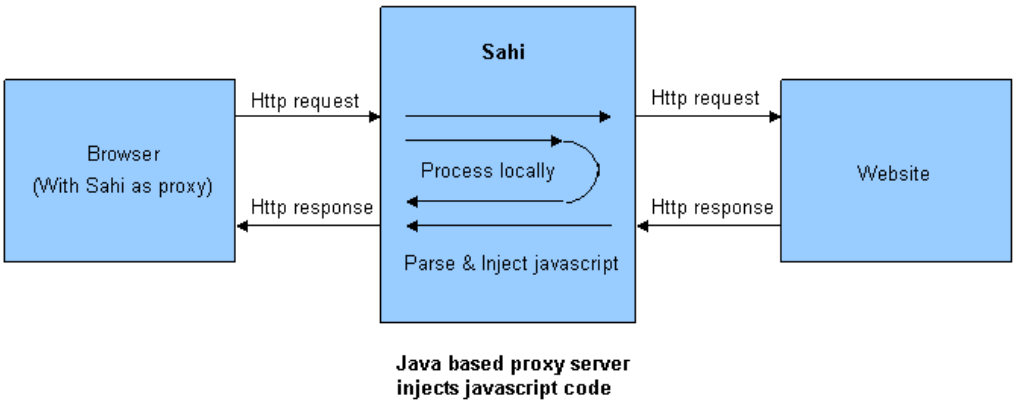


Figura 8 - Sahi injecció de javascript

Un dels problemes més grans que es troben a l'utilitzar eines d'automatització de navegadors és la identificació d'elements. Quan un test falla la identificació d'elements acostuma a ser apuntada com la causant de l'error. Sahi utilitza JavaScript DOM per a identificar els elements. La API de Sahi utilitza diversos atributs DOM d'un element per a identificar-lo.

3.2.4 Telerik Testing Framework

Telerik Testing Framework és una eina propietat de l'empresa Telerik. Consta de dos components diferenciats, Test Studio i Testing Framework.

Testing framework és una eina gratuïta que ens dona una solució per escriure test que interactuen amb el navegador, donant-nos una capa d'abstracció per a poder realitzar accions sobre la pàgina web oberta al navegador, i poder automatitzar-les. Te suport multi navegador. No proveeix de cap eina per a gravar els test, aquests han de ser gravats mitjançat codi. Cal dir que és una solució gratuïta

Test Studio són un conjunt d'eines o aplicacions que permeten facilitar l'ús de Testing framework. Entre d'altres coses ens ofereix una interfície de gravació de circuits, tot i que després podrem modificar el codi per optimitzar els nostres test. També ofereix un repositori de test per a facilitar-ne la gestió. Aquesta solució es de pagament.

A continuació veiem un diagrama on s'exposa com interactuen les dues solucions:

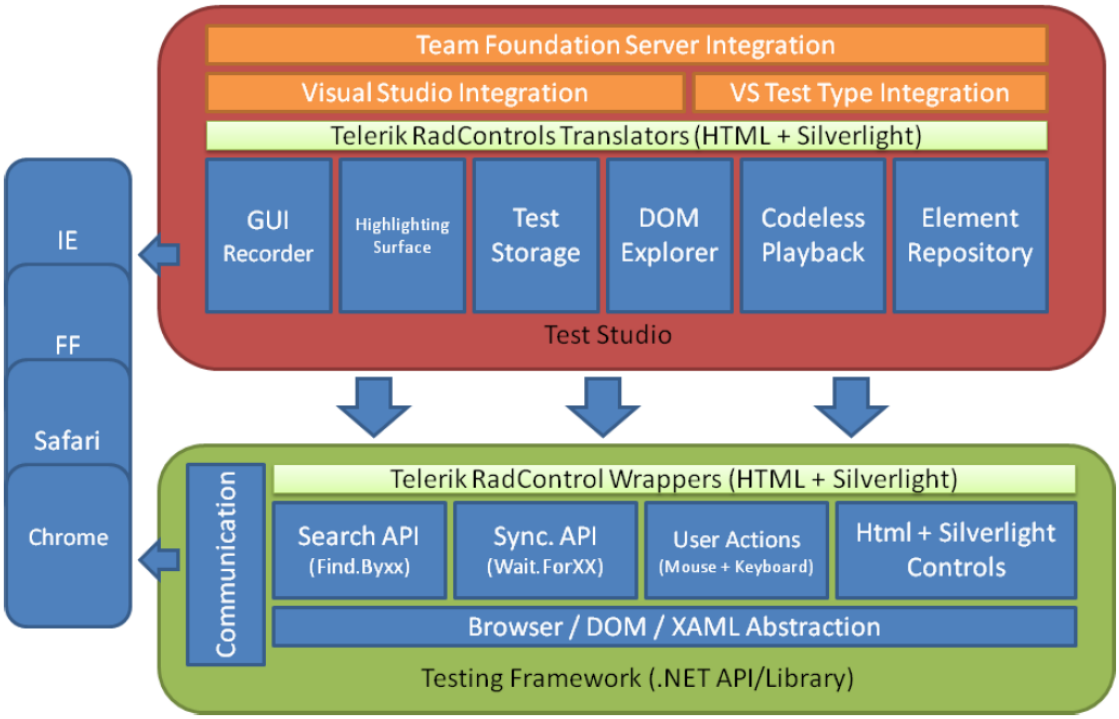


Figura 9 - Telerik testing framework i Test Studio

Com podem veure, la capa d'automatització correspon a testing framework, i test studio el que fa es dotar d'eines per a facilitar-ne l'ús i l'administració.

Les dues solucions s'integren amb Microsoft Visual Studio, permetent escriure els test en C# o VB.NET.

3.2.5 Resum

	Selenium	Sahi	WatiN/WatiR	Telerik Testing Framework
Gravador	Només disponible en firefox	Tots els navegadors	Tots els navegadors	Tots els navegadors
Suport IFrames	✓	✓	✓	✓
Identificació d'objectes	Id, Nom o XPATH	Algorisme propi	Id, Nom o XPATH	Algorisme propi
Suport multi navegador	✓	✓	IE i FIrefox	✓
Llenguatges de programació dels tests	Java, Ruby, Perl, Python, C#, etc	Sahi Script (propi), Java, Ruby Sahi Script	.NET / Ruby	.NET
Independent del SO	✓	✓	✗	✗
Suport HTTPS	✓	✓	✓	✓
Gratuït	✓	✗ Hi ha una part gratuïta, i l'aplicació (més funcionalitats) es de pagament	✓	✗ Hi ha una part gratuïta, i l'aplicació (més funcionalitats) es de pagament

3.2.7 Elecció final

Després d'estudiar varies possibilitats que ofereix el mercat finalment s'ha escollit utilitzar Selenium com a framework d'automatització de navegadors. El que ha motivat aquesta elecció és:

- Suport de diversos llenguatges de programació, entre ells Java, el que ens permet una gran flexibilitat i potencia per a poder realitzar desenvolupaments sobre aquesta base.
- Projecte viu i en constant evolució. La versió dos de Selenium amb la inclusió de WebDriver ha suposat un avanç molt gran a l'eina i li ha dotat de moltes de les funcionalitats que li faltaven a la primera versió.
- Solució gratuïta, existeixen solucions de pagament que ofereixen funcionalitats espectaculars, però després ens trobem amb el problema de voler afegir noves funcionalitats o integrar-ho amb eines pròpies, ja que són solucions tancades. També prometen moltes funcionalitats i que tot funciona a la perfecció però normalment acostumen a tenir el mateix problema que les solucions gratuïtes.
- Solució open source. Això ens permet afegir aquelles funcionalitats que no tinguem incloses.
- La inclusió de WebDriver com a estàndard d'internet per la W3C. Això mostra que la feina feta per l'equip de desenvolupament de Selenium s'està prenent seriosament dins de la comunicat d'internet.
- La nombrosa documentació existent, tant oficial com per part de la comunitat internauta.

Cal destacar que s'han provat totes les solucions aquí presentades, i tot i que a les seves respectives pàgines prometen que són la millor solució ens hem trobat amb mancances o problemes que es produeixen en totes elles. Tot i això Selenium es la que ens permet un entorn més obert, i per tant permet que dissenyem una solució a mida utilitzant-lo com a base.

3.3 Frameworks i APIs utilitzats

3.3.1 Selenium

Comentat a l'apartat 2.2.1

3.3.2 Hibernate

Es tracta d'un framework de ORM (object/relational mapping) llicenciat sota GNU LGPL, que facilita el mapeig d'atributs entre bases de dades relacionals i el model d'objectes d'una aplicació, en altres paraules agilitza la relació entre l'aplicació i la base de dades. Les entitats/classes són mapejades a les taules, les instancies son mapejades a les files i els atributs de les instancies es mapegen a les columnes de la taula, aconseguint una base de dades d'objectes virtuals. Hibernate és un framework de persistència[1] que s'utilitza per a fer persistir la informació des del llenguatge de programació a la base de dades.

A grans trets, pel que fa a l'arquitectura d'Hibernate i com podem veure en la següent figura, aquest crea una capa entre la bades de dades i l'aplicació, crea objectes persistents que sincronitzen les dades entre l'aplicació i la base de dades.

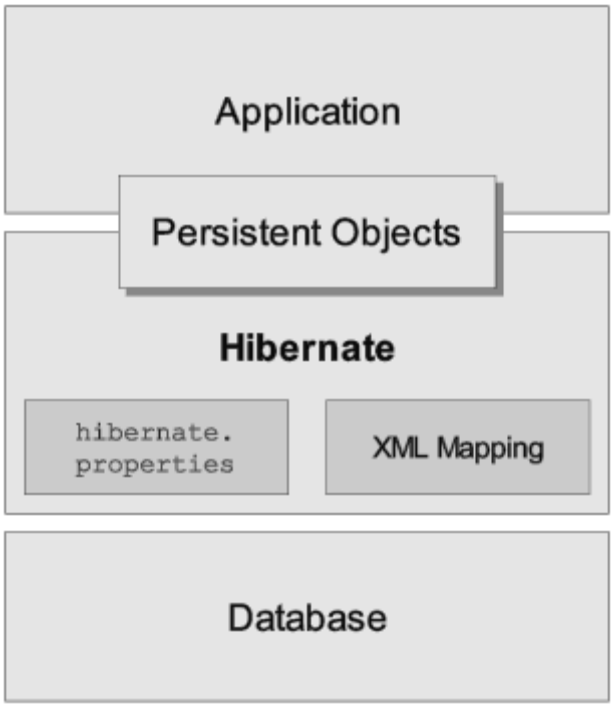


Figura 10 - Arquitectura general Hibernate

Per tal de que les dades siguin persistents contra la Base de dades, Hibernate crea una instancia de la classe entitat, la classe Java mapejada amb la taula de la base de dades. A aquest objecte se l'anomena objecte transitori, ja que encara no esta associat a la sessió ni

[1] Persistència: procés d'emmagatzemament de dades a algun mitja permanent i recuperar-la en qualsevol punt i moment inclús quan l'aplicació que han creat les dades ha finalitzat.

esta persistit a la base de dades. Per a fer persistent un objecte a la base de dades, es crea la instancia de la interfície “SessionFactory”. “SessionFactory” es una instancia única (definició en l’apartat 3.1.3) que implementa el patró de disseny “Factory Method” (definició a l’apartat 3.1.2). El “SessionFactory” carrega el fitxer de configuració d’Hibernate i amb l’ajut de “TransactionFactory” i “ConnectionProvider” implementa totes les dades de configuració sobre la base de dades.

Cada connexió a base de dades amb Hibernate es crea amb una instancia de la interfície de sessió. La sessió representa una connexió única amb la base de dades. Els objectes de sessió es creen des de l’objecte “SessionFactory”.

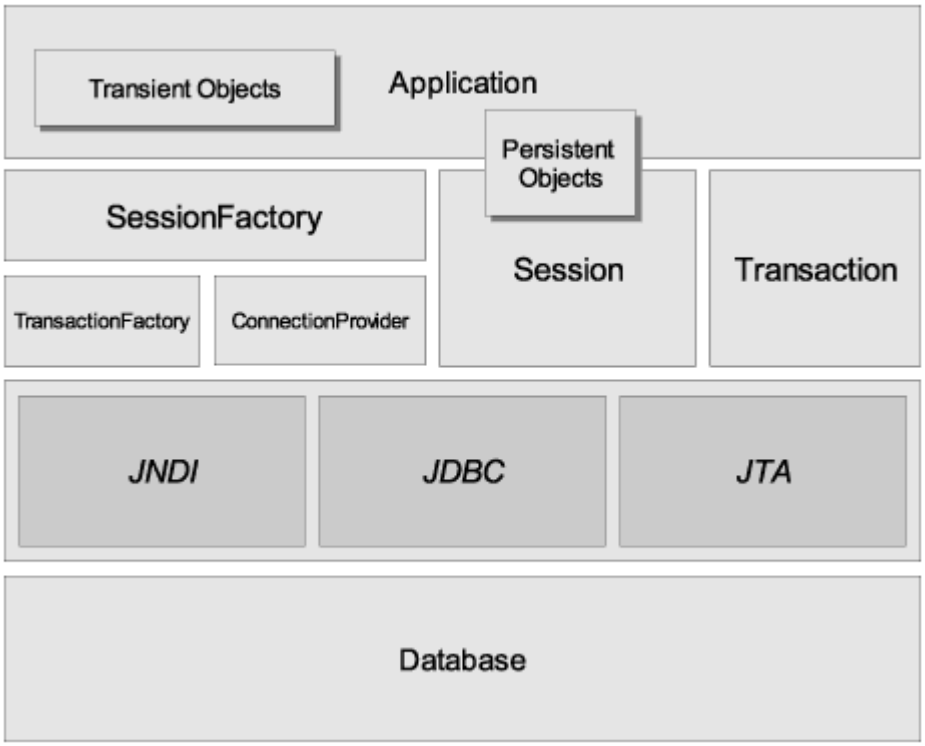


Figura 11 - Arquitectura Hibernate (detall)

3.3.3 Bootstrap

Bootstrap és un framework de front-end ,desenvolupat per Twitter (és el framework amb el que està desenvolupat twitter), per a desenvolupar aplicacions “responsive” combinant CSS i JavaScript. La seva principal funció és la de simplificar la creació de dissenys web, tenint com a principal avantatge la creació d’interfícies que s’adaptin a diferents navegadors (tant de d’escriptori com mòbils). Per tant no haurem de crear versions diferents per a navegadors d’escriptori i per a mòbils, sinó que la mateixa pàgina s’adaptarà a cada un dels casos (navegadors d’escriptori, tables, mòbils i diferents resolucions). Ofereix un disseny sòlid utilitzant LESS (llenguatge que este de CSS per a donar de comportament dinàmic) i estàndards com CSS3 i HTML5.

3.3.4 REST

Tot i no ser un framework explicarem en que consisteix aquesta família d’arquitectures, per entendre millor el pròxim apartat. REST (Representational State Transfer, com dèiem és una família d’arquitectures, qualsevol arquitectura de sistemes distribuïts que compleixi amb un seguit de requisits es pot considerar com una arquitectura REST. Per tant és un conjunt de principis que defineixes la interacció entre diferents components

I quines són les regles que les aplicacions han de complir per tal de poder ser anomenades REST:

- Arquitectura client-servidor: consisteix en una separació clara i concisa entre els dos agents bàsics en un intercanvi d’informació, el client i el servidor. Aquests dos agents han de ser independents entre si, el que permetrà una flexibilitat molt alta en ambdós sentits
- Stateless: el nostre servidor no podrà emmagatzemar dades del client per a mantenir l’estat del mateix. Això facilita el disseny, el desenvolupament i la distribució a través de múltiples servidors. La responsabilitat de mantenir l’estat recau en el client de l’aplicació. El servidor només és responsable de generar respostes i proveir d’una interfície que permeti al client mantenir l’estat pels seus propis mitjans.
- Cacheable: el servidor que serveix les peticions del client ha de definir algun mètode de caching per a les peticions, per tal d’augmentar el rendiment i l’escalabilitat. HTTP ja implementa això amb la capçalera “Cache-control”.
- Sistema per capes: el sistema no ha de forçar al client a saber per a quina capa es tramita la informació, el que permet al client conservar la seva independència respecta a aquestes capes.
- Interfície uniforme: aquesta regla simplifica el protocol i augmenta l’escalabilitat i el rendiment del sistema. No es vol que la interfície de comunicació entre un client i el servidor depengui del servidor al que estem fent les peticions, ni molt menys del client, pel que aquesta regla ens garanteix que no importa qui faci les peticions ni qui les rebi, sempre i quan ambdós compleixin una interfície definida amb anterioritat.

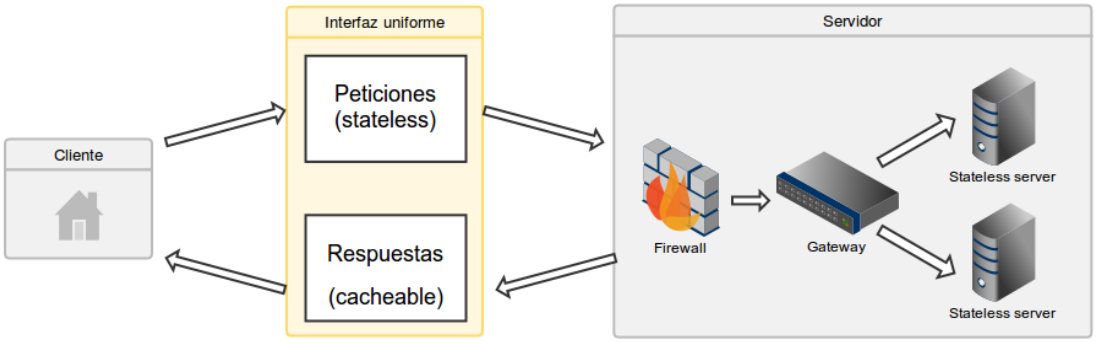


Figura 12 - Funcionament arquitectura REST

Propietats:

- No es publiquen serveis RPC, en aquesta arquitectura no es publiquen un conjunt arbitrari de mètodes i operacions, el que es publiquen són recursos. Un recurs es pot considerar com una entitat que representa un concepte de negoci que pot ser accedit públicament. Entenem per recursos les dades i les funcionalitats.
- Als recursos s'hi accedeix utilitzant URIs (Uniform Resource Identifier), que normalment són adreces HTTP. Els recursos són accionats mitjançant l'ús d'un conjunt d'operacions simples i ben definies. Als serveis web RESTful típicament s'utilitzen els quatre mètodes HTTP sobre les operacions que realitzen:

Mètode HTTP	Operació que realitza
GET	Agafa un recurs
POST	Crea un recurs i altres operacions
PUT	Crea o actualitza un recurs
DELETE	Esborra un recurs

Taula 1 - Relació dels mètodes HTTP amb les operacions que realitza

- Cada recurs posseeix un estat intern, que no pot ser accedit directament des de l'exterior. El que si és accessible des de l'exterior és una o varies representacions d'aquest estat. Entenem per representació el format de dades concret utilitzat per a la transferència de l'estat públic del recurs entre el client i el servidor. Aquest format pot ser: XML, JSON o ambdós.

Resumint, REST permet crear serveis WEB distribuïts seguint un seguit de principis.

3.3.4.1 REST vs SOAP

Existeix una alternativa per desenvolupar WebServices, anomenada SOAP. A continuació detallem les característiques principals d'ambdós mètodes:

En general si l'aplicació que hem de publicar a l'exterior es complexa SOAP serà més útil, ja que requereix menys codi d'interconnexió que REST, per tant facilita la implementació, sobretot en les funcions més complexes.

Si per el contrari l'aplicació a publicar necessita una corba d'aprenentatge menor, transaccions simples i ràpids resultats, la millor elecció és REST.

4 Anàlisis i Disseny de l'Agent MEU

La nova solució de monitorització d'experiència d'usuari en entorn web, és un projecte propi anomenat MEU. La seva principal funció serà la d'oferir als seus usuaris una solució completa per tal de conèixer en tot moment l'estat dels seus serveis com aplicacions web. Al tenir experiència en l'ús d'eines d'aquest tipus, el que es pretén és oferir una solució completa que ofereixi:

- Pels explotadors del servei:
    - Facilitat d'ús: implica que no requerirà de personal amb alts coneixements tècnics per a fer servir l'eina.
    - Manteniment baix: un manteniment baix implica utilitzar menys hores dels tècnics, i per tant el cost en manteniment es reduirà podent així oferir preus més competitius
    - Escalabilitat: si la solució és escalable, no necessitarem de grans inversions quan vulguem ampliar el servei ofert
- Pels usuaris del servei:
    - Fiabilitat: el client vol saber quan te una indisponiblitat. Quan això succeeix, haurà de dedicar recursos tècnics en investigar les causes d'aquesta indisponiblitat. Investigar falsos positius repercuteix negativament en els costos de TI, i sobretot en la credibilitat de la solució.
    - Facilitat de diagnòstic: la solució no només ha de dir que ha aparegut un error, sinó que ha de proporcionar les eines suficients per tal de poder determinar on pot estar el problema. Hi ha altres eines, que poden arribar a indicar quin servidor està donant problemes, quina consulta SQL és problemàtica, etc. No és la finalitat de MEU, ja que MEU te la mateixa informació que te un usuari, per tant pot arribar al mateix nivell de detall que podria un usuari.

4.1 Requeriments

Per tal de poder desenvolupar aquesta solució, s'ha optat per un llenguatge de programació d'ús global. Es podria optar per desenvolupar la solució amb tecnologia propietària de Microsoft, .Net, ja que el framework Selenium ho permet així com la resta de components de la solució, però si s'utilitzés aquesta tecnologia no podríem complir un dels principals objectius d'aquest projecte: multiplataforma.

Per aquest motiu s'ha optat per a utilitzar com a llenguatge de programació Java 1.6, J2EE per la part d'aplicacions web i J2SE per la part d'aplicació d'escriptori.

4.2 Dependències

La solució MEU fa ús de llibreries externes per a funcionar. A continuació es detallen, per cada un dels components de la solució, les seves dependències. A més requereix d'un seguit de components externs, que també s'han desenvolupat, per tal de funcionar, i que es detallen en punts posteriors. A mode resum l'aplicació requereix:

- BBDD SQL Server 2008 R2: per a emmagatzemar les dades generades per a la seva posterior explotació.
- WebService: per tal de realitzar la comunicació entre l'Agent i la BBDD. Aquest serà l'encarregat de tramitar les peticions que el client realitzi, i comunicar-se amb la BBDD, retornant o els missatges corresponents a l'agent.
- Aplicació d'administració: aplicació web per tal d'administrar tot el servei.
- Eina Fiddler per extreure mètriques

A continuació podem veure el diagrama de dependències de l'agent:

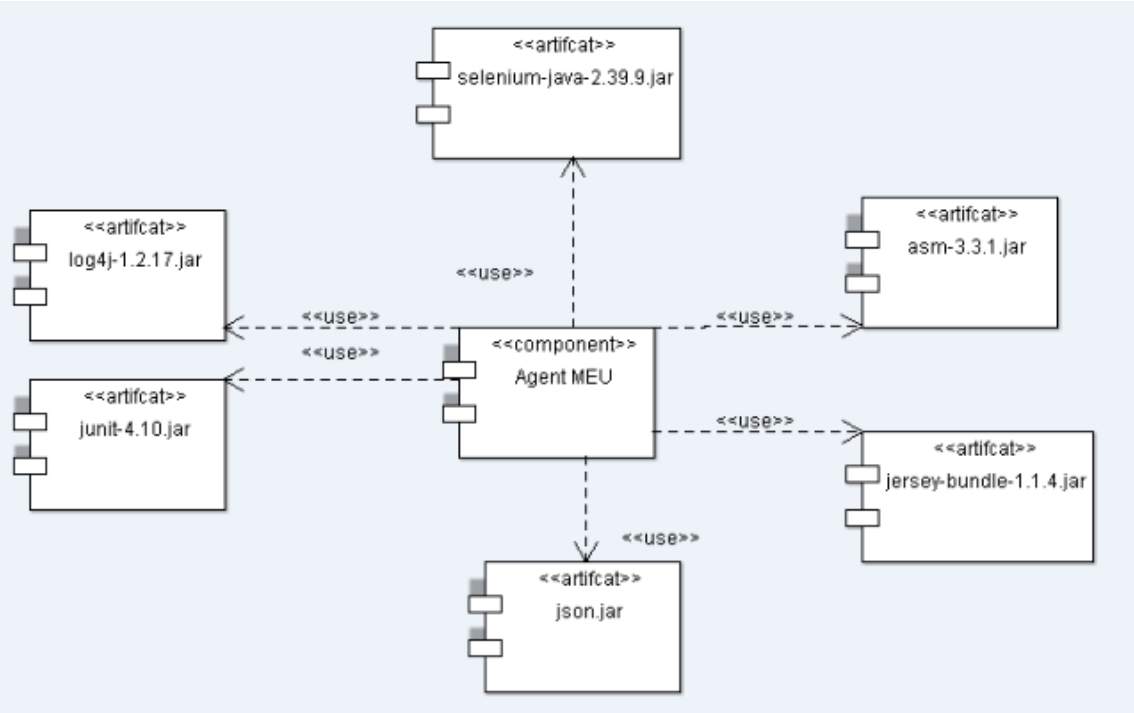


Figura 13 - Diagrama de dependències Agent MEU

4.3 Estructura i jerarquia de paquets

Aquest projecte s'ha estructurat en diversos paquets Java, agrupant les classes en aquests paquets segons les seves funcionalitats.

Paquet	Descripció
Com.pfc.selenium.agete	Classe principal de l'agent on s'inicialitza l'aplicació
Com.pfc.selenium.classes	Seguit d'objectes que l'aplicació necessita instanciar per tal d'emmagatzemar certes dades
Com.pfc.selenium.connections	Classes de connexió a l'exterior: WebService i FTP
Com.pfc.selenium.testRunner	Classes d'execució dels test
Com.pfc.selenium.UI.Agente	Classe de la interfície d'usuari de l'agent
Com.pfc.selenium.UI.listeners	Classes d'escolta d'esdeveniments sobre la interfície gràfica
Com.pfc.selenium.UI.Runnable	Classe per controlar els botons de la interfície gràfica i parar o començar l'execució dels tes
Com.pfc.selenium.utils	Classe d'utilitats complementaries.

Taula 2 - Paquets de l'agent MEU

A continuació mostrem el diagrama de paquets:

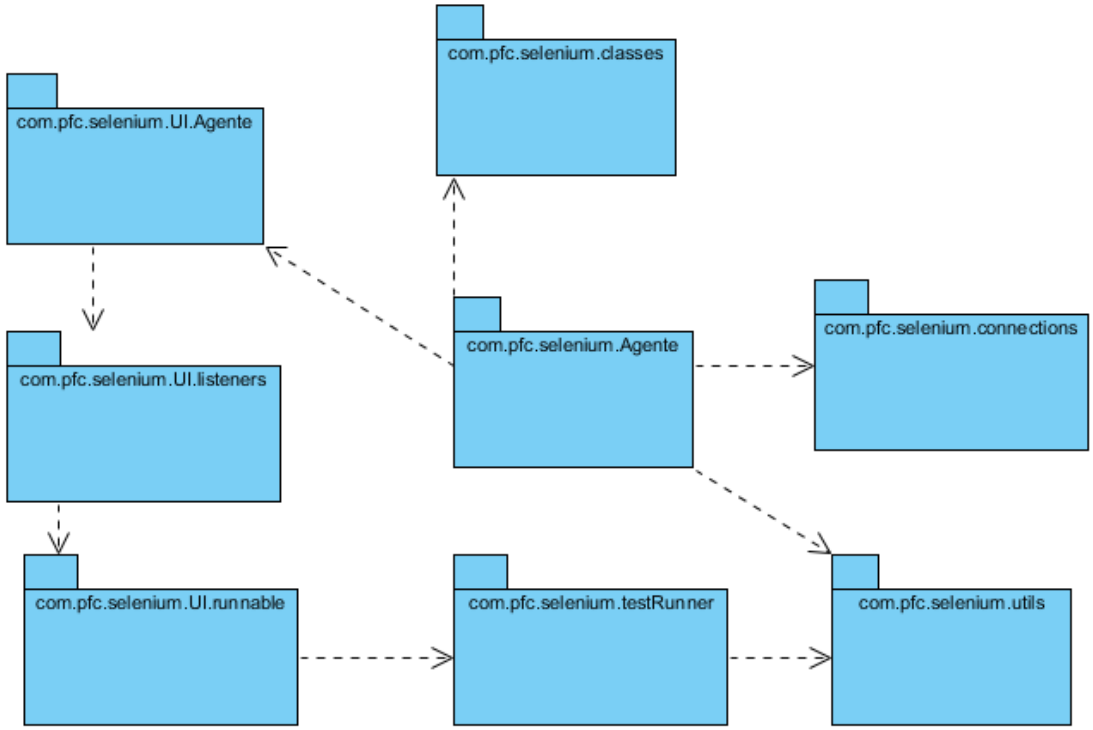


Figura 14 - Diagrama de paquets Agent

4.4 Configuració

L'agent de la nostre solució és un component que per si sol no sap que fer, per tal de saber quins circuits ha d'executar i quina és la seva configuració inicial es necessari que reculli aquesta informació d'un fitxer de configuració. Per tal de dur a terme això s'ha creat un fitxer XML: Agente.xml on apareixen els paràmetres bàsics necessaris per a que l'agent comenci a funcionar. Per tal de llegit el fitxer XML s'utilitza el parser DOM, que facilita molt el tractament de fitxers XML. L'estructura bàsica d'aquest fitxer és:

```
<?xml version="1.0" encoding="UTF-8"?>
<Agente>
        <Robot>
                <Nombre> </Nombre>
        </Robot>
        <WebService>
                <URL> </URL>
        </WebService>
        <FTPServer>
                <FTPIP> </FTPIP>
                <FTPUser> </FTPUser>
                <FTPPass> </FTPPass>
        </FTPServer>
</Agente>
```

Figura 15 - Fitxer de configuració agent

S'han definit tres seccions bàsiques de configuració:

- Robot: aquí especificarem el nom del robot que volem que executi aquest agent. Aquest nom és necessari per tal que l'agent realitzi una consulta al WebService indicant que te assignat aquest robot i rebre totes les dades necessàries per a començar a executar circuits
- WebService: es defineix la URL on està publicat el WebService per tal que el robot pugui llançar les diferents peticions
- FTPServer: s'indicarà la informació necessària per tal de poder connectar al servidor FTP on es pujaran les captures d'error i d'on es baixaran els circuits.

4.5 Disseny de l'aplicació

Després de l'estudi realitzat en apartats anteriors ha ajudat en gran part en com volem realitzar el disseny d'aquesta eina. No només per tal de complir els objectius marcats, sinó per fer-ho de la forma més eficient possible.

4.5.1 Patró singleton

Com hem descrit anteriorment, és un patró de tipus creacional, que s'utilitza per assegurar que en tot moment, la classe sobre la que s'implementa tingui una i només una instancia. Això s'aconsegueix fent que els constructors de la classe siguin privats, i per tant no puguin ser cridats de l'exterior, assegurant-nos així el control de l'instanciament de la classe.

Aquest patró s'ha utilitzat en les següents classes:

- WebServiceClient
- FTPClient
- Fiddler

De les que només volem que existeixi una única instancia durant tota l'execució. Això ens permet sobretot tenir un mètode d'accés a aquest objectes, sense necessitat de crear variables globals o bé anar passant l'objecte entre diferents classes, ja que en cap cas hi haurà modificacions en els seus atributs un cop creades.

L'estructura del patró és la següent:

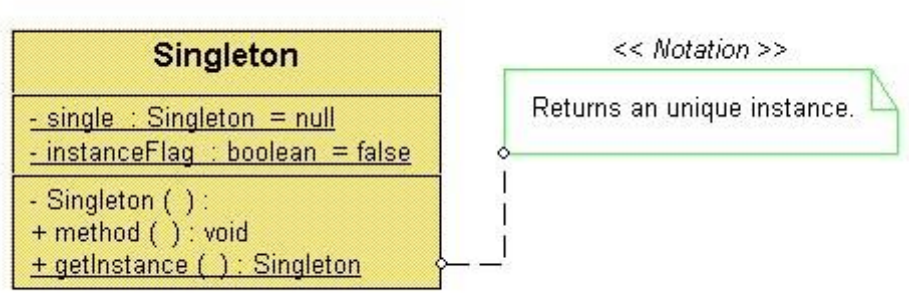


Figura 16 - Estructura patró Singleton

Com podem veure el seu constructor és privat, per tant no serà accessible des de l'exterior. Quan alguna classe necessiti instanciar aquest objecte el que farà es cridar al mètode getInstance, que detallem a continuació:

```
public Singleton getInstance(){
      if (Singleton == null){
            Singleton = new Singleton();
      }
      return Singleton;
}
```

Com podem veure en cas que la classe no hagi estat instanciada, cridarem al constructor per instanciar-la. En cas que aquesta ja estigues instanciada simplement retornarem la instancia.

4.5.2 ClassLoader

La funció bàsica de l'agent és l'execució de test per tal de recollir mètriques de temps de resposta i detecció d'indisponiblitats. Aquests test són classes Java ja compilades. El problema que existeix es que l'agent ha de ser capaç d'executar aquests .class i extreure'n els resultats de l'execució. A priori mai sabrem quins test té assignats un robot, a l'iniciar l'aplicació l'agent es comunicarà amb el WebService, indicant-li quin robot té configurat i demanant-li les dades associades a aquest robot. Una de les dades són els tests assignats. A part de la informació bàsica d'un test com el nom, el seu identificador, etc. També requerirà del fitxer Java prèviament compilats associat a aquell test. Aquest fitxer .class conté el codi necessari per a executar l'automatització del navegador i els passos associats, així com recollir els resultats. Executar, des de codi Java, un JUnit és senzill, només hem de tenir les llibreries de JUnit, instanciar-ne un objecte i cridar a la funció runTest, tal com podem veure a l'API de JUnit Core:

```
Result | run(java.lang.Class<?>... classes)
         Run all the tests in classes.
```

Figura 17 - runTest JUnit Core

Com podem veure aquesta funció retorna un objecte Result, i demana com a paràmetre un objecte de tipus Class. En el mapa actual, ens trobem que els objectes .class no són part de la nostre aplicació i per tant no formen part de cap paquet Java. Això va complicar aquest punt, ja que era indispensable dotar a l'aplicació de la capacitat de carregar classes externes a ella de forma dinàmica. Per aquesta tasca, i després de realitzar una investigació, es va decidir que per les possibilitats que ens oferia la llibreria nativa de Java l'ús del ClassLoader s'ajustava a les nostres necessitats.

ClassLoader és una classe encarregada de carregar altres classes en memòria segons la necessitat que tingui la màquina virtual.

Al iniciar qualsevol aplicació Java, ens trobem amb la següent estructura de carregadors de classe:

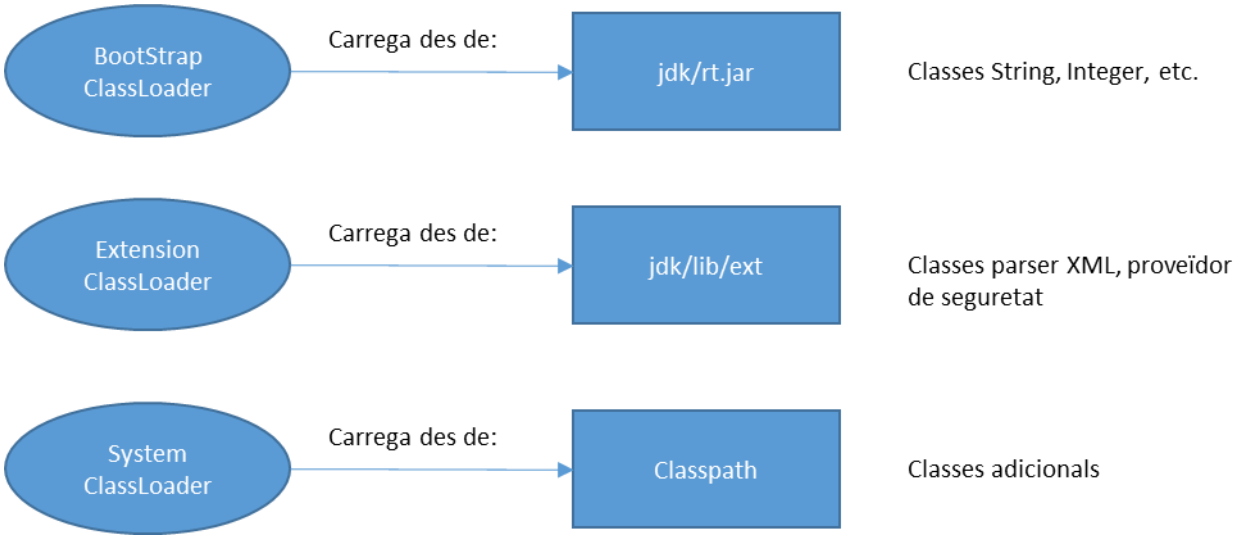


Figura 18 - Estructura de carregadors de classe

Com podem veure hi ha tres tipus de ClassLoader:

- **BootStrapClassLoader:** encarregat de carregar les classes principals (core) del JDK, com pot ser String, Integer, Date, etc. Com podem veure aquestes classes estan dins la llibreria rt.jar dins del propi JDK.
- **ExtensionClassLoader:** encarregat de carregar les classes ubicades a la carpeta jdk/lib/ext i fa referencia a extensions que el JDK, tipus de proveïdors de seguretat, parser XML, etc.
- **SystemClassLoader:** és un dels més habituals i s'encarrega de carregar les classes contingudes dins del classpath de l'aplicació. Normalment llibreries que nosaltres necessitem, com Hibernate, Spring, etc.

La relació entre aquests diferents ClassLoaders és una relació pare/fill:

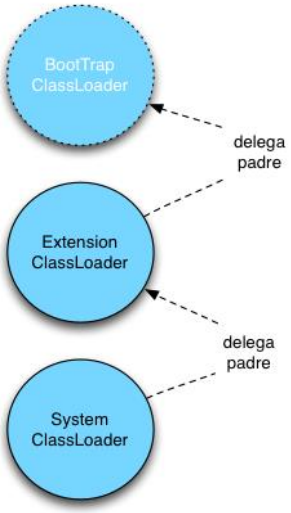


Figura 19 Relació entre classLoaders

Quan un ClassLoader va a realitzar una carrega primer pregunta al seu ClassLoader pare si ell te la classe disponible, en cas afirmatiu la carrega, en cas contrari el pare delega la petició al següent fins arribar a BootStrapClassLoader. Si cap dels ClassLoaders pot carregar la classe, el carregador actual (originant de la petició) intenta carregar-la de les seves rutes disponibles, si no la troba es produeix un error ClassNotFoudException.

Coneixent el funcionament del ClassLoader i les seves limitacions, veiem que per a carregar una classe de forma dinàmica aquesta ha d'estar dins dels nostres rutes, però com diem, a l'iniciar l'aplicació no tenim disponibles encara els fitxers .Class. Per tant com la carrega és en temps d'execució haurem d'utilitzar reflexió.

Exposat tot això ens veiem en la necessitat de fer-nos el nostre propi ClassLoader que compleixi amb totes les nostres necessitats, aquest ClassLoader serà capaç de carregar una classe només passant-li com a paràmetre la ruta on està el fitxer. Donat això instanciarà la classe i ens retornarà el Class que necessitem per executar amb JUnit.

Es crea una classe pròpia que estendrà de ClassLoader on es desenvolupa una funció on li passarem com a paràmetre la ruta i el nom del fitxer .class que volem carregar dinàmicament. Un cop fet això creem un objecte URL per a poder accedir al fitxer i l'obrim amb un

InputStream, finalment copiem el fitxer binari a un ByteArrayOutputStream. En aquest punt tenim en memòria el contingut del fitxer binari. L'instanciem a un objecte de tipus Class i el definim com una classe. Hem aconseguit carregar dinàmicament un .class.

4.5.3 Classes customitzades

L'execució de test unitaris JUnit te com a resultat un objecte Result, aquest objecte ens proporciona certa informació com:

- Nombre d'errors
- Descripció dels errors
- Temps d'execució total

Els requisits de la solució que es presenta requereixen d'un nivell de detall major pel que fa el resultats oferts. No només volem saber que ha tardat la totalitat de l'execució d'un circuit sinó que volem una granularitat menor de temps de resposta, volem poder veure que tarda cada un dels passos (navegació entre pàgines) per tal de poder detectar, en cas que el temps de resposta creixi en quin pas s'ha incrementat aquest.

Per aquest motiu veiem la necessitat de crear un seguit de classes que ens permetin obtenir aquestes dades.

Ens trobem amb dos problemes. El primer d'ells es com fer que el nostre test generi aquestes mètriques que comentem. Recordem que els test es generen mitjançant una eina externa, i que la generació d'aquests test no és objectiu d'aquest projecte, sinó que son quelcom que ens ve donat. Però això no vol dir que no puguem modificar-los d'alguna forma. La forma "fàcil" es afegir un seguit de funcionalitats al codi del nostre test per tal que pugui recollir les mètriques necessàries, però si per algun motiu volguéssim variar la forma en que fem les mesures o afegir-ne de noves hauríem de modificar el codi de tots els nostres test. Si en tenim poc no és problema, però en el moment que tinguem desenes de circuits la feina es converteix en quelcom laboriós i amb moltes possibilitats de cometre errors. Per tant es pensa una solució que ens blindi contra aquests problemes. El que es fa es crear una nova classe on definirem totes aquelles funcions que requerim realitzar a tots els test per tal de recollir les mètriques que ens interessen. El que fem un cop realitzada aquesta nova classe es modificar el codi dels nostres test per tal que heretin de la nova classe. Ara tots els test tenen les funcionalitats del pare, i per tant podrem cridar a les funcions de la classe pare des de les classes filla, i obtenir les mètriques desitjades. Per tant utilitzant el concepte d'herència aconseguim solucionar aquest problema.

El segon dels problemes es com recollir totes aquestes noves mètriques que s'estan generant. Recordem que la funció run de JUnit retorna un objecte Result, i aquest objecte no conté les dades que nosaltres hem generat, per tant necessitem alguna forma per a poder recollir les dades que ens genera el nostre test.

Primer de tot ens creem una nova classe de resultat anomenada ResultCustom que ens permet emmagatzemar tant el Result que ens retorna el run de JUnit, com totes aquelles mètriques extra que generem a dins del nostre test.

Per altre banda ens hem creat una nova classe que la seva funció principal serà la d'executar el test que li passem. Un cop executat el test amb JUnit recollirà l'objecte Result que ens retorna

l'execució, així com recollirà les mètriques extra generades, emmagatzemant-ho tot en la classe ResultCustom que hem creat.

Per tant amb aquestes modificacions tenim un nou sistema d'execució de test JUnit, on la funció encarregada d'executar-los en genera un nou objecte Resultat.

A continuació podem veure el detall de les classes que s'han creat per a poder resoldre els problemes anteriorment comentats.

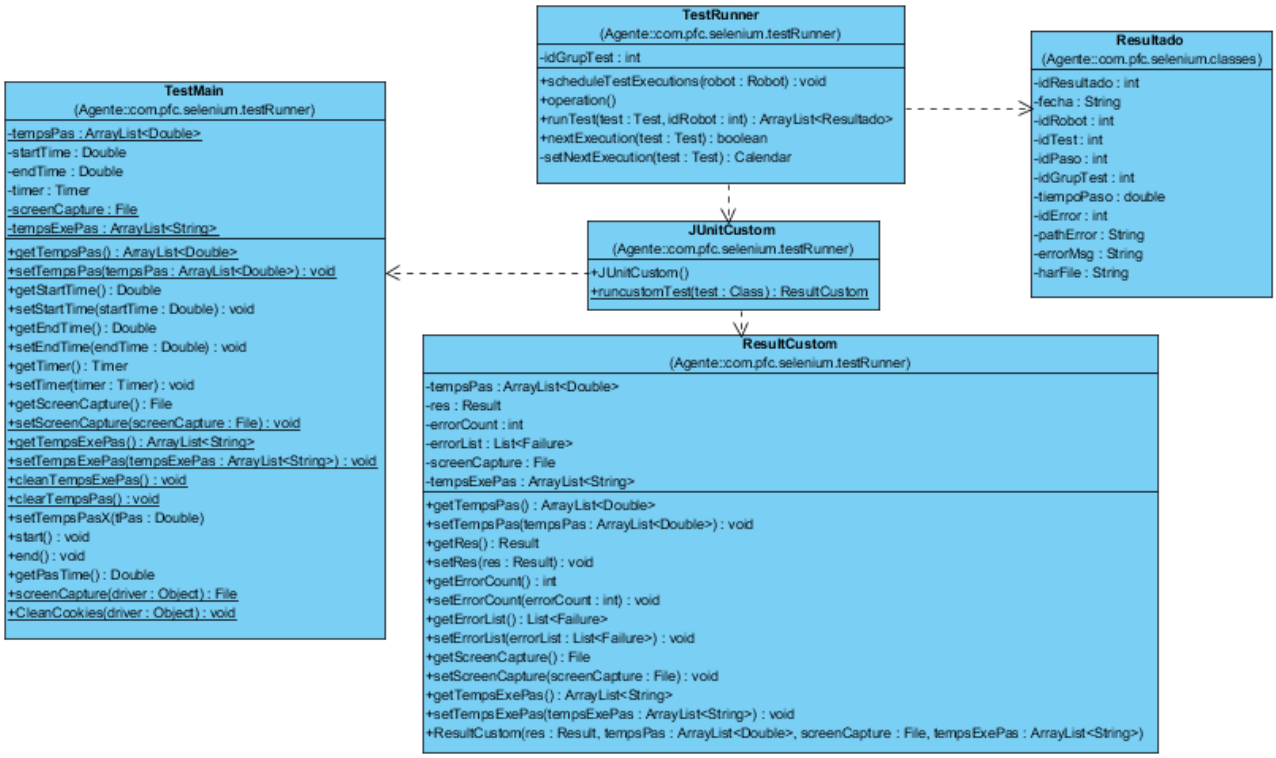


Figura 20 - Diagrama de classes customitzades

La classe TestRunner és l'encarregada de gestionar les execucions dels nostres test, no només és qui s'encarrega de processar l'execució, sinó que també valida si és el moment per a que s'executi el test.

La classe JUnitCustom s'encarrega de l'execució pròpiament dita, i de recollir tant el Result que retorna el run de JUnit com la resta de resultats que hem capturat.

La classe TestMain és la que fem estendre als nostres tests i també forma part de l'aplicació, per tal de poder-hi fer crides. Tenim un seguit de variables i funcions estàtiques, així que podrem accedir-hi sense necessitat d'instanciar l'objecte. Això ho fem perquè realment la instancia de l'objecte la crea el test a l'executar-se per reflexió. Des de l'aplicació necessitem accés a aquestes dades i per tant necessitem que siguin estàtiques per a que no variïn els seus valors durant l'execució. S'han desenvolupat un seguit de funcions per a poder inicialitzar les variables quan ens convingui, sinó repetiríem valors en diferents execucions.

Finalment la classe ResultCustom s'encarrega d'agrupar tots els resultats, com el Result així com els temps de cada pas, el nombre d'errors que s'han trobat durant l'execució del test, la descripció dels errors, la captura de pantalla de l'error (en cas que es produeixi un error), i l'hora d'execució de cada pas.

4.5.4 Fiddler

L'eina Selenium proporciona el temps que tarda en executar tot un circuit, des del temps d'apertura del navegador fins que finalitza l'execució. Com hem vist anteriorment, mitjançant desenvolupaments propis hem aconseguit extreure també el temps de cada una de les accions o passos. Però volíem poder extreure més dades que siguin rellevants per al client i que els permeti veure fàcilment que tardà en descarregar-se cada objecte, en quin objecte s'ha produït un error, o quin objecte és el causant de l'increment del temps de resposta. S'han estudiat diverses opcions, una d'elles ha aparegut recentment amb les últimes versions dels navegadors, ens permet extreure dades del temps de DNS, temps de servidor, etc. Mitjançant l'execució d'un Javascript sobre el navegador en cada una de les execucions. Tot i donar més informació que la obtinguda inicialment, veiem que no ens dona el detall que volem. Per aquest motiu es decideix d'utilitzar un "web debugging proxy" que ens permeti, a mode de sniffer, capturar tots els objectes web i extreure'n informació molt detallada. Després d'estudiar diverses opcions s'ha optat per utilitzar Fiddler, eina de l'empresa telerik, gratuïta, de fàcil instal·lació i sobretot no intrusiva i transparent.

Mitjançant aquesta eina no només podem capturar el detall de tots els objectes sinó que ens permet exportar-ho en un format que es pot obrir amb moltes aplicacions, el format HAR (HTTP Archive).

Com podem veure a continuació en aquest fitxer veiem el temps que tarda cada objecte en descarregar-se, així com el detall de les capçaleres HTTP, el HTTP response code, etc.

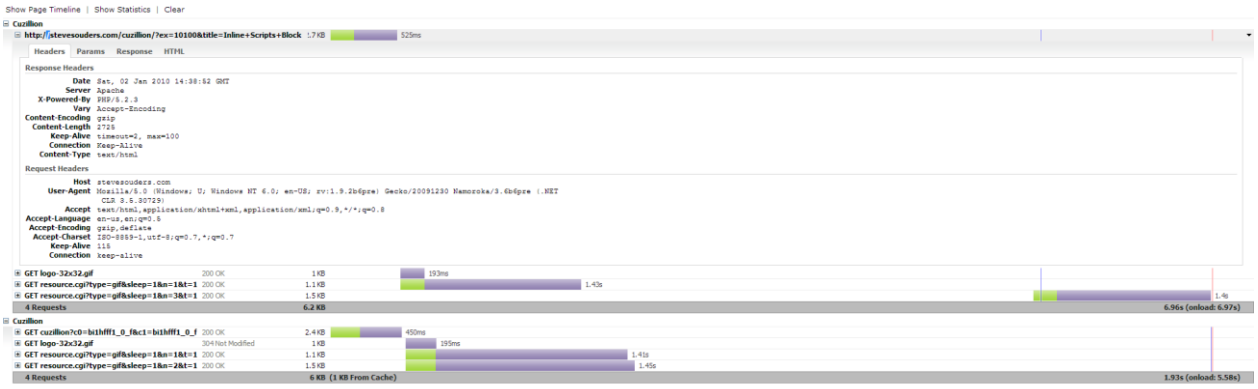


Figura 21 - HAR file

A continuació mostrem el detall de la classe Fiddler creada per a automatitzar-lo des del nostre codi.

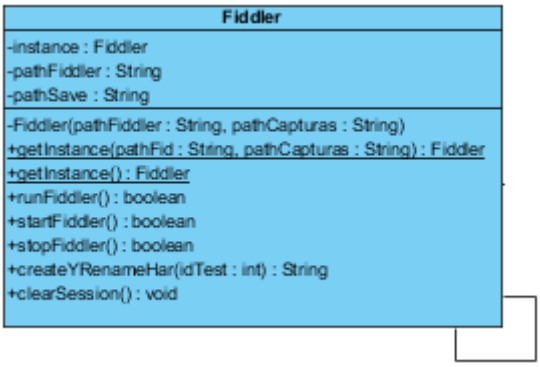


Figura 22 - Classe Fiddler

Com es pot veure per a dissenyar aquesta classe s'ha seguit el patró singleton, per tal d'assegurar-nos que en tota l'execució només existeix una única instancia. Si existís més d'una ens podria donar problemes al poder estar cridant a funcions antagòniques al mateix temps i per tant donant resultats no esperats. Per aquest motiu el constructor és una funció privada, i la forma d'accedir a la classe és mitjançant la funció getInstance() que ens retornarà la instancia de la classe. Internament aquesta funció mirarà si la classe ja ha estat instanciada, en cas contrari la instanciarà i sempre retornarà la instancia.

La funció runFiddler() s'encarrega d'executar l'aplicació. No hem posat un control per validar si l'aplicació ja estava arrencada, degut a que el propi Fiddler només permet una única instancia de la seva aplicació, així que si tornem a executar-la ell sol veurà que ja esta executant-se i no tornarà a executar-se.

La funció startFiddler() llança una comanda cap a Fiddler indicant-li que esborri totes les navegacions que havia realitzat, així evitem mostrar dades que no són del nostre test. stopFiddler() atura la captura i genera el fitxer amb aquesta. Finalment tenim createYRenameHar, el que fa és llançar una comanda a Fiddler per a que generi el fitxer Har, el renombri i el guardi a la carpeta corresponent dins de l'aplicació, a l'espera que sigui pujada al servidor.

4.5.5 Internacionalització

S'ha dotat a l'eina de la funció d'internacionalització, permetent-nos en un futur poder oferir-la a mercats estrangers. Per a aconseguir aquesta fita s'ha centralitzat tots els missatges que es mostren dins de l'aplicació en un fitxer de propietats. Per aquesta primera fase només s'ha creat un únic fitxer de propietats en castellà, però amb la possibilitat, i sobretot amb poc esforç, de poder mostrar l'eina en els idiomes que vulguem.

L'estructura d'aquest fitxer de propietats es senzilla, es defineix un nom fix identificatiu a un missatge i tot seguit s'escriu el missatge:

textIdentificatiu=missatge

Des de la classe Agent es crea un objecte tipus Properties, i l'instanciem passant-li com a paràmetre el fitxer de propietats corresponent a l'idioma que escollim, aconseguint que tots els missatges mostrats estiguin en aquell idioma.

Això no només ens serveix per a poder mostrar fàcilment l'eina en diferents idiomes, sinó que ens permet tenir de forma centralitzada tots els missatges a mostrar, i per tant facilita tant la seva localització dins del codi, ja que només haurem de buscar-lo pel seu identificador, sinó que també ens facilita la modificació d'aquests missatges.

4.5.6 Diagrama de classes de l'Agent MEU

A continuació podem veure el diagrama de classes de l'aplicació agent.

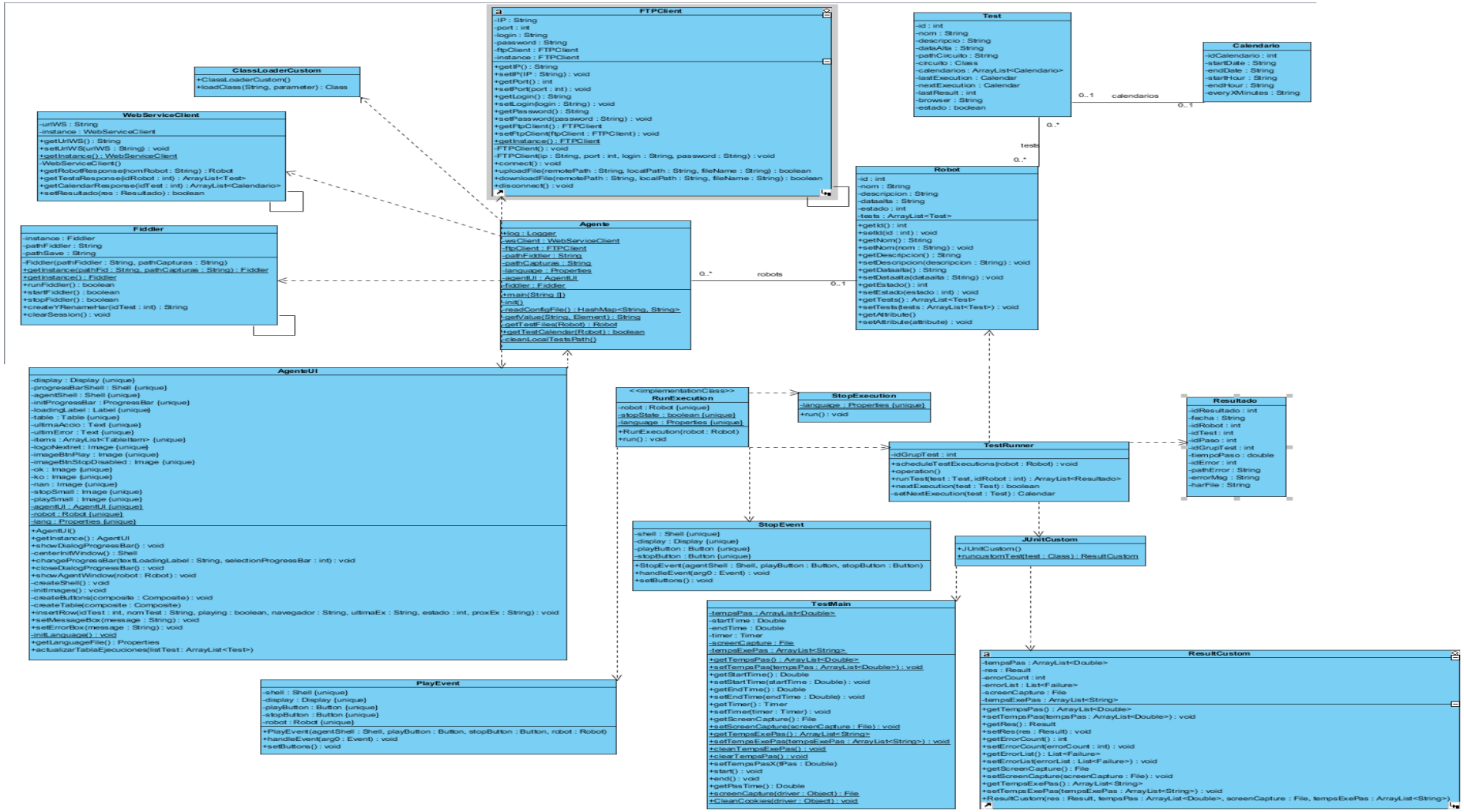


Figura 23 - Diagrama de classes Agent MEU

4.5.7 Diagrama de seqüencies de l'Agent MEU

A continuació mostrem el diagrama de seqüències de l'Agent, es tracta d'una aplicació d'escriptori la interacció de l'usuari es quasi inexistent, bàsicament l'usuari l'únic que podrà fer és arrancar o aturar l'aplicació. En aquest diagrama mostrem des del moment que l'usuari arrenca l'aplicació i pitja el botó play fins que s'executa el primer test i s'emmagatzemen els resultats

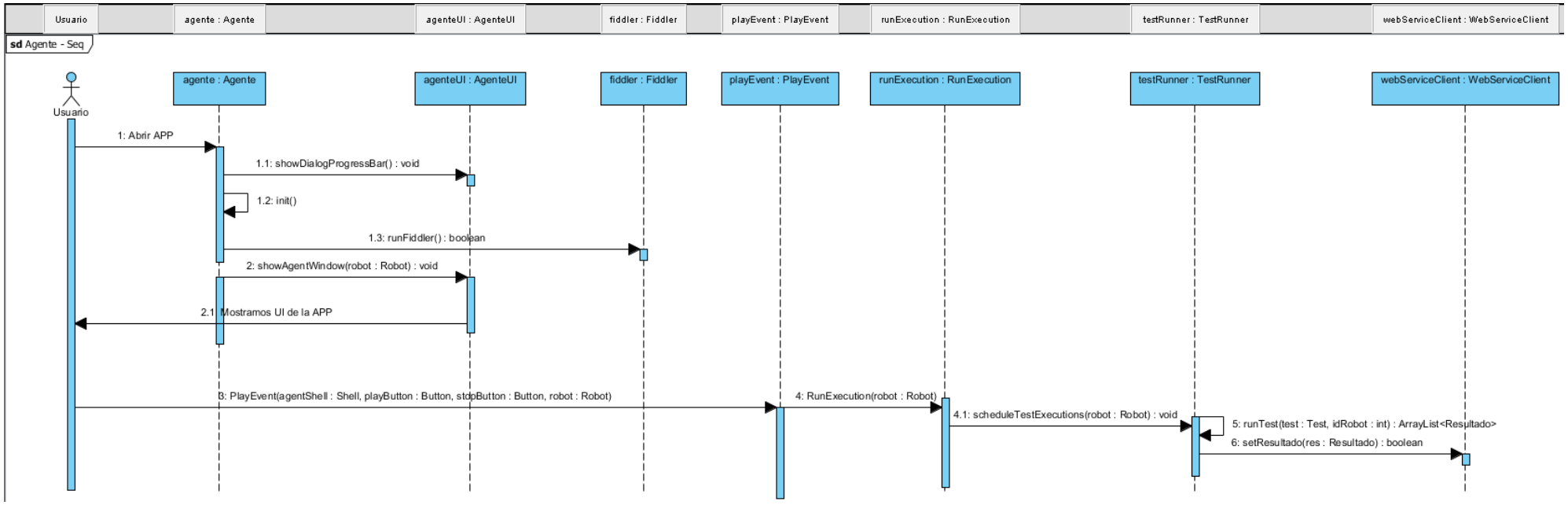


Figura 24 - Diagrama de seqüencies Agent MEU

4.5.8 Diagrama de flux de l'Agent MEU

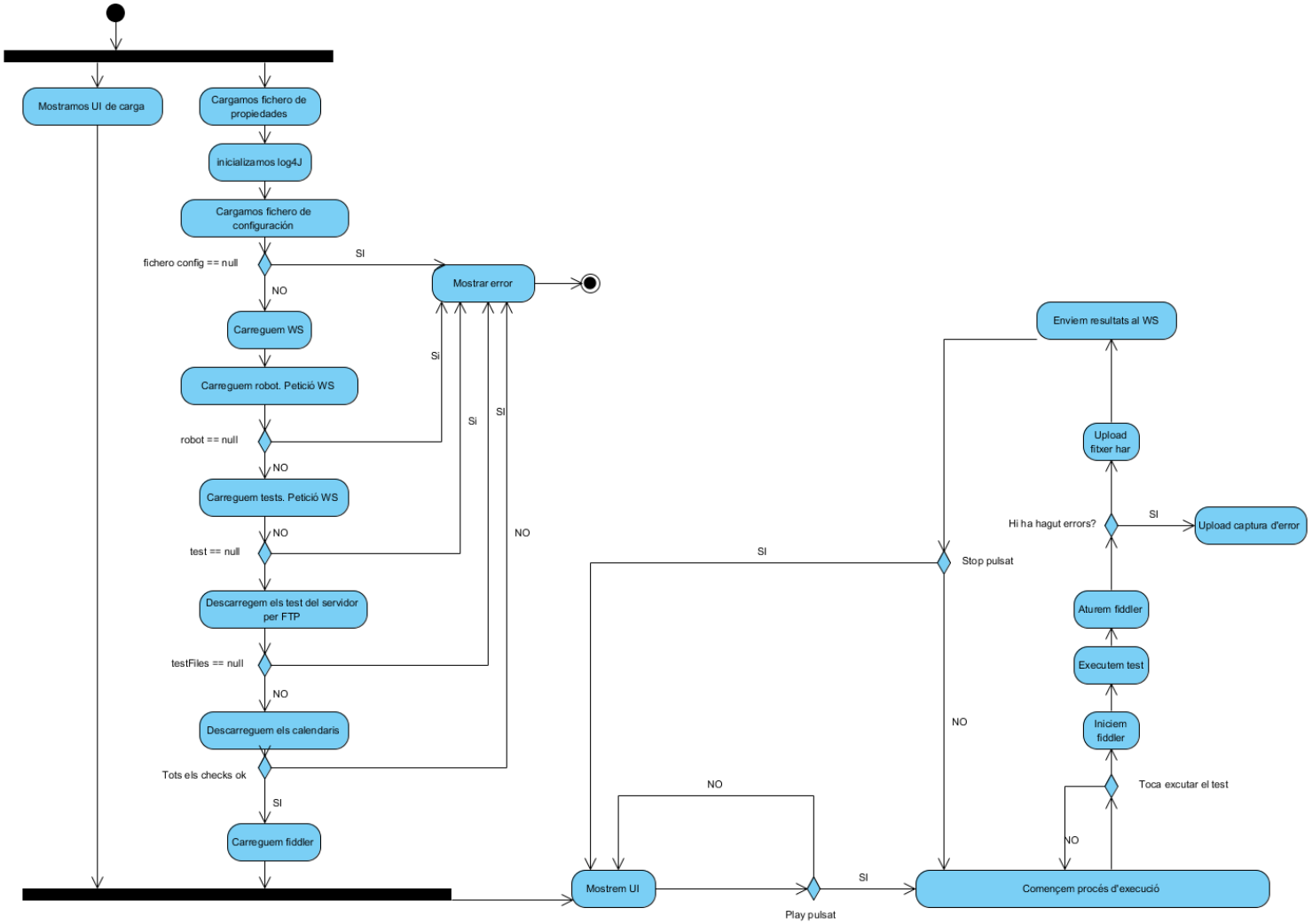


Figura 25 - Diagrama de flux Agent

5 Anàlisis i disseny del WebService

La funció principal del WebService és la de realitzar la comunicació entre els agents i la BBDD. Es podria haver optat per a que els diferents agents es comuniquessin directament amb la BBDD, però això implicaria:

- No poder distribuir els agents a qualsevol punt geogràfic, ja que tenir oberta a l'exterior la BBDD és un problema greu de seguretat. Per tant al tenir aquesta aplicació publicada a internet, i concretament al ser una aplicació web que es comunica per HTTP, els agents només requeriran d'accés a internet per a poder comunicar-se amb el WebService.
- Canvis en la BBDD implicarien haver de generar una nova versió de l'agent i distribuir aquesta nova versió a tots els robots.

Per tant l'ús d'un WebService per a comunicar els agents i la BBDD ens aporta flexibilitat, seguretat i poder distribuir els agents globalment.

5.1 Requeriments

Els requeriments per a fer funcionar aquesta eina són:

- Servidor d'aplicacions (WebSphere, JBoss, Bea WebLogic) o contenidor de servlets (Apache Tomcat)
- BBDD

5.2 Dependències

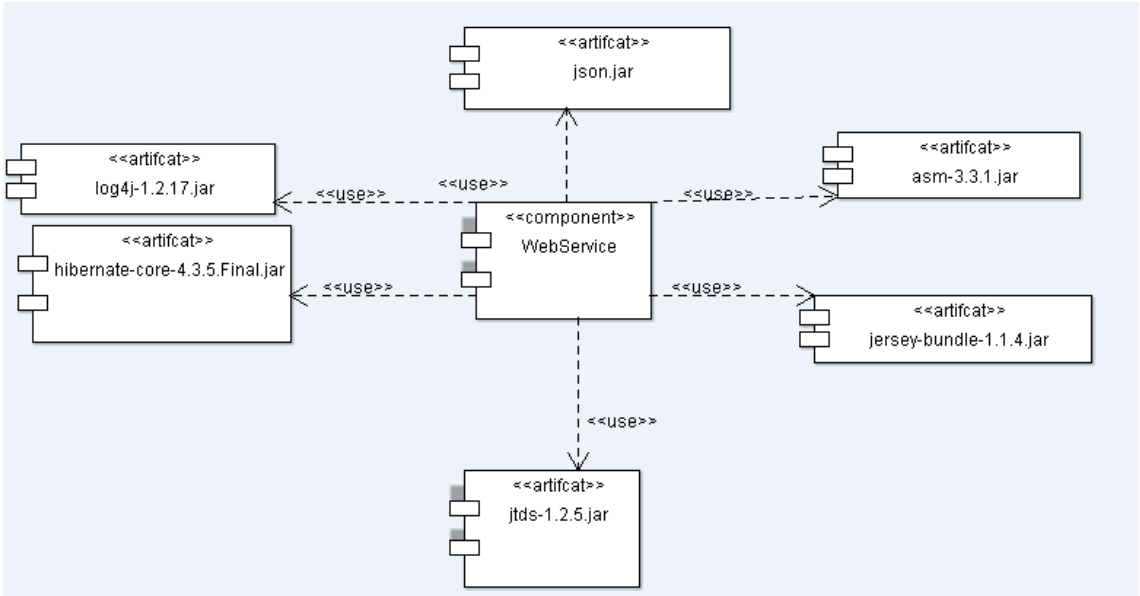


Figura 26 - Diagrama de dependències WebService

5.3 Estructura i jerarquia de paquets

Aquest projecte s'ha estructurat en diversos paquets Java, agrupant les classes en aquests paquets segons les seves funcionalitats.

Paquet	Descripció
Com.pfc.meu.WebService.Hibernate.dao	Classe principal de l'agent on s'inicialitza l'aplicació
Com.pfc.meu.WebService.Hibernate.mapping	Seguit d'objectes que l'aplicació necessita instanciar per tal d'emmagatzemar certes dades
Com.pfc.meu.WebService.petitions	Classes d'execució dels test

TAULA 3 - ESTRUCTURA DE PAQUETS WEBSERVICE

A continuació mostrem el diagrama de paquets:

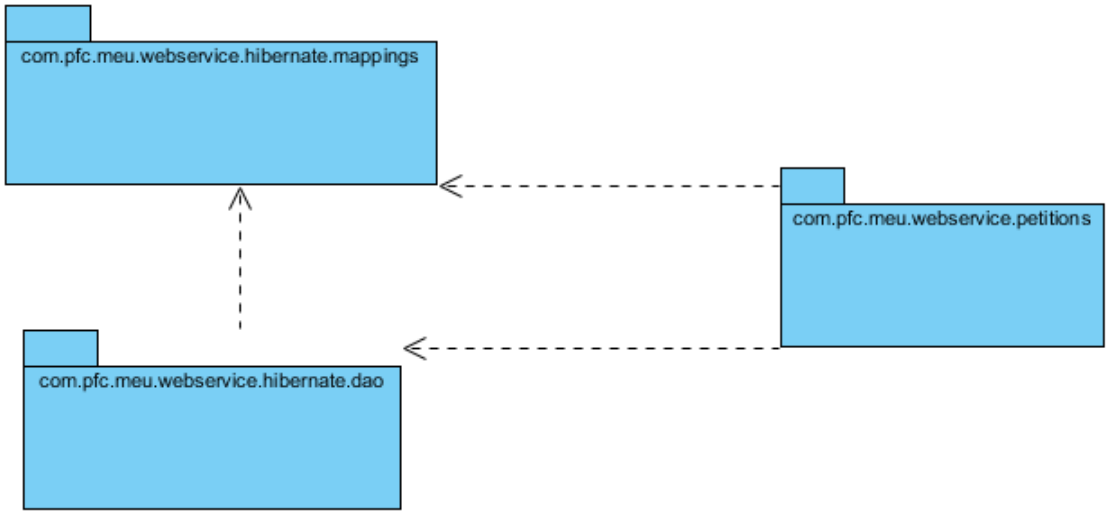


Figura 27 - Diagrama de paquets WebService

5.4 Configuració

El WebService necessita un seguit d'informació per a poder funcionar, concretament s'ha de carregar la configuració que utilitza Hibernate.

Hibernate utilitza el fitxer Hibernate.cfg.xml per a carregar la informació, aquest fitxer té la següent estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Hibernate-configuration PUBLIC
             "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
             "http://Hibernate.sourceforge.net/Hibernate-configuration-
3.0.dtd">
<Hibernate-configuration>
    <session-factory>
        <property
name="Hibernate.connection.driver_class">net.sourceforge.jtds.jdbc.Driver</pr
operty>
        <property name="Hibernate.connection.password">password </property>
        <property
name="Hibernate.connection.url">jdbc:jtds:sqlserver://IP_BBDD:Port_BBDD/Nom_B
BDD</property>
        <property name="Hibernate.connection.username">login</property>
        <property
name="Hibernate.dialect">org.Hibernate.dialect.SQLServerDialect</property>
        <mapping
resource="com/pfc/meu/WebService/Hibernate/mappings/Robot.hbm.xml"/>
        <mapping
resource="com/pfc/meu/WebService/Hibernate/mappings/Test.hbm.xml"/>
        <mapping
resource="com/pfc/meu/WebService/Hibernate/mappings/RobotTest.hbm.xml"/>
        <mapping
resource="com/pfc/meu/WebService/Hibernate/mappings/Calendario.hbm.xml"/>
        <mapping
resource="com/pfc/meu/WebService/Hibernate/mappings/TestCalendario.hbm.xml"/>
        <mapping
resource="com/pfc/meu/WebService/Hibernate/mappings/Resultado.hbm.xml"/>
    </session-factory>
</Hibernate-configuration>
```

Taula 4 - WebService: Hibernate.cfg.xml

Dins de la clau <session-factory> s'ha de configurar l'accés a la BBDD, tant les credencials, IP, port, així com el driver de connexió que s'utilitzarà.

A part de les propietats també s'ha d'incloure les taules que tenim mapejades mitjançant <mapping-resource>

5.5 Disseny

5.5.1 Diagrama de classes del WebService

A continuació mostrem el diagrama de classes del WebService:

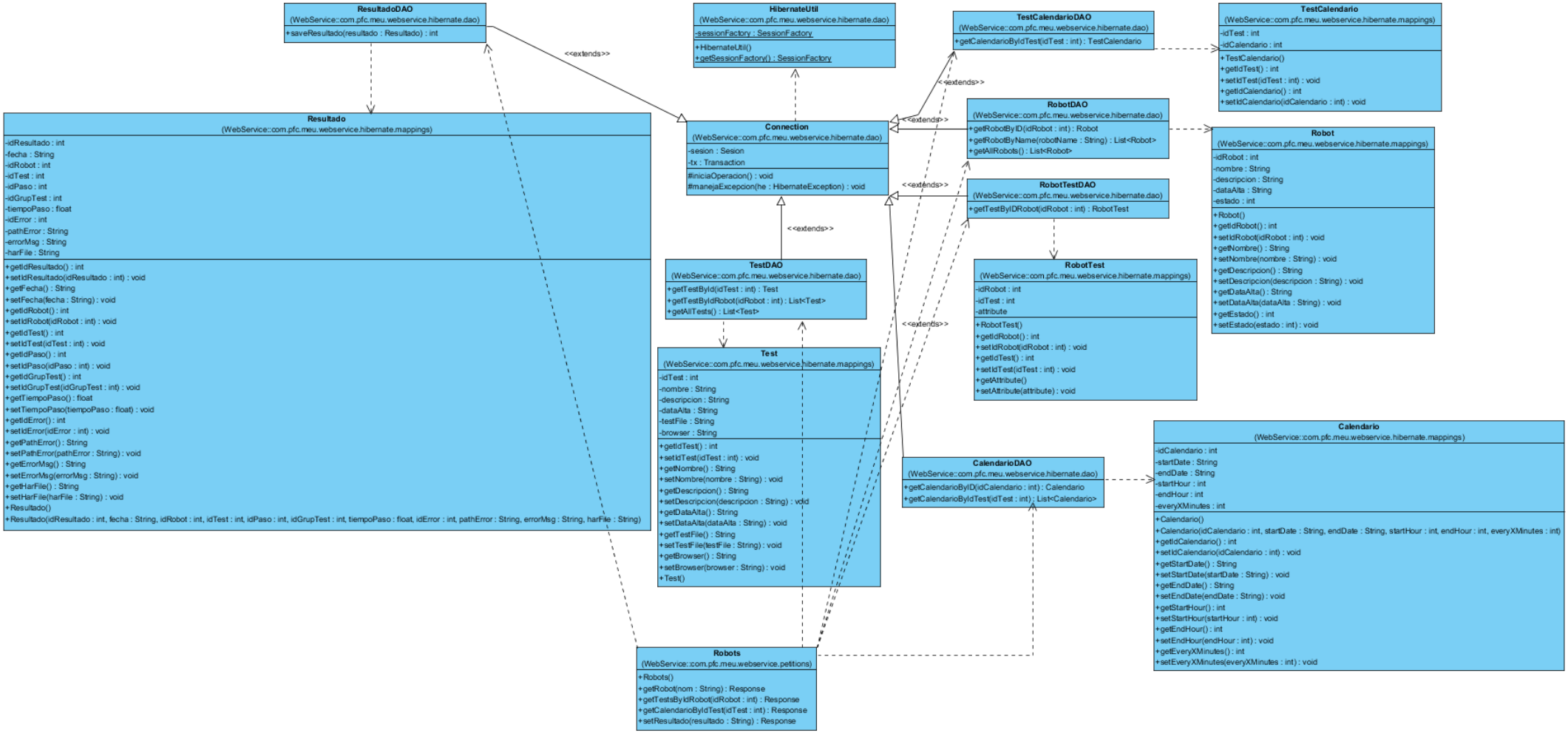


Figura 28 - Diagrama de classes WebService

5.5.2 Diagrama de seqüencia WebService

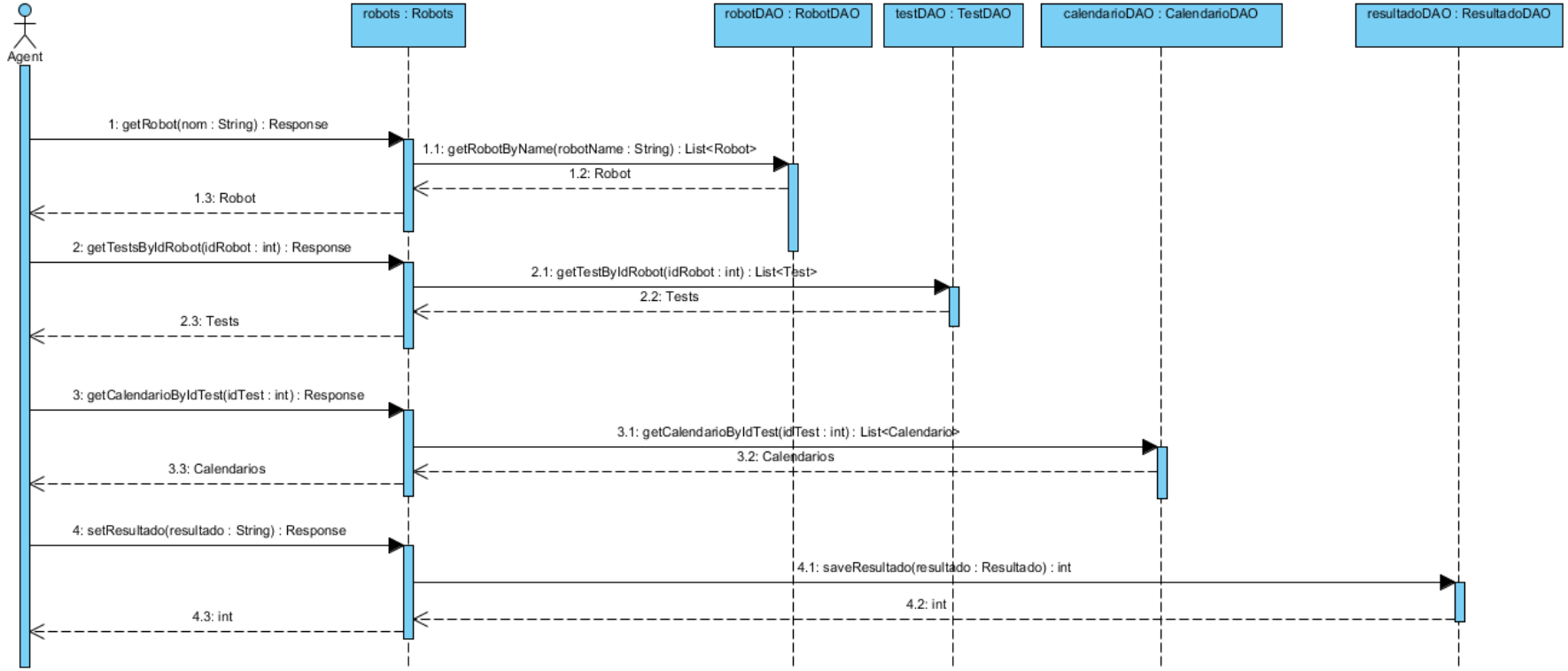


Figura 29 - Diagrama de seqüencia WebService

5.5.3 Hibernate

La funció del WebService es la realitzar la comunicació entre el WebService i la BBDD. Com ja hem comentat anteriorment es necessària una capa que comuniqui ambdós elements, per temes de seguretat (no obrir la BBDD a internet) com per poder instal·lar un robot a qualsevol part del món, sempre que tingui una connexió a internet.

Per tal que el WebService accedeixi a la BBDD existeixen nombroses alternatives, des d'utilitzar un JDBC i realitzar nosaltres el control de connexió a la BBDD, crear classes per a realitzar consultes, etc. Però actualment existeixen diverses solucions, que no només faciliten la tasca de connexió sinó que ens permeten modelar l'estructura de la nostre BBDD a dins de la nostre aplicació. Per aquests motius s'ha escollit Hibernate a l'oferir-nos un mapping objecte-relacional.

Hem estructurat les nostres classes Hibernate de la següent forma:

- HibernateUtils: aquesta classe s'encarrega d'establir la connexió amb la BBDD.
- Classes DAO: tenim una classe DAO per cada un dels objectes de la BBDD, la seva funció és la de llançar les consultes a la BBDD i retornant-nos els resultats, en objectes de la nostre aplicació, per a poder treballar amb ells directament, sense haver de fer parsings.
- Classes BBDD: a l'igual que amb les classes DAO tenim una per cada element de la nostre BBDD, aquestes classes representen en codi l'estructura de la base de dades, convertint en objectes el que tenim a la BBDD, permetent-nos poder treballar amb la BBDD com si d'objectes es tractés.

5.5.4 REST WebService

Com ja hem explicat s'ha escollit per a desenvolupar aquest WebService la tecnologia REST. Aquesta ens permet publicar serveis per a que altres aplicacions, en el nostre cas l'agent, els consumeixi.

El funcionament és molt senzill. Tenim centralitzada tota la lògica en una única classe anomenada Robots. En aquesta definim els serveis que volem publicar. La forma de fer-ho és, agafant com exemple el servei getRobot que ens retorna un objecte Robot passant com a paràmetre el seu nom.

Primer de tot hem de definir on es pública el servei, això és el paràmetre que posarem a la URL per a cridar-lo (en aquest cas a la URL només li haurem de passar el paràmetre del nom del robot, així com el mètode de petició (s'ha escollit GET per motius de debbuging, ja que ens permet llançar còmodament comandes al WebService directament des del navegador. En un entorn productiu hauria de substituir-se per POST per raons de seguretat), finalment escollim el llenguatge amb el que respondrà el WebService, en aquest cas JSON.

```
@Path("{nom}")
@GET
@Produces("application/json")
```

A continuació definim la nostre funció:

```
public Response getRobot(@PathParam("nom") String nom) throws
JSONException {
```

es tracta d'una funció pública on definim que rebrà un paràmetre que anomenem nom i que serà del tipus String.

Dins d'aquesta funció ens encarreguem de recollir el nom rebut, obtenir l'objecte Robot utilitzant les funcionalitats de Hibernate, i retornar la resposta amb JSON al client.

REST ja s'encarrega ell sol de que al rebre una petició del servei sap a quina funció l'ha d'enviar, facilitant-nos molt la feina.

6 Anàlisis i disseny de la consola d'administració

L'administració del conjunt de robots, necessita d'una eina centralitzada que permeti realitzar totes les accions necessàries del dia a dia.

Per aquest motiu es decideix crear una eina centralitzada de gestió o administració de la solució, que permetrà realitzar aquestes accions bàsiques o complexes, com pot ser donar d'alta un nou robot, crear un nou test, modificar un test existent, etc.

Actualment s'ha dotat a la consola d'administració d'un seguit de funcions bàsiques, que han de ser ampliades en un futur.

6.1 Requeriments

Els requeriments per a fer funcionar aquesta eina són:

- Servidor d'aplicacions (WebSphere, JBoss, Bea WebLogic) o contenidor de servlets (Apache Tomcat)
- BBDD

6.2 Dependències

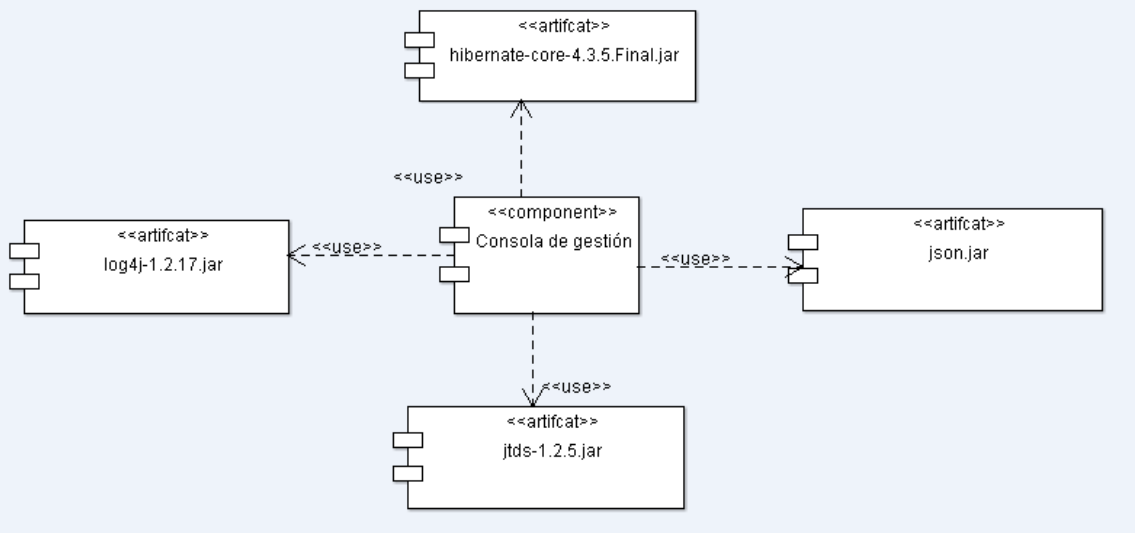


Figura 30 - Diagrama de dependències consola d'Administració

6.3 Estructura i jerarquia de paquets

Aquest projecte s'ha estructurat en diversos paquets Java, agrupant les classes en aquests paquets segons les seves funcionalitats.

Paquet	Descripció
Com.classes	Classes d'autenticació de l'eina
Com.Hibernate.dao	Classes d'accés a les dades - Hibernate
Com.Hibernate.map	Mapeig de les classes – Hibernate
Com.servlets	Servlets de l'aplicació

TAULA 5 - ESTRUCTURA DE PAQUETS ADMINISTRACIÓ

A continuació mostrem el diagrama de paquets:

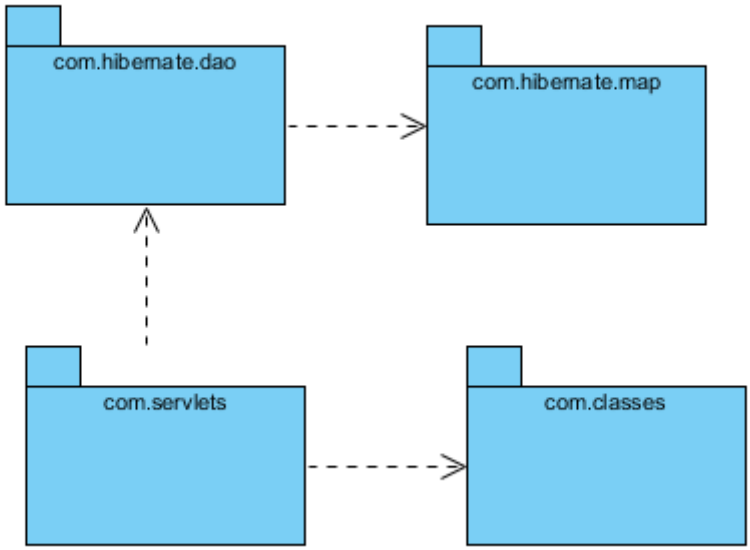


Figura 31 - Diagrama de paquets Administració

6.4 Configuració

La consola d'administració necessita un seguit d'informació per a poder funcionar, concretament s'ha de carregar la configuració que utilitza Hibernate.

Hibernate utilitza el fitxer Hibernate.cfg.xml per a carregar la informació, aquest fitxer té la següent estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Hibernate-configuration PUBLIC
             "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
             "http://Hibernate.sourceforge.net/Hibernate-configuration-
3.0.dtd">
<Hibernate-configuration>
    <session-factory>
        <property
name="Hibernate.connection.driver_class">net.sourceforge.jtds.jdbc.Driver</pr
operty>
        <property name="Hibernate.connection.password">password </property>
        <property
name="Hibernate.connection.url">jdbc:jtds:sqlserver://IP_BBDD:Port_BBDD/Nom_B
BBDD</property>
        <property name="Hibernate.connection.username">login</property>
        <property
name="Hibernate.dialect">org.Hibernate.dialect.SQLServerDialect</property>
        <mapping resource="com/Hibernate/map/Robot.hbm.xml"/>
        <mapping resource="com/Hibernate/map/Usuario.hbm.xml"/>
        <mapping resource="com/Hibernate/map/Test.hbm.xml"/>
        <mapping resource="com/Hibernate/map/Paso.hbm.xml"/>
        <mapping resource="com/Hibernate/map/RobotUsuario.hbm.xml"/>
        <mapping resource="com/Hibernate/map/TestUsuario.hbm.xml"/>
        <mapping resource="com/Hibernate/map/RobotTest.hbm.xml"/>
        <mapping resource="com/Hibernate/map/Calendario.hbm.xml"/>
        <mapping resource="com/Hibernate/map/TestCalendario.hbm.xml"/>
    </session-factory>
</Hibernate-configuration>
```

Taula 6 - Consola d'administració Hibernate.cfg.xml

Dins de la clau <session-factory> s'ha de configurar l'accés a la BBDD, tant les credencials, IP, port, així com el driver de connexió que s'utilitzarà.

A part de les propietats també s'ha d'incloure les taules que tenim mapejades mitjançant <mapping-resource>

6.5 Disseny

6.5.1 Disseny responsive

Per tal de fer el disseny d'aquesta aplicació s'ha optat per utilitzar un framework per a la part front-end anomenat bootstrap, i que ja hem comentat amb més detall anteriorment.

El que m'ha dut a escollir aquest framework és la seva capacitat per a construir dissenys responsive, permetent-nos amb un únic disseny que l'aplicació es mostri correctament en qualsevol dispositiu o mida de pantalla evitant haver de fer dissenys diferents per a la versió d'escriptori i per a terminals mòbils.

Actualment l'accés a aplicacions o pàgines web des de dispositius mòbils està a l'ordre del dia, quasi tothom disposa de terminals mòbils o tauletes i accedeixen a internet des d'aquests, amb la comoditat de poder fer-ho des de qualsevol lloc, per tant és molt important que l'eina sigui accessible des de qualsevol terminal i es vegi correctament.

Bootstrap es basa en les últimes tecnologies web con són HTML5 i CSS3.

S'ha volgut tenir molta cura amb la part de front-end i fer-la el més estàndard possible per tal d'evitar incompatibilitats amb els diferents navegadors. S'han realitzat proves amb els navegadors més utilitzats per validar que funciones correctament amb tots ells.

Boostrap també ha ajudat a crear una interfície gràfica molt usable i fàcil d'utilitzar. També hem volgut que fos atractiva, ja que és la part que els clients veuran i utilitzaran habitualment, per tant ha de ser visualment atractiva. Boostrap ens permet de forma relativament fàcil crear interfícies gràfiques atractives i usables.

6.5.2 Hibernate

Per al portal d'administració també s'ha utilitzat el framework Hibernate per tal de fer el mapping objecte-relacional. S'ha seguit una estructura igual a la utilitzada amb el WebService. Una classe per a centralitzar el connexionat cap a la BBDD, un seguit de classes DAO per accedir a les dades, i un altre seguit de classes on hem replicat l'estructura de la BBDD al nostre codi.

S'ha intentat acorar al màxim tot l'accés a la base de dades i el mapping dels objectes d'aquesta per tal de poder simplificar al màxim el codi.

6.5.3 Diagrama de classes

A continuació podem veure el diagrama de classes de la consola d'administració

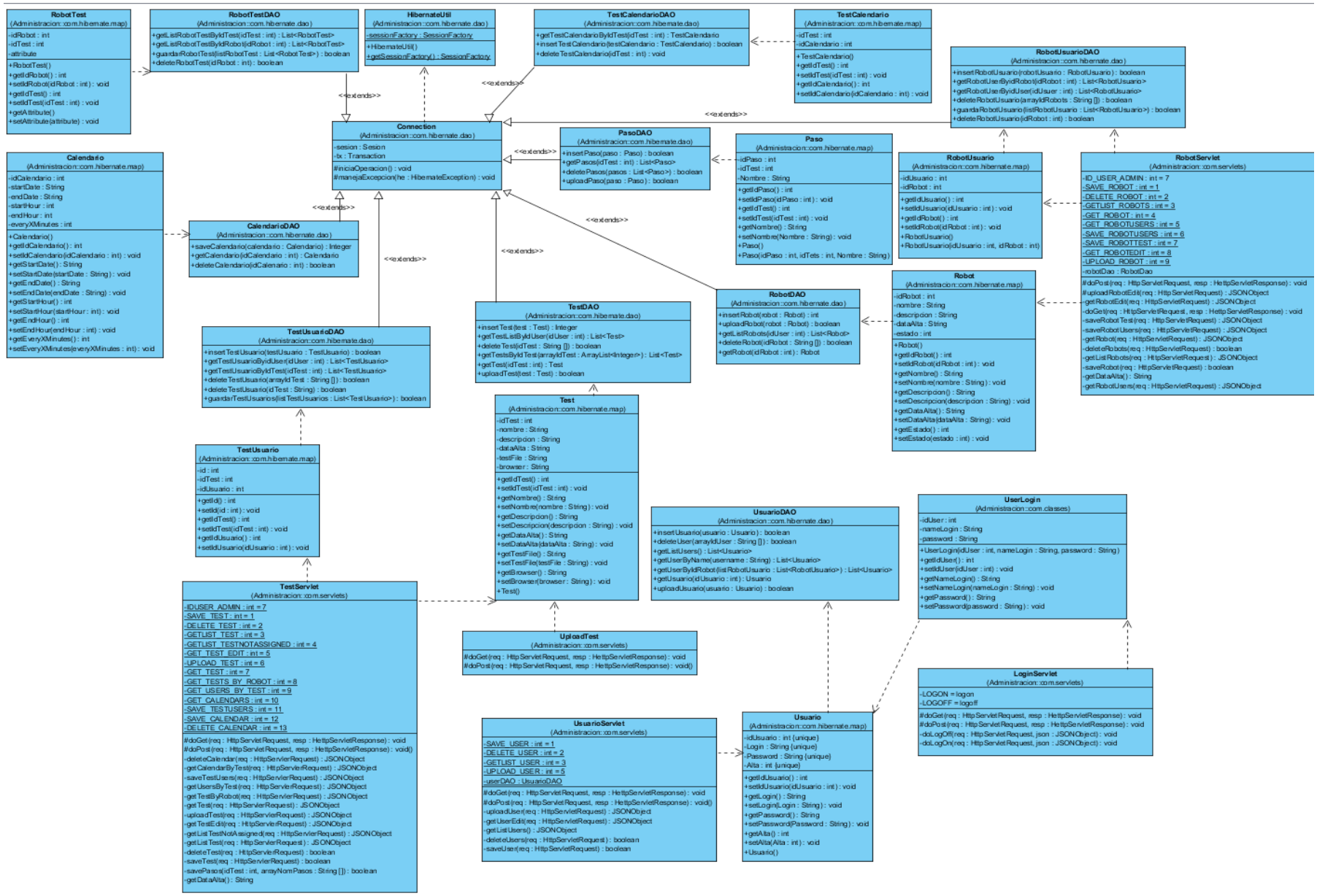


Figura 32 - Diagrama de classes consola d'Administració

6.5.4 Diagrama de casos d'ús

A continuació es pot veure el diagrama de casos d'ús de la consola d'administració:

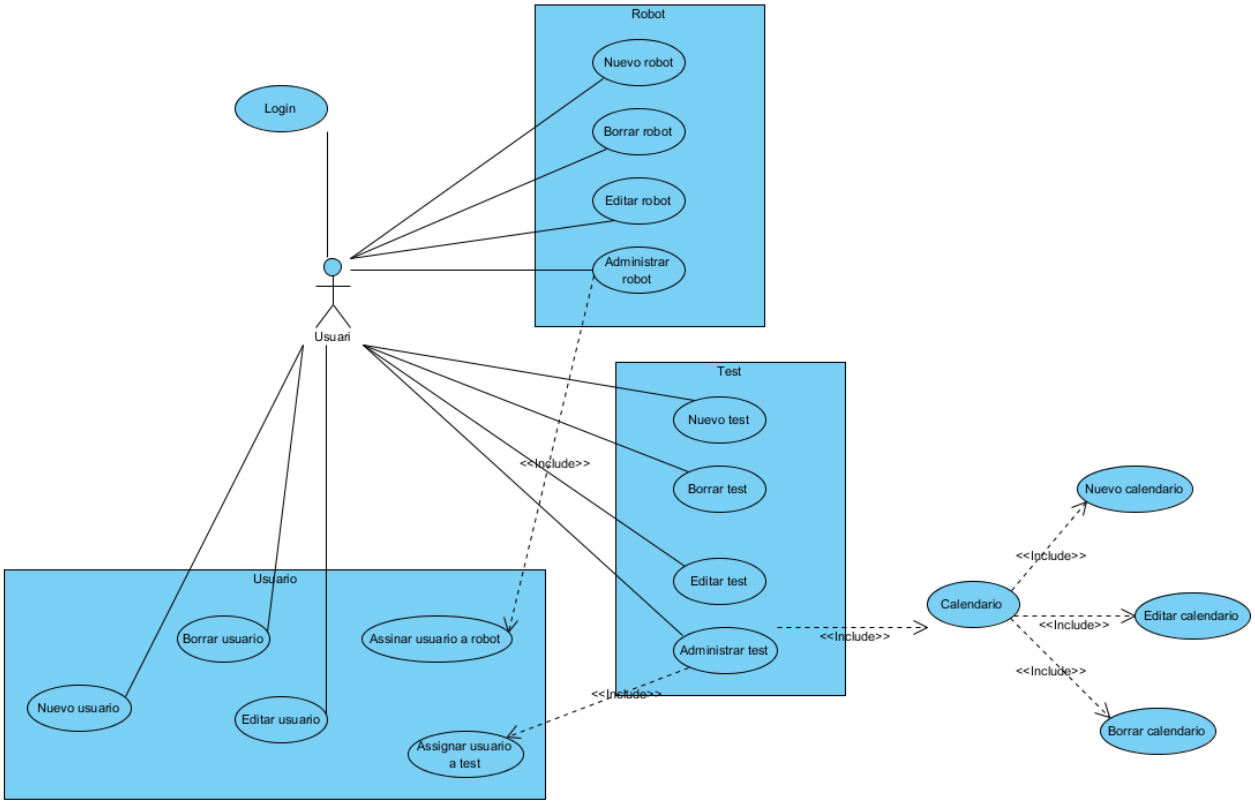


Figura 33 - Diagrama de casos d'ús consola d'administració

6.5.5 Diagrama de seqüències

6.5.5.1 Login

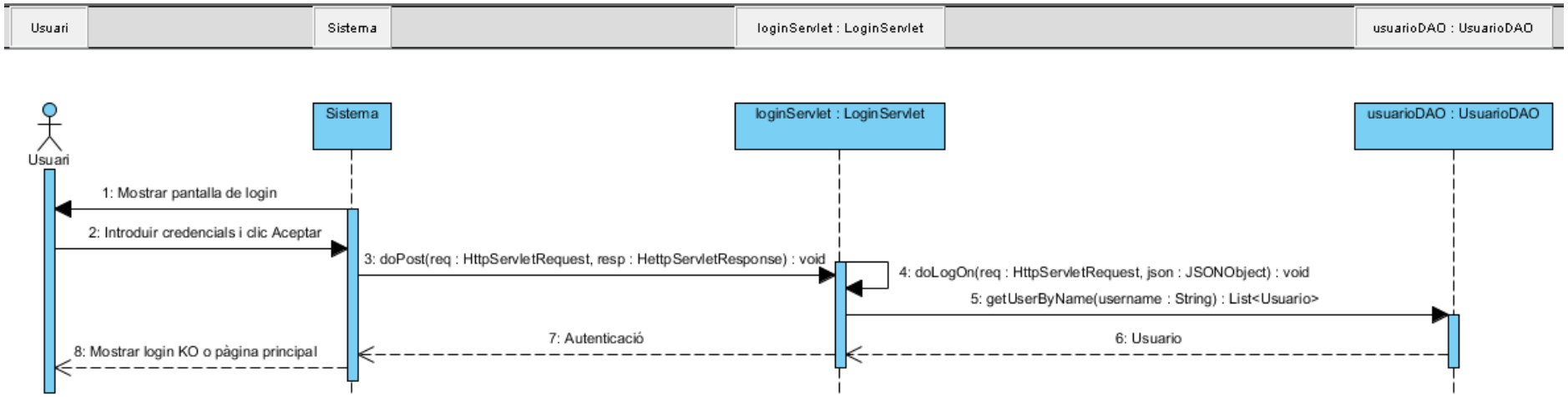


Figura 34 - Diagrama de seqüencies login

6.5.6 Nou Robot

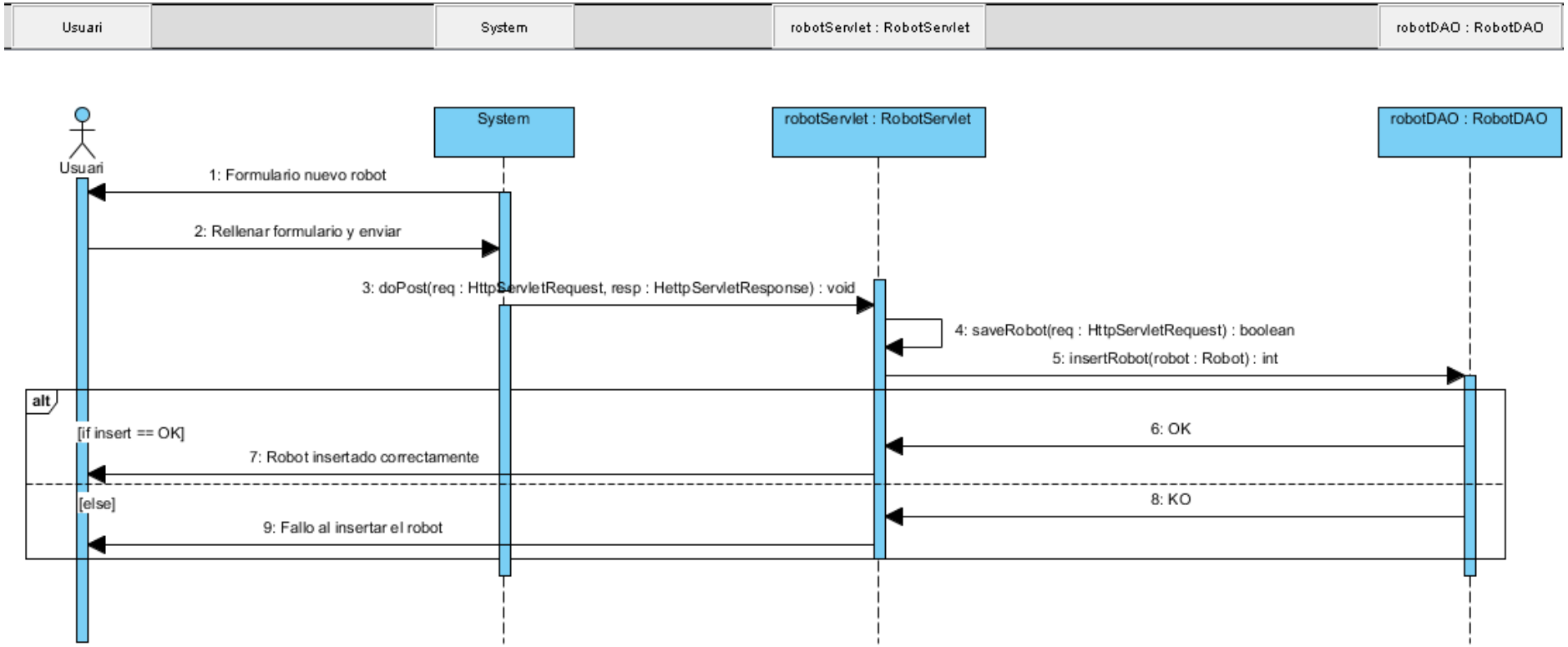


Figura 35 - Diagrama de seqüencies nou robot

6.5.6.1 Esborrar robot

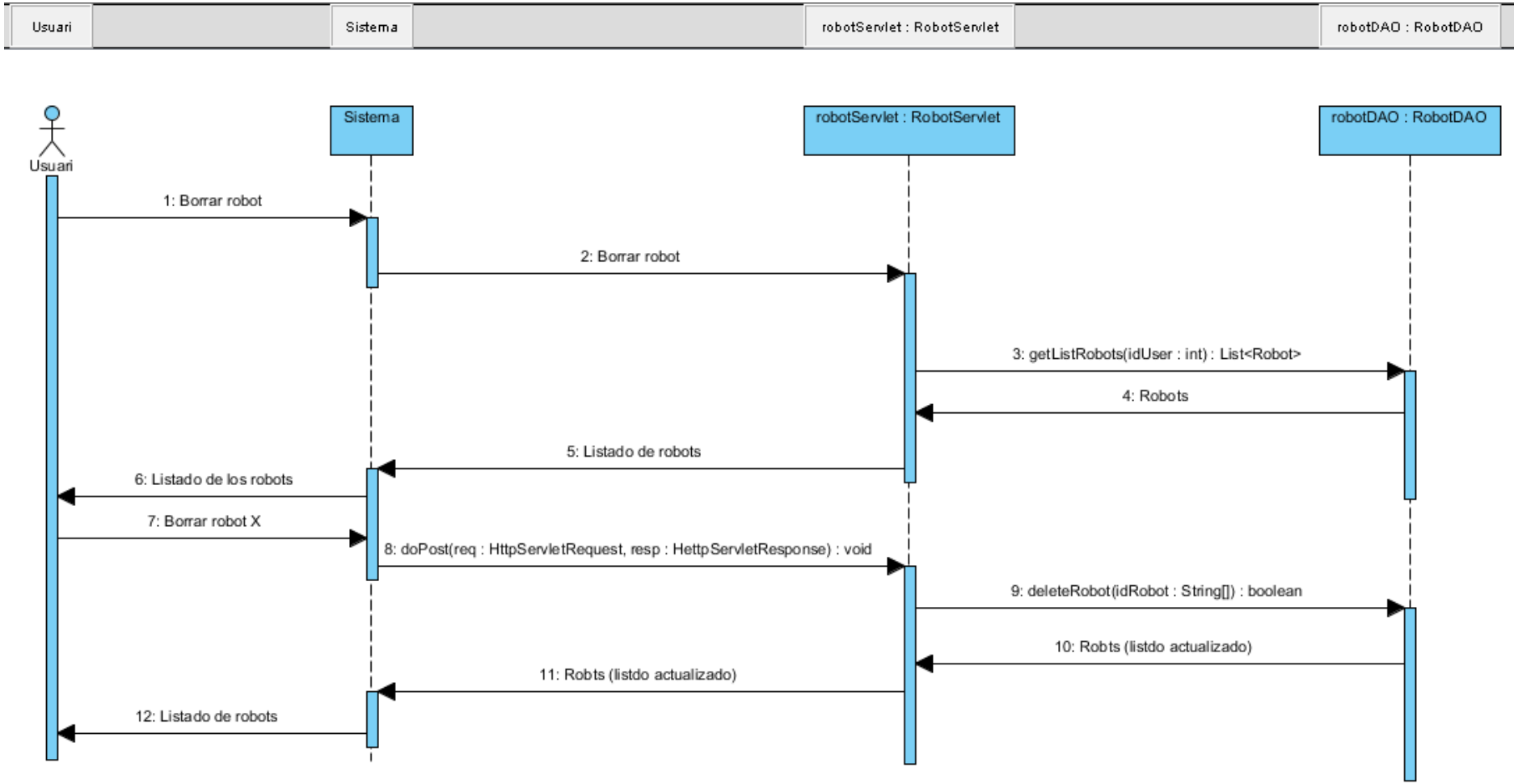


Figura 36 - Diagrama de seqüencies esborrar robot

6.5.6.2 Modificar robot

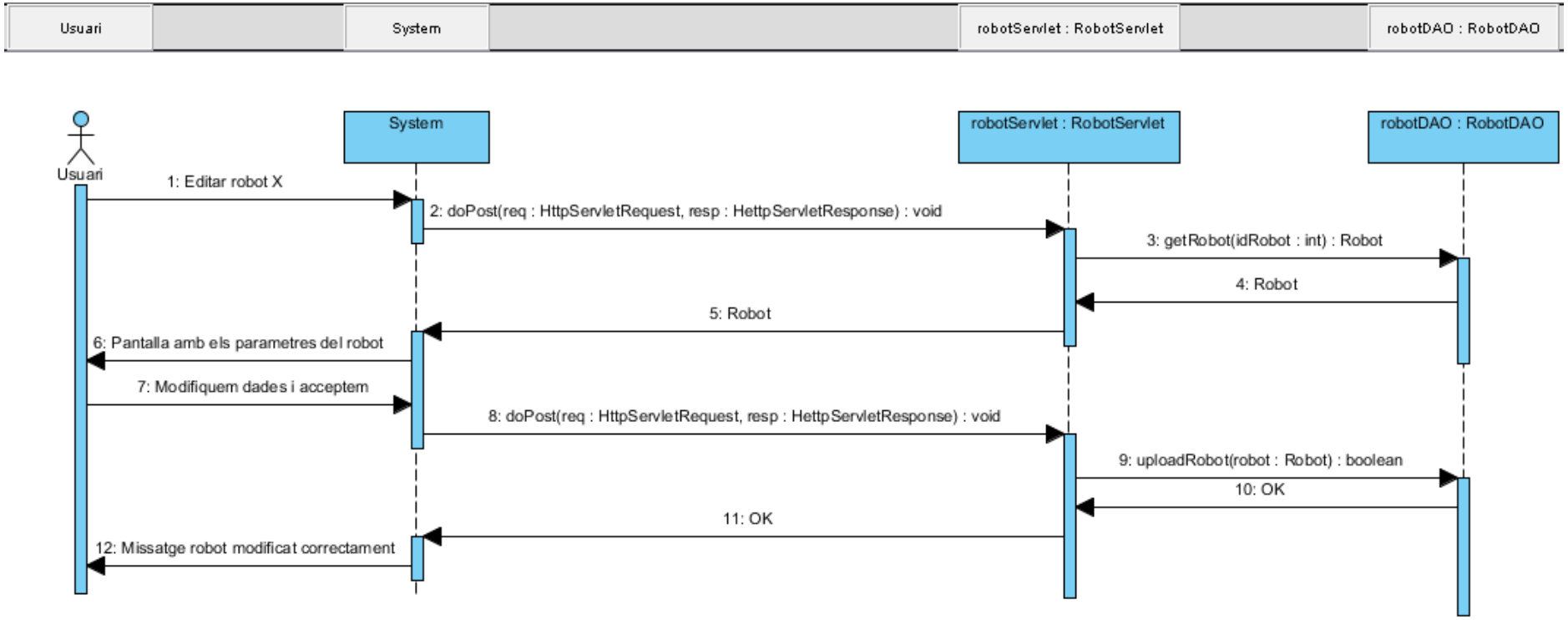


Figura 37 - Diagrama de seqüencia modificar robot

6.5.7 Nou calendari d'un test

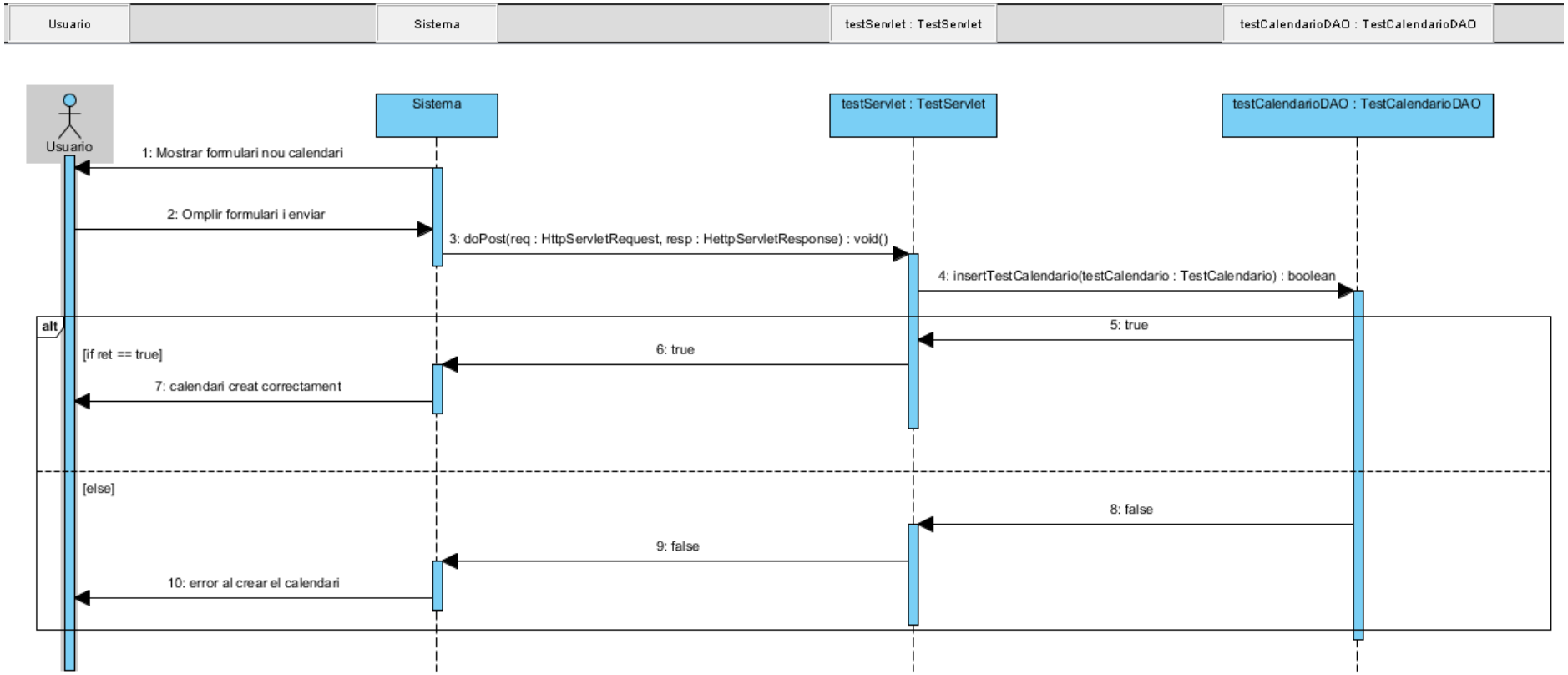


Figura 38 - Diagrama de seqüencia nou calendari d'un test

6.5.8 Editar calendari d'un test

Figura 39 Diagrama de seqüencia editar calendari de test

6.5.9 Esborrar calendari de test

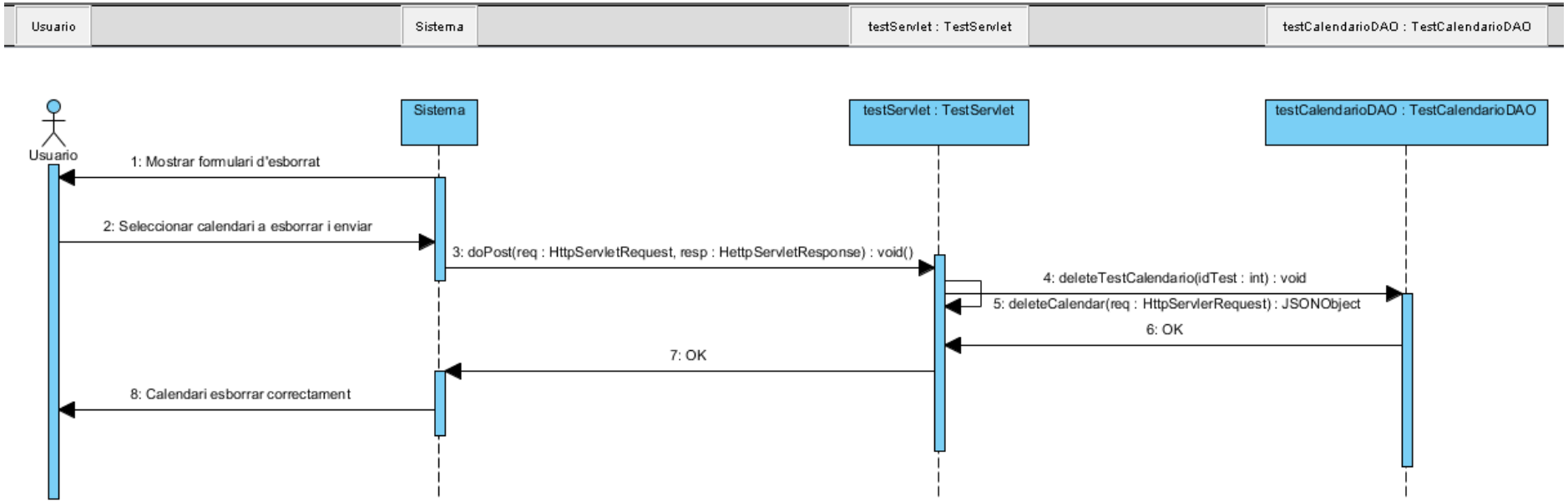


Figura 40 - Diagrama de seqüencia esborrar calendari de test

6.5.10 Nou test

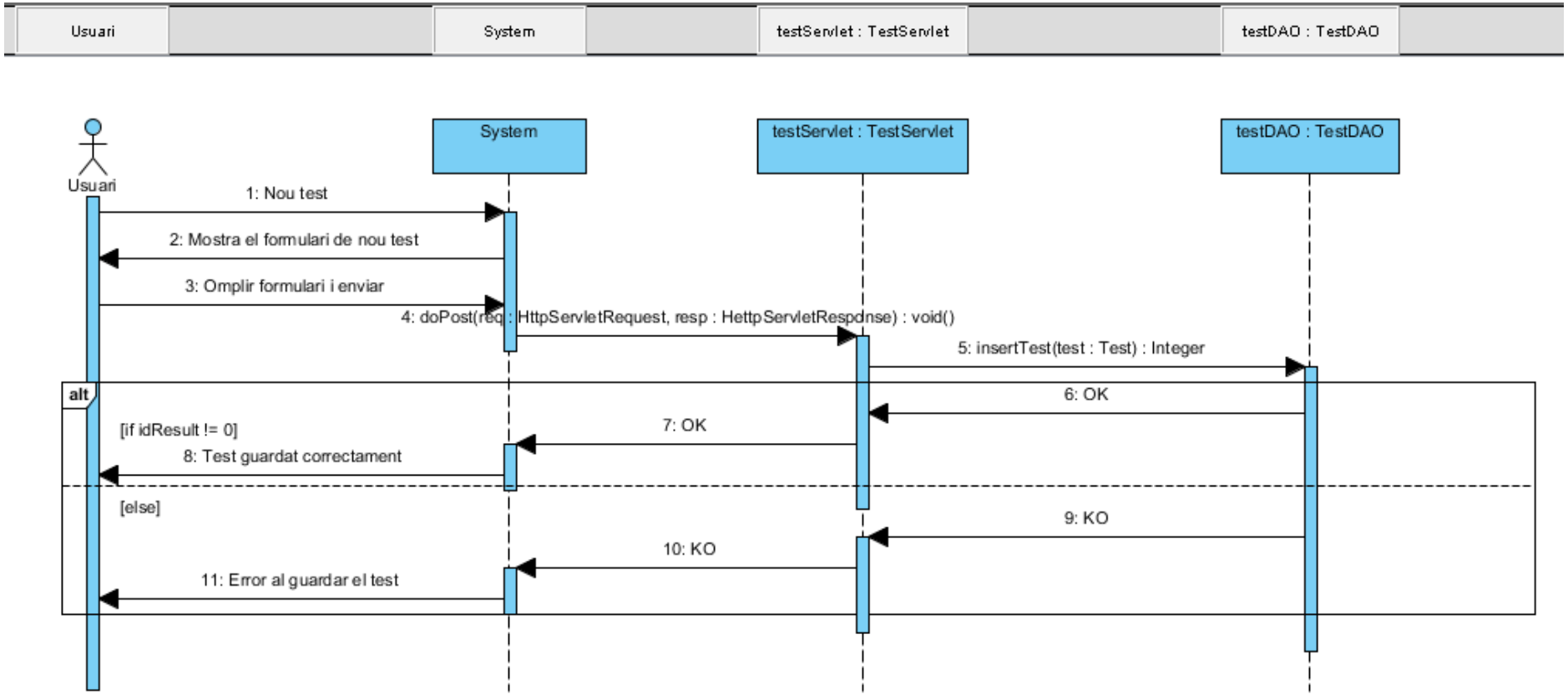


Figura 41 - Diagrama de seqüencia nou test

6.5.11 Editar test

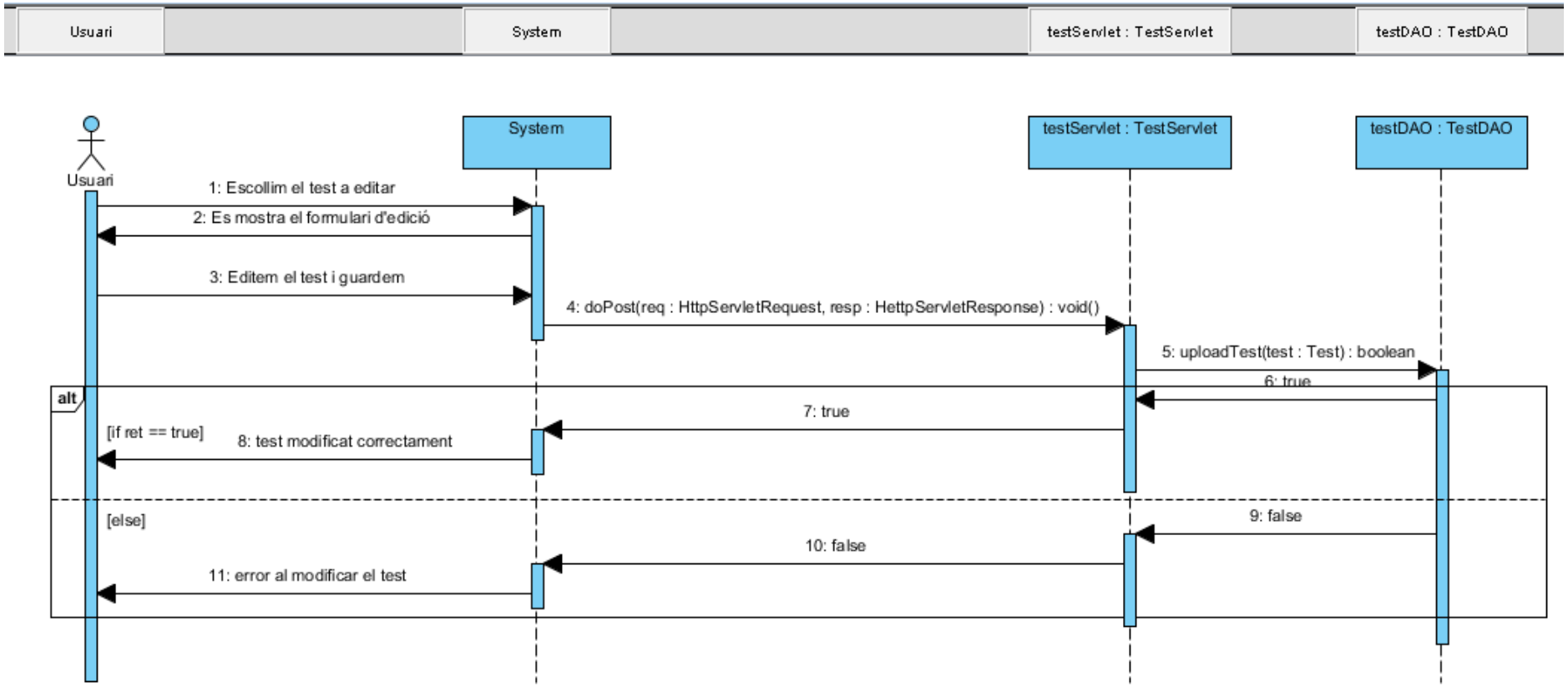


Figura 42 - Diagrama de seqüencia editar test

6.5.12 Esborrar test

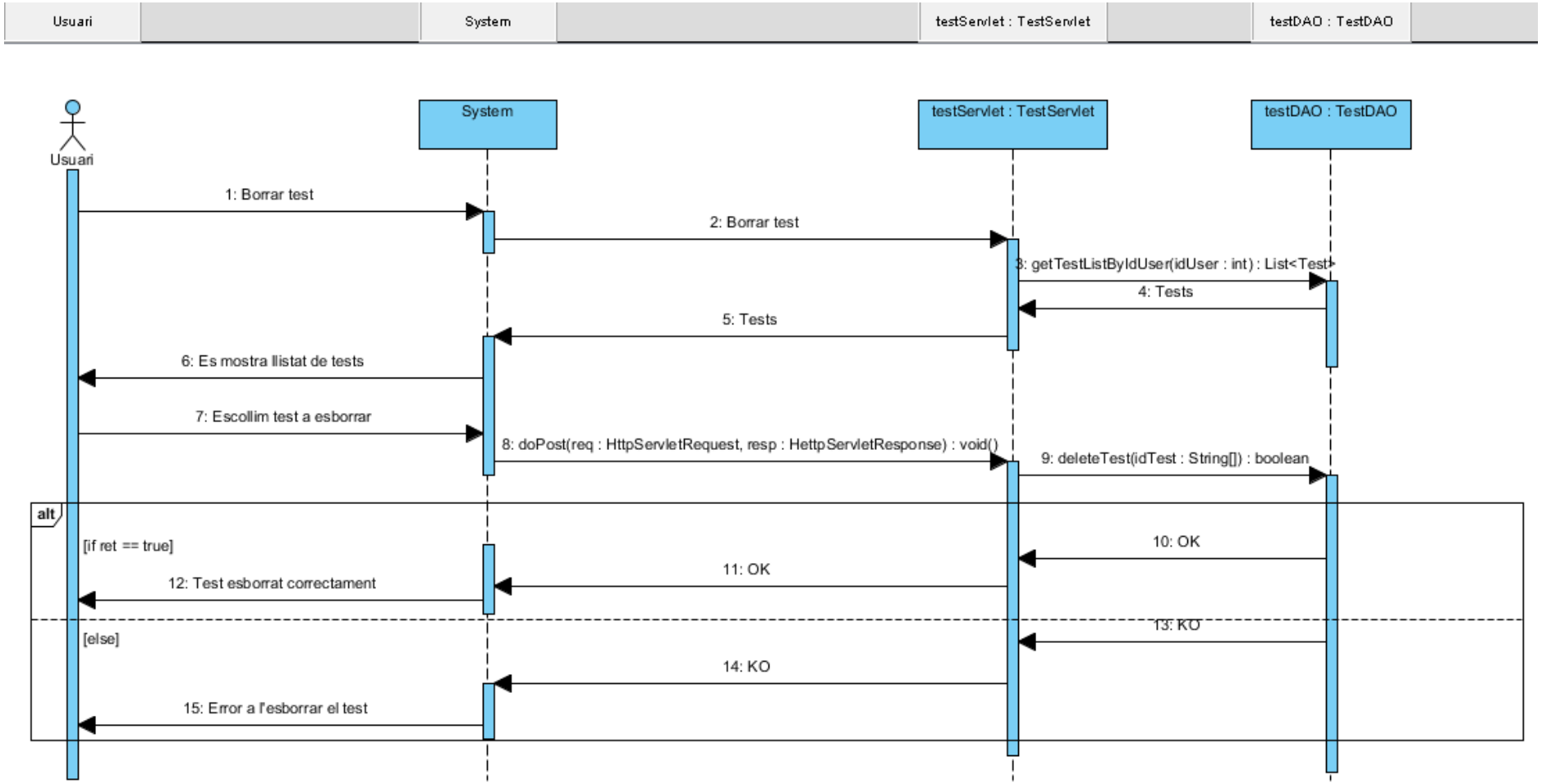


Figura 43 - Diagrama de seqüencia esborrar test

6.5.13 Nou usuari

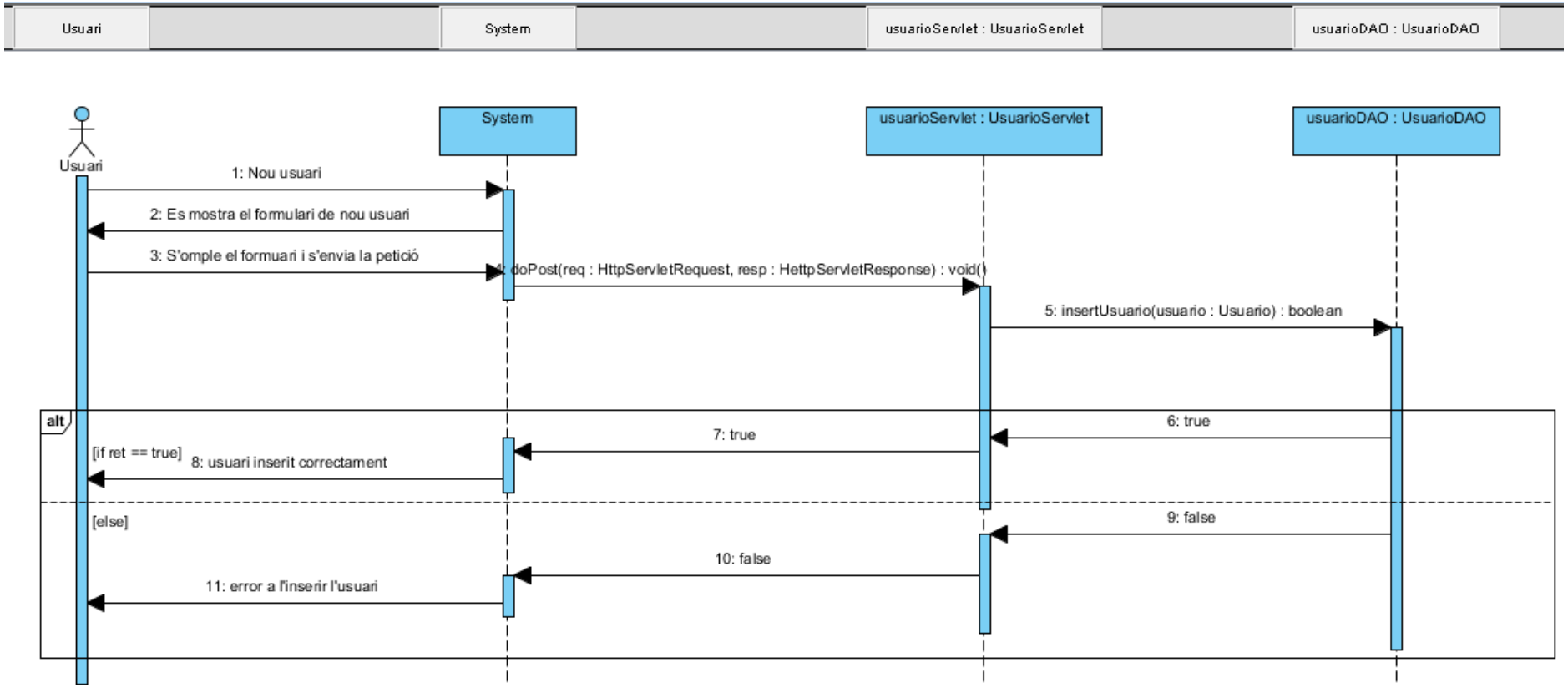


Figura 44 Diagrama de seqüencia nou usuari

6.5.14 Editar usuari

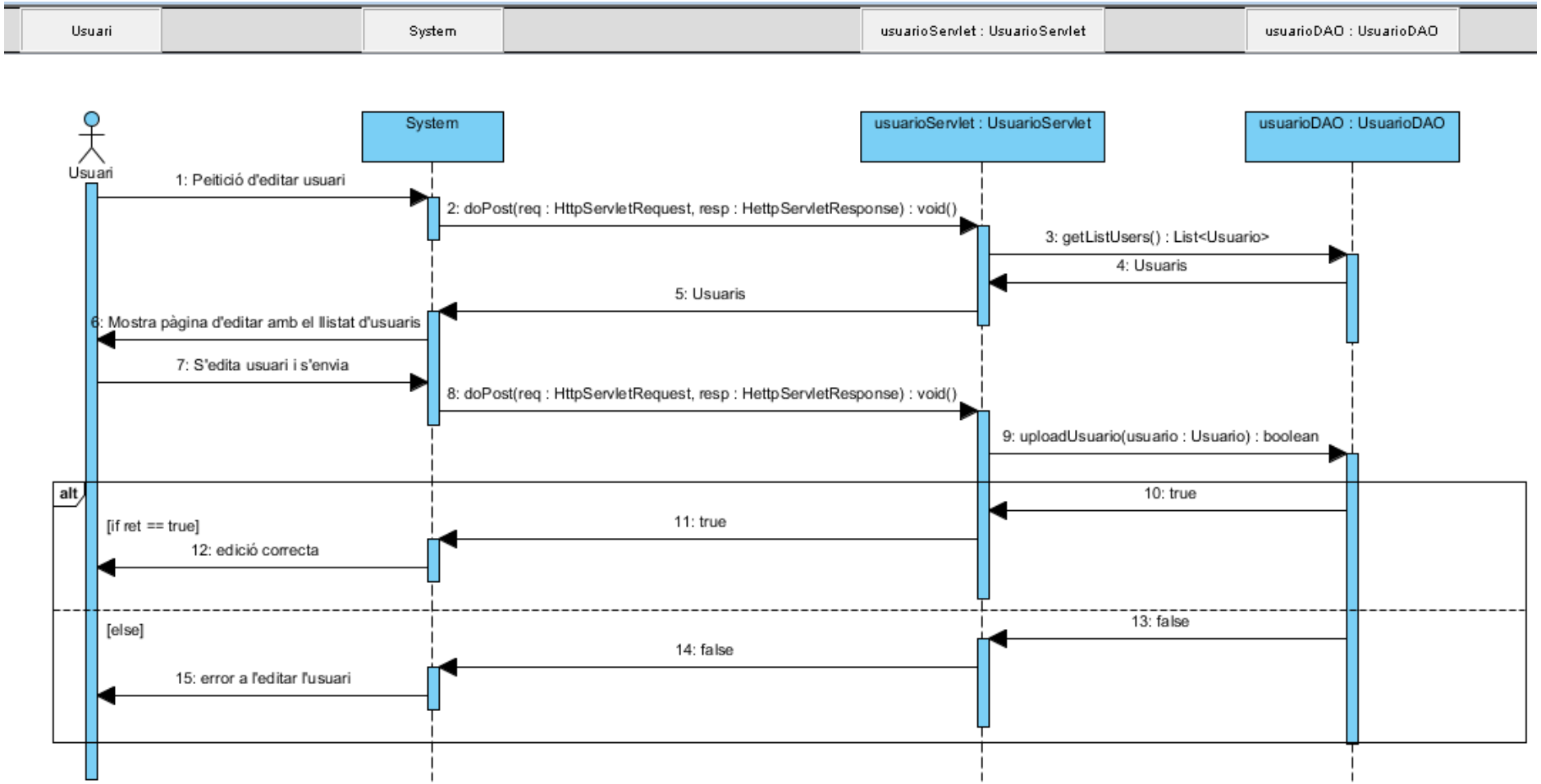


Figura 45 - Diagrama de seqüencia editar usuari

6.5.15 Esborrar usuari

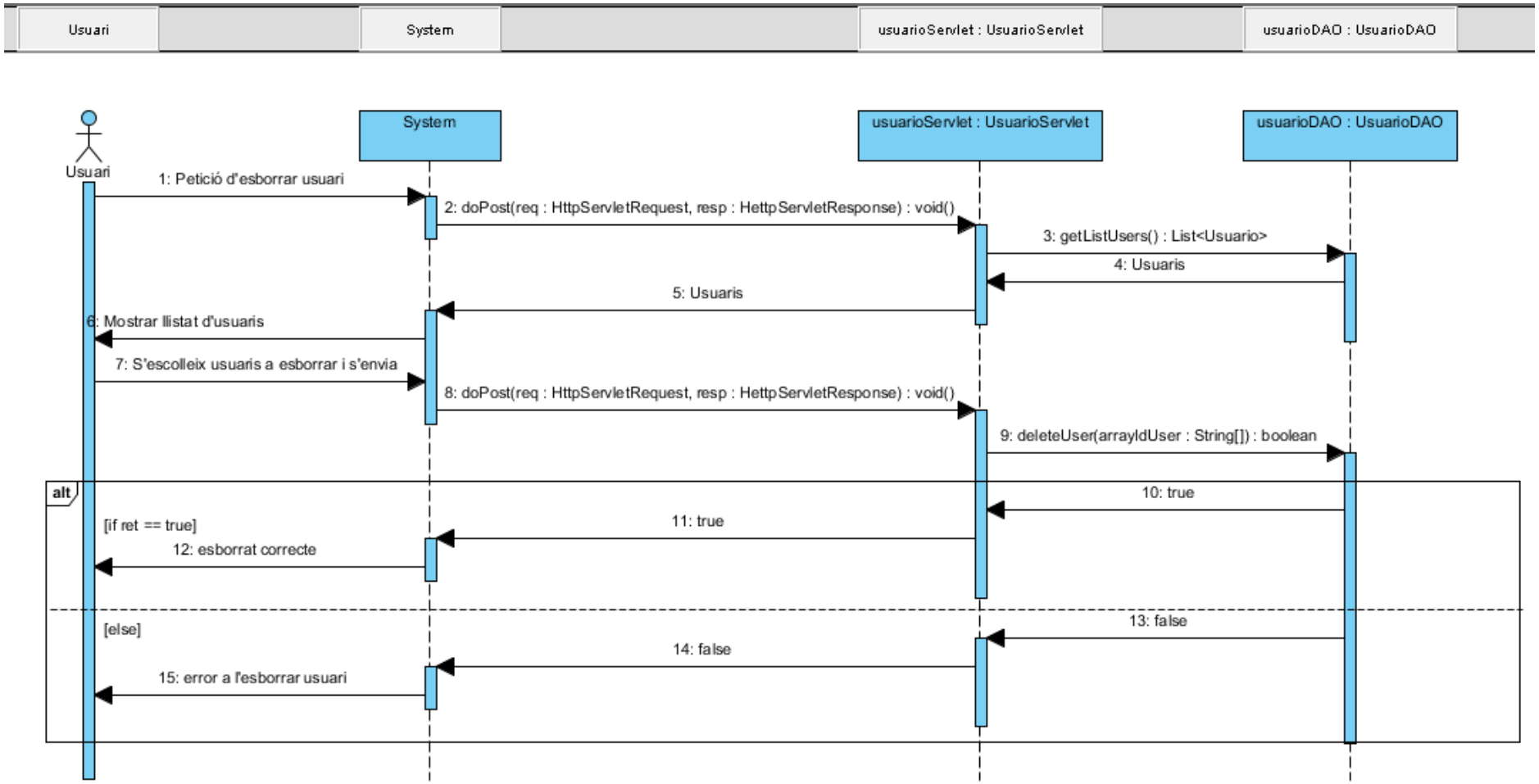


Figura 46 - Diagrama de seqüencia esborrar usuari

7 Base de dades

Els diferents robots estan pensats per a un mode de funcionament continu. Aquests aniran generant resultats amb les mètriques recollides en les diferents execucions. Si tenim diversos robots funcionant de forma continuada es generarà una gran quantitat d'informació que ha de ser emmagatzemada d'alguna forma. Una opció es guardar localment tota aquesta informació, però això no ens permetria poder explotar amb posterioritat totes les da des recollides. Per aquest motiu s'ha pensat que la millor solució és utilitzar una base de dades relacional que centralitzi tota la informació recollida. Concretament s'ha utilitzat SQL Server de Microsoft. El principal motiu d'escollir aquesta solució i no un altre com podria ser MySQL és el coneixement personal que tinc sobre SQL Server. En l'àmbit laboral treballo diàriament amb aquesta eina, cosa que em dona un alt coneixement sobre aquest.

S'ha dissenyat la BBDD de forma relacional,

7.1 Disseny de les taules

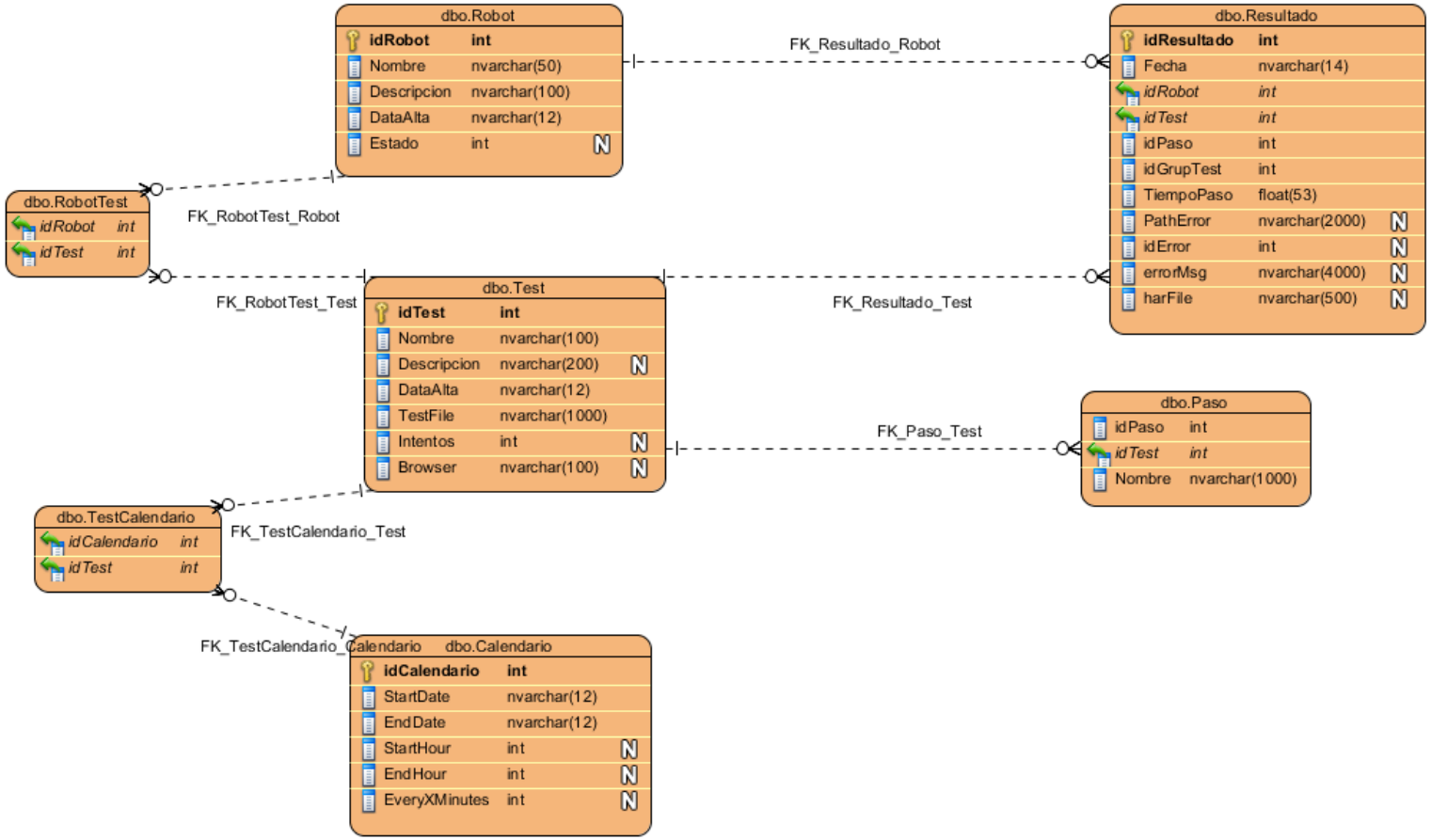


Figura 47 - Diagrama de base de dades

8 Conclusions

El desenvolupament d'aquest projecte ha estat un gran repte personal. Fa anys que treballo amb aplicacions de monitorització d'experiència d'usuari, tant en l'explotació de serveis, com desenvolupant noves funcionalitats sobre un software ja existent (no en Java), per tant tinc experiència i coneixements en aquest àmbit. Quan treballes amb software en el que no has participat en l'etapa de disseny i desenvolupament et trobes amb mancances que posteriorment són difícils de solucionar, ja sigui per complexitat i/o bé per cost. Per tant interioritzes tot allò que voldries en una solució d'aquest tipus si poguessis fer-la des de 0.

Quan es va presentar el moment de realitzar el meu PFC, no vaig tenir dubte que volia poder desenvolupar una eina d'aquest tipus, i que complís tot allò que crec personalment i els anys d'experiència m'han ensenyat, requereix una solució d'aquest tipus, tant per a les persones que exploten el servei i fan el manteniment de la solució com per als clients.

Revisant les fites que es volien assolir inicialment, i després de moltes setmanes de feina, veig que aquestes eren massa ambicioses. No s'han pogut assolir totes elles, com per exemple la integració amb una eina de reporting que pertany a l'empresa on treballo, la integració amb un sistema d'alertes així com realitzar els processos d 'historificació de les mètriques recollides. Tot i això, aquestes fites eren complementaries al projecte en si. Per tant un cop vaig veure que no totes es podrien complir vaig preferir deixar-les de banda i centrar-me en el disseny i desenvolupament del nucli de la solució.

El motiu principal va ser voler que aquest nucli complís amb un seguit de requisits, bàsics en la meva opinió:

- Estabilitat: l'eina havia de ser molt estable, per tant el codi havia de controlar tots els possibles errors
- Mantenibilitat: degut a la dura experiència que resulta mantenir o evolucionar codi que no te un bon disseny, tenia molt clar que el nucli de la solució havia de ser molt mantenible. Un bon disseny inicial i l'ús de patrons de disseny han ajudat en gran mesura a que això sigui possible
- Fàcil d'evolucionar: un producte d'aquest tipus evoluciona constantment, ja que internet evoluciona cada dia, així com els requisits que van marcant els diferents clients. Per aquest motiu la solució havia de tenir un disseny que permetés fàcilment realitzar evolutius
- Fiable: l'experiència m'ha ensenyat que en eines d'aquest tipus el mínim error fa que la confiança en l'eina per part del client decreixi ràpidament. Els resultats han de ser fiable, s'han de reduir al mínim els falsos positius.

Veient el resultat obtingut crec que tots aquests requisits s'han complert àmpliament, i el resultat, tot tenir totes les funcionalitats esperades inicialment, és molt bo.

A mode resum, el desenvolupament d'aquest projecte m'ha ajudat a profunditzar molt més en el desenvolupament de Java, així com en el disseny d'una aplicació, aprofundint i posant en pràctica molts conceptes apresos durant els anys d'estudi, com poden ser els patrons de disseny

9 Bibliografia

- Hibernate. [En línia]. [Data de consulta: 18 d'abril de 2014]. http://Hibernate.org/
- viralpatel [En línia]. [Data de consulta: 18 d'abril de 2014]. http://viralpatel.net/blogs/introduction-to-Hibernate-framework-architecture/
- Apunts EPPO [Físic]. [Data de consulta: 10 d'abril de 2014].
- getbootstrap [En línia]. [Data de consulta: 11 d'abril de 2014]. http://getbootstrap.com/
- wikipedia [En línia]. [Data de consulta: 9 d'abril de 2014]. http://en.wikipedia.org/wiki/Representational_state_transfer
- ArquitecturaJava [En línia]. [Data de consulta: 19 d'abril de 2014]. http://www.arquitecturaJava.com/el-concepto-de-classloader/
- Selenium wiki [En línia]. [Data de consulta: 15 d'abril de 2014]. https://code.google.com/p/selenium/wiki/PageObjects
- Watir [En línia]. [Data de consulta: 15 d'abril de 2014]. http://watir.com/
- Telerik [En línia]. [Data de consulta: 15 d'abril de 2014]. http://www.telerik.com/teststudio/testing-framework
- Sahi [En línia]. [Data de consulta: 15 d'abril de 2014]. http://sahi.co.in/

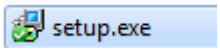
11 Annexos

11.1 Manual d'instal·lació

11.1.1 Agent

11.1.1.1 Instal·lació

L'agent té un setup que instal·larà l'aplicació, les seves dependències i l'estructura de directoris automàticament.

El primer pas serà executar setup.exe 

Això ens guiarà, mitjançant un assistent, per totes les pantalles de configuració:

- Selecció d'idioma: permet escollir l'idioma de la instal·lació

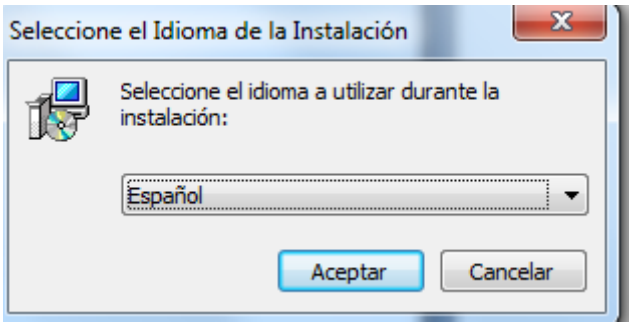


Figura 48 Agent - Instal·lació 1

- Carpeta d'instal·lació

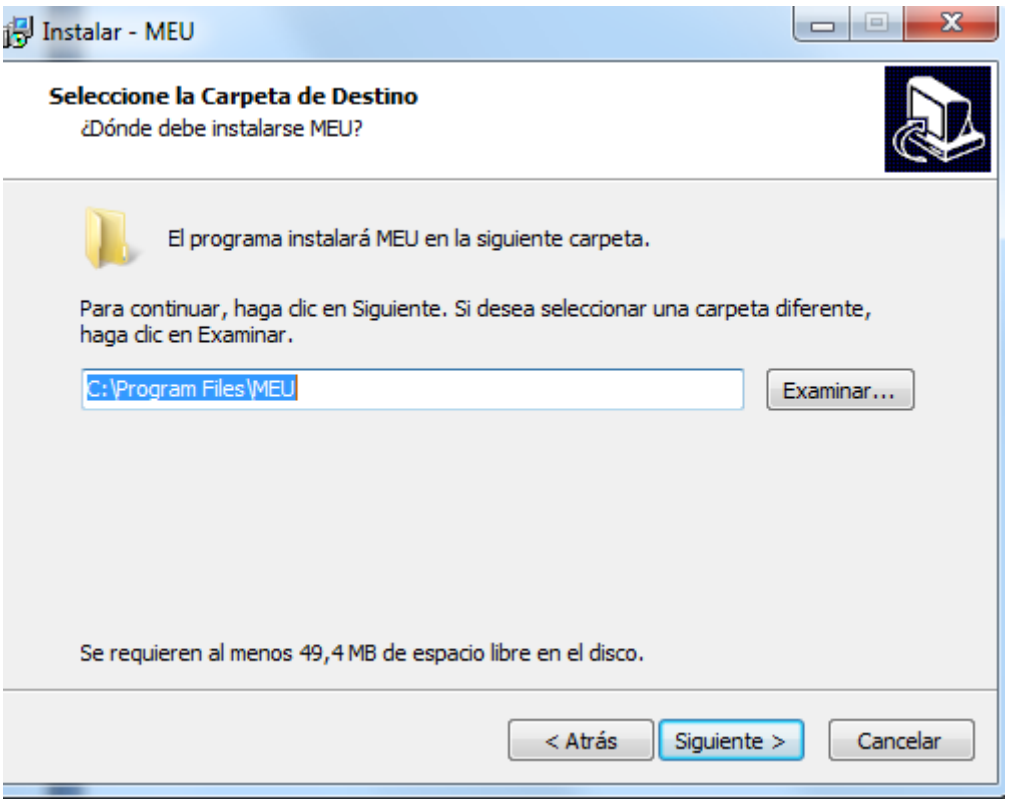


Figura 49 Agent - Instal·lació 2

- Un cop finalitzada la instal·lació de l'aplicació s'instal·larà l'aplicació Fiddler, necessària per a poder realitzar la captura d'esdeveniments HTTP

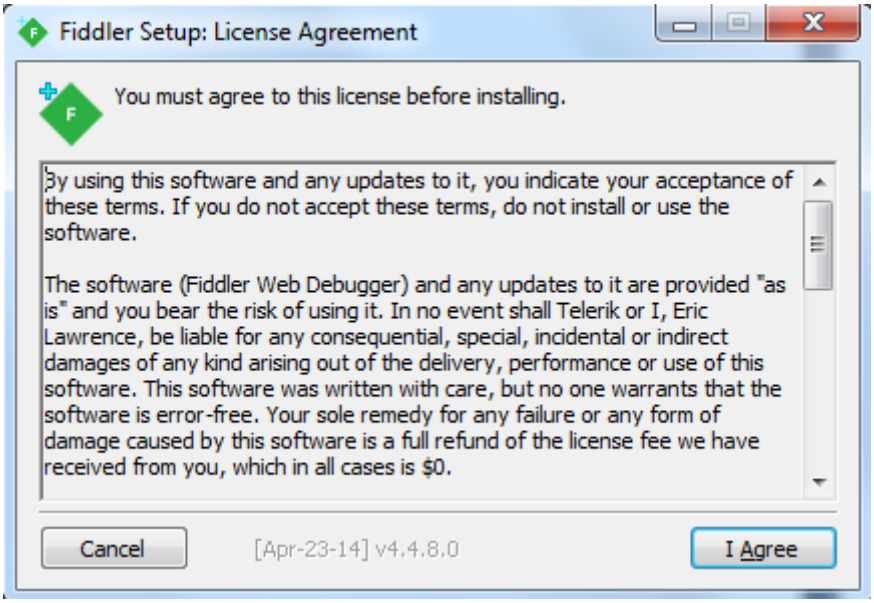


Figura 50 Fiddler - Instal·lació

- Seguim el procés d'instal·lació de Fiddler amb les opcions per defecte
- Un cop seguits tots aquests passos es mostrarà la pantalla de finalització de la instal·lació

11.1.1.2 Configuració

El fitxer de configuració esta situat a la carpeta on s'ha instal·lat l'aplicació, concretament el subdirectori config. Dins d'aquesta veurem un fitxer anomenat Agente.xml, on es troba el fitxer de configuració de l'aplicació. Si editem el fitxer veure que te la següent estructura. Marquem en vermell els camps a editar per el correcte funcionament de l'aplicació.

```
<?xml version="1.0" encoding="UTF-8"?>
<Agente>
      <Robot>
            <Nombre>Nombre del robot</Nombre>
      </Robot>
      <WebService>
            <URL>URL donde esta corriendo el WebService, por ejemplo:
http://localhost:8080/WebService/WebService/robot/</URL>
      </WebService>
      <FTPServer>
            <FTPIP>Ip del servidor FTP</FTPIP>
            <FTPport>Puerto donde escucha el servidor FTP. Per defecte
el 21</FTPport>
            <FTPUser>usuario de acceso al servidor FTP</FTPUser>
            <FTPPass>password del servidor FTP</FTPPass>
      </FTPServer>
      <Fiddler>
         <PathFiddler> Path donde tenemos instalada la aplicación
         Fiddler, por defecto: C:\Program Files\Fiddler2\</PathFiddler>
            <PathCapturas>Path donde almacena Fiddler las capturas. Por
         defecto
         C:\Users\usuario\Documents\Fiddler2\Captures\</PathCapturas>
      </Fiddler>
   </Agente>
```

Per a facilitar la instal·lació de l'eina, sobretot pel que fa a la BBDD, s'han publicat les aplicacions web. La URL del WebService és:
http://ismsilk.nextret.net/WebService/WebService/robot/

També s'ha publicat el servidor FTP per facilitar la instal·lació de l'eina, aquest és accessible des de ftp://ismsilk.nextret.net amb les credencials:

- Login: xxxx
- Password: xxxx

11.1.2 WebService

Per a les aplicacions web s'ha utilitzat Apache Tomcat 7 (http://tomcat.apache.org/download-70.cgi) amb Java v7. Per a desplegar l'aplicació WebService copiarem el fitxer war proporcionat (WebService.war) a la carpeta webapps del directori d'instal·lació d'Apache Tomcat. El propi servidor s'encarregarà d'instal·lar l'aplicació. Un cop Tomcat finalitzi la instal·lació de l'aplicació, haurem de configurar l'accés a la BBDD.

Per això anirem a webapps\WebService\WEB-INF\classes I editarem el fitxer hibernatge.cfg.xml I haurem d'editar els camps:

```
<property name="Hibernate.connection.password">Password de la BBDD</property>
<property
name="Hibernate.connection.url">jdbc:jtds:sqlserver://IP_Servidor_BBDD:1433/S
elenium</property>
<property name="Hibernate.connection.username">usuari de la BBDD</property>
```

Després haurem de reiniciar l'aplicació per a que agafi els canvis.

La URL per defecte de l'aplicació és: http://IP_servidor:8080/WebService/WebService/robot

11.1.3 Portal d'administració

El portal d'administració és també una aplicació Web. La instal·lació d'aquesta aplicació és la mateixa que per al WebService. Copiarem el fitxer WAR proporcionat (Administracion.war) i un cop desplegat realitzarem la configuració especifica d'Hibernate.

Per això anirem a webapps\Administracion\WEB-INF\classes I editarem el fitxer hibernatge.cfg.xml I haurem d'editar els camps:

```
<property name="Hibernate.connection.password">Password de la BBDD</property>
<property
name="Hibernate.connection.url">jdbc:jtds:sqlserver://IP_Servidor_BBDD:1433/S
elenium</property>
<property name="Hibernate.connection.username">usuari de la BBDD</property>
```

Després haurem de reiniciar l'aplicació per a que agafi els canvis.

La URL per defecte de l'aplicació és: http://IP_servidor:8080/Administracion

S'ha publicat aquesta aplicació a internet i la seva URL d'accés és:

http://ismsilk.nextret.net/Adminitracion

Les credencials d'accés són:

- Login: xxxxx
- Password: xxxx

11.1.4 Base de dades

Aquesta solució esta preparada per funcionar amb una BBDD SQL Server 2008. En l'entorn de desenvolupament s'ha utilitzat la versió Enterprise, però es pot utilitzar la versió Express, que pot ser descarregada de: http://www.microsoft.com/es-es/download/details.aspx?id=1695

Si volem saltar-nos aquest pas es poden utilitzar les dos aplicacions ja publicades a internet i que tenen accés a una base de dades ja configurada.

Un cop instal·lada obrirem l'aplicació SQL Management Studio crearem una nova base.

Fem botó dret sobre "Bases de datos" i fem clic sobre "Nueva base de datos..."

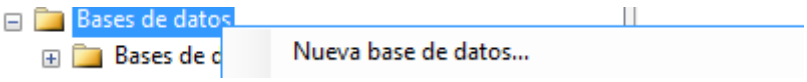


Figura 51 Instal·lació BBDD 1

Al camp “Nombre de la base de datos” posem “Selenium”:

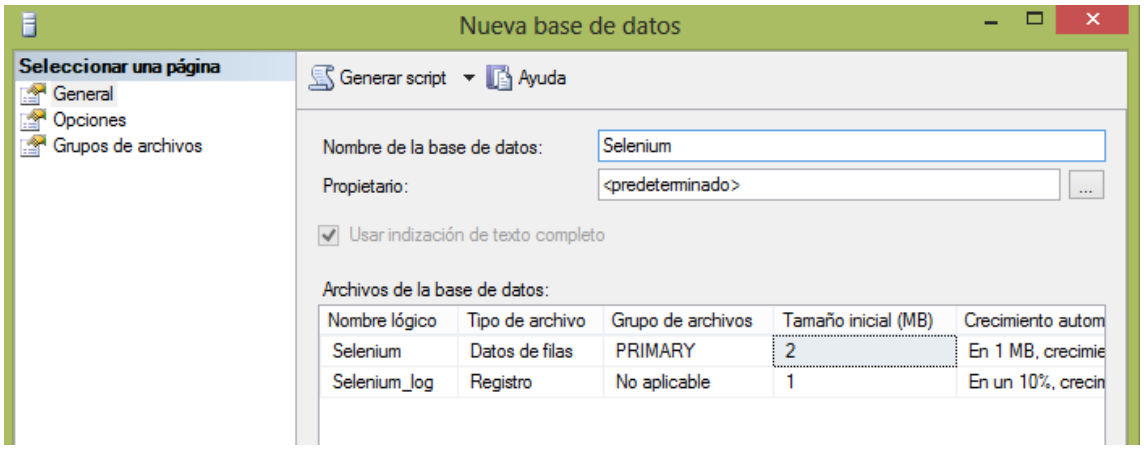


Figura 52 Instal·lació BBDD 2

I fem clic a “Aceptar”.

Un cop creada la base de dades fem botó dret sobre aquesta, tareas, restaurar, base de datos...

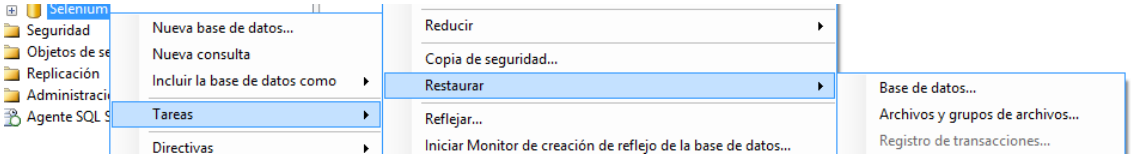


Figura 53 Instal·lació BBDD 3

Primer marquem la opció “Des de dispositivo” i fem clic a la icona de la dreta per escollir el fitxer de restauració de la base de dades que s’ha proporcionat (Selenium.bak)

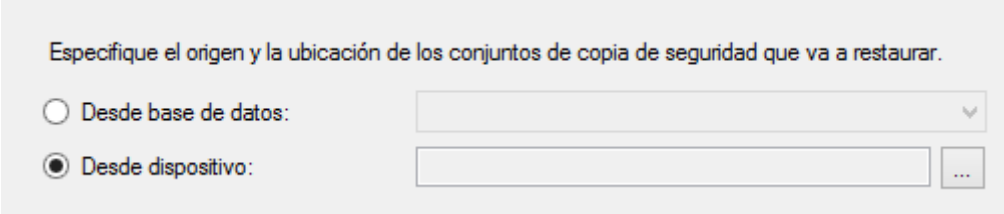


Figura 54 Instal·lació BBDD 4

Del llistat que apareixerà escollim l’última:

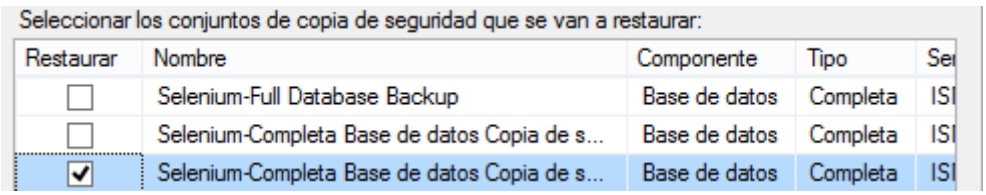


Figura 55 Instal·lació BBDD 5

Anem a la part “Opciones” que està a l’esquerra de la pantalla i marquem:

“Sobreescribir la base de datos existente (WITH REPLACE)”

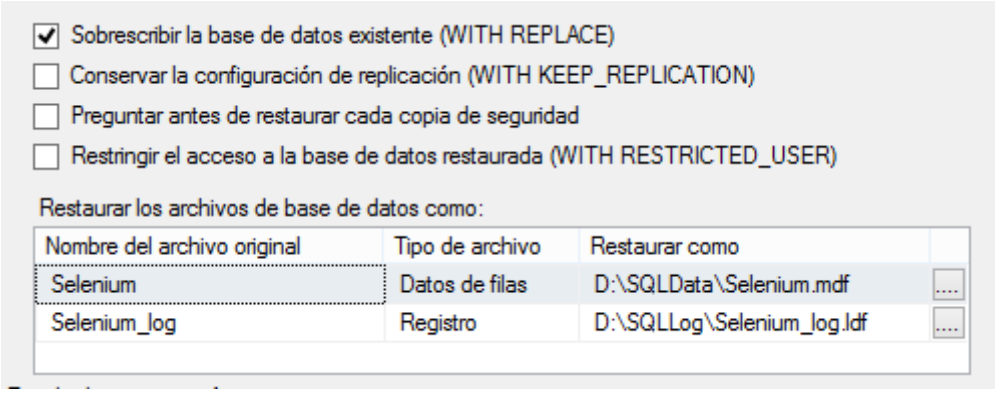


Figura 56 Instal·lació BBDD 6

En aquesta part ens hem d’assegurar que les rutes que apareixen són les correctes.

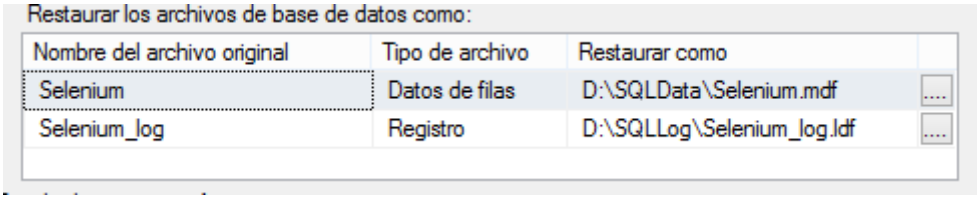


Figura 57 Instal·lació BBDD 7

Finalment fem clic a acceptar i ja tindrem la BBDD llesta.

11.2 Manual d'usuari

11.2.1 Agent

L'ús de l'agent és molt senzill. No requereix de quasi intervenció per part de l'usuari, ja que la seva funció principal és la d'automatitzar execució de circuits.

Al obrir-se la interfície gràfica veurem el següent:

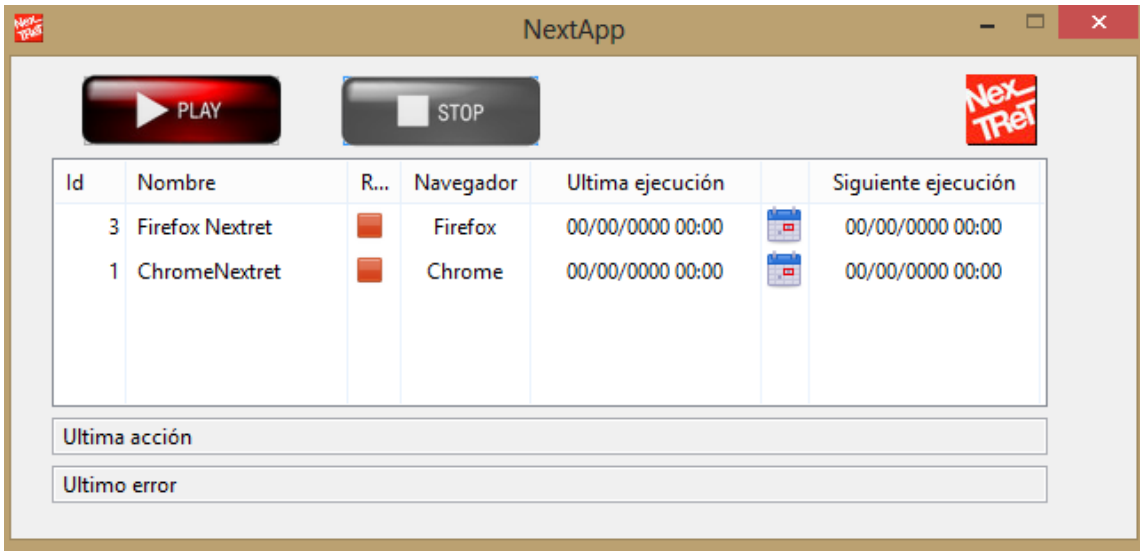


Figura 58 Interfície gràfica agent

Per a començar a executar els test haurem de fer clic sobre el botó "Play".

Quan vulguem parar l'execució dels circuits farem clic sobre el botó "Stop". Al fer "Stop" l'execució no pararà immediatament, sinó que aquesta es farà al finalitzar tota la roda d'execució de tests, això vol dir que si per exemple tenim 4 test i fem clic a la meitat de l'execució del test 3, no passarem a estar parat fins que finalitzi l'execució del test actual i la del següent (si per condicions de calendari li toca executar-se).

11.2.2 Portal d'administració

L'eina d'administració és una aplicació web, per accedir-hi s'haurà d'obrir un navegador i anar a la URL on estigui publicada l'aplicació.

El primer que veurem al accedir-hi és la pàgina d'autenticació:

Figura 59 Home portal d'administració

Ens autentiquem amb les credencials que ens han proporcionat i veurem la pàgina principal:

Figura 60 Menú portal d'administració

A la part esquerre de la pantalla podrem veure el menú amb les diferents opcions que tenim, i que detallem a continuació.

11.2.2.1 Robots

11.2.2.1.1 Nou

Opció per a crear un nou robot. Veurem la següent pantalla:

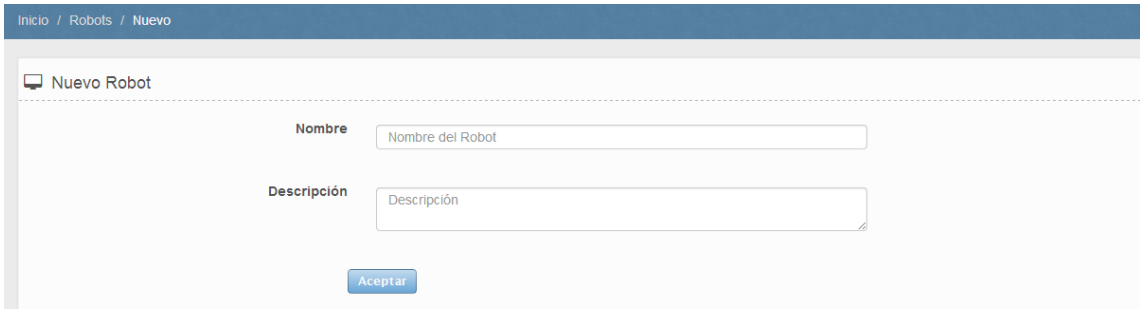


Figura 61 Nou robot

On haurem d'introduir el nom del robot i una descripció

11.2.2.1.2 Esborrar

Opció per esborrar un robot. Ens apareix un llistat de tots els robots. Mitjançant el checkbox de la part dreta seleccionarem el robot o robots que vulguem esborrar i finalment pitjarem el botó vermell per procedir amb l'esborrat

Figura 62 Esborrar robot

11.2.2.1.3 Editar

Permet editar els robots. Al accedir a aquesta opció haurem de seleccionar un robot mitjançant el menú que apareix a la part esquerre:

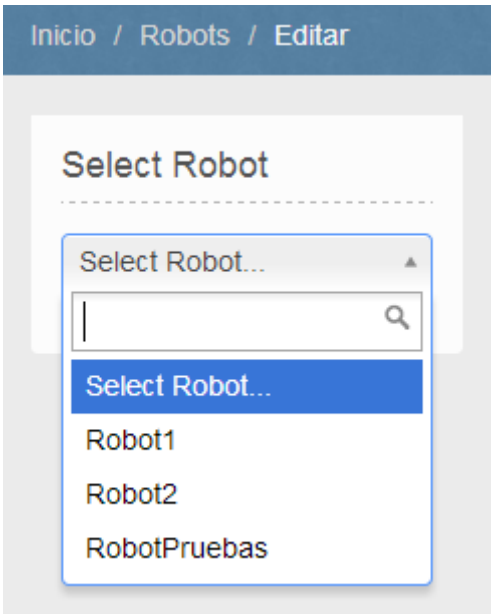


Figura 63 Selecció de robot

Un cop escollit un robot podrem editar els seus valors:

Figura 64 Edició de robot

Un cop realitzades les modificacions pertinents polsarem el botó “Aceptar” per a guardar els canvis.

11.2.2.1.4 Administrar

Pantalla d'administració on podrem assignar tests als diferents robots.

Primer de tot hem d'escollir el robot que volem administrar, mitjançant el menú de l'esquerra.

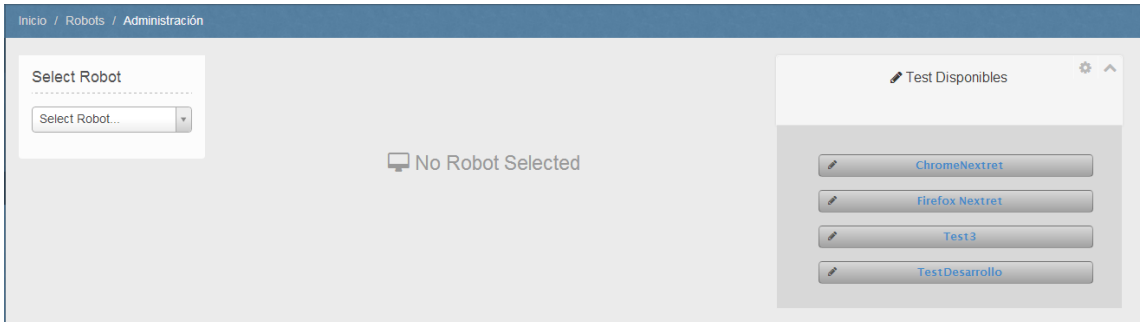


Figura 65 Pantalla d'administració de robots

Veurem que a la part central s'obre l'administració del robot, i a la part dreta tenim els tests disponibles per assignar.

La primera pestanya ens permet assignar o treure tests del robot

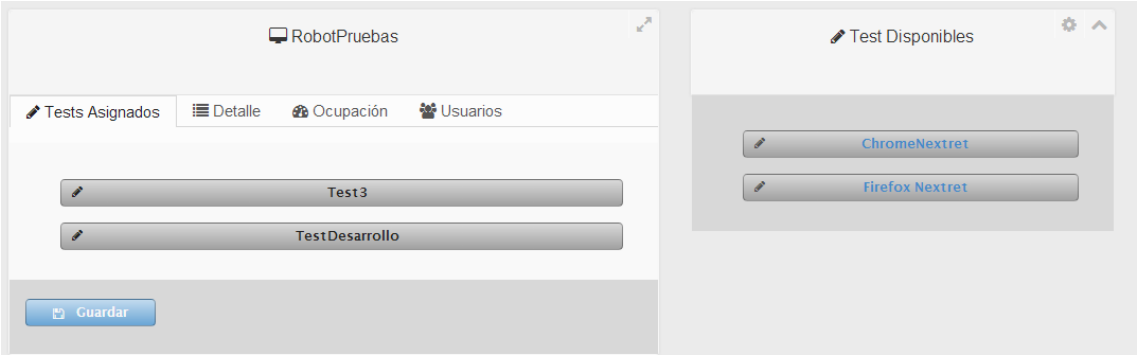


Figura 66 Assignació de test a robot

Per a assignar un test farem clic sobre el test disponible (a la part dreta de la pantalla) i l'arrastrarem a la part esquerre. Quan finalitzem l'assignació de tests farem clic al botó "Guardar"

Per a desassignar un test farem clic sobre un test situat a la part central i l'arrosseguem fins a la part dreta. Quan finalitzem les operacions farem clic al botó "Guardar".

Si fem clic a la pestanya “Detalle” podrem veure informació del robot:

Figura 67 Detall d'un robot

Si fem clic a la pestanya “Ocupación” veurem l’ocupació total del robot. Actualment no es calcula l’ocupació del robot i la dada que es mostrà no és real.

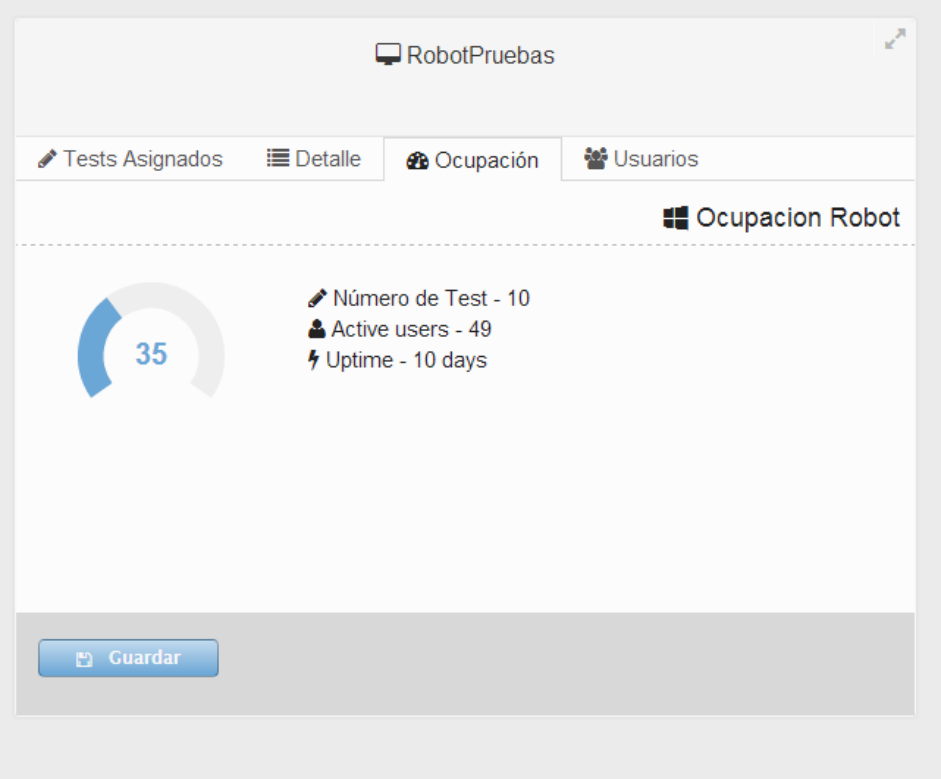


Figura 68 Ocupació d'un robot

Finalment, la pestanya “Usuarios” ens permetrà assignar o desassignar usuaris al robot. Aquests usuaris són de la consola d’administració i permetrà als usuaris assignats poder administrar aquell robot.

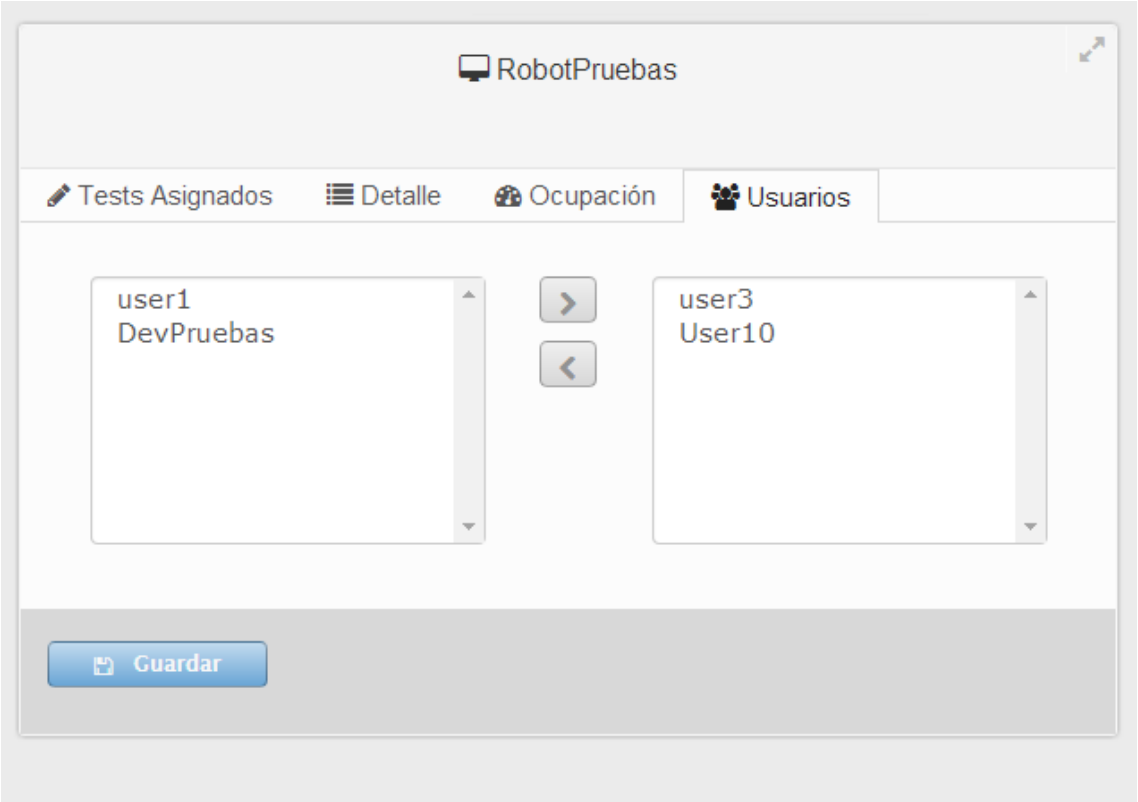


Figura 69 Administració d'usuaris associats a un robot

11.2.2.2 Test

Des d’aquest apartat podrem realitzar totes les accions que fan referencia als tests. A continuació veurem en detall cada una de les subopcions.

11.2.2.2.1 Nou

Permet crear un nou test. Per tal de donar d’alta un nou test haurem de complimentar el formulari:

- Nombre: nom que volem donar al circuit
- Descripció: text descriptiu del que fa el circuit
- Intentos: Indica el nombre de vegades que s’executarà el circuit en cas d’error. Per defecte el valor és 1. Això significa que el circuit s’executarà una única vegada. Si per exemple el valor fos dos, el circuit s’executa i falla, i torna a intentar-ho. Això ens permet veure si l’error ha estat causat per algun fet puntual
- Navegador: tipus de navegador amb el que s’executarà el circuit
- Pasos: nombre de passos que te el circuit
- Nombre de passos: haurem de posar un nom descriptiu a cada un dels passos. Per exemple al pas 1: home, pas 2: login, etc.

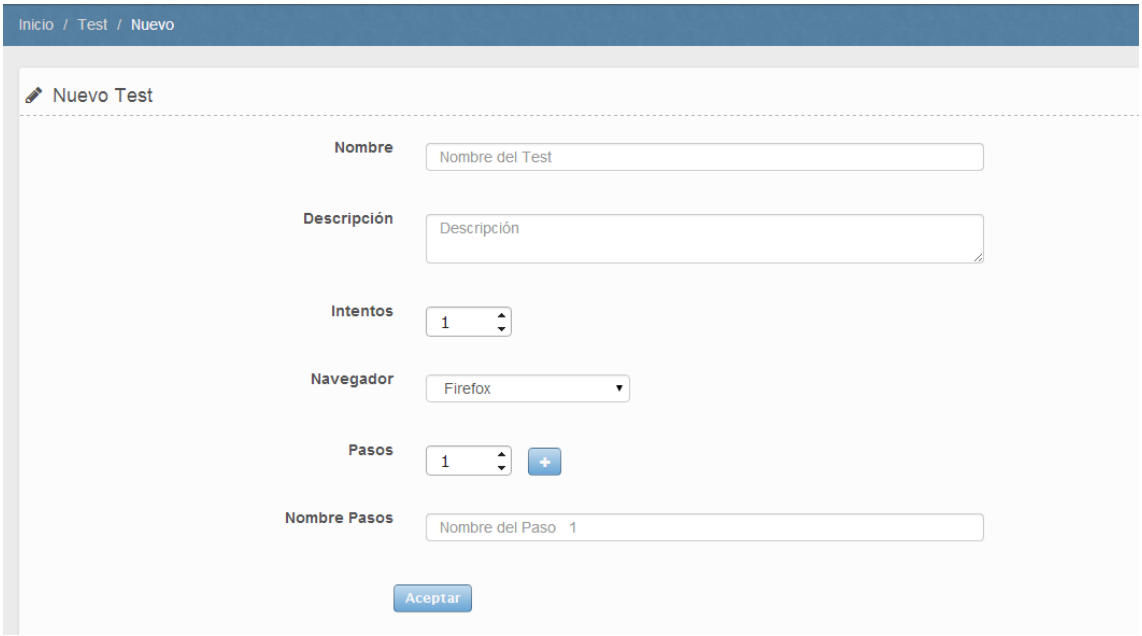


Figura 70 Formulari d'alta d'un test

Finalment polsarem “Aceptar” i s’obrirà una pantalla per a pujar el fitxer del circuit. Hem de buscar el nostre fitxer JUnit (.Java) que hem generat amb el plugin de firefox (https://addons.mozilla.org/es/firefox/addon/selenium-expert-selenium-ide/)

Un cop escollit el fitxer farem clic a enviar.

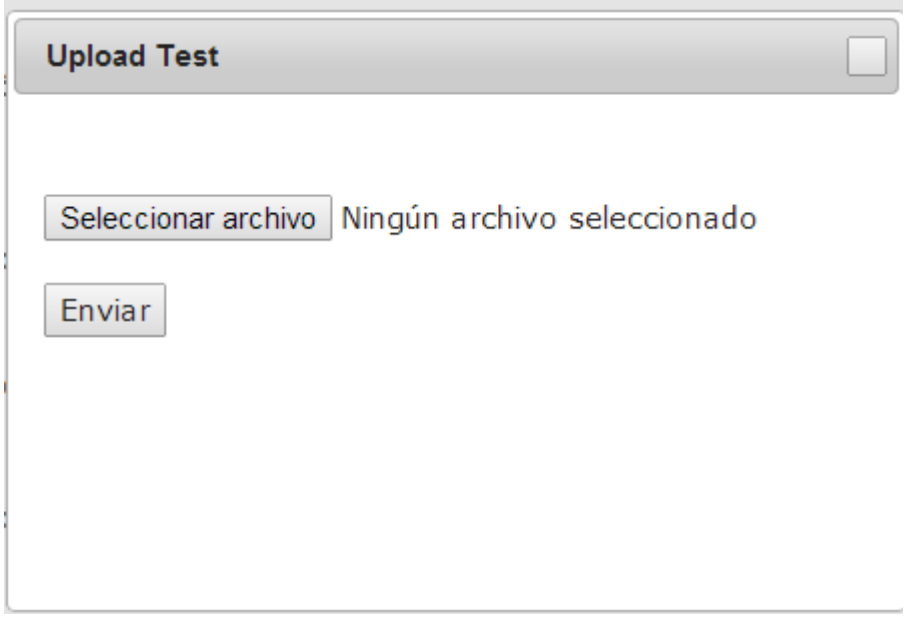


Figura 71 Pantalla de pujada de test

11.2.3.1.1 Esborrar

Permet esborrar un test, ens mostrarà un llistat de tots els tests. Marcarem el checkbox dels test que vulguem esborrar i finalment polsarem sobre el botó vermell per tal d'esborrar-los.

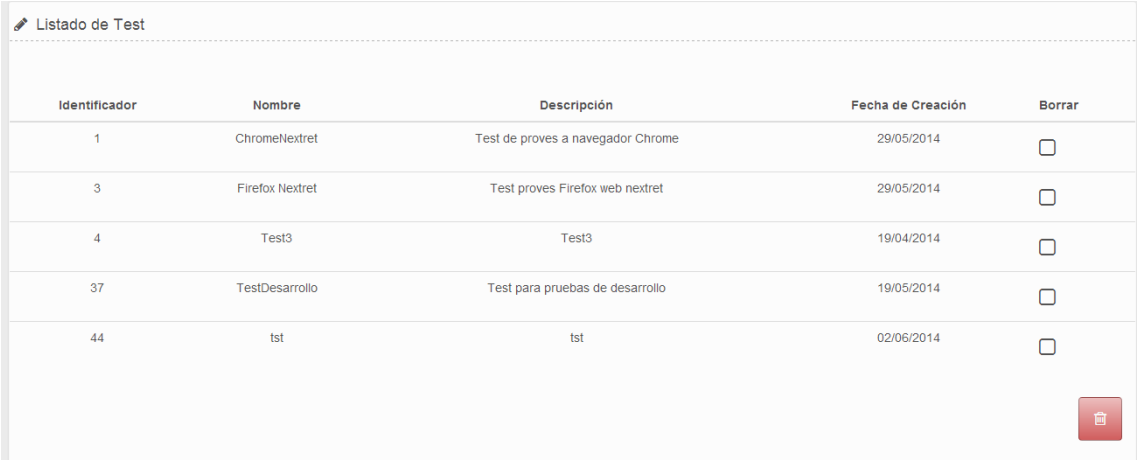


Figura 72 Pantalla per esborrar tests

11.2.3.1.2 Editar

Edició de test, primer de tot escollirem el test que vulguem editar del llistat.

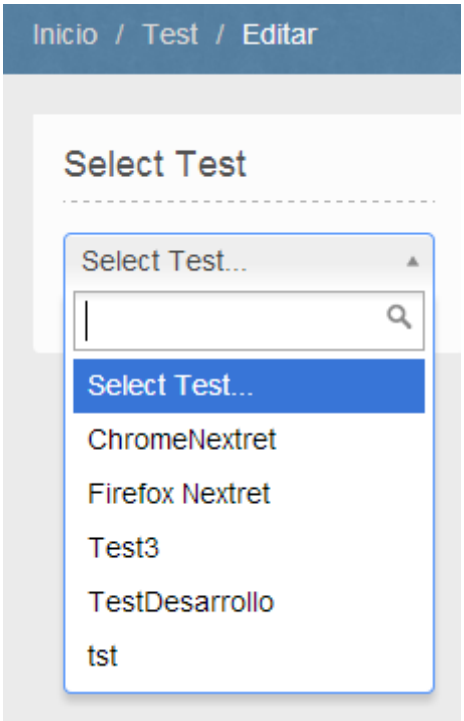


Figura 73 Selecció de test a editar

S'obrirà un formulari on podrem editar els següents paràmetres:

- Nombre
- Descripción
- Intentos
- Pasos
- Nombre de passos

11.2.3.1.3 Administrar

Administració del tests. Primer de tot escollirem el test que vulguem administrar del llistat:

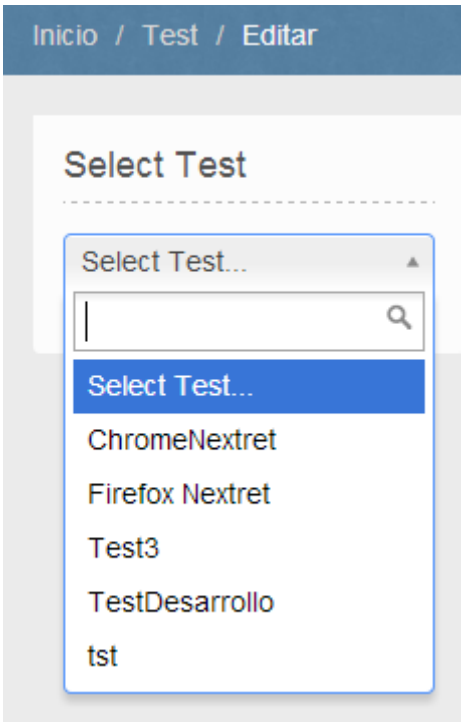


Figura 74 Selecció de test a administrar

S'obrirà el detall del test on podrem veure:

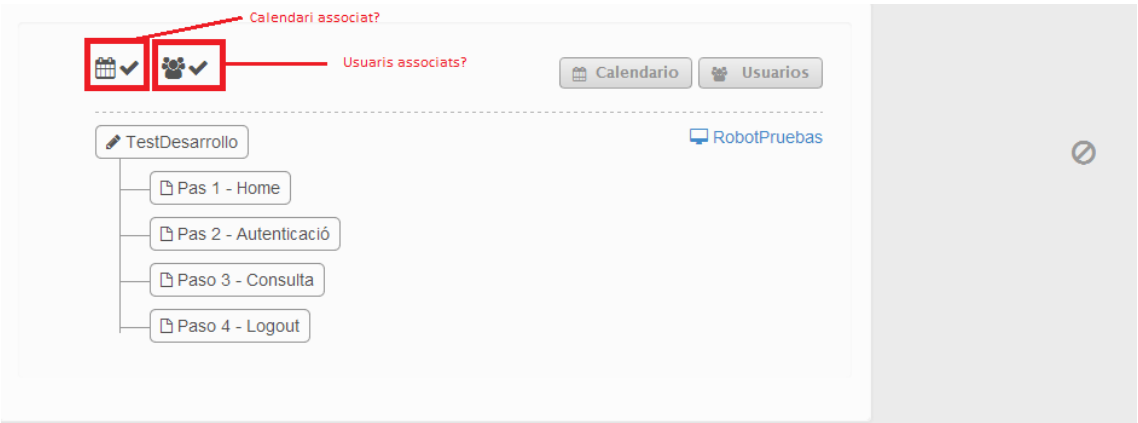


Figura 75 Detall d'un test

Les dues icones superiors ens indiquen si el test te un calendari associat o si te algun usuari associat.

A la part inferior veurem el detall del test: el nom del test, i la informació dels passos d'aquests. A la dreta d'aquesta informació veurem el robot o robots on aquest circuit s'executà.

Si fem clic sobre el botó “Calendario”, s’obrirà a la part dreta la informació del calendari i les accions que podem realitzar sobre ell:

Figura 76 Detall del calendari d'un test

- Nuevo: en cas de no tenir cap calendari associat aquesta opció estarà activada, si fem clic sobre “Nuevo” s’obrirà el formulari per a donar d’alta un nou calendari:
    - Inicio: Indicarem la data inicial i l’hora en la que volem que el circuit comenci a executar-se
    - Final: indicarem la data final i l’hora en la que volem que el circuit finalitzi la seva execució, sempre que ho activem marcant el checkbox que hi ha a la dreta. Si no posem cap valor el circuit no tindrà data final d’execució.
    - Minutos: temps mínim per a que el circuit torni a executar-se. Si posem 10 minuts, el circuit s’executarà cada 10 minuts o més (segons la capacitat del robot).

Figura 77 Pantalla de creació de calendari

Un cop finalitzem la introducció de dades polsarem el botó acceptar. Ara, a la part dreta, podrem veure que ja tenim un calendari assignat.

Figura 78 Detall d'un calendari associat a un test

En aquest cas com ja en tenim un de creat veurem la informació

- Editar: obrirà un formulari on podrem editar la configuració del calendari

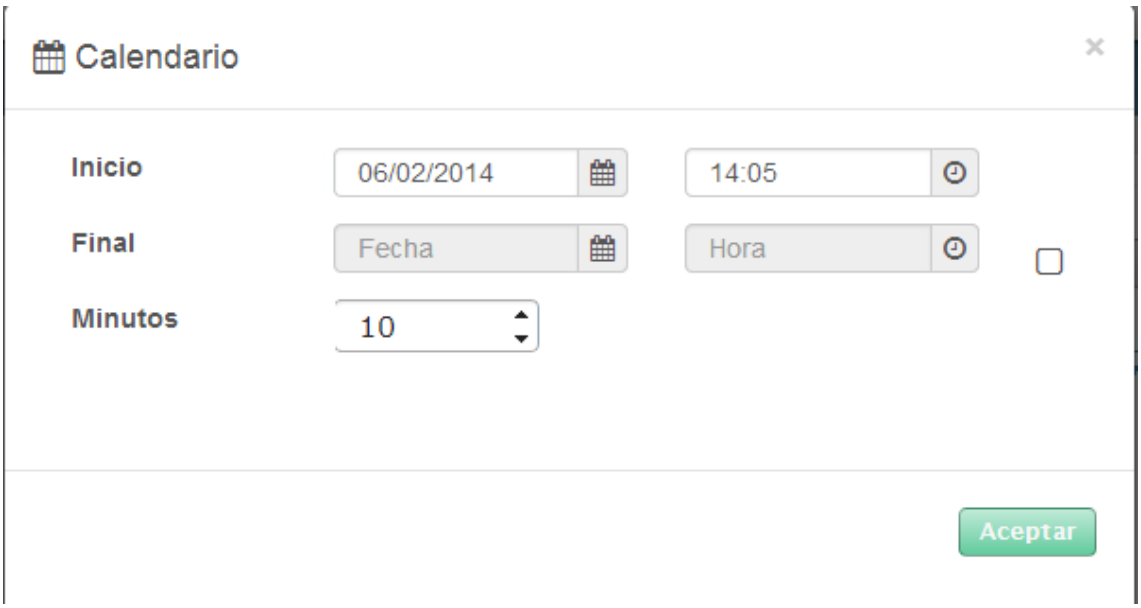


Figura 79 Edició d'un calendari

- Borrar: esborra el calendari.

Si fem clic sobre el botó “Usuarios” a la part dreta veurem tant els usuaris disponibles com els usuaris assignats a aquest test. Podrem assignar i des assignar usuaris seleccionant-los i fent us de les fletxes. Finalment polsarem el botó “Guardar” per a guardar tots els canvis realitzats.

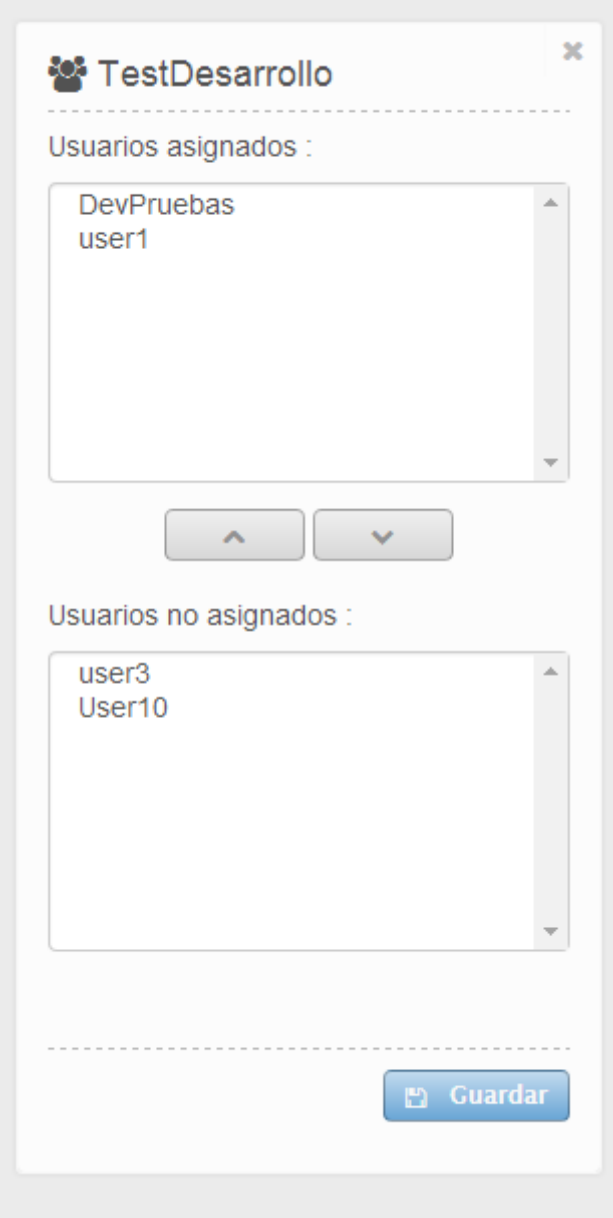


Figura 80 Vista d'usuaris assignats a un test

11.2.3.2 Usuaris

11.2.3.2.1 Nou

Permet crear un nou usuari d'administració. Al fer clic en aquesta opció s'obrirà un formulari que haurem de completar:

- Nombre: nom del usuari. S'utilitzarà com a login
- Contraseña: paraula de pas d'accés a l'eina
- Repetir contrasenya: paraula de pas d'accés a l'eina, s'ha de repetir per a evitar errors al escriure-la
- Usuario activado: podem marcar l'usuari com actiu o inactiu.

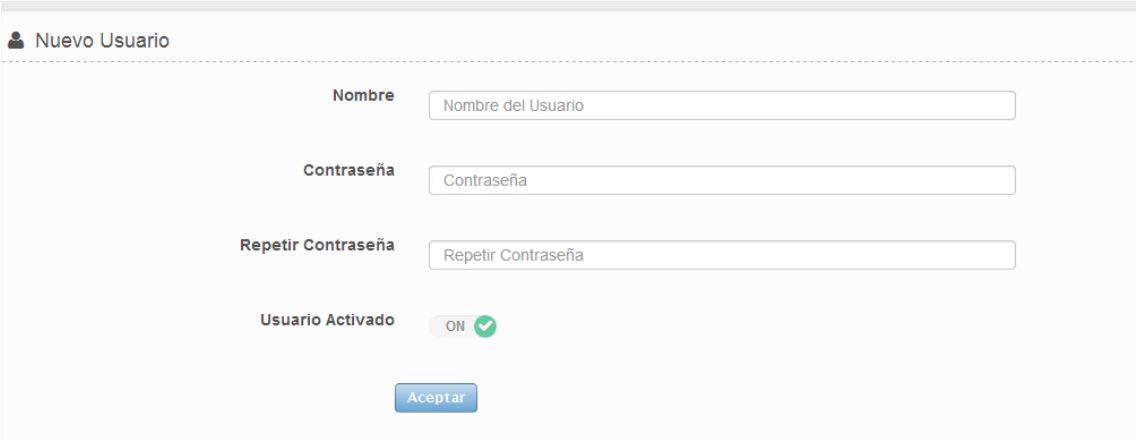


Figura 81 Formulari d'alta d'usuari

11.2.3.2.2 Esborrar

Ens permet esborrar un o diversos usuaris. Per tal d'esborrar marcarem el o els usuaris que vulguem fent clic sobre el seu checbox i finalment polsarem el botó vermell.

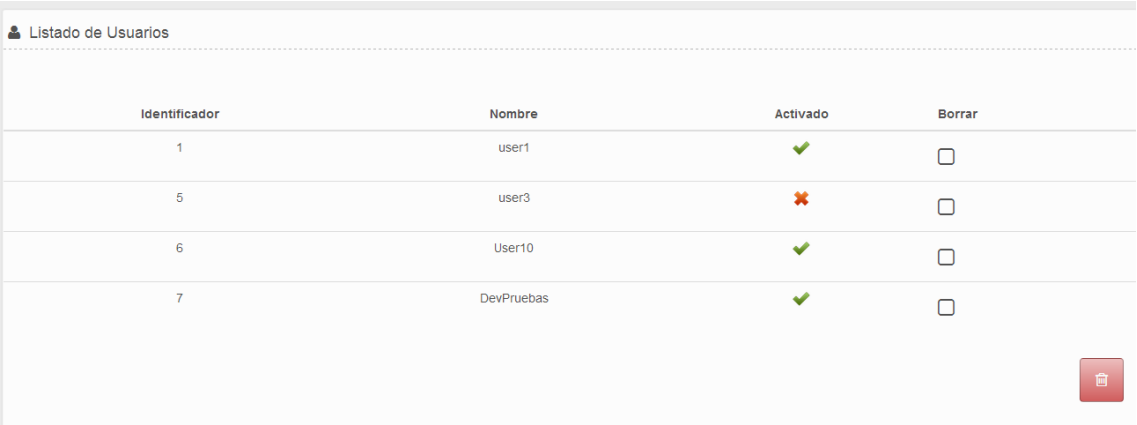


Figura 82 Pantalla d'esborrar usuari

11.2.3.2.3 Editar

Permet editar la configuració de l'usuari. Primer de tot escollirem un usuari del menú desplegable:

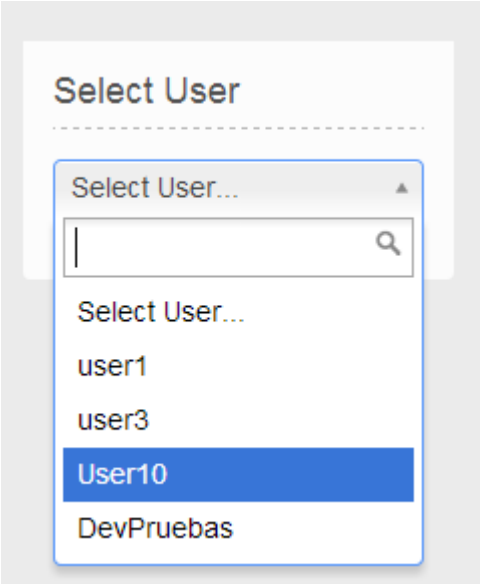


Figura 83 Selecció d'usuari

S'obrirà un formulari on podrem modificar tots els camps, un cop realitzades les modificacions pertinents polsarem sobre "Aceptar" per a guardar els canvis.

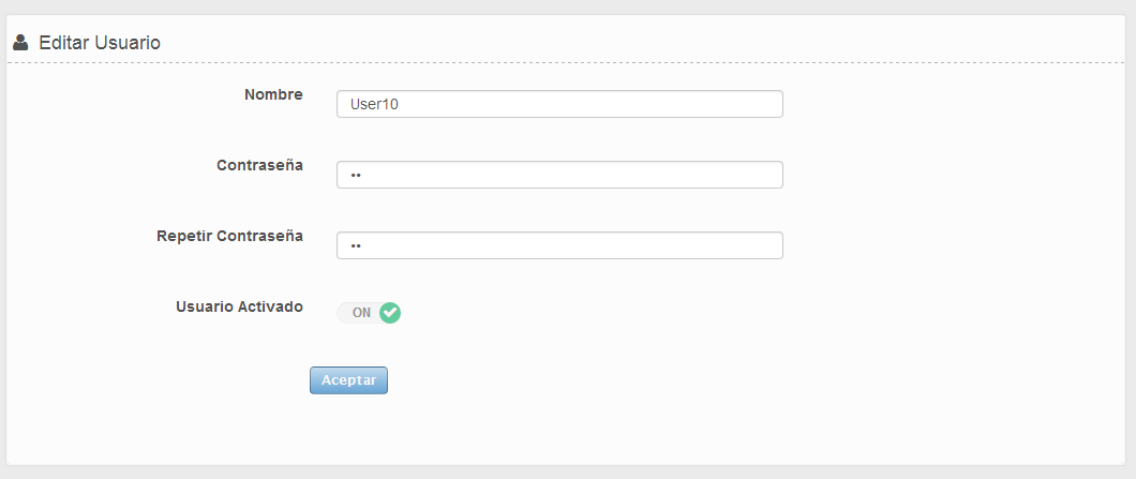


Figura 84 Edició d'usuari