RIA AMB SISTEMA MODULAR PER A GENERAR CONSULTES DINÀMIQUES A UNA BASE DE DADES

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Jordi Masip Balart
i dirigit per
Pilar Gómez Sánchez
Bellaterra, 25 de gener de 2010

Agraïments

Voldria donar les gràcies a Pilar Gómez per dirigir el meu projecte de manera clara i directe.

A Ruben Celada, el meu tutor d'empresa, per oferir-me aquest projecte.

També m'agradaria agrair a Javier Vacas per compartir els seus coneixements, a Iñaki Bustero per la seva ajuda desinteressada i a tots els companys d'universitat que he tingut en aquests anys.

I molt especialment a Nuria Latorre per estar sempre al meu costat i a la meva família per ajudar-me quan ho he necessitat.

TAULA DE CONTINGUTS

1.- INTRODUCCIÓ

1.1- Objectius

L'empresa Playoff Informàtica vol aconseguir unificar les dades de l'empresa Solmania i generar estadístiques a través d'un quadre de comandaments.

Actualment a Solmania no existeix cap sistema automatitzat i unificat per facilitar el procés de recollida de dades i el posterior integrament de totes les dades de cada franquícia de la cadena. Fins ara es realitzen connexions via *PC-Anywhere* i s'accedeix directament a l'aplicació del franquiciat.

L'objectiu principal és crear una aplicació que integri sota un sistema de base de dades la informació existent, i es combini per extreure la informació estadística necessària per fer un control dels paràmetres clau del negoci, ajudant en el pla estratègic de la companyia per aconseguir els objectius marcats d'una forma exacte i eficaç.

El *look&feel* ha de ser amigable, la interfície d'usuari fàcil de fer servir, amb un disseny final espectacular i modern.

1.2- Estat de l'art

L'inici del desenvolupament de les aplicacions el trobem amb els *mainframes* que ens oferien una interfície sense gràfics i totalment basada en text. Amb l'arribada de Windows i l'evolució de les màquines de l'època, s'imposa la tecnologia client-servidor, i d'aquesta manera apareixen els primers models visuals en que l'usuari hi podia interactuar. Les màquines client anaven evolucionant d'una manera molt ràpida i això va permetre millorar molt la utilitat de les aplicacions amb elements com tabuladors, menús i taules de dades. Tot i així, aquesta tecnologia era difícil d'administrar i de mantenir actualitzada.

A mitjans dels 90 apareix la web com a mitjà fonamental per a transmetre informació per als usuaris de tot el món. Poc a poc, totes les organitzacions van començar a migrar les seves aplicacions actuals cap a les anomenades aplicacions web, però llògicament no es va aconseguir mantenir el grau d'utilitat de les anteriors aplicacions client-servidor. Ara ens trobem en una nova situació en que la tecnologia ha fet possible combinar lo millor de cada món (web i client-servidor).

Les *Rich Internet Application* (RIA) combinen els beneficis que s'obtenen de la facilitat de distribució i manteniment de les aplicacions web amb les experiències més efectives i interessants de l'usuari final. A més, la tecnologia que ha fet possible aquesta evolució no para de millorar ràpidament.

1.3.- Planificació

La planificació per mesos per realitzar el projecte ha estat la següent:

- Requeriments: Desembre 2008.
- Estudi de la tecnologia: Gener 2009 - Febrer 2009.
- Disseny: Març 2009 – Abril 2009.
- Implementació: Maig 2009 – Juliol 2009.
- Proves: Agost 2009.
- Memòria: Setembre 2009 – Desembre 2009.

1.4.- Estructura de la memòria

Aquesta memòria està dividida en 5 capítols, presentats a continuació:

En el capítol actual es presenten els objectius d'aquest projecte i s'expliquen els orígens de les aplicacions web tot introduint el concepte d'aplicació RIA. En el segon capítol es presenta la tecnologia utilitzada, detallant les opcions existents al món RIA i presentant els *frameworks* de Flex[1] i el protocol que utilitza per connectar-se amb Java. En el tercer capítol s'explica com s'ha dissenyat el projecte i les parts que el composen. En la quarta part els passos seguits per implementar el projecte i finalment en el cinquè capítol les conclusions i les millores.

2.- TECNOLOGIA EMPRADA

Aquest capítol està dividit en dos parts clarament diferenciades. A la primera part s'argumenta l'elecció de Flex com a tecnologia i més concretament el *framework* Cairngorm per crear l'aplicació RIA. Es realitza un estudi comparatiu entre les tecnologies RIA i els *frameworks* de Flex i s'explica el funcionament detallat del framework Cairngorm amb un exemple detallat. També es presenta el protocol *Action Message Format* (AMF)[2], que serveix per connectar la capa de presentació amb la del servidor mitjançant el producte Blazeds[3].

A la segona part es presenten tècniques teòriques per aprofitar tota la potència del Flex per crear un *dashboard* que comuniqui la informació de manera eficient.

2.1.- El món de les RIA

2.1.1.- Necessitats RIA

Com s'ha comentat a la introducció, les aplicacions RIA permeten crear una aplicació web amb l'aparença d'una aplicació feta per escriptori.

A priori, utilitzar la tecnologia RIA per la capa de presentació del projecte sembla una bona idea ja que ens ofereix les següents característiques:

- Madura, amb una gran massa d'usuaris que la utilitzin i que hi hagi una bibliografia extensa.
- Lleugera i que ofereixi un rendiment acceptable.
- Que s'hi puguin crear aplicacions de tipus empresarials.
- L'accés a la capa de base de dades s'ha de poder realitzar d'una manera senzilla.
- S'hi pugui accedir mitjançant *Web Services*.
- Eines que ajudin a crear formularis ràpidament.
- Mostrar informació amb una gran varietat de gràfiques.
- Crear aplicacions multi-idioma.
- Interfícies que siguin fàcilment utilitzables pels usuaris.
- Modificar l'aparença (*skins*) d'una manera fàcil.
- Que no ens tinguem que preocupar per l'entorn on s'executarà l'aplicació (sistema operatiu i navegador).

- Un bon entorn de desenvolupament.

2.1.2- Oferta RIA

Existeixen una àmplia gamma d'opcions per implementar una aplicació RIA, cadascuna té uns avantatges i uns inconvenients, a continuació es descriuen breument les principals opcions i posteriorment es fa un estudi detallat entre els dos candidats principals:

- Flex: Tecnologia d'Adobe, es desenvolupa la vista amb MXML i la part de lògica amb ActionScript3. Per desenvolupar la capa de la lògica de negoci pròpiament dita (per exemple, l'accés a base de dades) s'utilitza generalment J2EE, però també es permet .NET, Php o altres tecnologies existents en la part del servidor. Al final, es genera un *Small Web Format* (SWF). També és possible crear una aplicació d'escriptori (amb Air[4]). L'entorn de desenvolupament de referència és el Flash Builder.
- Silverlight [5] : Tecnologia de Microsoft, va començar més tard que Flex, però ha anat millorant ràpidament. El principal problema és que ve de Microsoft i per tant, costa aconseguir la confiança necessària per part dels desenvolupadors. Es desenvolupa la vista amb XML (*Extensible Markup Language*) tot i que la lògica de negoci serà desenvolupada habitualment amb .NET o cridant web services (per exemple, desenvolupats amb Java). Al final, l'aplicació necessita descarregar un plugin per ser vist en el navegador. L'entorn de desenvolupament de referència és Visual Studio.
- JavaFX [6] : Tecnologia de Sun, li queda molt per madurar. Es programa amb el llenguatge JSON[7]. Lo millor és que acaba sent Java però de moment produeix una certa decepció. El principal entorn de desenvolupament és NetBeans però també existeix un plugin per Eclipse.
- GWT (*Google Web Toolkit*) [8] :Tecnologia de Google. Es tracta de fer més senzilla la tasca de desenvolupar una aplicació AJAX. La idea és desenvolupar amb JAVA i el compilador GWT ens genera codi JavaScript. Mitjançant *Remote Procedure Call* (RPC) s'invoquen els serveis desenvolupats a la part dels servidor. No hi ha un entorn de desenvolupament de referència, cal utilitzar algun plugin per l'Eclipse. El disseny final resulta poc atractiu i cal reforçar-ho amb llibreries addicionals.

- AJAX : És un conjunt de tecnologies (XHTML, CSS, JavaScript, DOM i XMLHttpRequest) amb la que s'aconsegueix millorar l'experiència dels usuaris en el sentit que s'obté més interactivitat que quan s'utilitzen aplicacions de contingut simple a la web. AJAX es recomanat per millorar els continguts d'una web però no per crear aplicacions empresarials completes.

2.1.3.-Flex vs Silverlight

Adobe ha esta líder en el mercat de les aplicacions RIA durant aquests anys. Va sortir per primer cop al 2004 i va revolucionar la programació Flash. Fins ara, han dominat el mercat en contra de diversos competidors, ja siguin comercials o open source.

El principal competidor és Microsoft amb el seu producte Silverlight, va aparèixer per primer cop al 2007 i tot i que inicialment estava pensat per especialitzar-se en apartats multimèdia ha evolucionat molt gràcies a que permet desenvolupar amb el llenguatge C#.

A continuació ens centrarem en els avantatges i inconvenients d'aquestes tecnologies, els possibles resultats i quin impacte poden tenir.

Història de Flex i Silverlight

Per oferir una comparació el més justa possible entre aquests dos productes, és important familiaritzar-se amb la seva història. Sense conèixer les diferents versions dels productes és fa complicat realitzar una comparació equitativa.

La versió del Flex 1.0 va sortir a la llum el març del 2004. Flex necessita la màquina virtual de Macromedia Flash que es pot executar en qualsevol sistema operatiu. Al 2006, Adobe va treure tres versions beta de Flex 2.0 abans d'entregar la última versió 2.0 al juny del mateix 2006. La nova versió es basa en Eclipse, una plataforma de desenvolupament de codi font que és molt popular en la majoria d'entorns de desenvolupament Java. Aquesta nova versió va coincidir amb la nova versió del Action Script 3, d'aquesta manera per programar amb Flex s'utilitzen els llenguatges MXML i Action Script. Action Script és el llenguatge bàsic de desenvolupament del popular reproductor de Flash. Al 2007, Adobe treu un altre cop tres versions beta de Flex 3.0, la versió final 3.0 surt al mercat el febrer del 2008. Aquesta tercera versió suposa un gran

canvi ja que el SDK de Flex passa a ser un producte "open source". Ara, els desenvolupadors són lliures per contribuir al SDK i llògicament aquesta mesura va ser ben rebuda per la comunitat. A més inclou altres millores importants com la integració amb els productes Adobe Creatives Suites i sobre tot la novetat d'una primera versió del Adobe Air (*Adobe Integrated Runtime*) per fer aplicacions d'escriptori. Al segon semestre del 2009 va sorgir la beta de la versió 4.0

Silverlight és un plug-in del navegador web que significa el primer producte de Microsoft que veritablement és compatible en diferents navegadors i plataformes. La primera versió de Silverlight va sortir a l'abril del 2007. L'enfoc d'aquesta versió va ser més orientada cap a la part de multimèdia i li faltaven moltes de les característiques que Flex ja tenia a la versió 2.0. La versió 1.0 de Silverlight es basava en XAML com a llenguatge de programació, XAML és un llenguatge XML extensible que va ser creat per Microsoft i que s'utilitza àmpliament en el Framework 3.0 del .NET. La versió 2.0 va ser llançada el març del 2008 i la gran novetat va ser la integració amb Visual Studio. Això implica que els programadors poden utilitzar llenguatges més comuns com C# i Visual Basic.

És important comprendre les novetats que incorporava cada versió per tal de poder comparar correctament els productes. Hi ha molt debat per decidir quin producte és millor. Molts seguidors d'Adobe comparaven el Flex 3.0 amb el Silverlight 1.0. no hi ha comparació possible. Silverlight 1.0 es centrava simplement en aparèixer al mercat, però les grans novetats estaven previstes per la versió 2.0. Per tant, en el següent punt compararem les versions de Flex 3.0 amb Silverlight 2.0.

Avantatges i inconvenients de Flex

Adobe Flex té un gran avantatge sobre Microsoft Silverlight en la maduresa del producte i en la gran comunitat de desenvolupadors que hi ha al darrera. Però realment la gran diferència la trobem en que la màquina virtual de Flash està instal·lada en més del 90% de tots els PC's i portàtils de tot el món.

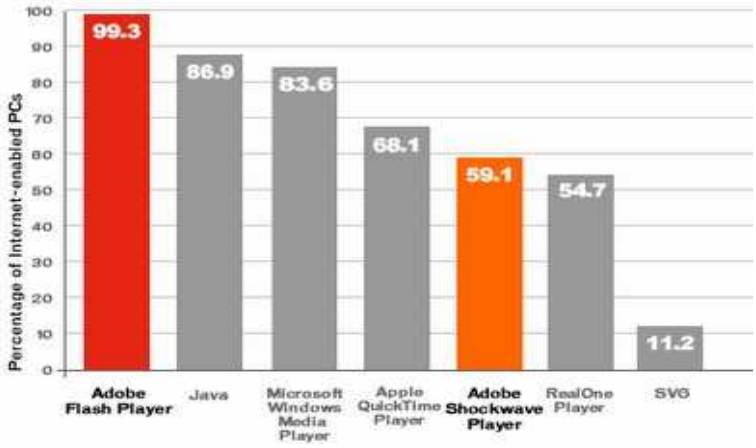


Figura 1 - Plugins més instal·lats als PC's. [9]

El *runtime* de Flash també es pot executar en múltiples plataformes (Windows, Mac, Linux, Solaris, HP-UX, Pocket PC, OS / 2, i altres). Flash es pot veure en més del 85% de tots els navegadors i és un producte multi-plataforma. Hi ha una gran comunitat de desenvolupadors i podem dir que els productes Flex són madurs, ja que han estat evolucionats a través de cicles de distribució molt importants. Un gran valor afegit del Flex és que el SDK és open source.

Una de les principals queixes sobre Flex és la corba d'aprenentatge pels desenvolupadors. Flex utilitza ActionScript 3, que és un llenguatge propi que es requereix per executar el reproductor de Flash. Per tant, és un altre llenguatge a aprendre pels desenvolupadors i no és tant comú com ho pot ser el Java o el C#.

	FLEX	SILVERLIGHT
Plug-in instal·lat	Més del 90% de PC's.	No és gens habitual.
Dependències	Totalment independent.	Depèn molt de .NET
Integració amb J2EE	Sí, amb BlazeDS	No
Multiplataforma	Sí, amb Windows, Mac, Linux i Sun Solaris.	No és compatible amb Linux.
RPC	SOA[10], HTTP i Remote Objects.	SOA.
Aplicacions per a mòbils	No és possible.	Sí, però hi ha poca informació al respecte.

Conclusions

La primera opció en ser descartada és AJAX ja que no es recomana fer una aplicació complerta amb aquesta tecnologia. JavaFX i GWT també es descarten ja que la manca d'informació existent fa que siguin opcions arriscades. Per tant, els productes que ens ofereixen més garanties són Flex i Silverlight.

El producte d'Adobe sembla la opció més segura degut a la gran comunitat que hi ha al darrera i a que porta més anys al mercat. Silverlight és un producte molt interessant ja que a partir de la versió 2.0 permet desenvolupar amb C#. Però com que les dos tecnologies s'han d'aprendre des de cero, és millor optar per la que fins ara s'utilitza més i en la que hi ha més informació per anar solucionant els requeriments.

Tot i així, la idea no és fer l'aplicació esclava de Flex, si en el futur apareix una nova tecnologia o alguna de les existents passa a adaptar-se millor a les necessitats actuals no hi hauria problema en canviar-la ja que les tres capes estan clarament separades.

2.1.4.-Frameworks de Flex

Per crear l'aplicació Flex és important escollir un *framework* ja que al ser una tecnologia desconeguda, el framework ens normalitza la manera de fer les coses. Existeixen el següents 4 *frameworks* que estudiarem a continuació:

Cairngorm[11]

És el més antic i per tant el més conegut en el món Flex. Bàsicament son un conjunt de patrons de disseny que funcionen bé junts. Segueix una metodologia molt a l'estil Java i es centra en tres àrees: accions de l'usuari, interaccions amb el servidor i l'encapsulació de la lògica de negoci.

Avantatges:

- És un projecte Open Source d'Adobe, té una gran comunitat de desenvolupadors al darrera.
- Utilitza estratègies del món Java que han donat grans resultats i que s'han utilitzat per crear projectes a gran escala.

- És molt recomanable per a projectes amb equip, ja que la metodologia de treball està molt ben definida i és fàcil dividir les tasques.

Inconvenients:

- La principal crítica és que cal escriure un gran número de classes. Cada event queda *mapejat* en un *command* i per tant cal escriure gran quantitat de codi. A part de l'*event* i del *command* també cal definir el *delegate* que utilitzarà cada *command*. Per tant, el número de classes s'incrementa ràpidament sigui gran o petit el projecte.
- Cairngorm implementa un mètode propi de control d'*events* i això pot arribar a complicar el model d'*events* integrat amb Flex.

Mate[12]

Es basa en un sistema de *tags*, dirigit per *events*, aquests *tags* són definits amb el llenguatge MXML.

Per crear un projecte amb Mate, només hi ha dos requeriments: tenir un o més events i definir un mapa d'*events* on hi hagi definits tots aquests events de l'aplicació i detallant com han d'actuar. Aquest framework també es coneix per seguir *el principi de Hollywood*[13], és a dir, els objectes estan construïts de tal manera que les dades que es necessiten s'injectaran a cada classe només quan siguin requerides.

Avantatges:

- No hi ha acoblament gràcies a que segueix *el principi de Hollywood*. Per tant cada component és molt lliure de ser reutilitzat. També supera la limitació de Cairngorm ja que no obliga a tenir una única resposta per event.
- Hi ha molta documentació i exemples de codi.

Inconvenients:

- Mate té dificultats per treballar amb BlazeDS.
- Només és llenguatge MXML, no es poden generar classes amb ActionScript.

- No té una rutina clara de funcionament i cal coordinar la metodologia de treball en projectes d'equip.

PureMVC [14]

Tot i que s'utilitza per a Flex, PureMVC no va ser realment dissenyat per a ser un framework de Flex, és una idea que es pot implementar en múltiples llenguatges i un d'ells és el Flex.

L'objectiu del PureMVC és seguir el *Model-View-Controller* (MVC) per separar les capes del model, la vista i el controlador. Es divideix el projecte en paquets i utilitza diversos patrons de disseny semblants a Cairngorm però afegint el *Facade* [15].

Avantatges:

- PureMVC actua en un marc molt ben definit i té una gran comunitat al darrera.
- Adequat per treballar en equips.

Inconvenients:

- Té els mateixos inconvenients que Cairngorm ja que utilitza la majoria dels patrons (*Singleton*[16], *FrontController*[17], *Command*[18], *Observer*[19]).
- Cal definir moltes classes.
- No va ser dissenyat per Flex i per tant li falten algunes de les característiques que incorpora el llenguatge MXML.
- Corba d'aprenentatge elevada.

Swiz [20]

Swiz segueix el patró de *Inversion of Control* (IoC)[21], una manera de fer que simplifica la gestió dels events i crides asíncrones a *Remote Objects*. L'objectiu és proporcionar una arquitectura MVC d'una manera senzilla i eficient. A diferència de Cairngorm i PureMVC, no imposa cap patró de Java i no segueix cap estructura en la creació de les carpetes del projecte.

En essència Swiz és un patró de *Factory*[22], és a dir, els components són carregats en aquesta *factory* mitjançant una classe anomenada *BeanLoader*. Quan s'inicia l'aplicació, la *factory* s'encarrega de la creació de les instàncies d'aquests components.

Swiz també proporciona la gestió de les dependències mitjançant un sistema de *tags* personalitzat que rep el nom de *Autowire*. Aquesta etiqueta és una manera de definir les dependencies entre les classes Swiz.

Avantatges:

- Swiz és fàcil d'utilitzar i no imposa una estructura predefinida en el projecte.
- Amb el *Autowire*, es redueix al màxim l'acoblament entre components.

Inconvenients:

- No defineix l'estructura del projecte i és una feina a fer per nosaltres. Per tant, pot ser complicat treballar amb equip.
- Utilitza uns *metatags* personalitzables que són mesures addicionals per realitzar un projecte i afegeix més arguments al compilador, no són passos difícils però només són requerits quan es treballa amb aquest framework.

Un cop presentats els 4 *frameworks* principals ja hi ha informació suficient per escollir-ne un, el projecte ha d'estar preparat per evolucionar en el futur i incorporar-hi més recursos, per tant sembla evident que s'hauria d'escollir una opció que presenti una metodologia clara de treball i ajudi des de bon principi a desenvolupar l'aplicació d'una manera més o menys ràpida tenint en compte que cal aprendre Flex des de cero.

Dit això, els dos *frameworks* que semblen més adequats són Cairngorm i PureMVC. Tenint en compte que cal aprendre la tecnologia, la creació d'excessives classes és un peatge que val la pena pagar.

Entre Cairngorm i PureMVC s'escull Cairngorm ja que si bé els dos *frameworks* són semblants el fet que Cairngorm es crees exclusivament per a Flex fa decantar la balança al seu favor.

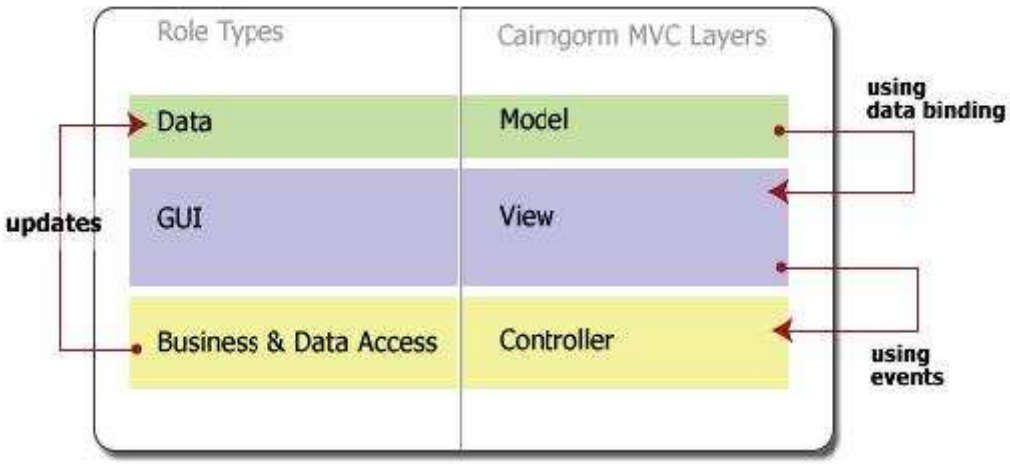


Figura 2 – Capes de Cairngorm

A continuació es detallarà com funciona el framework de Cairngorm ja que és el punt clau per aconseguir l'èxit en el projecte.

Cairngorm

Cairngorm té 5 components principals que s'utilitzen per fer una aplicació RIA amb Flex:

- *ModelLocator*: Repositori de dades globals a l'aplicació.
- *Services*: Repositori de serveis.
- *Commands*: Components que no formen part de la interfície d'usuari i que processen ordres cap a la part del *Business Delegate*.
- *Events*: Esdeveniments personalitzats que activen *commands*.
- *Controller*: Component necessari per a *enrutar* els events als *commands* corresponents.

En la següent figura podem veure com es relacionen aquests 5 components per acabar completant el framework sencer:

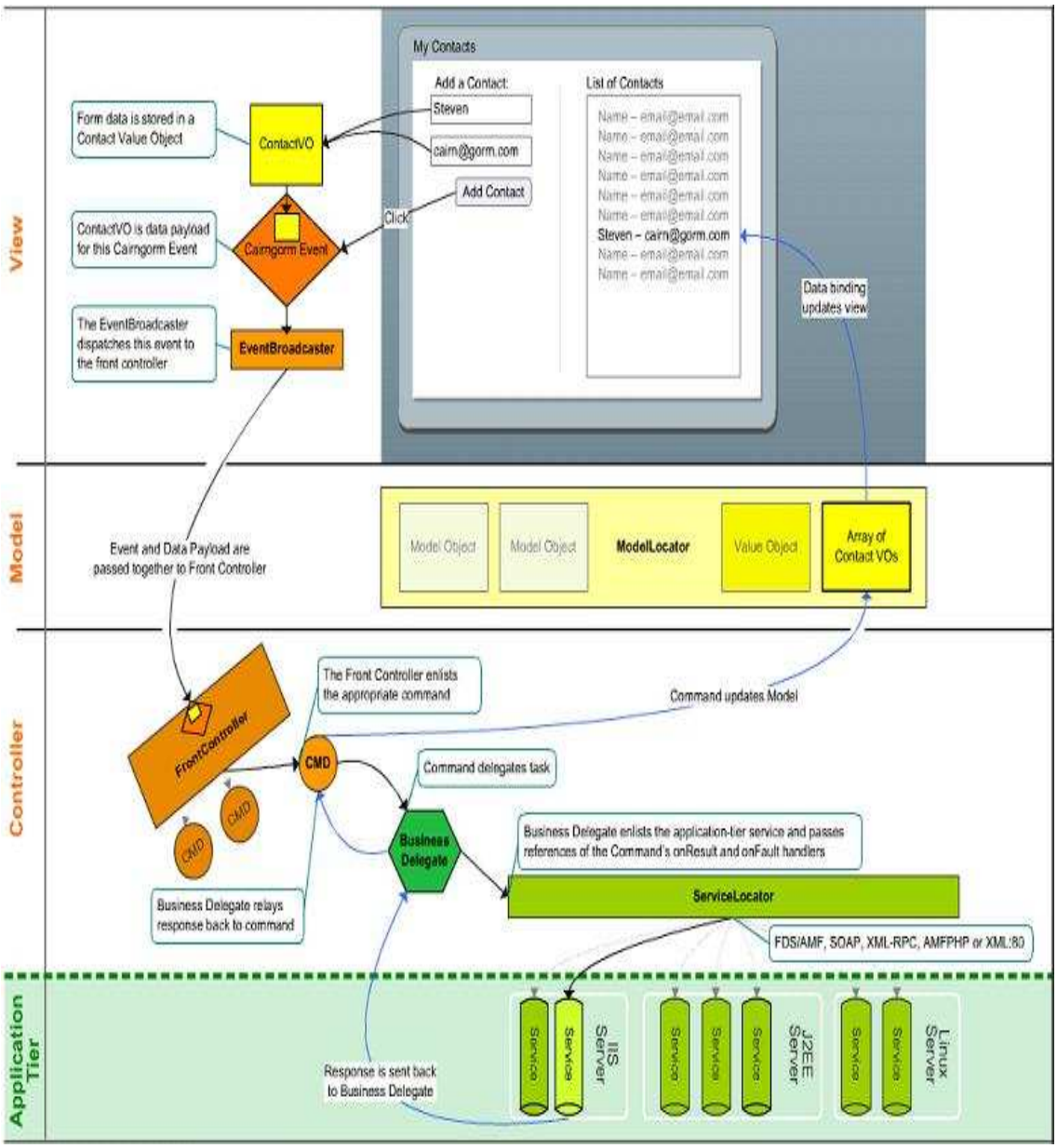


Figura 3 – Camí complet del framework de Cairngorm [23]

Mitjançant el següent exemple es detalla el camí que s'ha de seguir per implementar la nostra aplicació. En aquest exemple, presentarem la implementació d'afegir un contacte a una llista de contactes.

Solució amb Cairngorm

A l'exemple trobem el següent escenari senzill realitzat amb components Flex:

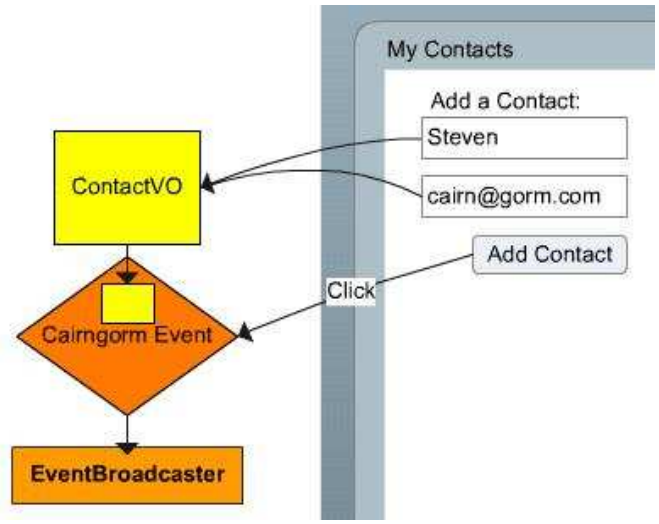


Figura 4 – Generació d'un event a Cairngorm

La informació que apareix en el formulari està representada a la classe *ContactVO*. Aconseguir aquesta perspectiva és molt senzilla, simplement s'han anat afegint els següents components pre-configurats:

Panel → Vbox → Form → FormItem1, FormItem2 i Button.

El primer pas del flux de Cairngorm succeeix quan es genera un event i correspon a prémer el botó de *Add Contact*. L'*EventBroadcaster* dispara l'*event* i el dirigeix cap al *Controller*.

En el *Controller* tenim definits tots els possibles events de l'aplicació i que cal fer quan s'executi cada un d'ells. És un arxiu on hi trobem una llista de la següent forma:

addCommand (EventDeterminatNumero1, VesAlCommandNumero1);

addCommand (EventDeterminatNumero2, VesAlCommandNumero2);

addCommand (EventDeterminatNumero3, VesAlCommandNumero3);

Per tant, el *Controller* té la intel·ligència necessària per saber cap a on s'ha de tirar en rebre un event.

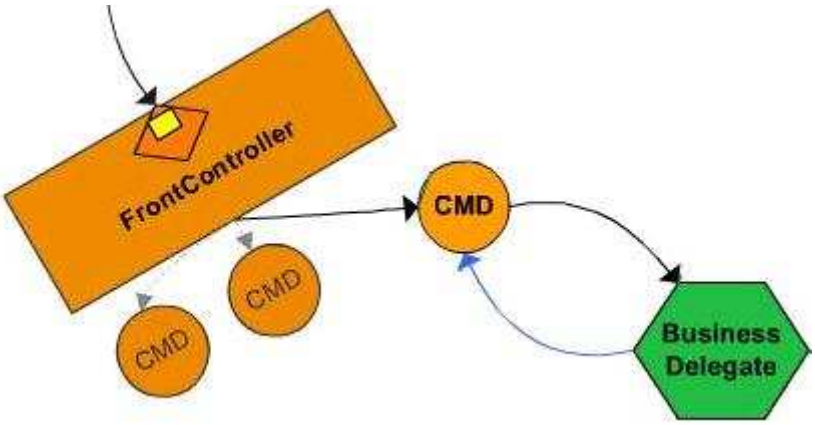


Figura 5 – El *FrontController* delega la responsabilitat

Arribats al *Command* corresponent ha de delegar la feina al *Business Delegate* assignat. I el *Business Delegate* és l'encarregat de fer la crida al servei Java.

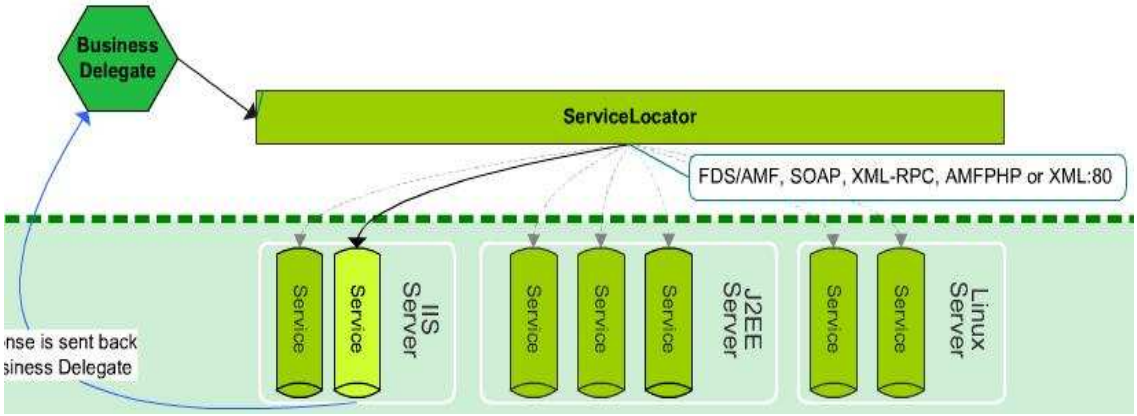


Figura 6 – *ServiceLocator* es comunica amb la part del servidor

La resposta a aquest servei retorna al *Business Delegate* i aquest ràpidament notifica al *Command* si tot s'ha executat correctament. En cas afirmatiu, el *Command* ha de notificar els canvis al *Model*, que és on tenim guardades totes les variables globals del projecte i per tant la informació que s'ha d'acabar mostrant a l'usuari.

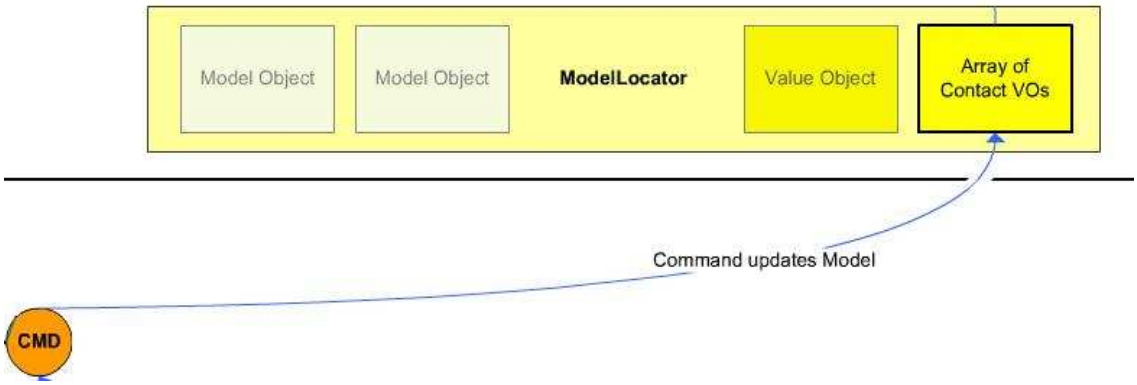


Figura 7 – El *Command* actualitza el Model

Finalment i d'una manera totalment automàtica el model és l'encarregat d'actualitzar la vista de l'aplicació i és quan l'usuari rep la resposta a la petició que acaba de realitzar.

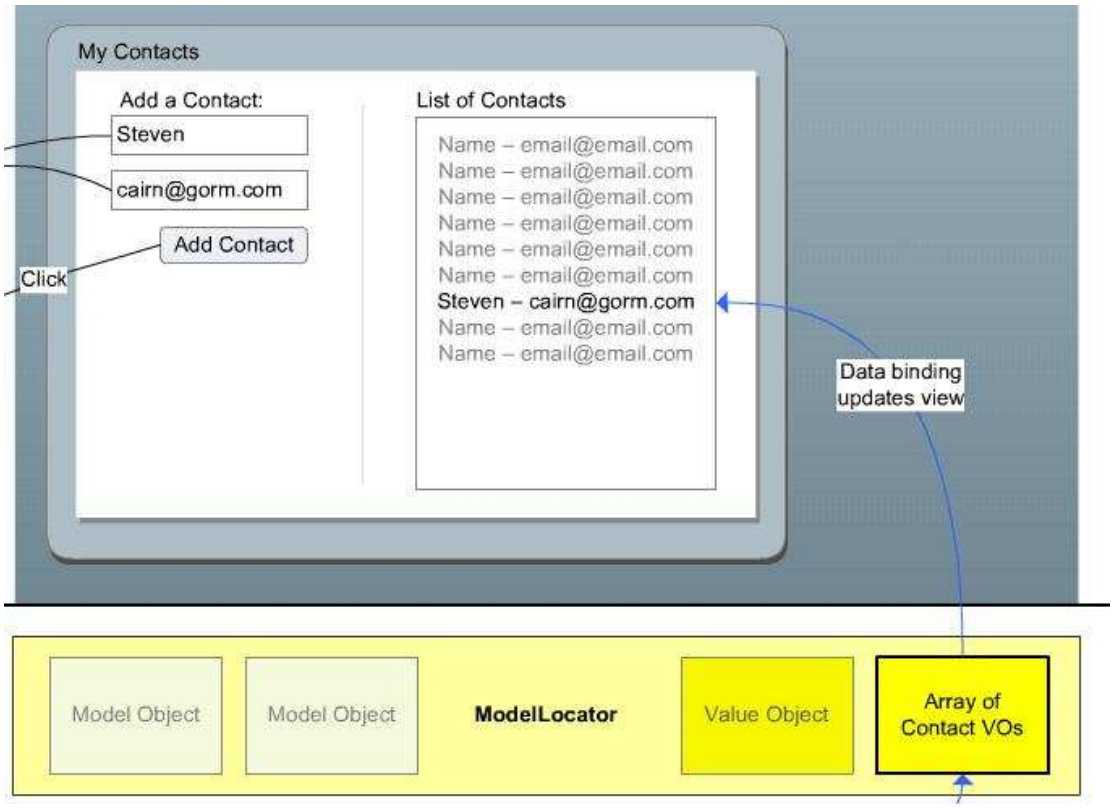


Figura 8 – La vista queda actualitzada

2.2.- Base de dades

Tant MySQL com Oracle són solucions contrastades per realitzar projectes però cal valorar quin s'adapta millor a l'aplicació concreta de Solmania buscant la relació entre preu/rendiment/escalabilitat.

- MySQL és relativament lleuger, i pot ser molt ràpid si tenim dissenyada una bona arquitectura.
- Oracle ofereix moltes característiques i és molt complet per resoldre problemes complexos.

A continuació es mostra una comparativa entre les principals característiques de MySQL i Oracle.

Funcionalitat	MySQL	Oracle
Punts forts	Relació preu/rendiment excel·lent per aprofitar-se d'una arquitectura d'aplicació	Capaç de donar solucions a base de dades molt grans i d'última tecnologia
Versions dels productes	Enterprise (€): Més estable. Community (gratis): La més utilitzada.	Enterprise (€€€€) Standard (€€) Standard One (€) Express (gratis):Fins a 4Gb
Perspectiva segons les aplicacions	Aplicacions web normalment no aprofiten al màxim les funcionalitats de la base de dades. Es concentren en operacions de lectura ràpida.	Com més complicades siguin les coses a fer, més s'aprofiten les característiques d'Oracle.
Administració	Trivial per la instal·lació i la posada en marxa. La configuració per a projectes grans pot ser molt	Es necessita molt coneixement previ. Pot ser molt complicat d'instal·lar però també molt potent si

	complicada.	es fa bé.
Popularitat	Molt popular per a projectes petits i mitjans.	Molt popular a les empreses més poderoses, també als projectes més grans.
Entorns de desenvolupament (de més a menys comuns)	1.- Php 2.- Java 3.- Ruby on Rails 4.- .NET 5.- Pearl	1.- Java 2.- .NET 3.- APEX 4.- Ruby on Rails 5.- Php
Taules	Les taules utilitzen motors d'emmagatzament. Cadascun d'aquests motors té característiques pròpies.	Poques taules amb moltes funcionalitats.
Opcions de partició	Lliure, funcions bàsiques	€€€€ amb moltes opcions
Opcions de replicació	Lliure, fàcil de configurar	€€€€ amb moltes opcions
Exportació/Importació	Opcions bàsiques.	Moltes opcions.
Procediments	Les característiques bàsiques. Escalabilitat limitada.	Característiques avançades. Escalabilitat molt bona.

Per tant, comparar MySQL amb Oracle no seria just ja que són dos productes enfocats per a necessitats molt diferents:

- Oracle té moltes funcionalitats de XML i moltes eines per gestionar la base de dades. Oracle:
    - Es pot utilitzar Oracle Express en un punt inicial i evolucionar fins a les versions més completes i cares per a fer qualsevol projecte imaginable.
    - Moltes característiques pròpies que redueixen el fet d'instal·lar-hi complements d'altres empreses (*3rd party software*).

- MySQL és excel·lent quan cal llegir dades amb molta velocitat per aplicacions web. MySQL:
    - Per a projectes petits que permeten reduir costos de desenvolupament.
    - Li falten moltes característiques d'Oracle, però generalment en els projectes amb MySQL no es necessiten moltes funcionalitats a nivell de base de dades.

Com es pot comprovar, l'aplicació de Solmania s'adapta molt millor a un MySQL ja que es desenvoluparà una aplicació web amb poca funcionalitat a nivell de base de dades i on a més a més sortirà molt més econòmic.

2.3.- Tecnologia emprada en el servidor

Un cop decidit que la capa de presentació serà amb Flex mitjançant el framework de Cairngorm i que la capa de base de dades serà amb MySQL, només queda escollir la tecnologia emprada a la part del servidor.

L'empresa PlayOff Informàtica és experta en Php i en un primer moment es va optar per anar per la solució més econòmica que suposa treballar amb una tecnologia ja coneguda.

Necessitem un component que ens faci de pont entre la capa de presentació i la part del servidor i semblava que el més adequat era Weborb. Aquesta tecnologia ens fa transparent l'enllaç entre Php i Flex, però el primer problema apareix en conèixer les limitacions que té la versió gratuïta del producte Weborb, l'aplicació de Solmania treballa amb grans volums de dades i cal escollir molt bé el tipus de connexió entre Flex i Php ja que sinó ens trobarem amb un gran coll d'ampolla.

Existeix el producte BlazeDS que és oficial d'Adobe, es pot utilitzar de manera gratuïta i és OpenSource, l'únic problema és que cal utilitzar una tecnologia menys coneguda per l'empresa a la part del servidor com és Java.

Java té avantatges respecte Php en el sentit que està totalment desenvolupat per ser orientat a objectes i és més robust, tot i que també és més pesat, però aquesta aplicació no és una web tradicional i per tant el principal avantatge de Php que és la immediatesa queda reduït, no així que aquesta aplicació ha de créixer molt i que probablement serà

més senzilla de mantenir amb Java (hi ha frameworks molt potents que en el futur es podrien afegir) que no pas amb Php.

Per tant, la tecnologia amb la que dissenyarem l'aplicació és la següent:

- Base de dades → MySQL
- Capa de servidor → Java
- Enllaç amb el client → BlazeDS
- Presentació → Flex

A continuació descriurem que és el BlazeDS i explicarem el protocol AMF que és el que utilitza el Blazeds i el compararem amb altres protocols com JSON i XML.

2.3.1.- BlazeDS

BlazeDS és una tecnologia a la part del servidor per a components desenvolupats en Java que permet connectar les aplicacions Flex amb Java utilitzant missatgeria asíncrona.

Actualment hi ha la necessitat de millorar les opcions de connectivitat entre les dades del client i el servidor. BlazeDS automatitza i simplifica el procés de fer crides entre el client de Flash i els mètodes Java que estan al servidor.

Utilitza el protocol binari AMF per la transferència de dades i d'aquesta manera s'eleva el rendiment considerablement, permetent que les aplicacions carreguin les dades fins a 10 vegades més ràpidament que amb formats basats amb text, com poden ser XML o SOAP[24].

BlazeDS està publicat dintre de la llicència *GNU Lesser General Public License Version 3* (LGPL) i les principals característiques són:

- Connexió per aplicacions Flex amb Java.
- Alt rendiment en la transferència de dades en aplicacions crítiques.
- Permet el *push* de dades per part del servidor, per tant s'actualitzen les dades en temps real.
- Lliure i de codi obert.

2.3.2.- Protocol AMF (Action Message Format)

AMF és un protocol binari inspirat en SOAP i té especificació oberta (LGPL), aconsegueix serialitzar els objectes ActionScript de forma compacta.

És un protocol que s'utilitza nativament en diverses API's del Flash Player per emmagatzemar i intercanviar dades i es transmet a través dels protocols HTTP/S, RTMP/S i RTMFP/S.

L'any 2002 va sortir la primera versió del protocol, la AMF0 que funcionava pel Flash Player 7 i 8 i que servia per aplicacions creades amb les versions 1 i 2 d'Action Script. Al 2007 surt la versió AMF3 (no hi va haver AMF2) que funciona amb el Flash Player 9 i serveix per les aplicacions creades amb Action Script 3.

Aquests són els principals avantatges que guanyem al utilitzar AMF:

- Els objectes AMF al ser representacions binaries, són molt lleugers i es comprimeixen amb *zlib*.
- Serialització/Deserialització:
    - El procés de conversió de AMF a objectes ActionScript i a l'inrevés, es realitza mitjançant API's natives de FlashPlayer implementades amb C, per tant és un procés extremadament lleuger.
    - És un procés automàtic i totalment transparent pel desenvolupador.
    - El protocol suporta tant objectes amb ActionScript natius com personalitzats:
        - *Boolean* (natiu)
        - *Rectangle extends Forma implements IDibuix* (personalitzat)
- Permet transferir vídeo (només amb RTMP/RTMFP), àudio i imatges.
- Hi ha diferents implementacions disponibles del protocol AMF i una d'elles és BlazeDS, en la següent imatge en podem veure d'altres:

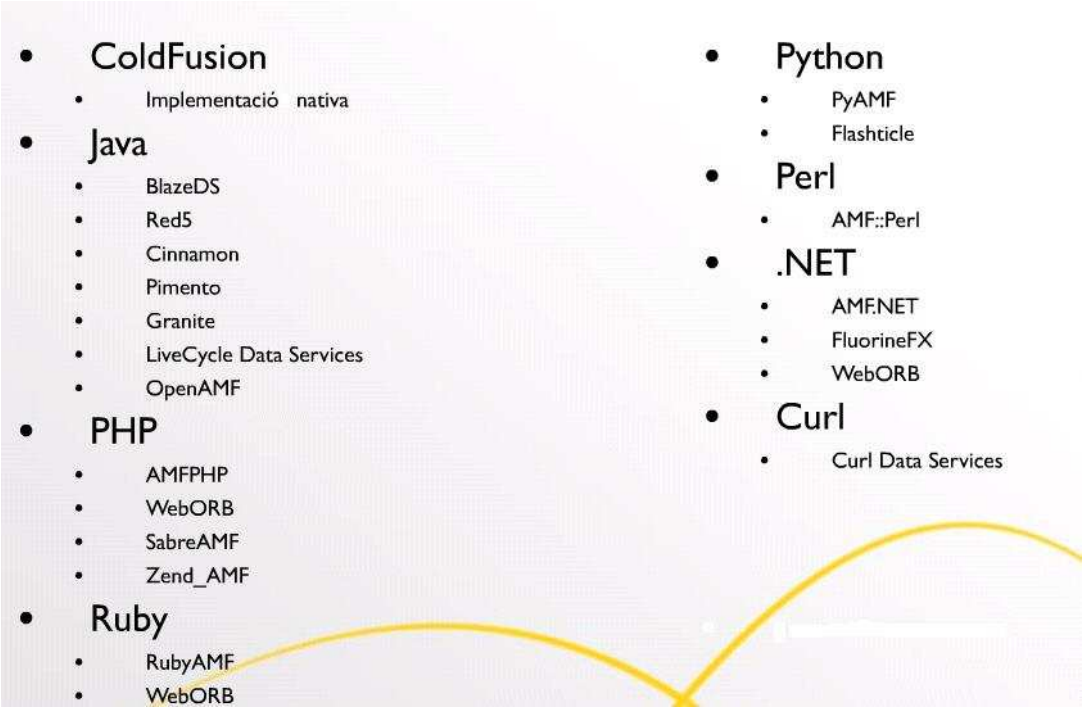


Figura 9 – Implementacions disponibles del protocol AMF

En el següent punt es compara AMF amb altres protocols.

Comparativa entre AMF, JSON i XML

El coll d'ampolla més gran a qualsevol sistema distribuït és el flux de dades, ja sigui d'escriptura i lectura sobre un disc dur o l'ús de protocols d'aplicació per les comunicacions en xarxa. Aquesta transmissió anomenada d'Entrada/Sortida (E/S) pot limitar enormement el rendiment d'una aplicació RIA.

Hi ha moltes coses que poden fer reduir la quantitat d'E/S que requereix una aplicació i aquestes tècniques normalment segueixen patrons. En el disseny de l'aplicació cal decidir com l'aplicació es connectarà amb altres processos, ja siguin locals o remots.

A part dels patrons d'E/S també tenim que tenir en compte els protocols de l'aplicació en si mateixos. Per protocol d'aplicació s'entén qualsevol cosa sobre els protocols TCP/IP, que pels propòsits de RIA, inclouen HTTP parametritzat, XML mitjançant HTTP, serveis basats en SOAP, CORBA IIOP, Java RMI, DCOM, JSON, AMF. Aquests protocols orientats a l'aplicació s'anomenen *Remote Procedure Call* (RPC).

JSON va ser desenvolupat per Douglas Crockford i va rebre molta atenció al aparèixer. És molt més eficient que l'ús del XML quan cal intercanviar informació amb missatges complexos, en canvi, HTTP és més ràpid quan la comunicació es fa amb dades simples.

Action Message Format (AMF), va ser desenvolupat per Adobe i és OpenSource, és un format de RPC que està molt estès quan cal desenvolupar una aplicació RIA. Tot i que el AMF és utilitzat principalment per desenvolupadors de Flash i Flex existeixen implementacions d'aquest protocol en PHP, Java i fins i tot .NET.

JSON i AMF semblen ser molt més eficients que XML sobre HTTP, però també depenen de l'existència de llibreries dissenyades per a codificar-ne el contingut. Per altra banda, XML rep el suport de tots els llenguatges de programació més o menys moderns i els parsers de XML són inclosos en les llibreries bàsiques de la majoria de les plataformes RIA (per exemple, Flex, Ajax, Silverlight,...). A més, XML és més fàcil d'interpretar quan es busca en una interfície web qualsevol.

Per tant, sembla que hi ha bons motius per separar els protocols RPC en dos categories dintre del món RIA: XML per a API's públiques i protocols com JSON o AMF per a les comunicacions privades.

En el cas d'APIs públiques, si es vol publicar una API per poder ser utilitzada per a qualsevol aplicació (per exemple, Amazon) cal proporcionar com a mínim suport per a interfícies XML. També es pot proporcionar accés per JSON o AMF, però cal fer-ho obligatòriament amb XML per tenir més acceptació.

En canvi en les API's privades, és a dir, les API's web utilitzades per alguna aplicació RIA individual o per la pròpia empresa, es pot triar entre XML, AMF, JSON o altres opcions. Si no cal comunicar-se amb l'exterior l'elecció s'ha de prendre estrictament per al propi ús de l'aplicació i escollir l'opció amb més sentit. En aquest cas, és molt recomanable escollir JSON o AMF. Qualsevol dels dos tendeix a ser molt més ràpid en la creació dels missatges i en el *parseig* de les dades que el XML i suposen una diferència significativa en el rendiment.

La qüestió és entre JSON i AMF, i cal veure si alguna de les dues aconseguirà dominar les RIA. Moltes plataformes utilitzen les dues opcions. Per exemple, PHP suporta tant JSON com AMF.

JSON va ser dissenyat específicament per a JavaScript. Al mateix temps AMF és un protocol dissenyat exclusivament per Adobe i per tant s'adapta molt millor al projecte Flex ja que AMF és el seu protocol natiu. Això no vol dir que XML mitjançant HTTP es deixi d'utilitzar, però a nivell de RIA és una opció remota.

En la següent gràfica podem comprovar com responen els protocols per enviar 5000 registres d'una taula. A la primera part de la figura 10 es mostra el temps que tarden les diferents opcions. Les opcions amb AJAX tenen una penalització molt alta en el pas de *parsejar* les dades, sobretot amb XML i SOAP. Utilitzant Flex amb AMF3 el temps de *parseig* queda pràcticament anul·lat ja que AMF3 actua com a capa transparent entre els objectes Java i els Flex i d'aquesta manera no cal traduir res.

A la segona part es mostra que Flex amb AMF3 és la opció que utilitza menys ample de banda, això és degut a que inicialment ens descarreguem la pel·lícula Flash i construïm la taula (seguint l'exemple de construir una taula amb 5000 registres). Per tant només cal enviar els registres ja que la taula es construeix inicialment i ja perdura. D'aquesta manera ens estalviem utilitzar molt ample de banda i s'aconsegueixen grans millores si ho comparem amb les opcions amb AJAX.

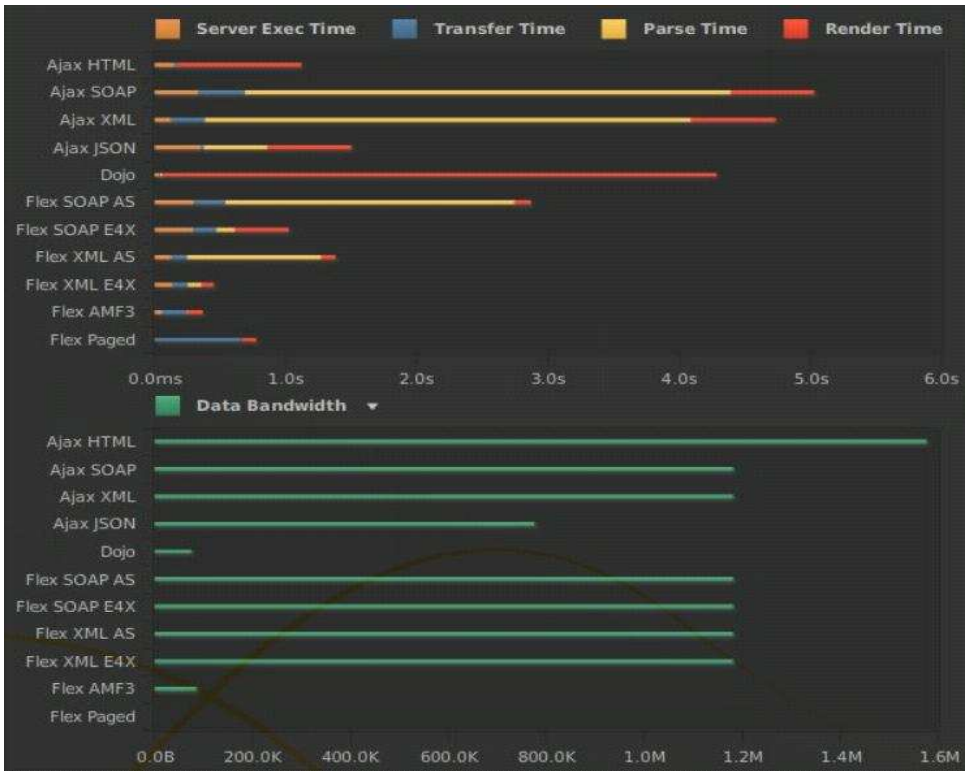


Figura 10 – Prova d'eficiència amb diferents tecnologies

Per finalitzar el capítol es definirà que és un *dashboard* i quines són les pràctiques a seguir per aconseguir crear un *dashboard* realment útil.

2.4.- Dashboard

Els *dashboards* ens ofereixen una solució única i potent per mostrar informació, però generalment acaben estant molt per sota del seu potencial real si no es tenen en compte alguns conceptes. Un *dashboard* fet amb Flex pot ser molt espectacular però cal donar importància al disseny visual per aconseguir que la informació que es mostra sigui desxifrada fàcilment per l'usuari.

L'objectiu principal d'un *dashboard* és el de comunicar, i per tant aquest concepte decidirà si el projecte té èxit o no. Podem utilitzar les llibreries gràfiques de Flex més avançades que si el disseny visual no és el correcte en pocs dies deixarà de ser útil. En aquest sentit, no és tant important si s'escull Flex, Silverlight o una altre tecnologia, l'èxit del *dashboard* es basa en si és capaç de comunicar-nos una informació de manera ràpida i clara. El *dashboard* pot aprofitar la potència del Flex per a comunicar, però això només servirà si s'entén com actua la percepció visual i se'ls hi apliquen els principis del disseny.

Uns efectes molt espectaculars poden semblar atractius en un primer moment, però al cap de pocs dies aquest factor es perd i si el *dashboard* no està realment ben fet, es perd la seva utilitat i passa a estar en desús en poc temps. En el següent gràfic es mostra un *dashboard* que a primera vista és molt atractiu però que falla en transmetre la informació. Té uns gràfics molt espectaculars però és fa molt difícil extreure'n la informació important, a la part esquerra es podria substituir els dos gràfics que segueixen l'estil de marcadors de velocitat per simples números que aconseguirien guanyar molt d'espai, de la mateixa manera uns gràfics de barres amb colors suaus anirien molt millor per la part central ja que l'excés de colors no facilita la comunicació i tantes ralles en els gràfics només dificulten el procés de transmetre la informació..

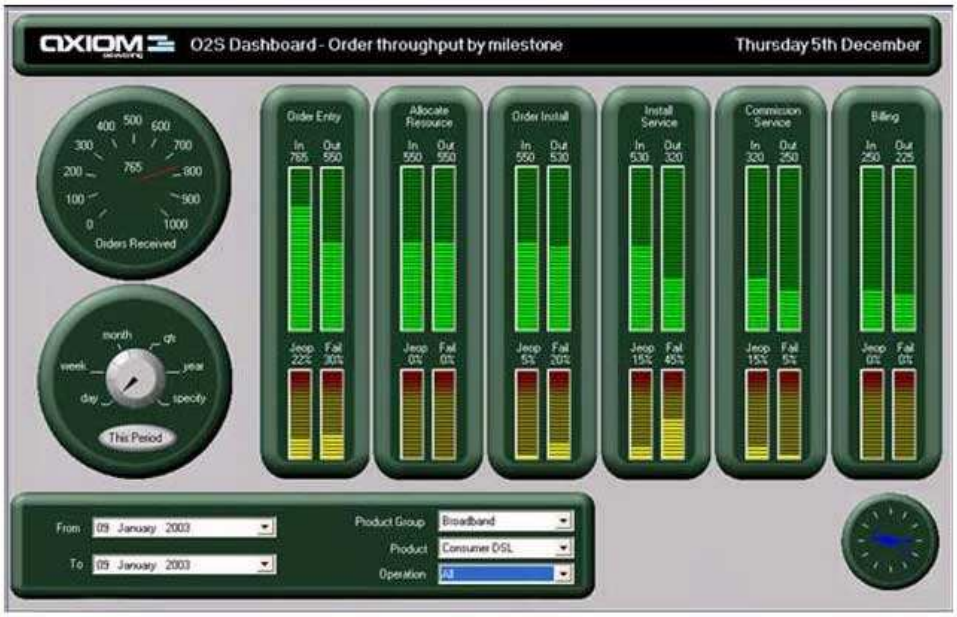


Figura 11 – *Dashboard* ineficient

2.4.1.- Què és un dashboard?

Un *dashboard* és un indicador visual de la informació més important necessària per aconseguir un o més objectius; organitzats en una mateixa pantalla de manera que la informació es pot visualitzar amb un cop d'ull [25].

A continuació es descriu cada punt una mica més en detall:

- *Un dashboard és un indicador visual*. La informació en un *dashboard* es presentada visualment, normalment com a combinació de text i gràfics, però emfatitzant els gràfics. Els *dashboards* tenen més part gràfica que textual, no perquè siguin més atractius que el text, sinó perquè la presentació ajudada de gràfics comunica més eficientment que només amb text. Per tant, per aconseguir un bon *dashboard* cal aprendre alguna cosa sobre la percepció visual.
- *Un dashboard mostra la informació necessària per aconseguir un objectiu*. Per aconseguir l'objectiu principal de comunicar, generalment es necessita accedir a una col·lecció d'informació que no necessariament ha d'estar relacionada. No ha de ser informació d'un tipus determinat, però ha de ser de qualsevol tipus que permeti arribar a l'objectiu.
- *Un dashboard cap en una sola pantalla*. La informació ha de cabre en una pantalla, totalment disponible a la mirada de l'usuari per veure-ho tot al mateix

temps. Si ens hem de desplaçar amb la barra d'*scroll* vol dir que s'han transgredit els límits del *dashboard*. Si cal canviar de pantalla per veure un altre informació vol dir que hi ha més d'un *dashboard*, l'objectiu és tenir la informació a l'abast i que sense cap tipus d'esforç es pugui assimilar.

- *Un dashboard s'utilitza per monitoritzar la informació d'una ullada*. Tot i que la informació de qualsevol tipus es pot representar en un *dashboard*, hi ha un tret característic en gairebé tots els *dashboards*: la informació apareix molt resumida. Això es perquè l'ull humà no pot quedar-se amb tots els detalls necessaris per aconseguir un determinat objectiu. Un *dashboard* ha de ser capaç de mostrar aspectes que mereixen l'atenció, serveixen com a primera mesura per conèixer en quin punt s'ha d'actuar en més detall.

Els punts anteriors descriuen l'essència del *dashboard*. Els *dashboards* són un tipus de visualització, una manera de presentar unes dades, no són un tipus de tecnologia. L'objectiu és clar, comunicar.

2.4.2.- La percepció visual

La visió és el sentit més poderós. Mirar i pensar estant molt relacionats. Per mostrar dades eficientment, cal entendre una mica la percepció visual.

Per crear un bon *dashboard* hi ha àrees que cal entendre:

- Els límits de la memòria a curt termini.
- Codificació visual per a la percepció ràpida.
- *Els principis de Gestalt*[26] per a la percepció visual.

Els límits de la memòria a curt termini

Les persones no mirem pels ulls, realment mirem pel cervell. Els ulls són els mecanismes sensorials que transmeten la informació al cervell, i és el cervell el que decideix que s'ha de percebre.

En els nostres ulls no es registra tot el que és visible en el món. Només una petita part del que els nostres ulls veuen es converteix en un focus d'atenció.

Hi ha diferents tipus de memòria:

- La memòria icònica (els registres sensorials de la vista).
- La memòria a curt termini.
- La memòria a llarg termini.

La memòria important per al *dashboard* és la de curt termini ja que és aquí on la informació es manté durant el seu processament. Aquestes són algunes característiques importants:

- És temporal.
- Una petita porció es dedica a la informació visual.
- Té una capacitat molt limitada.

Només podem guardar de tres a nou trossos d'informació visual alhora a la memòria de curt termini. És molt relatiu el que es considera com a "tros" d'informació i varia depenent de la naturalesa del que observem. Per exemple, un número individual en un *dashboard* es guarda en un tros complet, però un bon disseny gràfic implementat amb gràfics de varies línies també s'acaba guardant com a un sol tros i aquí radica l'avantatge dels gràfics respecte del text. Els *dashboards* s'han de dissenyar d'una manera que permetin guardar tota la informació alhora en els trossos que hi tenim disponibles.

Codificació visual per a la percepció ràpida

Existeix una primera etapa prèvia al processament de la informació a la que hem prestat atenció que ve marcada per un conjunt d'atributs visuals. Després hi ha la fase pròpia del procés de la informació que és una fase seqüencial i per tant molt més lenta que la fase prèvia. És millor mostrar la diferència de les dues fases amb un exemple que consisteix en trobar el nombre de vegades que surt el número 5 de la manera més ràpida possible:

12134243243124235789403817476432895912734104285943970327590638987092957409328572394007398973092789078190572349832857239400739897309278907819057234983285723940073989730927890781905723498

La resposta és que el número 5 surt 7 vegades, per aconseguir el resultat ha estat necessari un procés molt lent de cerca que equival a la pròpia fase de processament de la informació a la que prestem atenció. En canvi ara es presenta la llista de números així:

12134243243124235789403817476432895 91
27341042859439703275906389870929574 09
32857239400739897309278907819057234 98

Ara ha estat molt més senzill i ràpid trobar la solució, això és degut a que hi havia un atribut visual en forma de color que ens ha ajudat enormement. Aquesta fase sí que correspon a la part prèvia de processar la informació a la que hem prestat atenció. Cal destacar que les formes dels números també són atributs visuals que poden ajudar a la fase prèvia però en aquest cas eren massa complexes i no ens han servit, sí només hi haguessin hagut quadrats o rodones sí que n'hauriem tingut suficient sense la necessitat dels colors.

En general, podem distingir entre 4 categories d'atributs de percepció visual: color, forma, posició espaial i moviment.

Els principis de Gestalt per a la percepció visual

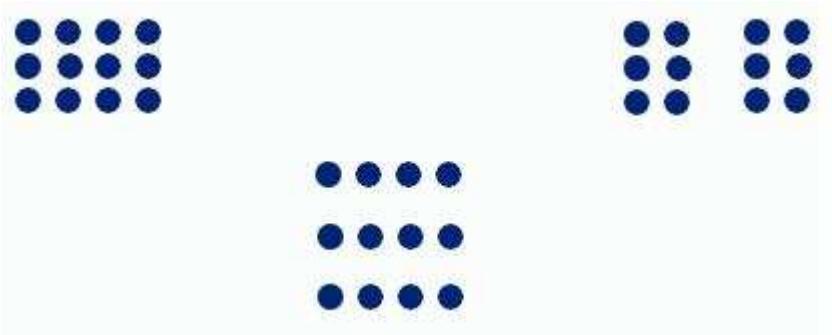
En el 1912, l'escola Gestalt de Psicologia va fer una gran tasca per trobar patrons de com organitzem la informació que veiem. *Gestalt* és un terme alemany que vol dir patró. Aquesta escola va fer un estudi que va suposar els principis de Gestalt, n'examinarem els sis més significatius per al disseny del *dashboard*:[27]

- Proximitat
- Tancament
- Similitud
- Continuïtat
- Englobació

El principi de la proximitat

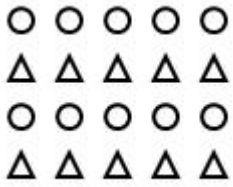
Nosaltres percebem els objectes que estan situats un a prop de l'altre i els situem en un mateix grup. La següent figura il·lustra aquest principi. Basant-nos amb la seva posició

relativa, nosaltres automàticament veiem els punts pertanyent a tres grups separats. Aquesta és la manera més fàcil de connectar dades que volem que estiguin agrupades. Els espais en blanc són una bona manera de crear la separació entre grups.



El principi de la similitud

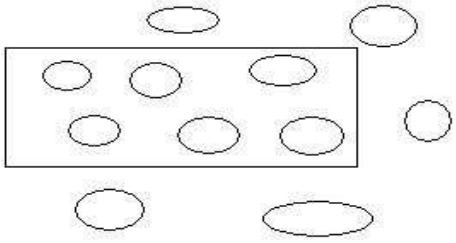
Tendim a agrupar els objectes que són semblants en forma, orientació, color i mida. La següent figura il·lustra el procés:



Aquest principi funciona especialment bé com a mitjà d'identificació de diferents conjunts de dades en un gràfic (per exemple, ingressos, despeses i beneficis). Tot i que les dades estiguin situades en diversos llocs del *dashboard*, aquest principi es pot utilitzar per crear enllaços entre les dades a vincular.

El principi de l'englobació

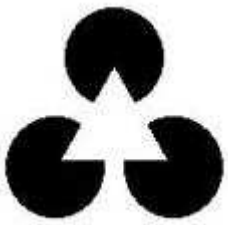
Percebem els objectes junts quan estan tancats per alguna frontera visual (per exemple, una línia o un camp comú de color). Aquesta frontera aconsegueix més separació entre els objectes de la que hi ha realment.



Aquest principi s'utilitza habitualment en forma de fronteres a les taules per tal de separar la informació clarament. No fa falta grans separacions per crear un efecte d'agrupació.

El principi de tancament

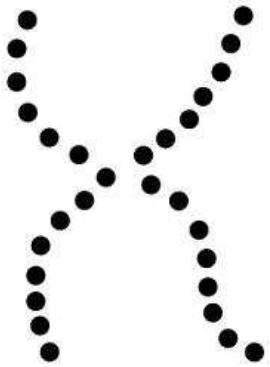
Quan ens enfrontem amb figures incompletes o amb formes estranyes, les acabem percebent com figures completes o les traduïm a formes més normalitzades. El principi de tancament diu que percebem les estructures obertes com a tancades, completes i regulars. La següent figura ho il·lustra:



Podem utilitzar aquesta tendència per percebre la totalitat de les estructures d'un *dashboard*, especialment en les gràfiques. Per exemple, podem agrupar objectes (punts, línies, o barres d'un gràfic,...) dintre de regions visuals sense la necessitat d'afegir-hi fronteres o colors de fons per definir l'espai.

El principi de continuïtat

Percebem els objectes com a un part completa, si estan alineats un al costat de l'altre. En el següent gràfic es poden veure els diferents punts com una gran X sencera.



Aquesta tècnica es pot utilitzar per alinear seccions i donar més importància a unes que a les altres en un *dashboard*.

Aplicació dels principis de la percepció visual al dashboard

Un dels reptes més importants alhora de dissenyar un *dashboard* és el d'aconseguir separar les dades importants de la resta, i aconseguir que la gran quantitat de la informació es mostri amb sentit i per tant es pugui percebre d'una manera eficient. Entendre les tècniques prèvies al processament de la informació i conèixer els principis de Gestalt ens pot ajudar a complir els objectius. D'aquesta manera podrem anar creant noves gràfiques o taules seguint unes pautes que ens garanteixen que la informació serà assumible per l'usuari.

3.- DISSENY

En aquest capítol partim de la situació inicial de Solmania on s'explica com funcionava l'empresa per obtenir informació dels franquiciats i treure'n estadístiques. Tot seguit es fa un primer disseny per conèixer totes les opcions que tindran els usuaris a l'aplicació. En la darrera part, es mostra com queda l'arquitectura de l'aplicació i quines opcions tindran els usuaris.

3.1.- Punt de partida

Fins ara, cada centre tenia un programa de gestió integrat que anava registrant les ventes i els usos de les diferents màquines. Aquesta informació es guardava en format de base de dades Paradox. Cada cert temps, la franquícia central de Solmania accedia mitjançant control remot a totes les franquícies i n'extreia la informació, això suposava que la franquícia a la que s'estava accedint no podia continuar treballant amb els inconvenients que això comporta. Un cop realitzat manualment aquest procés per a les més de 100 franquícies, la oficina central disposava d'una gran quantitat de dades que no sabia com gestionar i que amb prou feines en podia treure algun llistat.

Per tant, el primer pas és programar un procés diari que per FTP vagi enviant les dades de cada franquícia i d'aquesta manera queda superada la primera limitació d'aturar la producció del centre determinat. Un cop el servidor disposa d'aquesta informació s'ha de migrar la base de dades a MySQL per facilitar-ne el tractament.

Finalment, caldrà fer un estudi de la base de dades de l'antic programa de gestió i crear unes taules resum que seran les que ens serviran per representar visualment la informació requerida. Aquest darrer procés es realitzarà cada nit i durarà aproximadament unes 3 hores.

A continuació es mostra el disseny de les dos parts principals del projecte.

3.2.- Integració de la base de dades

Aquest part representa el pas inicial del procés i consisteix en la recuperació de totes les bases de dades de les franquícies. El sistema, haurà de ser robust, ràpid i amb capacitat

per treballar amb múltiples descàrregues FTP simultànies.

A més, s'haurà de poder verificar fàcilment si el procés de sincronització de dades ha tingut alguna incidència i en cas de succeir s'hauria d'enviar un correu electrònic automàticament al departament informàtic de Solmania.

El volum de dades a traspassar diàriament serà aproximadament de 12 Gb (100 Mb per franquícia). S'haurà de definir un horari (nocturn) pel traspàs d'informació. Aquest procés serà totalment transparent per l'usuari franquiciat.

Un cop tinguem tota la informació de les bases de dades Paradox al servidor central Solmania, ja podrem passar a la fase d'integració.

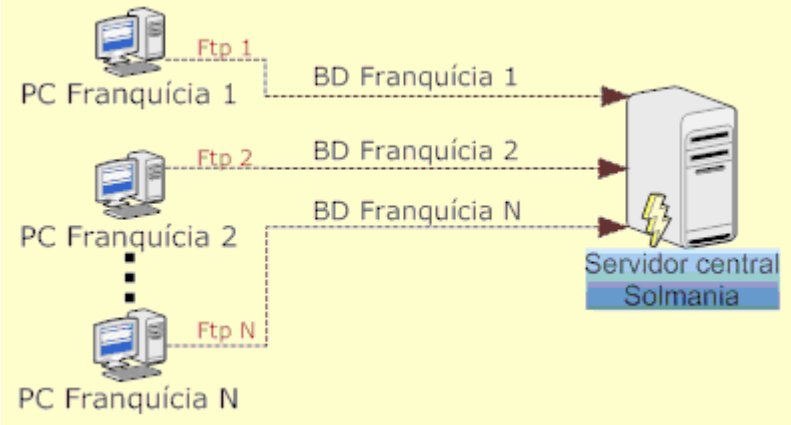


Figura 12 – Integració de les bases de dades

Aquest segona fase és el nucli del sistema, s'encarrega de transformar la informació de Paradox al sistema gestor de base de dades MySql. Aquesta transformació consisteix en un conjunt de processos que adapten les dades del sistema original a un nou model, molt més ric, afegint nous camps clau (zona, tipus franquícia,...) i amb la possibilitat d'agrupar la informació, a diferència del model actual que es basa en un sistema mono-franquícia.

El fet de definir correctament el nou model de dades és molt important per aconseguir una fiabilitat del 100% necessària per treure estadístiques.

3.3.- Aplicació RIA: Estadístiques Flex

Un cop extreta la informació més important arriba la part de treure'n partit, aquí és quan s'obtindran els resultats de totes les anteriors fases mitjançant la creació d'una aplicació RIA completa utilitzant Flex.

Flex actua a la capa de presentació i no és capaç de connectar-se directament amb base de dades, per tant necessitarem una capa de lògica de negoci implementada amb JAVA que ens aportarà els serveis necessaris per connectar-nos amb la base de dades i extreure la informació pertinent. Per comunicar Flex i JAVA s'utilitza Blazeds, que aconsegueix la connexió de manera transparent.

Tot això produirà un mòdul especialitzat en la presentació de les dades, generació de gràfiques i informes implementat en Adobe Flex 3.

Tindrem un motor de generació de gràfiques, alimentat amb serveis Java proporcionats pel mòdul anterior i un filtre per classificar el tipus i el nivell de detall de la informació:

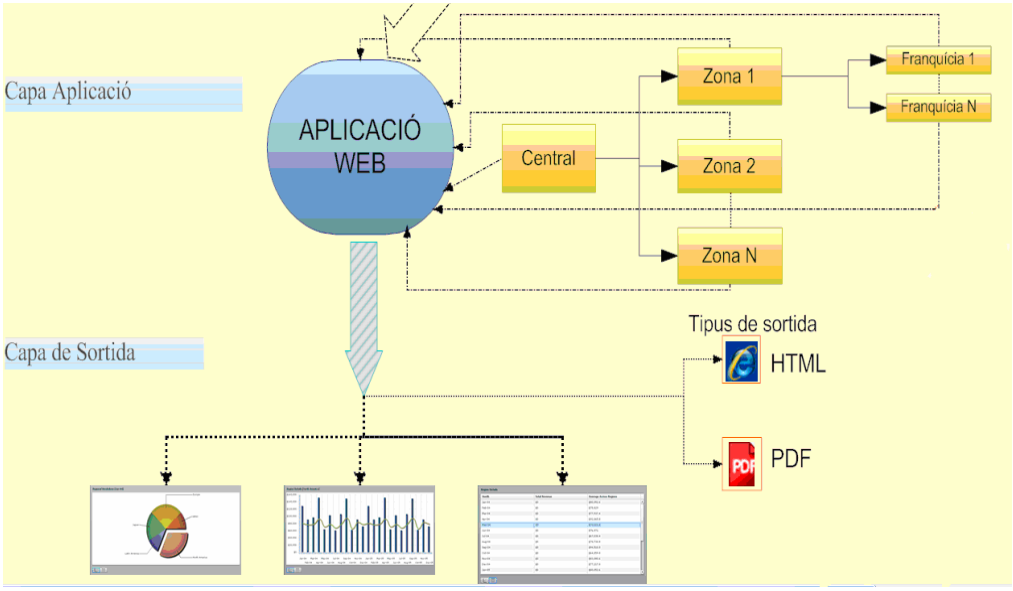


Figura 13 – Capa visual de l'aplicació

3.4.-Diagrama d'ús

Ara que ja tenim les dades que ens interessen cal dissenyar com utilitzarem tota aquesta informació a nivell visual, en una primera versió es va proposar aquesta diferenciació entre 4 grans línies d'estudi representades en el següent gràfic:

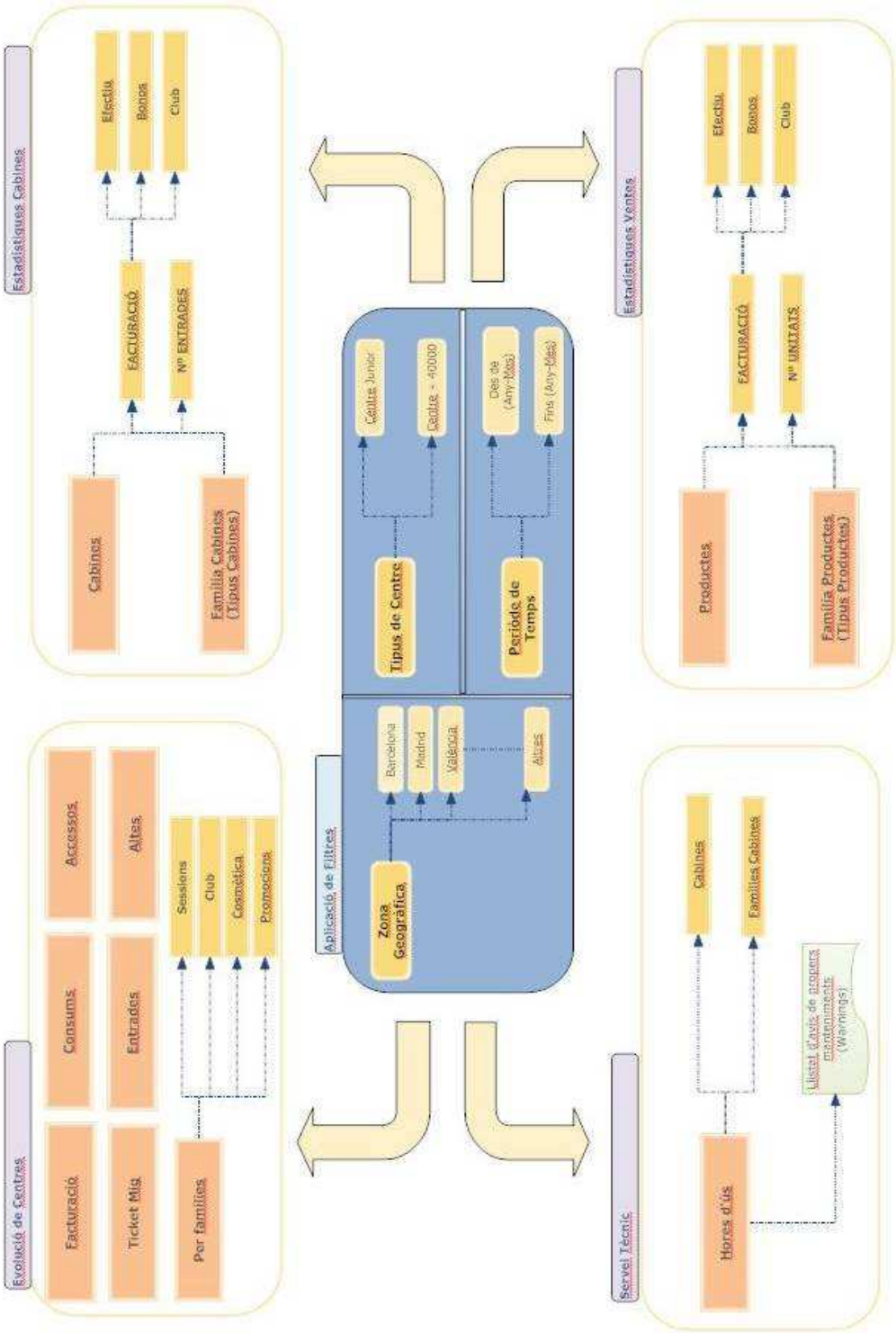


Figura 14 – Opcions de filtratge

Sempre hi haurà un filtre a nivell de centre, ja sigui depenent de la situació geogràfica, del tipus de centre o determinant un període de temps.

Un cop realitzat el filtre temporal i geogràfic tindrem 4 grans àrees de negoci per accedir. La principal és la que fa referència a l'evolució del centre corresponent, podrem comprovar diàriament com evoluciona la facturació, la venta de productes determinats, les noves altes i altres aspectes de cada centre.

També volem comprovar quines són les màquines que tenen més èxit i com les utilitza cada client, això seria la segona àrea de "estadístiques de les cabines". La tercera àrea faria referència al manteniment tècnic per conèixer possibles error ja sigui en un mal funcionament de les màquines de cada centre o si falla el procés d'importació de dades de cada franquícia.

Finalment trobarem les estadístiques de ventes del productes agrupades per la venta a socis o a clients externs i la manera de pagar aquests productes.

3.5.- Disseny detallat de l'aplicació

En aquest apartat entrarem al detall de l'aplicació explicant-ne el flux de l'execució i com queda finalment l'arquitectura de l'aplicació.

3.5.1.-Flux d'execució

Seguint la idea exposada en el gràfic del diagrama d'ús, cal decidir definitivament quin serà el flux d'execució, és a dir, les opcions de les que disposarà l'usuari per obtenir informació. Una opció senzilla i molt intuïtiva es mostra a continuació:

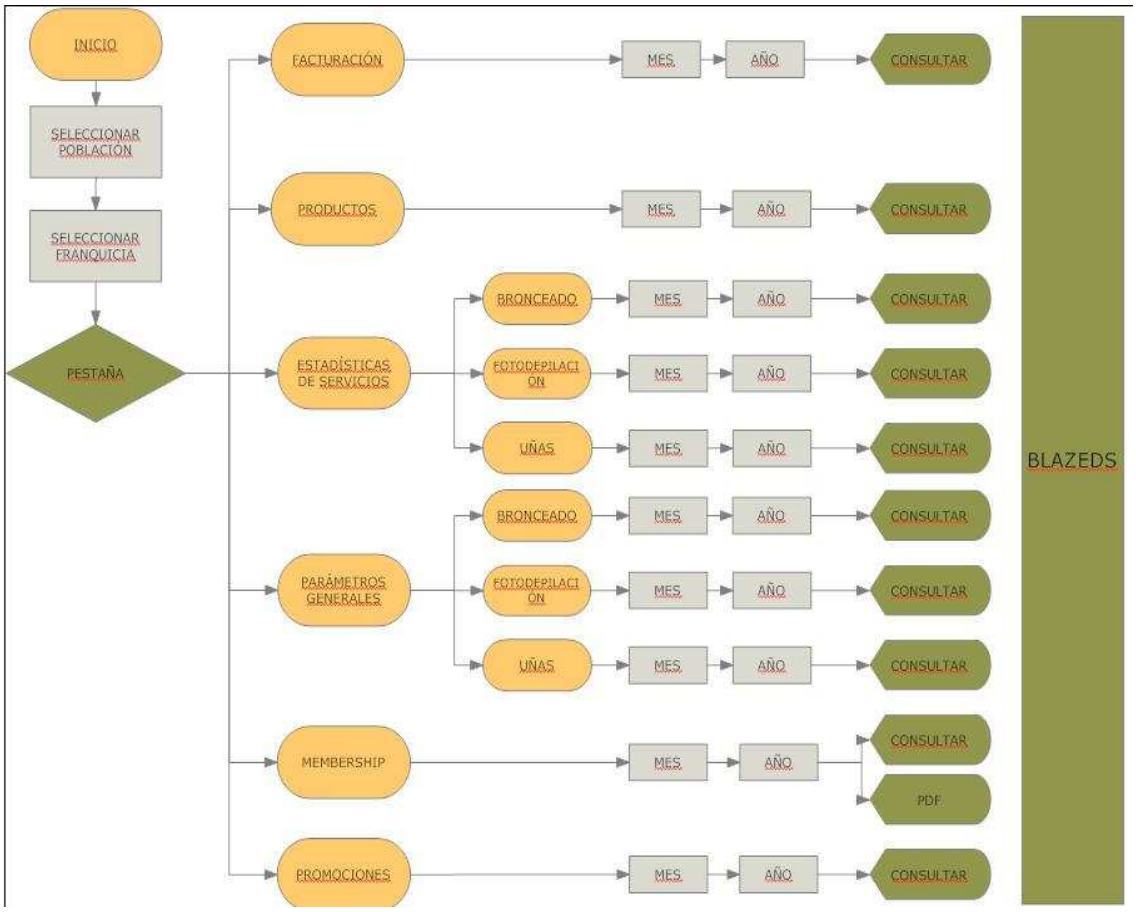


Figura 15 – Flux d'execució

El camí a seguir serà sempre el mateix, l'usuari seleccionarà una població i una franquícia, haurà d'escollir quina pestanya li interessa i un cop allà filtrar la informació per dates. Premerà el botó de consultar i en pocs instants obtindrà la informació requerida. D'aquesta manera l'usuari es farà seu el sistema en pocs instants i només polsant un o dos botons tindrà tota la informació.

3.6.- Arquitectura final

En el següent gràfic es mostra l'arquitectura completa de l'aplicació. Hi podem observar que hi ha clarament diferenciades les 3 capes:

- Front-End → Flex
- Back-End → Java
- Base de dades → MySQL

La informació és desada en un servidor MySQL i hi accedirem definint uns serveis amb Java que seran cridats directament des de Flex, d'aquesta manera s'obtindran informes

"en calent" de qualsevol dels productes de Solmania, amb gràfiques que comuniquin eficientment i la possibilitat de generar informes en format PDF o Excel.

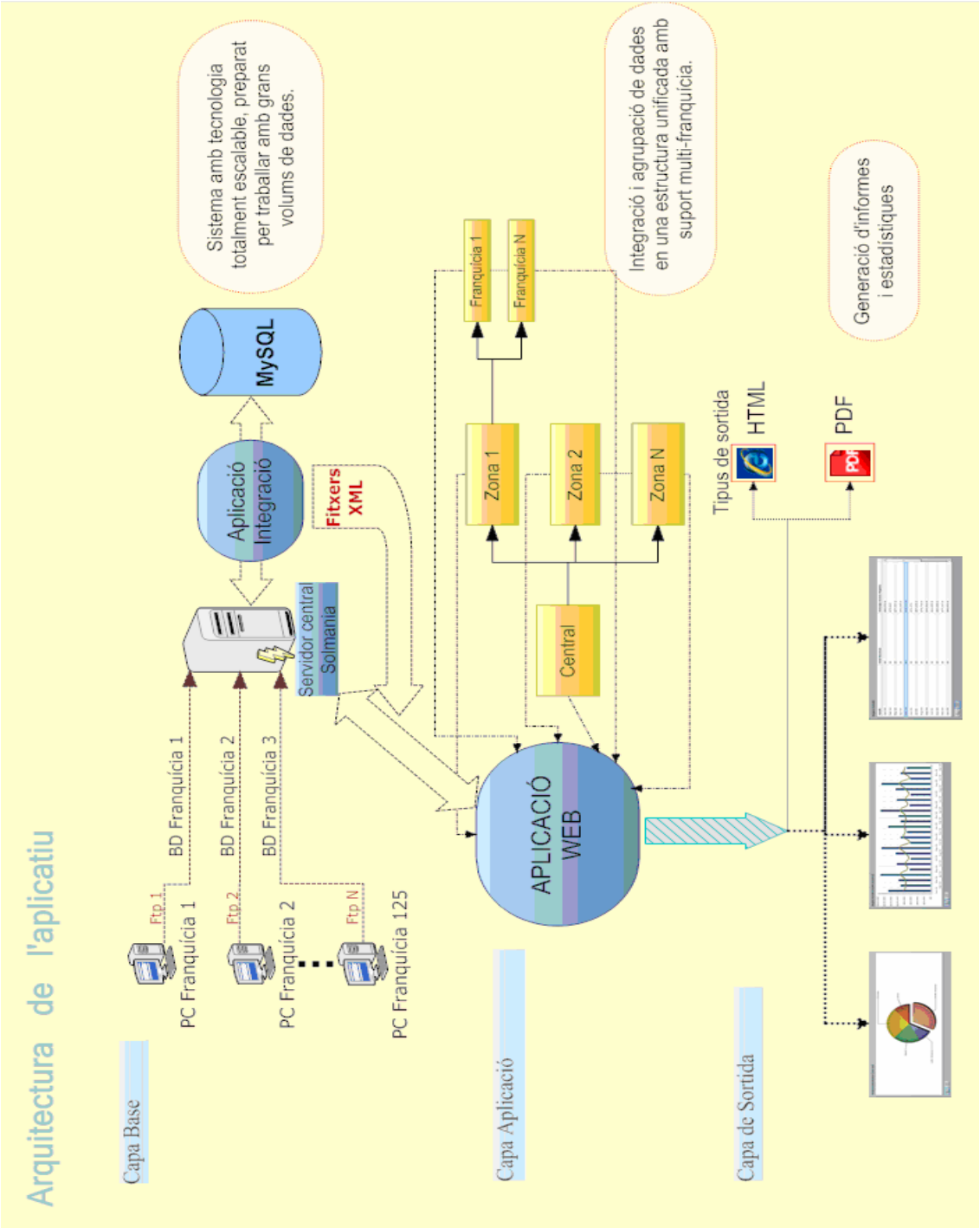


Figura 16 – Arquitectura final

4.- IMPLEMENTACIÓ

4.1.- Entorn de desenvolupament

Per codificar aquest projecte s'han fet servir dos entorns de programació. Per la part Java, s'ha utilitzat l'Eclipse. Per desenvolupar aquesta part, fa falta tenir el plugin de Java. A més, s'ha utilitzat l'eina Ant per compilar i crear un arxiu de tipus Jar.

Flash Builder és l'entorn amb el que s'ha codificat la part de Flex. És un software basat en l'Eclipse i permet escriure amb el llenguatge Action Script i amb MXML.

També necessitem tenir configurat el BlazeDS per connectar de manera transparent la capa Flex amb Java.

4.2.- Configuració del BlazeDS

Un cop descarregada la versió binaria del BlazeDS es tenien que seguir aquests passos:

El primer que cal fer és col·locar els arxius del BlazeDS dintre del servidor Java, en aquest cas és un servidor Apache Tomcat i els arxius BlazeDS es col·loquen dintre de [tomcat-home]\webapps.

Dintre de webapps el BlazeDS instal·la una sèrie de carpetes:

- WEB-INF: En aquesta carpeta aniran les classes Java que utilitza l'aplicació Flex.
- FLEX: Carpeta amb arxius xml que configuren la connexió.

El fitxer FLEX/remoting-config.xml és l'encarregat de definir les destinacions que s'utilitzen per la connexió, per exemple si la nostra aplicació Flex ha de cridar a un servei de la classe Java anomenada "facturación" dintre del package solmania, ho definirem així:

```
<destination id="facturacion">
        <properties>
                <source>solmania.Facturacion</source>
        </properties>
        <adapter                        ref="java-object"                       />
</destination>
```

L'altre fitxer important és el FLEX/services-config.xml on es defineixen les diverses opcions per realitzar la connexió, en el següent codi hi ha la definició dels canals AMF sobre HTTP i sobre HTTPS:

```
<channels>
        <channel-definition                                    id="my-amf"
        class="mx.messaging.channels.AMFChannel">
                <endpoint
                url="http://localhost:{server.port}/{context.root}/messageb
                roker/amf" class="flex.messaging.endpoints.AMFEndpoint" />
        </channel-definition>
        <channel-definition                                  id="my-secure-amf"
        class="mx.messaging.channels.SecureAMFChannel">
                <endpoint
                url="https://{server.name}:{server.port}/{context.root}/me
                ssagebroker/amfsecure"
                class="flex.messaging.endpoints.SecureAMFEndpoint" />
        </channel-definition>
</channels>
```

Per aconseguir implementar el projecte a l'empresa Solmania el servidor requeria un MySQL com a base de dades i l'Apache Tomcat com a servidor per la part Java. Pel que fa a la part dels client era necessari tenir instal·lada la versió 10 o posterior del plug-in de Flash.

4.3.- Part Flex

Aquestes són les pestanyes que s'han implantat a l'aplicació:

Figura 17 – Pestanyes de l'aplicació

Cadascuna d'aquestes pestanyes disposa dels seus serveis propis a la part Java i per aconseguir arribar fins a ells tenim que implantar tot un camí seguint la metodologia descrita al framework de Cairngorm. Aquesta és la manera com queden organitzades les carpetes per implementar el projecte Flex seguint Cairngorm:

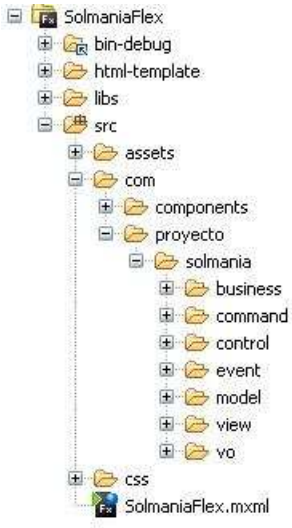


Figura 18 – Estructura de carpetes

A continuació es mostra la implementació completa de la pestanya de "Facturación" a la part Flex, aquesta pestanya probablement és la més utilitzada pels propietaris de Solmania. La resta de pestanyes segueixen el mateix funcionament però llògicament mostren dades i gràfics diferents.

El primer que farem es mostrar la imatge completa de com ha quedat la pestanya *Facturación* a la franquícia que Solmania té a Sabadell.

Figura 19 – Pestanya completa de Facturación

Anem a veure els processos que s'han seguit per aconseguir els resultats finals de la pestanya de facturació des del punt inicial de l'aplicació que és el "login":

L'entrada del programa és una finestra de login:

Figura 20 – Pantalla de login

1.- Hi ha dos tipus de rols d'usuaris, el soci que és propietari d'algunes franquícies, i que per tant només tindrà accés a les seves franquícies i el rol de tipus administrador que podrà accedir a totes les franquícies.

Un cop l'usuari introdueix *usuari* i *password* i es prem el botó de login l'aplicació està codificat per a que es generi un event de tipus *FranquiciesEvent*:

```
var evt:FranquiciasEvent =
        new FranquiciasEvent(FranquiciasEvent.EVENT_VERIFICAR_USUARIO);
evt.propiedadesConsulta.usuari = log_username.text;
evt.propiedadesConsulta.password = log_password.text;
evt.dispatch();
```

A l'arxiu *FranquiciesEvent* hi ha definits tots els possibles events de tipus *FranquiciesEvent*, com per exemple el de verificar usuaris, el de retornar totes les poblacions d'una determinada franquícia,...

2.- Cairngorm ens diu que els events han d'estar registrats en algun lloc, aquest lloc és el *controller*, allà hi tenim tots els possibles events del projecte i és on es decideix el camí que ha de seguir cada event individualment:

```
addCommand(FranquiciasEvent.EVENT_VERIFICAR_USUARIO,
        ConsultaUsuarioCommand);
addCommand(FranquiciasEvent.EVENT_CONSULTA_POBLACIONES,
        ConsultaPoblacionesCommand);
```

3.- Per tant, codifiquem l'aplicació per tal que quan es generi un event de tipus *EVENT_VERIFICAR_USUARIO*, el *controller* ens dirigeixi cap al *command ConsultaUsuarioCommand*. En el *ConsultaUsuarioCommand* recollim l'*event* amb els valors de l'usuari i el password i enviem la petició cap al *delegate* corresponent que s'anomena *consultaVerificarUsuario*.

```
var delegate:PoblacionesDelegate = new PoblacionesDelegate(this);
var franquiciasEvent: FranquiciasEvent = FranquiciasEvent(evento);
delegate.consultaVerificarUsuario(franquiciasEvent.propiedadesConsulta.usuari,franqui
ciasEvent.propiedadesConsulta.password);
```

4.- Ara ja estem a l'arxiu *PoblacionesDelegate* (*delegate*) i aquest és l'encarregat de cridar al servei Java anomenat *verificarUsuario* d'una manera transparent:

```
public function consultaVerificarUsuario(usuari:String,password:String):void
{
        var call : Object = service.verificarUsuario(usuari,password);
        call.addResponder(responder);
}
```

5.- El servei Java ens retorna el resultat de la consulta indicant si l'usuari i el password introduïts són correctes, aquesta informació arriba un altre cop a l'arxiu *command ConsultaUsuarioCommand* i si tot ha anat bé ens omplirà la variable *poblaciones* amb totes les poblacions a les que té accés aquell soci i ens passarà de la vista del login a la principal de les pestanyes; si el login és incorrecte ens mostra un missatge d'error i seguirem a la vista del login.

```
public function result (data:Object):void{
        if(data.result.length > 0){
                _model.poblaciones = data.result as ArrayCollection;
                _model.estadoAplicacion=_model.SELECCIONADA_PRINCIPAL;
        } else {
                Alert.show("LOGIN INCORRECTO!");
                _model.estadoAplicacion = _model.SELECCIONADA_LOGIN;

                }
        }
```

6.- Considerem que el login s'ha realitzat correctament, per tant ja se'ns carrega la vista principal amb el combo de *poblacions* omplert i on la de *Facturación* apareix per defecte.

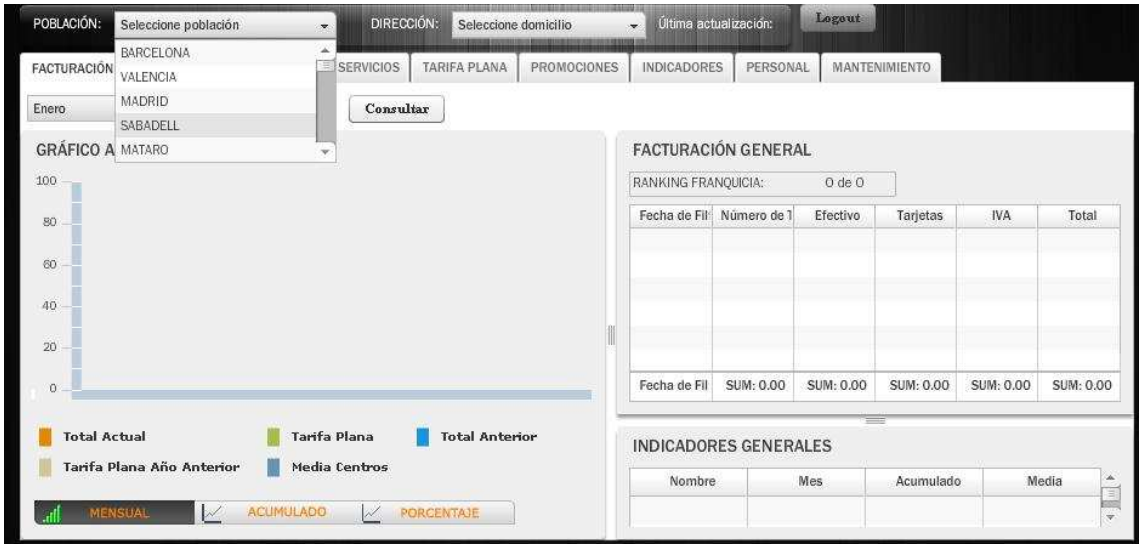


Figura 21 – Pestanya de facturació buida amb el combo de poblacions

7.- Quan seleccionem una de les poblacions es genera un nou event de tipus *FranquiciesEvent*, en aquest cas aquest event ens retorna totes les direccions de les franquícies d'aquella població.

Figura 22 – Selecció del combo del domicili

8.- Un cop tenim clar que volem consultar la facturació de la franquícia Rambla, 180 de Sabadell i havent seleccionat un mes i un any concret, cal prémer el botó de *Consultar* i se'ns disparen diferents events.

```
private function ejecutaEventos():void{
        if(comboDomicilio.selectedItem == null){
                Alert.show("Por favor, seleccione una franquicia");
        }else{
                visualizarFacturacionGeneral();
                visualizarFacturacionGeneralAnyo();
                visualizarRanking();
                visualizarIndicadores();
        }
}
```

9.- Tots aquests 4 events estan registrats al fitxer *FacturacionEvent* tal i com es mostra a continuació:

```
public static const EVENT_FACTURACION_GENERAL : String =
        "Consulta facturacion general";
public static const EVENT_FACTURACION_ANYO : String =
        "Consulta facturacion año";
public static const EVENT_FACTURACION_RANKING : String =
        "Consulta facturacion ranking";
public static const EVENT_FACTURACION_INDICADORES : String =
        "Consulta facturación indicadores";
```

10.- De la mateixa manera que hem vist per fer el login, aquests 4 events estan descrits al *Controller* de l'aplicació per tal de conèixer el camí a seguir. Per tant, cada un d'aquests events seguirà un camí diferent:

```
addCommand(FacturacionGeneralEvent.EVENT_FACTURACION_GENERAL,
        ConsultaFacturacionGeneral);
addCommand(FacturacionGeneralEvent.EVENT_FACTURACION_ANYO,
        ConsultaFacturacionAnyo);
addCommand(FacturacionGeneralEvent.EVENT_FACTURACION_RANKING,
        ConsultaFacturacionRanking);
addCommand(FacturacionGeneralEvent.EVENT_FACTURACION_INDICADORES,
        ConsultaFacturacionIndicadores);
```

11.- Estudiarem detalladament el *ConsultaFacturacionGeneral*, els altres 3 events funcionen d'una manera similar. Aquest arxiu recull l'event específic amb els paràmetres que li hem passat (idfranquicia, mes, any) i delega el funcionament al *delegate* corresponent.

```
public function execute (evento:CairngormEvent):void
{
        var delegate:FacturacionGeneral =
                new FacturacionGeneralDelegate(this)
        var facturacionGeneralEvent: FacturacionGeneral =
                FacturacionGeneral(evento);
        delegate.consultaFacturacionGeneral(idfranquicia,mes,anyo);
}
```

12.- El delegate relacionat amb el *ConsultaFacturacionGeneral* és el *FacturacionGeneralDelegat* i aquí definim tots els serveis Java existents.

```
public function consultaFacturacionGeneral(id:Number,mes:Number,anyo:String)
{
        var call : Object = service.getFacturacionGeneral(id,mes,anyo);
        call.addResponder(responder);
}

public function consultaFacturacionAnyo(id:Number,anyo:String,opcion:Num)
{
        …
}

public function consultaFacturacionRanking(id:Number,mes:Number,anyo:String)
{
        …
}

public function consultaFacturacionIndicadores(id:Number,mes:Number,anyo:String)
{
        …
}
```

13.- El servei Java realitza la consulta al MySQL i els resultats ens arriben en forma d'array al command *ConsultaFacturacionGeneral*.

```
public function result (data:Object):void
{
        _model.facturacionGeneral = new ArrayCollection;
        _model.facturacionGeneral = data.result as ArrayCollection;
}
```

14.- Els resultats queden registrats en unes variables globals que definim en el nostre *Model*, aquí hi ha definides totes les variables importants del projecte i a més es segueix el patró Singleton per crear aquestes variables només un cop.

```
public static function getInstance():Modelo
{
        if (_modelLocator == null) {
                _modelLocator = new Modelo();
        }
        return _modelLocator;
}
public var poblaciones:ArrayCollection = new ArrayCollection();
public var domicilio:ArrayCollection = new ArrayCollection();
public var facturacionGeneral:ArrayCollection = new ArrayCollection();
public var facturacionAnyo:ArrayCollection = new ArrayCollection();
public var facturacionAnyoTotal:ArrayCollection = new ArrayCollection();
public var facturacionPorcentual: ArrayCollection = new ArrayCollection();
…
```

15.- Un cop tenim totes les variables amb la informació correcta falta associar cada una d'aquestes variables amb la gràfica o la taula adequada. Fins ara tot era codi ActionScript, aquí és quan ens situem en la part gràfica de l'aplicació i aprofitem els components ja creats amb MXML que ens ofereix Flex i només fa falta associar-los a cada variable del nostre model [28]

```
<mx:ColumnChart
        dataProvider="{facturacionGeneral}"
        width="100%"
        height="100%"
        showDataTips="true"
        id="column">
        <mx:horizontalAxis>
                <mx:CategoryAxis categoryField="mes" />
        </mx:horizontalAxis>
        <mx:series>
                <mx:ColumnSet type="stacked">
                        <mx:ColumnSeries yField="totalActual"
                        displayName="Total Actual"/>
                        <mx:ColumnSeries yField="memberBanco"
                        displayName="Membership Banco" />
                </mx:ColumnSet>
                <mx:ColumnSet type="stacked">
                        <mx:ColumnSeries yField="totalAnterior"
                        displayName="Total Anterior" />
                        <mx:ColumnSeries yField="memberBancoAnterior"
                        displayName="Membership Banco Año Anterior" />
                </mx:ColumnSet>
                <mx:LineSeries displayName="Media Centros"
                yField="mediaActual" form="segment"/>
        </mx:series>
</mx:ColumnChart>
```

16.- A nivell gràfic, el camí que hem seguit acabaria produint el següent resultat on hi podem veure les columnes definides anteriorment com a *ColumnSet* i les línies com a *LineSeries*.

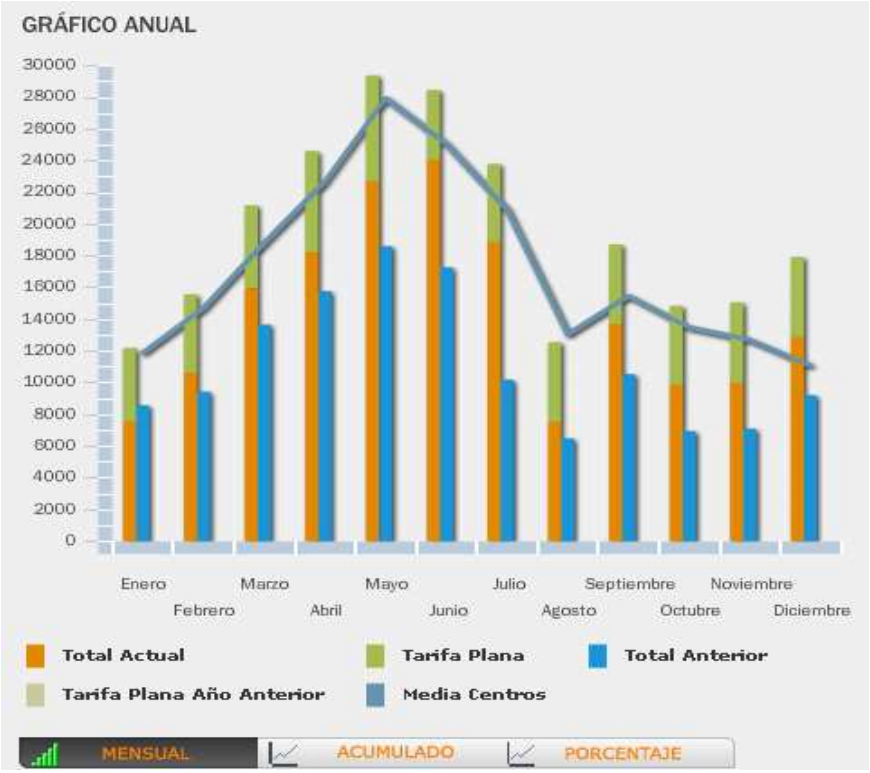


Figura 23 – Gràfica de la facturació anual

4.4.- Part Java

La part de Java actua sempre de la mateixa manera, rep una petició Flex i realitza una consulta a la base de dades.

Les entitats que cal codificar amb classes són les següents:

- *Facturació*
- *Productes*
- *Tarifa Plana*
- *Ungles*
- *Cabines*
- *Manteniment*
- *Personal*
- *Fotodepilació*

Cadascuna d'aquestes entitats queda codificada amb dos classes.

La primera classe correspon al Value Object (VO), aquí definim els estats i les variables de l'entitat. En el següent exemple es presenta l'entitat *ProductosVO*:

```
public class ProductosVO {

        private int idfranquicia;

        private Date datafiltre;

        private String datamin;

        private String datamax;

        private float serveis;

}
```

La segona classe de l'entitat és l'encarregada de crear la *query* que s'executa a la base de dades, i de retornar el resultat a la capa de Flex mitjançant un array de tipus *ProductosVO*, a continuació es mostra un mètode simplificat de la classe Productos:

```
public Vector<ProductosVO> getProductos(Number id){

        Connection conn = GetConnection.getSimpleConnection();
        Vector<ProductosVO> data = new Vector<ProductosVO>();

        Statement stmt = conn.createStatement();
        ResultSet res = stmt.executeQuery ("Consulta sobre productes");
        while(res.next()){
                float serveis = res.getFloat("serveis");
                int bons = res.getInt("bons");
                float total = res.getFloat("total");

                data.add(new ProductosVO(serveis,bons ,total));
        }
        return data;
}
```

Finalment hi ha una classe *GetConnection* que serveix com a punt d'entrada a les connexions a la base de dades, aquesta classe la utilitzaran totes les altres classes (*Productos, Facturacion, Personal*,...) per no crear infinitats de connexions.

4.5.- Passos finals

Els passos per implementar el projecte han estat els següents:

1. A l'entorn de desenvolupament del Flash Builder compilem el projecte Flex i creem una "Release Version".
2. A l'entorn de desenvolupament de l'Eclipse compilem el projecte Java i creem un arxiu Jar mitjançant la tasca Ant.
3. Mitjançant un client FTP ens connectem al servidor de Solmania.
4. Copiem els arxius Flash al servidor.
5. Copiem l'arxiu .JAR al servidor, aquest arxiu conté les classes Java compilades.
6. Reiniciem l'Apache Tomcat per a que els nous canvis siguin visibles.

Un cop obert el port 8080 no hi va haver majors complicacions per comprovar que l'aplicació s'havia implantat d'una manera correcta en el servidor de Solmania.

5.- CONCLUSIONS I MILLORES

5.1.-Conclusions

L'aplicatiu permet analitzar des de diversos punts de vista la informació que genera el negoci en quant a facturació, tipus d'ús de cabines, famílies de productes, manteniment de màquines, etc... A més es pot visualitzar des de diferents nivells de detall:

- A nivell global
- Per zones geogràfiques
- Per tipus de franquícies
- Per franquícia final

Això comporta un estalvi de temps important ja que totes les dades estan sempre actualitzades al servidor central i no s'ha de connectar ningú remotament per recollir informació. A més es guanya en transparència de cara al franquiciat, el qual no veu que li agafen el control de l'ordinador.

Cada soci té un codi d'accés per controlar com funcionen les seves franquícies d'una manera ràpida i segura.

A part d'aportar un benefici en quant a disposar de tota la informació de forma ordenada i fiable, marca un model de dades aplicable a una futura aplicació completament a mida personalitzada per Solmania, un sistema obert a qualsevol nova línea de negoci com estètica, massatges o altres.

El departament d'administració pot extreure informes "en calent" en format pdf i excel i fer-hi les modificacions pertinent.

Ara Solmania disposa d'una base de dades moderna correctament estructurada amb tot l'històric de dades i amb la possibilitat de realitzar una migració senzilla cap a una nova aplicació sense haver de començar de zero, amb l'estalvi econòmic i en temps que això suposa.

El model seguit per dissenyar la base de dades global ha estat pensat fent servir un concepte modular per poder afegir fàcilment noves línees de productes (estètica, massatge, ...).

Com a benefici col·lateral, després de la implantació, Solmania disposa de còpia de seguretat de les base de dades de tots els franquiciats de forma diària, protegint-los de qualsevol incidència respecte les dades (robatori, incendi, ...).

Finalment també serveix com a mecanisme de control per detectar incongruències com per exemple diferències entre l'import facturat per una cabina i les hores d'ús.

5.2.- Millores

El projecte està preparat per anar afegint noves pestanyes d'una manera senzilla, aquestes són les millores previstes per a versions futures:

- Aconseguir que els usuaris tinguin la possibilitat de configurar-se l'aplicació al seu gust, de manera que puguin definir quines gràfiques volen veure inicialment.
- Poder fer comparatives entre dos franquícies sobre qualsevol producte.
- Filtrar als clients de Solmania pels seus gustos mitjançant consultes totalment lliures. Per exemple, trobar els clients que consumeixen habitualment un tipus de producte i poder enviar e-mails amb campanyes publicitàries.

6.- BIBLIOGRAFIA

[1] Web oficial d'Adobe Flex. http://www.adobe.com/es/products/flex/

[2] Especificació del protocol AMF. http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf

[3] Web oficial de BlazeDS. http://opensource.adobe.com/wiki/display/blazeds/BlazeDS/

[4] Web oficial d'Adobe Air. http://www.adobe.com/es/products/air/

[5] Web oficial de Microsoft Silverlight. http://www.microsoft.com/SILVERLIGHT/

[6] Web oficial de JavaFX. http://javafx.com/

[7] Web oficial de JSON. http://www.json.org/

[8] Web oficial de GWT. http://code.google.com/intl/es-ES/webtoolkit/

[9] Estadística plugins als PC's. http://avtecmedia.com/web-site-design/adobe-flash-multimedia.htm

[10] Entrada a la wikipedia que s'explica que és SOA. http://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios

[11] Web oficial de Cairngorm. http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm;jsessionid=D5342CE2AC41E1E947DF3196EE96F2EF

[12] Web oficial de Mate. http://mate.asfusion.com/

[13] Entrada de la wikipedia on s'explica que és el Principi de Hollywood. http://en.wikipedia.org/wiki/Hollywood_Principle

[14] Web oficial de PureMVC. http://puremvc.org/

[15] Patró Facade. http://en.wikipedia.org/wiki/Facade_pattern

[16] Patró Singleton. http://en.wikipedia.org/wiki/Singleton_pattern

[17] Patró Front Controller. http://en.wikipedia.org/wiki/Front_Controller_pattern

[18] Patró Command. http://en.wikipedia.org/wiki/Command_pattern

[19] Patró Observer. http://en.wikipedia.org/wiki/Observer_pattern

[20] Web oficial de Swiz. http://swizframework.org/

[21] Patró Inversion of Control. http://en.wikipedia.org/wiki/Inversion_of_control

[22] Patró Factory. http://en.wikipedia.org/wiki/Factory_method_pattern

[23] Diagrama de Cairngorm. http://www-student.it.uts.edu.au/~vicvo/cairngorm.gif

[24] Especificacions de SOAP. http://www.w3.org/TR/soap/

[25] Colin Ware: “Information Visualization: Perception for Design”, Editorial Morgan Kaufmann, 2000.
[26] Els Principis de Gestalt.
http://graphicdesign.spokanefalls.edu/tutorials/process/gestaltprinciples/gestaltprinc.htm
[27] Stephen Few: “Information Dashboard Design”, Editorial O’Reilly, 2006.
[28] Michael Labriola, Matthew Boles, James Talbot: “Adobe Flex 3: Training from the Source”, Editorial Adobe Press, 2008.

ANNEX A: Imatges de l'aplicatiu

Figura 1 – Login de l'aplicació.

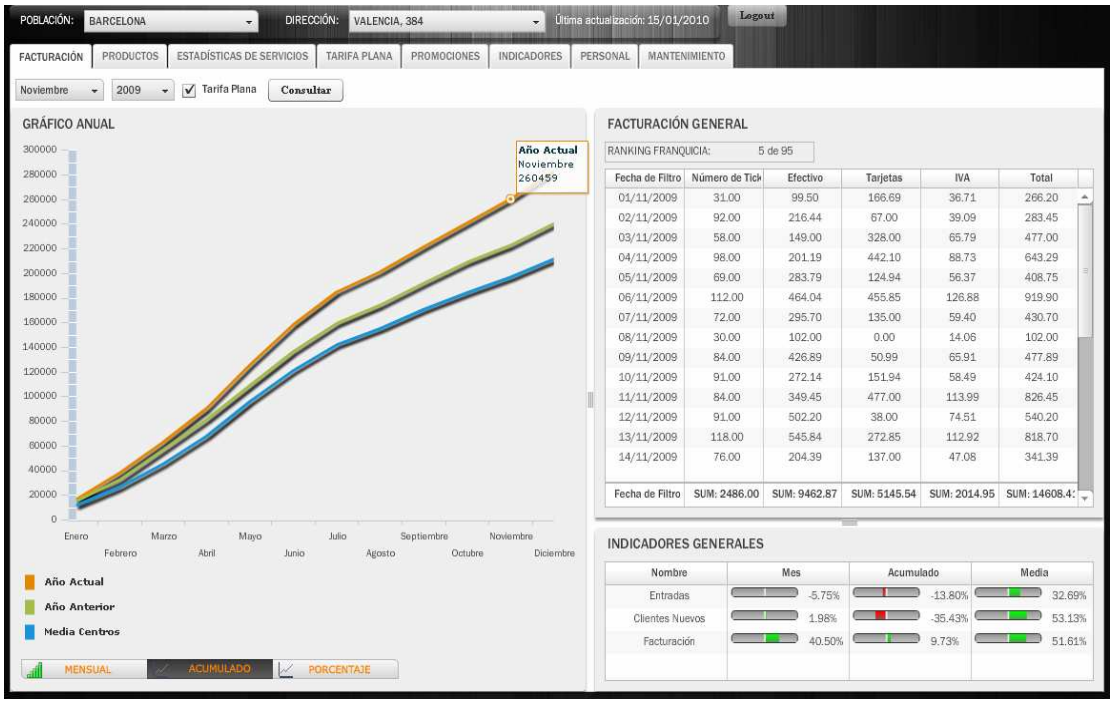


Figura 2 – Pestanya de "Facturación". Podem veure gràfiques referents a la facturació mensual d'aquest any comparades amb les de l'any passat. També en quina posició es trobaria aquesta franquícia en comparació amb la resta i un resum dels indicadors principals.

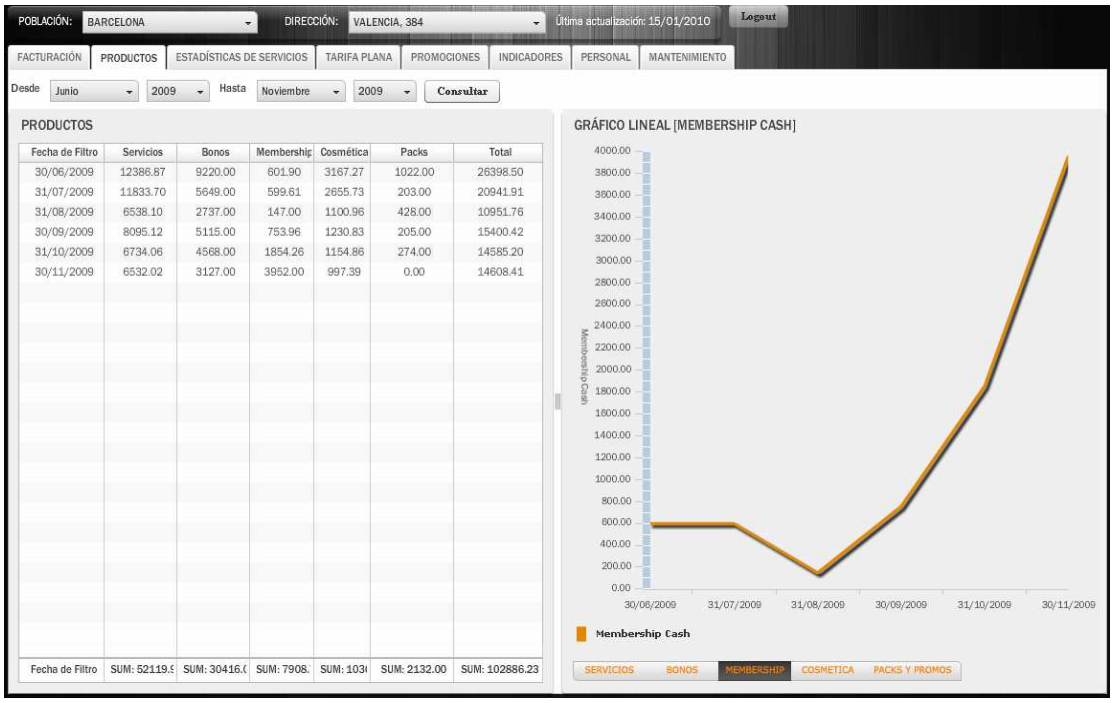


Figura 3 – Pestanya de "Productes". Podem veure una taula principal separant els diferents productes i al costat una gràfica de cada tipus per veure com evoluciona cada tipus al llarg del temps.

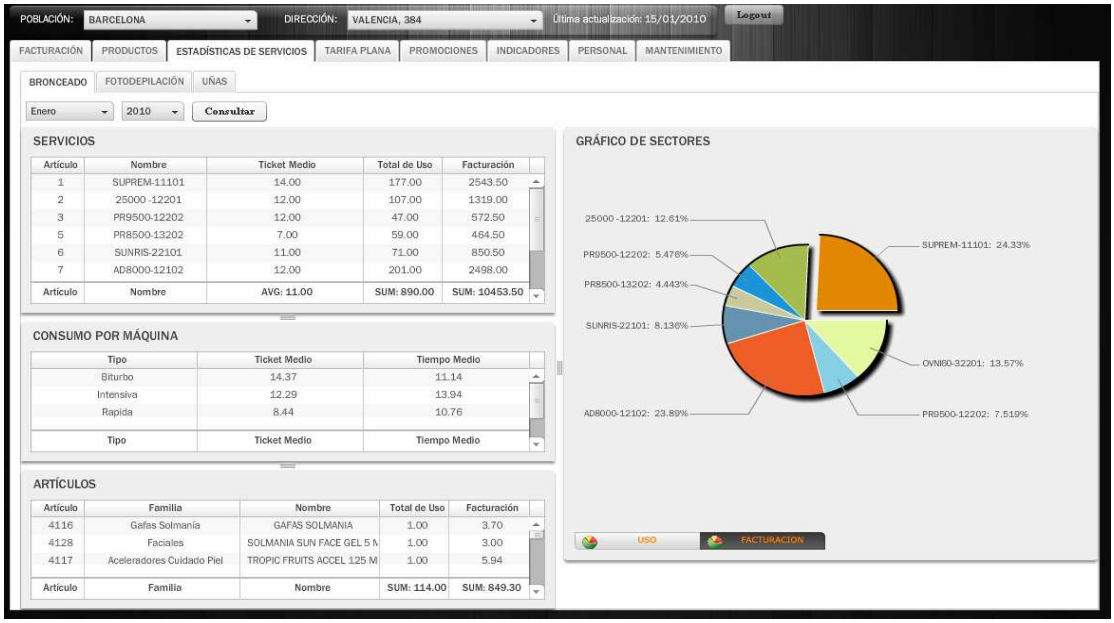


Figura 4 – Pestanya de "Bronceado". A la part esquerra podem veure quines són les màquines de bronzejat que donen més rendiment i a la part dreta el corresponent gràfic d'aquestes mateixes màquines en termes de facturació.

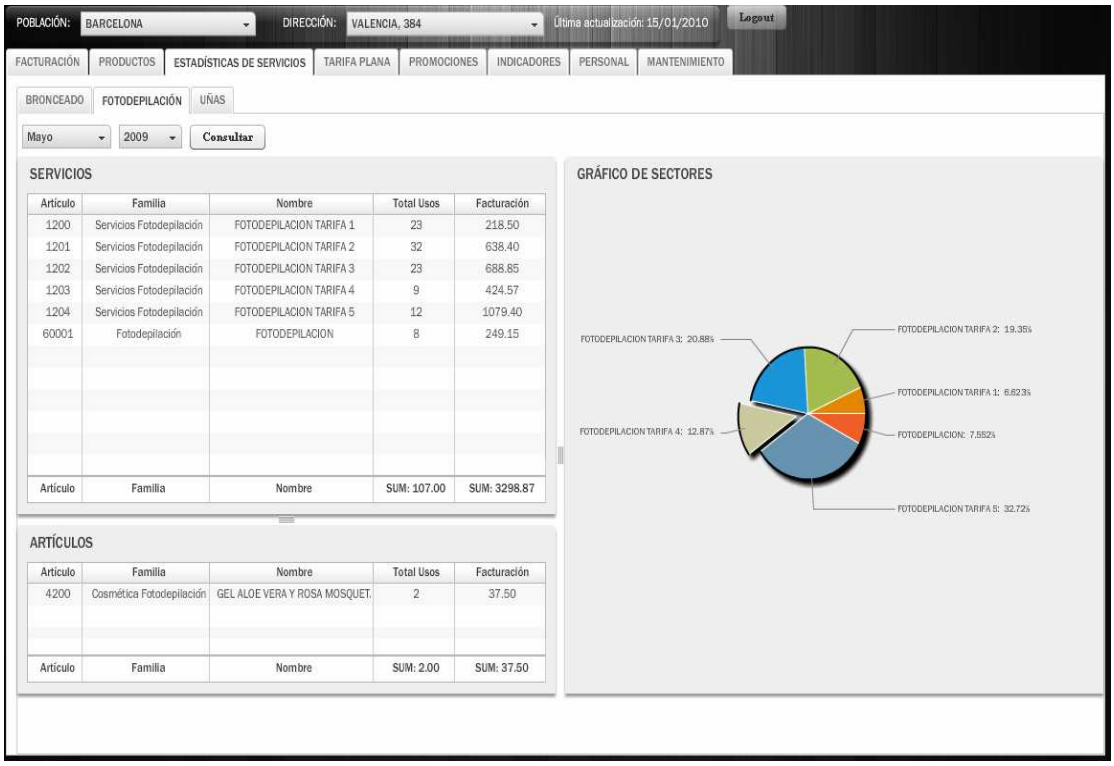


Figura 5 – Pestanya de "Fotodepilación". A la part esquerra podem veure quines són els serveis de fotodepilació que donen més rendiment i a la part dreta el corresponent gràfic d'aquests mateixos serveis per veure quins són els més utilitzats.

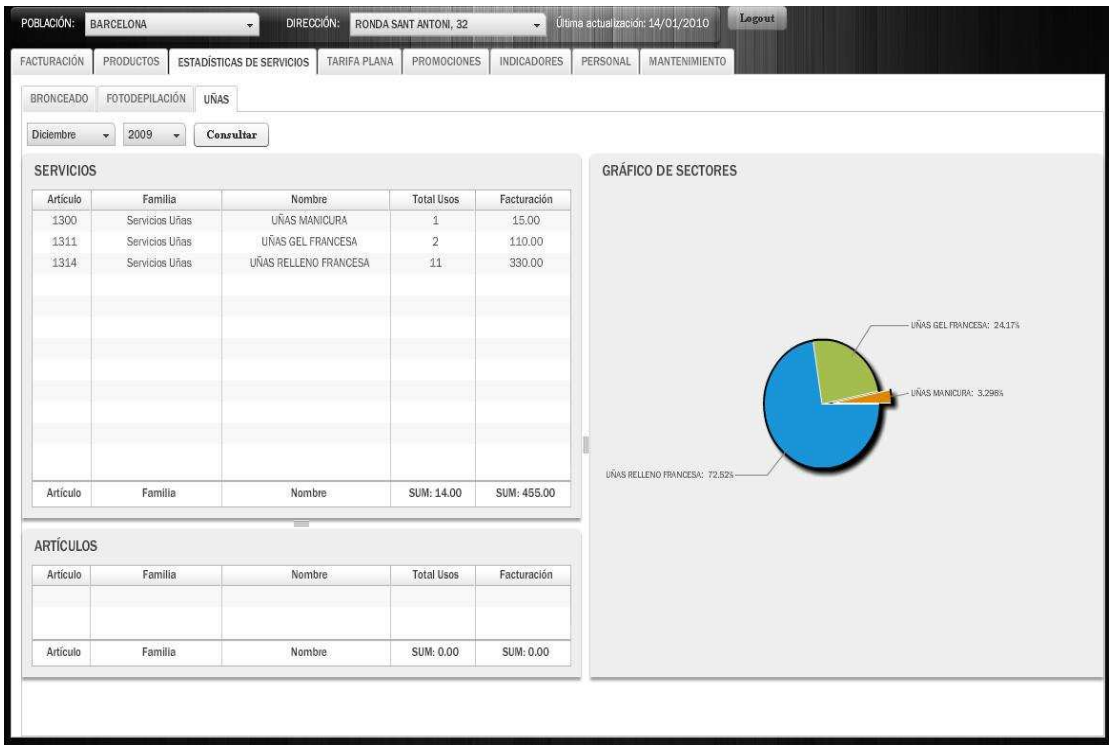


Figura 6 – Pestanya de "Uñas". A la part esquerra podem veure quines són els serveis de ungles que donen més rendiment i a la part dreta el corresponent gràfic d'aquests mateixos serveis per veure quins són els més utilitzats.

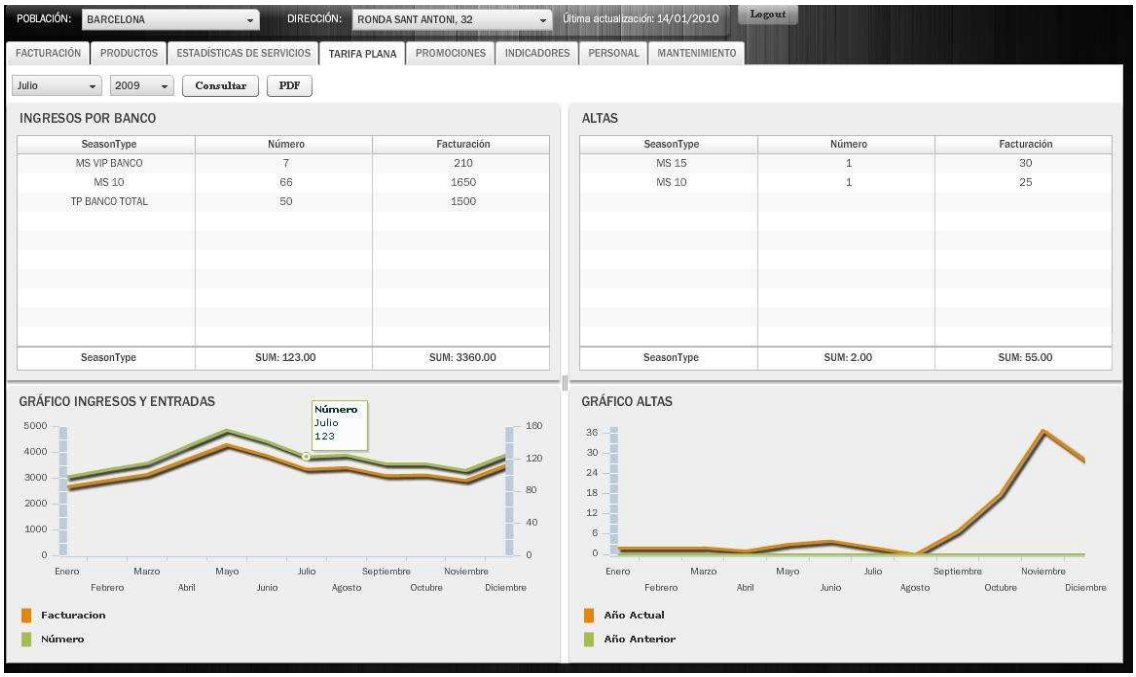


Figura 7 – Pestanya de "Tarifa Plana". A les taules es pot veure quins són els tipus de bons que es venen més i a la part inferior com han anat evolucionant. També es permet exportar informes a PDF.

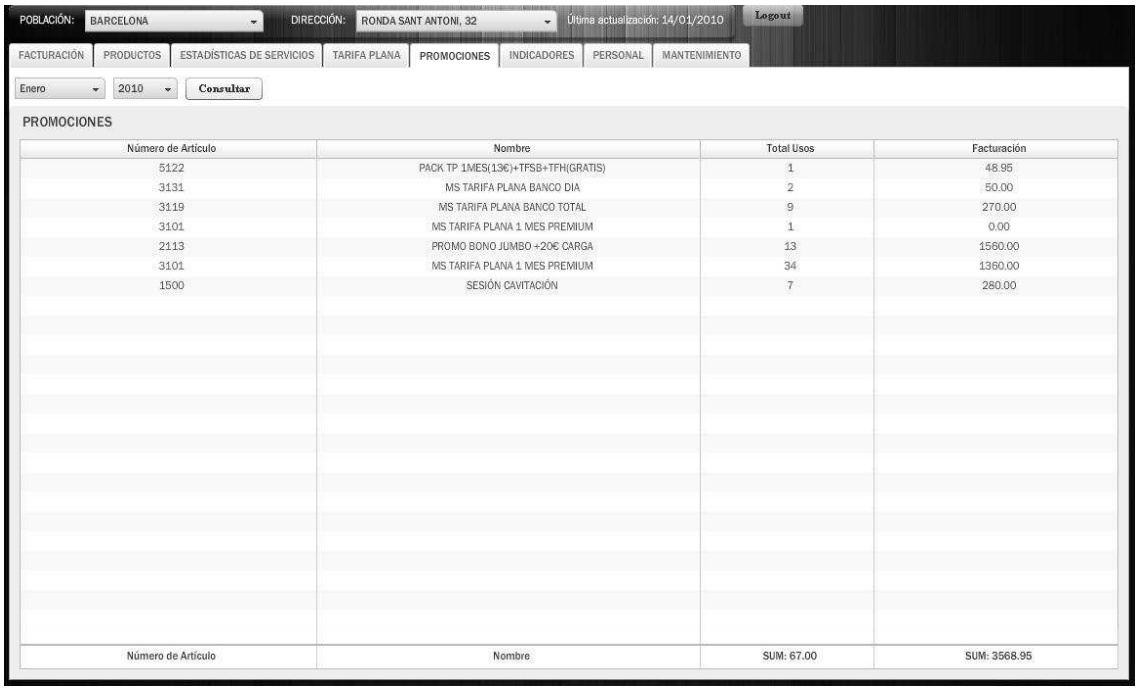


Figura 8 – Pestanya de "Promociones". Un resum de les promocions que s'ofereixen a la franquícia i quants d'aquests s'han facturat en un mes.

Figura 9 – Pestanya de "Indicadores de Servicios". A la part esquerra es presenta una comparació dels serveis mes a mes i a la part esquerra els resultats del mes concret.

Figura 10 – Pestanya de "Indicadores de Clientes". Un resum de com han evolucionat l'entrada dels clients a la franquícia i altres estadístiques d'interès.

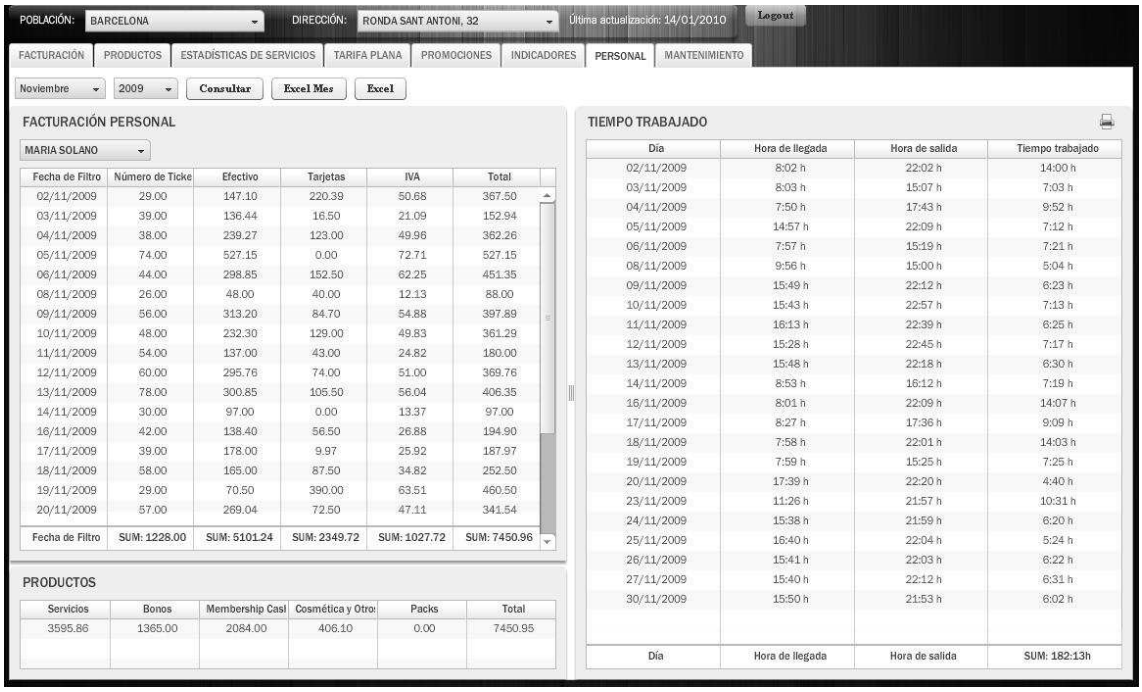


Figura 11 – Pestanya de "Personal". En aquesta pestanya es pot controlar les hores que treballa cada treballador i també exportar informes a Excel.

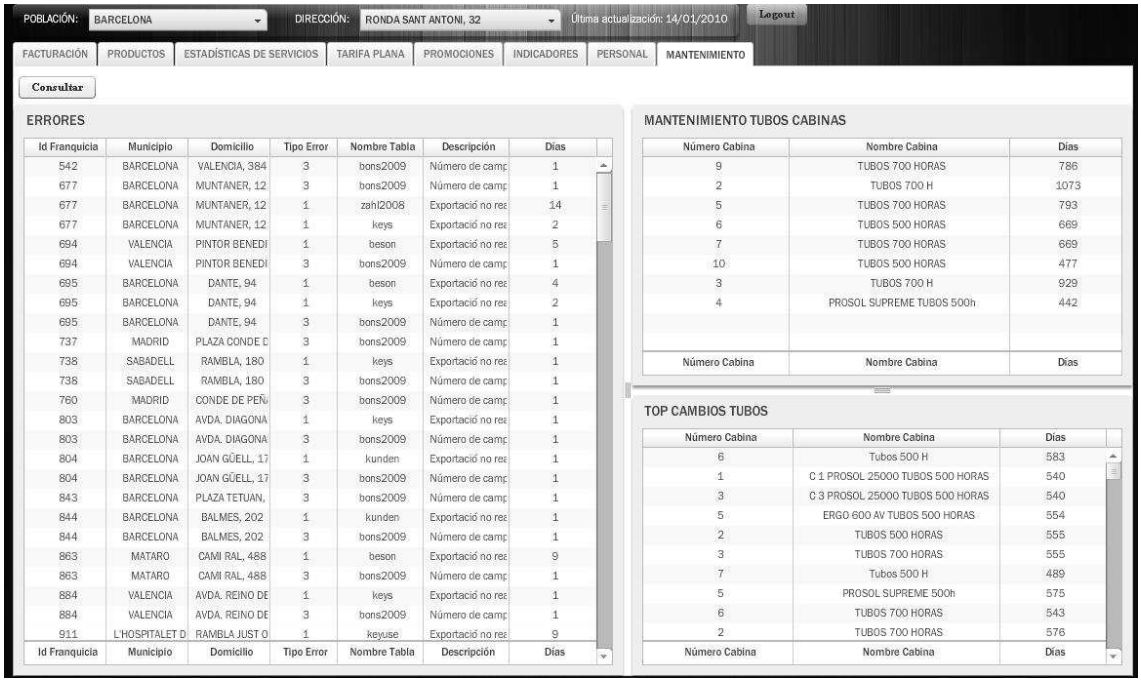


Figura 12 – Pestanya de "Mantenimiento". És un apartat només visible per l'usuari administrador, on es comproven que les importacions de les bases de dades s'han realitzat correctament.

Jordi Masip Balart

22 de gener del 2010

RESUM

En aquest projecte s'han unificat les dades de l'empresa Solmania i s'han generat estadístiques i informes a diferents formats mitjançant una aplicació RIA implementada amb Adobe Flex. D'aquesta manera es pot fer un control dels paràmetres clau del negoci per ajudar a la companyia a aconseguir els objectius marcats d'una manera eficient. L'aplicatiu permet analitzar des de diversos punts de vista qualsevol tipus d'informació que genera el negoci i fer-ne comparatives de rendiment.

RESUMEN

En este proyecto se han unificado los datos de la empresa Solmania y se han generado estadísticas e informes a diferentes formatos mediante la realización de una aplicación RIA implementada con Adobe Flex. De este modo se puede hacer un control de los parámetros clave del negocio para ayudar a la compañía a conseguir los objetivos marcados de un modo eficiente. La aplicación permite analizar desde diferentes puntos de vista cualquier tipo de información que genera el negocia y realizar comparativas de rendimiento.

ABSTRACT

Solmania business information has been treated in this project to generate statistics and different kind of reports by making a RIA application through Adobe Flex. This allows the company to control key parameters of business and achieve objectives in an efficient way. The application gives the power to analyze any type of business information and making performance comparisons from different points of view.