Sistema d'actualització d'aplicacions

Treball fi de carrera

Alumne: Claudi Godia Solanes

Consultor: David Gañan Jiménez

Universitat Oberta de Catalunya

Enginyeria Tècnica en Informàtica de Sistemes

11 de juny de 2010

Resum

En aquest treball de fi de carrera s'ha desenvolupat un sistema d'actualització d'aplicacions que permet de forma fàcil i intuïtiva la descàrrega de noves actualitzacions per a qualsevol tipus de programari. L'aplicació s'ha desenvolupat fent servir les següents tecnologies: la tecnologia .NET de Microsoft per al desenvolupament, ADO.NET per l'accés a la base de dades, WCF (Windows Communicacion Foundation) per desenvolupar el servei i WPF (Windows Presentation Foundation) per al disseny de les interfícies d'usuari.

El sistema s'ha implementat fent ús del llenguatge orientat a objectes C#. L'objectiu del projecte era, d'una banda, posar en pràctica els coneixements adquirits al llarg de la carrera (programació, bases de dades, xarxes, interacció humana amb els ordinadors, estructura de la informació, enginyeria de programari, etc), d'altra banda, l'objectiu era dur a terme gran part del cicle de vida d'una aplicació, es a dir: fer la planificació, l'anàlisi i la implementació d'una aplicació.

En aquest document s'exposa en detall cada etapa del cicle de vida, en primer lloc, s'explica la planificació que s'ha dut a terme i els imprevistos que han sorgit al llarg del procés de desenvolupament, en segon lloc, es detalla l'anàlisi del projecte: casos d'ús, diagrama entitat relació, diagrama de classes etc. En tercer lloc, s'exposa com s'ha dut a terme la implementació del projecte, es a dir, es descriuen les parts més interessants del procés de desenvolupament del sistema, en quart lloc, es mostra el manual d'ús de les diferents aplicacions que composen la solució, en cinquè lloc, es fa un resum del producte final i quines són les parts que el composen, els objectius assolits i l'avaluació de costos, per acabar, es proposen alguns canvis i es donen una sèrie d'idees per millorar el producte en un futur.

Índex

1. Introducció

Un sistema d'actualització es necessari en qualsevol aplicació informàtica, té especial importància en aplicacions empresarials on la seguretat i la estabilitat prenen un protagonisme especial.

Una aplicació empresarial requereix un manteniment per part de l'empresa que la desenvolupa, això implica actualitzacions per afegir noves característiques o millora en l'estabilitat de l'aplicació, aquesta tasca es prolonga durant tot el cicle de vida de l'aplicació.

Un sistema d'actualitzacions, es un sistema client-servidor que permet mantenir actualitzada una aplicació. L'usuari mitjançant una aplicació client es connecta a un servidor per comprovar si existeixen actualitzacions per una aplicació local que es vulgui actualitzar, es cas afirmatiu, es descarreguen tots els fitxers necessaris i es duen a terme totes les accions pertinents per actualitzar l'aplicació.

1.1 Justificació

S'ha triat l'àrea de .NET per al treball de final de carrera per dos motius; d'un costat la tecnologia .NET de Microsoft es moderna, potent i té molta sortida laboral. A nivell tècnic .NET ofereix una sèrie de tecnologies com per exemple LINQ i el potent llenguatge d'etiquetes XAML que d'altres plataformes com J2EE no tenen. A més a més, el llenguatge orientat a objectes C# inclou noves tècniques molt interessants per treballar amb subprocessos (també anomenats *threats*) com ara: els delegats anònims, reflexió, propietats auto implementades on no fa falta declarar les variables d'àmbit privat per guardar el valor de les propietats d'una classe, constructor anònims, etc. Totes aquestes característiques fan de .NET una tecnologia molt atractiva per treballar-hi.

D'un altre costat, s'ha triat el projecte del sistema d'actualitzacions perquè, en primer lloc, es obligat l'ús de les tecnologies WPF per el disseny de les interfícies d'usuari i WCF per la creació del servei, en segon lloc, la naturalesa del projecte obliga a fer una implementació quasi a baix nivell, es a dir, per fer un sistema de descàrregues ben optimitzat en indispensable implementar un control de fluxe de paquets de dades, això implica fer un ús adequat del tipus de dades binàries com ara variables de tipus *byte* i

tenir un bon coneixement del protocol TCP. Desenvolupar aquest projecte es una bona oportunitat per aprofundir en el coneixement de les xarxes de computadors, el protocol TCP i de l'estructura bàsica dels fitxers binaris.

1.2 Context

Un programa en producció necessita un manteniment constant, d'una banda, això indica que el programa es viu i es fa servir, d'altra banda, implica a nivell tècnic, que s'ha de modificar el codi font, generar una nova versió del programa i actualitzar-lo a l'equip del client. No obstant això, la majoria de vegades representa una tasca complexa i feixuga, en altres paraules, cal tenir en compte el nombre d'instàncies que seran afectades per l'actualització, a més a més, s'ha de tenir present que no tots el programes parteixen de la mateixa versió i per acabar, cal considerar el fet de que no tots els usuaris tenen una connexió de banda ampla eficient i sense talls de connexió.

Molt del programari que es pot trobar avui dia no incorpora una sistema d'actualitzacions còmode i eficient per l'usuari, els motius poden ser, a tall d'exemple: falta de previsió en la fase d'anàlisi, disseny arcaic no modular del programari o simplement manca de temps o de recursos. Algunes empreses solucionen el tema publicant les noves versions en format instal·lable en servidors FTP, així, l'usuari el pot descarregar i instal·lar-lo, aquest sistema es pràctic per l'empresa, però, té dos grans inconvenients, per una banda, es difícil saber l'abast de la nova actualització, es a dir, no es pot saber el nombre de còpies en producció que seran actualitzades i d'altra banda, es una tasca feixuga per l'usuari que implica: descarregar els fitxers, desinstal·lar la versió anterior, instal·lar-ne la nova etc.

En conclusió, un bon sistema d'actualitzacions ha de complir dues premisses fonamentals: fàcil i intuïtiu per l'usuari, pràctic i eficient per l'administrador. Per aquest motiu, s'ha vist la necessitat de crear un sistema d'actualitzacions extern, o sigui: que no formi part de cap aplicació, orientat a substituir els fitxers necessaris d'un programa per garantir-ne una correcta actualització, i així, que es pugui adaptar a qualsevol tipus de programari.

1.3 Objectius

L'objectiu principal és fer ús dels coneixements adquirits al llarg de la carrera, així com les tecnologies que proveeix la plataforma .NET de Microsoft per a desenvolupar l'aplicació proposada.

Els objectius secundaris estan basats principalment en conèixer i fer un ús adequat de tecnologies emprades per desenvolupar el projecte, en general, els objectius són els següents:

- Conèixer el llenguatge de programació orientat a objectes C#
- Fer ús del llenguatge de consultes estructurat SQL i de la base de dades Microsoft SQL Server.
- Fer ús de les tecnologies .NET Framework 3.0 com WPF i WCF.

L'objectiu del sistema desenvolupat és permetre l'actualització remota de qualsevol aplicació, per assolir-lo, el projecte ha d'incorporar el següent programari:

- Un servei amb WCF que publiqui les actualitzacions disponibles per un nombre indeterminat d'aplicacions.
- Una aplicació desenvolupada amb la tecnologia .NET i WPF que permeti actualitzar una aplicació a l'última versió disponible.
- Una aplicació desenvolupada amb la tecnologia .NET i WPF que permeti administrar les actualitzacions que, posteriorment, seran publicades pel servei WCF.

1.4 Metodologia

El projecte s'ha dut a terme en cinc etapes, la primera representa la planificació i l'anàlisi de requisits, la segona representa la implementació de les classes que faran servir les demés aplicacions del sistema, la tercera i la quarta és el desenvolupament de l'aplicació servidor i de l'aplicació client respectivament, la cinquena i última és la implementació i testeig del servei. El sistema s'ha dividit en quatre parts o mòduls per dos motius: un és a causa dels requisits, ja que, és imprescindible separar la lògica de les tres aplicacions que formen el projecte, o sigui, aplicació client, servidor i servei, l'altre per facilitar la implementació i futur manteniment del projecte.

- **Planificació i anàlisi de requisits**

  En primer lloc, per la planificació i l'anàlisi de requisits, s'ha dut a terme un estudi informal de les eines de desenvolupament necessàries per implementar el projecte. En segon lloc, s'ha elaborat un anàlisi bàsic del requisits. Finalment, en base a l'informació adquirida s'ha fet una planificació aproximada del dies necessaris per desenvolupar cada etapa.

- **Implementació de la llibreria base**

  S'han creat les entitats bàsiques del model lògic de l'aplicació, com ara les classes accio, actualitzacio, aplicacio, paquetInstalacio, peticio i paquetDades. A més a més, s'implementen les classes fitxer.cs i helperSerializer per comprimir i serialitzar fitxers respectivament.

- **Implementació de l'aplicació servidor**

  Es crea la base de dades, s'implementa la classe d'accés a dades dlGestorServeiAct i gestorServeiAct, es desenvolupa el model lògic de l'aplicació representat per la classe modelAplicació, es fa el disseny de les interfícies d'usuari mitjançant l'aplicació Expression Studio, d'aquest treball sorgeixen els següents fitxers XAML: edicioActualitzacio.xaml, edicioAplicacio.xaml i mainWindow.xaml. Finalment, es creen les classes frontera: controllerEdicioAplicacio i controllerMainWindow.

- **Implementació del servei**

  Es crea la classe d'accés a dades dlGestorPeticioActualitzacio, es desenvolupa la classe ServeiActualitzacio que fa d'enllaç entre la base de dades i el servei WCF. Per acabar, es crea un projecte de testeig per comprovar el correcte funcionament del sistema.

- **Implementació de l'aplicació client**

  Es crea el model lògic de l'aplicació implementant la classe modelAplicació.cs, es crea la porta d'enllaç entre l'aplicació client i el servei WCF a partir de les classes gateWay.cs i clientServeiActualitzacio.cs, es fa el disseny de les interfícies d'usuari, en aquest cas, el procés es més ràpid ja que, s'aprofiten molts del recursos que s'havien fet anteriorment per al disseny de l'aplicació servidor, en concret, es creen el fitxers XAML regionLlistaAct.xaml i mainWindow. Finalment, s'implementa la classe frontera controllerMainWindow.cs.

1.5 Planificació inicial

- Pla de treball del **25/02/2010 al 10/03/2010**
  - Seleccionar el projecte i fer una descripció d'aquest.
  - Anàlisi informal de la problemàtica.
  - Elaboració del document.

- Anàlisi i disseny del **11/03/2010 al 07/04/2010**
  - Familiaritzar-se amb les aplicacions de desenvolupament: Visual Studio 2008, SqlServer 2008, Expression Studio, així com les tecnologies a emprar: LINQ, XAML, WCF i WPF.
  - Estudiar els requisits funcionals/no funcionals de l'aplicació a desenvolupar.
  - Anàlisi exhaustiu del projecte: casos d'ús, diagrama de classes, etc.

- Implementació del **08/04/2010 al 25/05/2010**
  - A partir de l'anàlisi fet en el pas anterior:
    - Implementar el servei.
    - Implementar l'aplicació de gestió del servei.
    - Implementar l'aplicació del client.
  - Realitzar tests de funcionalitat.
  - Fer el manual d'instal·lació i posta en marxa del sistema, tant del client com del servidor.
  - Lliurar el codi font.

- Memòria i presentació virtual del **26/05/2010 al 11/06/2010**
  - Fer la memòria amb un resum del treball realitzat.
  - Fer la presentació en vídeo de l'aplicació.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Pla de treball**	**11 días**	**jue 25/02/10**	**mié 10/03/10**	
2	Selecció del projecte	3 días	jue 25/02/10	dom 28/02/10	
3	Anàlisi informal	4 días	lun 01/03/10	jue 04/03/10	2
4	Elaboració del document	4 días	vie 05/03/10	mié 10/03/10	3
5	**Anàlisi i disseny**	**20 días**	**jue 11/03/10**	**mié 07/04/10**	**4**
6	Instalació i estudi de les aplicacions	2 días	jue 11/03/10	dom 14/03/10	
7	Estudi de requisits	5 días	lun 15/03/10	vie 19/03/10	6
8	Diagrama de casos d´us	3 días	lun 22/03/10	mié 24/03/10	7
9	Disseny BD	2 días	jue 25/03/10	dom 28/03/10	8
10	Diagrama de classes	3 días	lun 29/03/10	mié 31/03/10	9
11	Disseny interfícies gràfiques	2 días	jue 01/04/10	vie 02/04/10	10
12	Altra documentació	3 días	lun 05/04/10	mié 07/04/10	11
13	**Implementació**	**34 días**	**jue 08/04/10**	**mar 25/05/10**	**5**
14	Implementació del servei	7 días	jue 08/04/10	dom 18/04/10	
15	Implementació de l´aplicació de gestió del servei	10 días	lun 19/04/10	dom 02/05/10	14
16	Implementació de l´aplicació client	10 días	lun 03/05/10	dom 16/05/10	15
17	Tests de funcionalitat	4 días	lun 17/05/10	jue 20/05/10	16
18	Manual d´instalació	3 días	vie 21/05/10	mar 25/05/10	17
19	**Memòria i presentació virtual**	**13 días**	**mié 26/05/10**	**vie 11/06/10**	**13**
20	Realització de la memòria	1 día	mié 26/05/10	mié 26/05/10	
21	Realització de la presentació virtual	1 día	vie 11/06/10	vie 11/06/10	

1.6 Planificació real

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Pla de treball**	**11 días**	**jue 25/02/10**	**mié 10/03/10**	
2	Selecció del projecte	3 días	jue 25/02/10	dom 28/02/10	
3	Anàlisi informal	4 días	lun 01/03/10	jue 04/03/10	2
4	Elaboració del document	4 días	vie 05/03/10	mié 10/03/10	3
5	**Anàlisi i disseny**	**20 días**	**jue 11/03/10**	**mié 07/04/10**	**4**
6	Instalació i estudi de les aplicacions	4 días	jue 11/03/10	mar 16/03/10	
7	Estudi de requisits	4 días	mié 17/03/10	lun 22/03/10	6
8	Diagrama de casos d´us	2 días	mar 23/03/10	mié 24/03/10	7
9	Disseny BD	2 días	jue 25/03/10	dom 28/03/10	8
10	Diagrama de classes	3 días	lun 29/03/10	mié 31/03/10	9
11	Disseny interfícies gràfiques	4 días	jue 01/04/10	mar 06/04/10	10
12	Altra documentació	1 día	mié 07/04/10	mié 07/04/10	11
13	**Implementació**	**34 días**	**jue 08/04/10**	**mar 25/05/10**	**5**
14	Implementació del servei	9 días	jue 08/04/10	mar 20/04/10	
15	Implementació de l´aplicació de gestió del servei	12 días	mié 21/04/10	jue 06/05/10	14
16	Implementació de l´aplicació client	9 días	vie 07/05/10	mié 19/05/10	15
17	Tests de funcionalitat	2 días	jue 20/05/10	vie 21/05/10	16
18	Manual d´instalació	2 días	lun 24/05/10	mar 25/05/10	17
19	**Memòria i presentació virtual**	**13 días**	**mié 26/05/10**	**vie 11/06/10**	**13**
20	Realització de la memòria	6 días	mié 26/05/10	mié 02/06/10	
21	Realització de la presentació virtual	7 días	jue 03/06/10	vie 11/06/10	

1.7 Discrepàncies entre la planificació inicial i la real

- Instal·lació i estudi de les aplicacions: s'han necessitat dos dies més del previstos inicialment per l'estudi de l'eina Expression Studio i del llenguatge XAML.

- Estudi de requisits: es va dedicar un dia menys del previst en aquesta fase, però, es va passar per alt un aspecte important que era el control de fluxe de les descàrregues.

- Diagrama de casos d'ús: s'ha necessitat un dia més del previst per fer l'anàlisi del casos d'ús i demés diagrames.

- Disseny d'interfícies gràfiques: s'han dedicat el doble de dies dels previstos. Això es degut a que no es va tenir en compte la corba d'aprenentatge del llenguatge XAML, ja que, tot i no ser un llenguatge molt complex és necessari entendre bé els conceptes de “template”, “controlTemplate”, “dataTemplate” i d'altres per fer-n'hi un ús adequat.

1.8 Productes obtinguts

A continuació, es presenten les diferents parts que formen la solució global. Una part es composa de la documentació que s'ha creat en la fase d'anàlisi de requisits, aquesta engloba els casos d'ús, el diagrama ER, el diagrama de classes de les aplicacions client, servidor, servei de IIS i llibreria base, el diagrama d'activitats i detalls referents a com s'ha dut a terme la implementació del projecte.
L' altra part està formada per tres aplicacions diferents i una llibreria de recursos, el codi font de cadascuna d'elles està estructurat en directoris per facilitar la organització del fitxers que la composen.

- Aplicació servidor:
  - Classes frontera: *controllerEdicioAplicacio.cs, controllerMainWindow.cs, currentPanelToVisibilityConverter.cs*
  - Accés a dades: *dlGestorServeiActualitzacions.cs, gestorServeiActualitzacions.cs*

- Model d'aplicació: *modelAplicacio.cs*
- Recursos: *buttons.xaml, converters.xaml, images.xaml, textBox.xaml*
- Fitxers de configuració: *app.config*
- Fitxers d'interfície: *edicioActualitzacio.xaml, edicioAplicacio.xaml, mainWindow.xaml*

- Aplicació client:
  - Classes frontera: *controllerMainWindow.cs, currentPanelToVisibilityConverter.cs*
  - Accés al servei: *clientServeiActualitzacio.cs, gateway.cs, modelAplicacio.cs*
  - Recursos: *buttons.xaml, converters.xaml, images.xaml, textBox.xaml*
  - Fitxers d'interfice: *regionLlistaLlistaActualitzacions, mainWindow.xaml*
  - Fitxers de configuració: *app.config*

- Servei:
  - Accés a dades: *dlGestorPeticioActualitzacions.cs*
  - Model d'aplicació: *gestorPeticioActualitzacions.cs, IServeiActualitzacio.cs, serveiActualitzacio.cs*
  - Fitxers de configuració: *app.config*

- Llibreria base:
  - Converters: *boolToNotVisibilityConverter.cs, boolToVisibilityConverter.cs, notConveter.cs*
  - Entitats bàsiques: *accio.cs, actualitzacio.cs, aplicacio.cs, paquetInstalacio.cs, peticio.cs, tipusAccio.cs, paquetDades.cs*
  - Utilitzats per l'accés a dades: *cadenaConnexióSQL.cs*
  - Accés a disc i serialització: *fitxer.cs, helperSerializer.cs*

2. Anàlisi de requisits

2.1 Requisits funcionals

El sistema està composat en tres parts ben diferenciades:

1. El servei d'actualitzacions: rep les peticions del client per comprovar si existeixen actualitzacions, en cas afirmatiu, el servei retorna un paquet amb els fitxers necessaris.

    - Donada una aplicació d'una versió en concret té que determinar si hi ha una actualització disponible.
    - Donada una aplicació amb actualització disponible, el servei ha de retornar un paquet amb la informació necessària per a que l'aplicació client pugi dur a terme l'actualització.

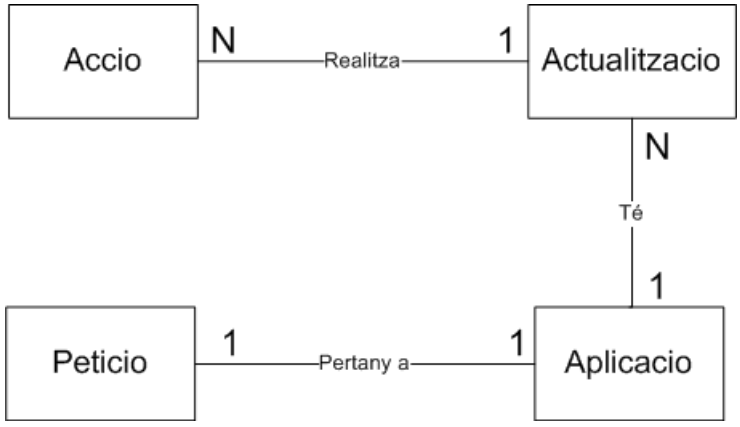
2. La aplicació client: estableix comunicació amb el servei per descarregar les actualitzacions disponibles i executa el procés d'instal·lació quan es rep un paquet d'actualització.

    - Ha d'enviar la petició d'actualització al servei.
    - Ha d'executar el procés d'instal·lació en cas de rebre un paquet d'actualització.

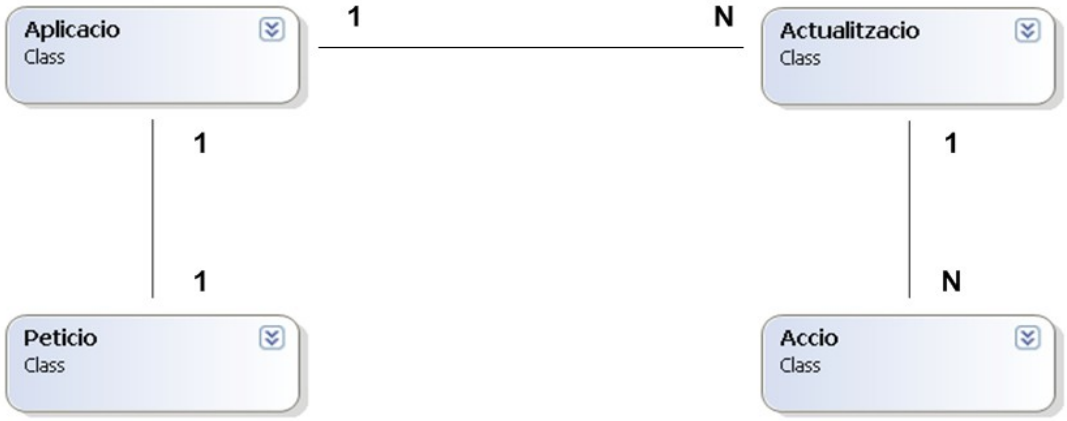
3. La aplicació d'administració del servei: gestiona les actualitzacions que publica el servei.

    - Ha de permetre afegir, modificar i eliminar noves actualitzacions publicades pel servei.

2.2 Diagrama ER



2.3 Diagrama conceptual



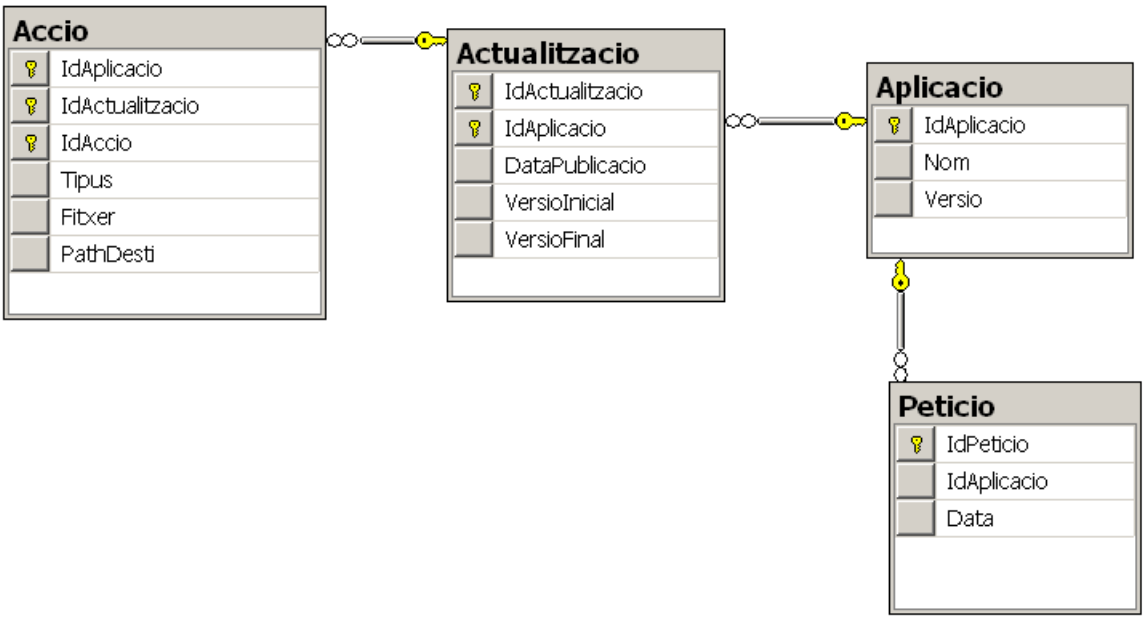
Entitat Aplicació: representa una aplicació actualitzable pel sistema.

Entitat Petició: representa una petició d'actualització feta per l'aplicació client.

Entitat Actualització: representa una actualització d'una aplicació.

Entitat Acció: representa una acció a fer per una actualització.

2.4 Diagrama de la base de dades



Aplicació

Column Name	Data Type
IdAplicacio	bigint
Nom	nvarchar(50)
Versio	nvarchar(10)

Petició

Column Name	Data Type
IdPeticio	bigint
IdAplicacio	bigint
Data	datetime

Actualització

Column Name	Data Type
IdActualitzacio	bigint
IdAplicacio	bigint
DataPublicacio	datetime
VersioInicial	nvarchar(50)
VersioFinal	nvarchar(50)

Acció

Column Name	Data Type
IdAplicacio	bigint
IdActualitzacio	bigint
IdAccio	bigint
Tipus	int
Fitxer	nvarchar(200)
PathDesti	nvarchar(200)

2.5 Requisits del sistema

Per desenvolupar el projecte es necessari disposar del següent programari:

- Planificació del projecte
  - Microsoft Project 2007

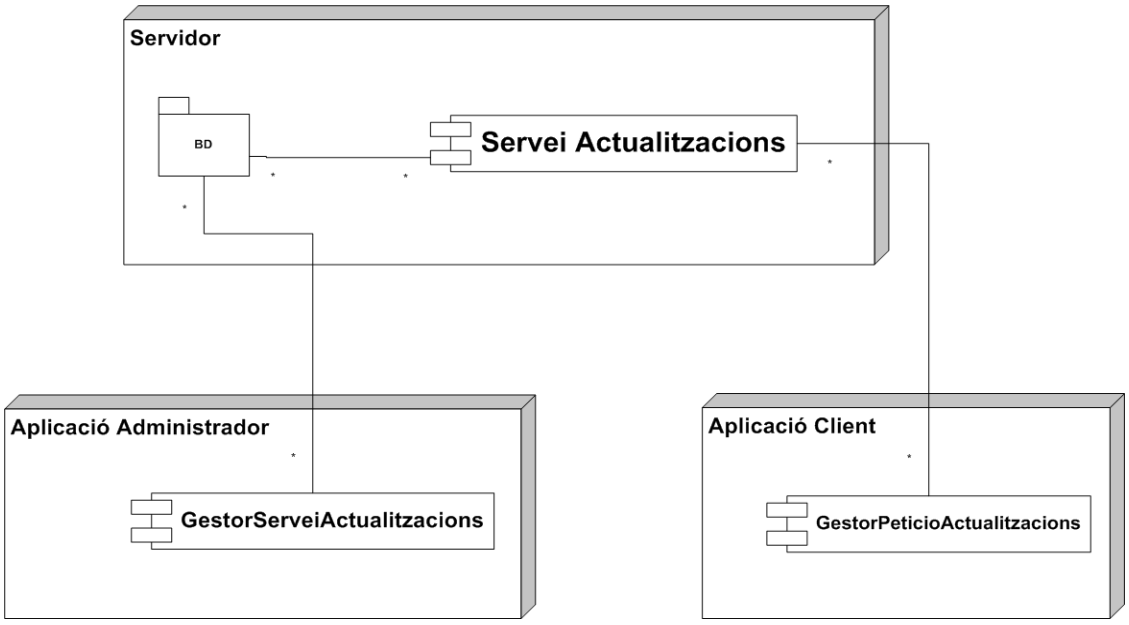
- Creació de la documentació
  - Microsoft Visio 2007
  - Open Office Writer

- Implementació del servei i la lògica de les aplicacions client i servidor:
  - Microsoft Visual Studio 2008 SP1
  - NET Framework 3.5 SP1

- Disseny de la interfície gràfica de l'aplicació client.
  - Microsoft Expression Studio.

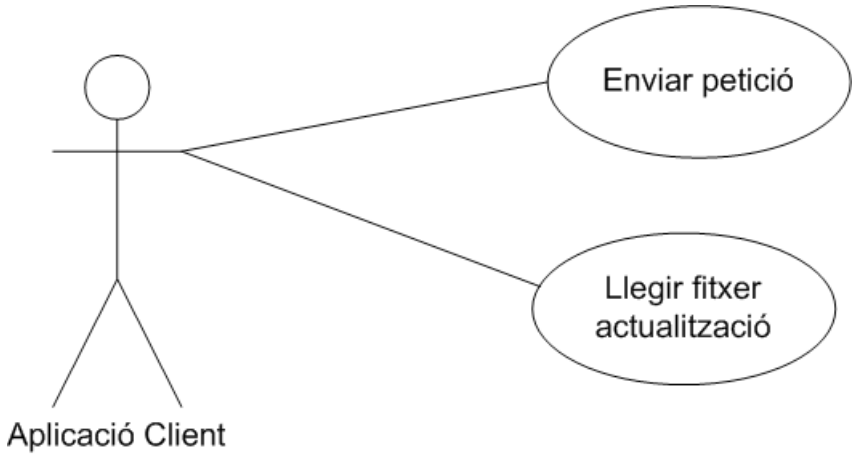
- Disseny de la base de dades.
  - Microsoft SQL Server 2008

2.6 Arquitectura



2.7 Casos d'ús

2.7.1 Cas d'ús “Enviar petició d'actualització”

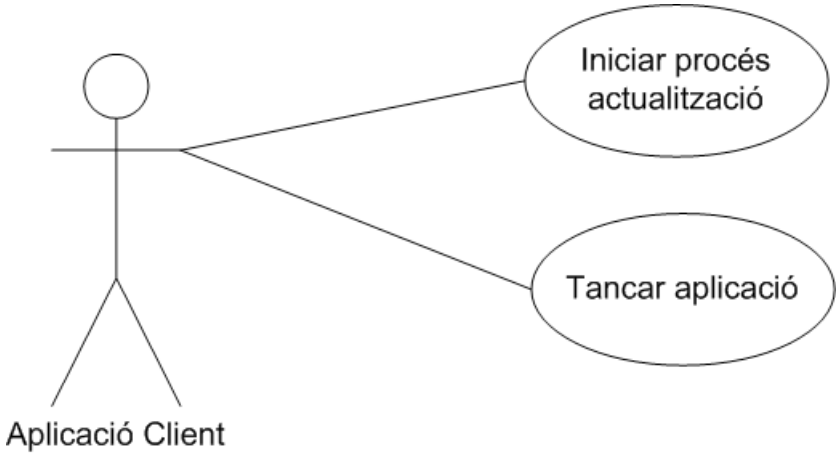


Resum de la funcionalitat: comprova que existeixen noves actualitzacions, en cas afirmatiu, aquestes es descarreguen i s'instal·len.
Paper dins el treball de l'usuari: es el cas d'ús principal de l' aplicació client.
Actors: aplicació client.
Casos d'ús relacionats: atendre peticions actualització.
Precondició: el servei ha d'estar disponible.
Postcondició: s'ha rebut el fitxer resposta del servei.

Descripció:

Quan s'inicia l'aplicació client s'envia un paquet de petició d'actualitzacions, el servei respon enviant un paquet amb les dades sol·licitades, l'aplicació client interpreta el fitxer i si existeixen actualitzacions mostra el contingut per pantalla.

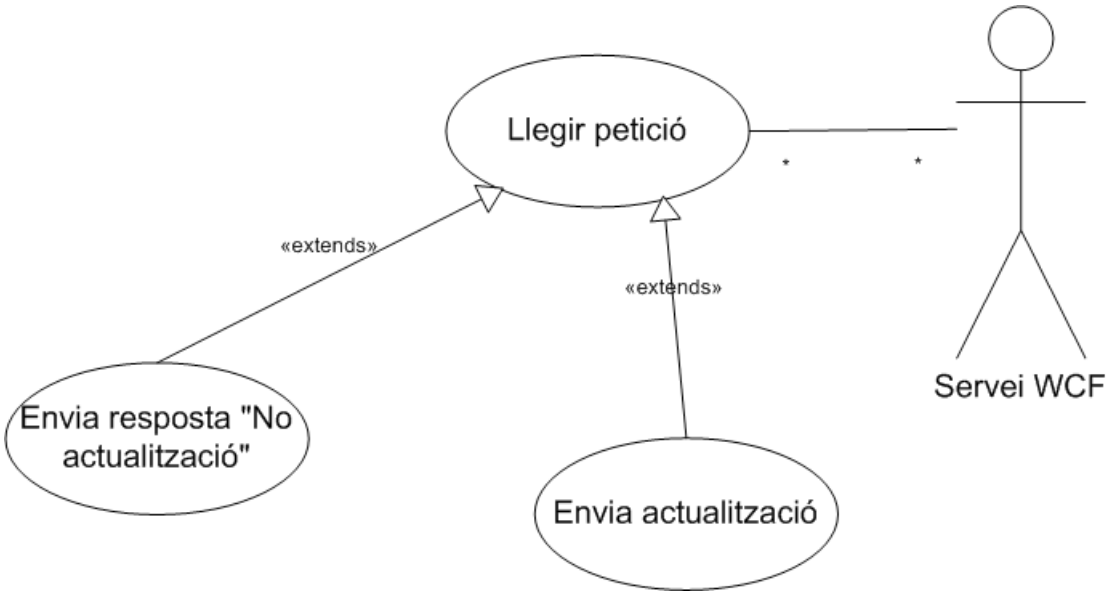
2.7.2 Cas d'ús “Iniciar actualització”



Resum de la funcionalitat: inicia el procés d'actualització.
Paper dins el treball de l'usuari: és el procés principal de l'aplicació client.
Actors: aplicació client.
Casos d'ús relacionats: enviar petició d'actualització.
Precondició: hi ha una actualització disponible.
Postcondició: la actualització s'ha descarregat del servidor i s'ha instal·lat.

Descripció:
Un cop s'ha rebut la resposta del servei, poden succeir dos casos diferents, d'un costat el paquet de resposta informa de que existeixen *n* actualitzacions disponibles, aleshores l'usuari pot iniciar el procés d'instal·lació. D'un altre costat, pot passar que no hi hagin actualitzacions disponibles, conseqüentment, s'informa a l'usuari i es tanca l'aplicació.

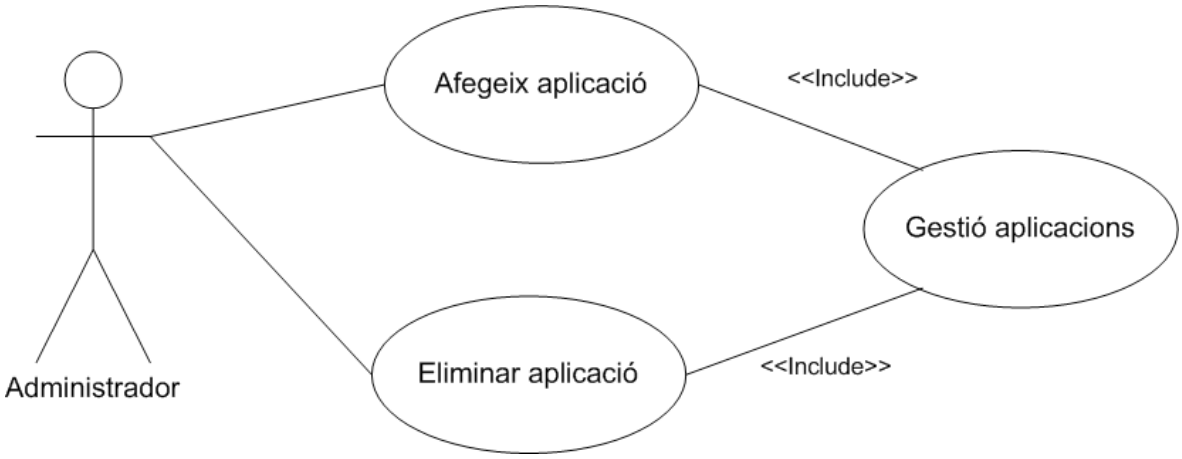
2.7.3 Cas d'ús “Atendre peticions d'actualització”



Resum de la funcionalitat: el servei rep les peticions d'actualització de les aplicacions clients del usuaris.
Paper dins el treball de l'usuari: es el cas d'us principal del servei d'actualitzacions.
Actors: el servei WCF.
Casos d'ús relacionats: enviar petició d'actualització.
Precondició: s'ha rebut una petició d'actualització.
Postcondició: s'ha enviat un paquet d'actualització a l'aplicació client.

Descripció:
Es llegeix la petició d'actualització d'una aplicació client, es comprova l'identificador i la versió de l'aplicació que fa la sol·licitud, tot seguit, es busca a la base de dades les actualitzacions existents en funció de la versió sol·licitada, en cas de que n´hi hagin s'envia un paquet amb totes les dades necessàries per fer l'actualització, en cas constrari, s'envia un paquet resposta on es notifica que no hi han paquets d'actualització disponibles.

2.7.4 Cas d'ús “Gestionar aplicacions”

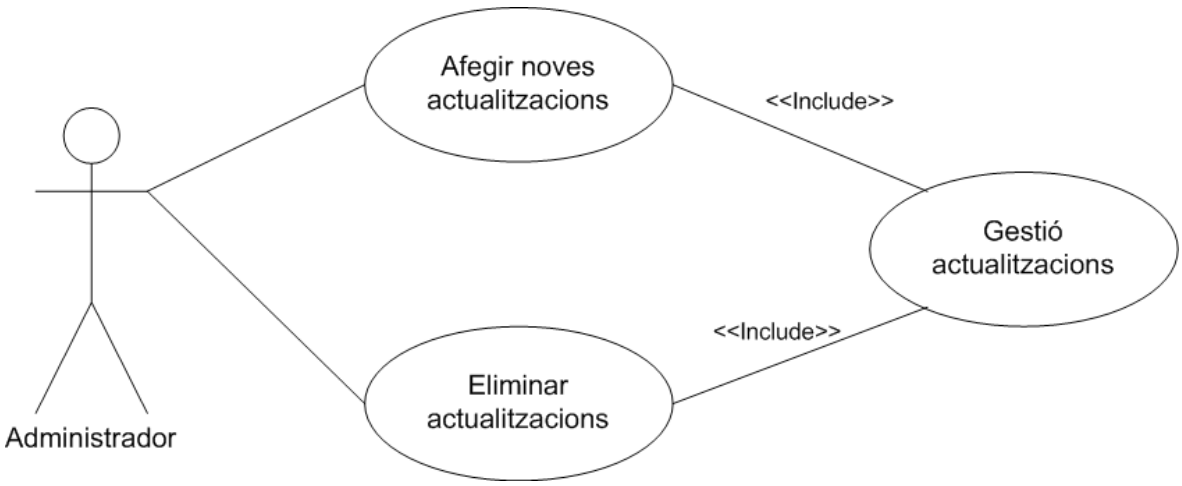


Resum de la funcionalitat: l'administrador crea noves aplicacions a la base de dades per poder publicar les actualitzacions.
Paper dins el treball de l'usuari: es el cas d'ús necessari per mantenir el servei.
Actors: l'administrador del servei.
Precondició: no hi ha cap aplicació a la base de dades per poder publicar actualitzacions.
Postcondició: s'ha creat una nova aplicació a la base de dades.

Descripció:

L'actor en aquest cas l'administrador gestiona les aplicacions de la base de dades, pot donar d'alta una nova aplicació introduint una breu descripció i una versió inicial, o bé, pot eliminar una aplicació existent, aquesta acció, a més a més, eliminarà totes les actualitzacions que depenen d'aquest registre.

2.7.5 Cas d'ús “Gestionar actualitzacions”

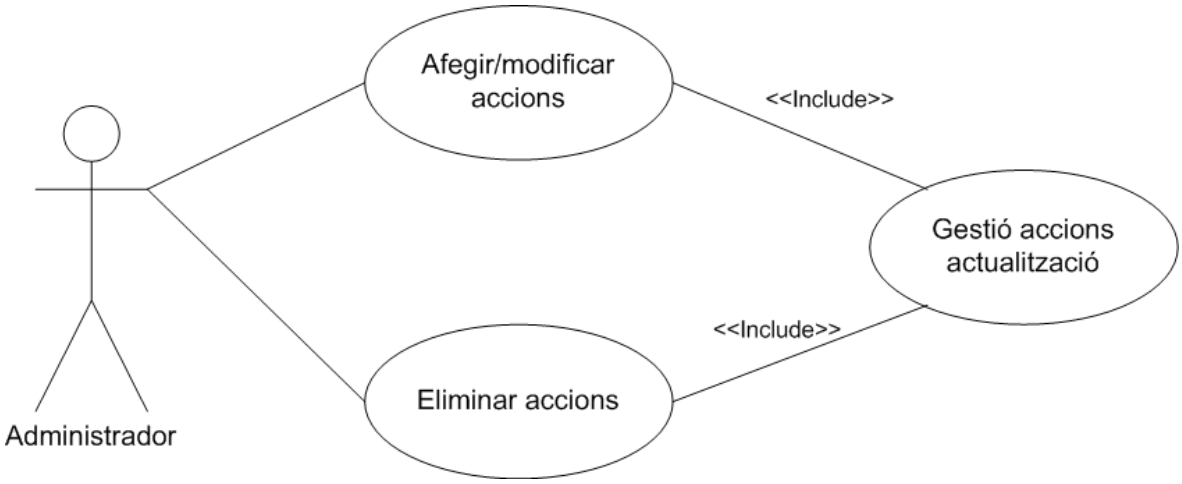


Resum de la funcionalitat: l'administrador publica noves actualitzacions per a que les aplicacions client puguin descarregar-les.
Paper dins el treball de l´usuari: es el cas d'ús necessari per mantenir el servei.
Actors: l'administrador del servei.
Precondició: no hi ha cap actualització publicada per una aplicació en concret.
Postcondició: s'ha publicat una nova actualització.

Descripció:

L'administrador gestiona les actualitzacions de la base de dades, d'un costat, pot donar d'alta una nova actualització introduint la data de publicació, la versió inicial i la versió final, això és la versió mínima per a que es pugui actualitzar l'aplicació i la versió que tindrà l'aplicació un cop s'ha dut a terme tot el procés. D'un altre costat, pot eliminar una actualització existent, a més a més, aquest procés eliminarà totes les accions (fitxers que s'han d'afegir o modificar) que depenen d'aquest registre.

2.7.6 Cas d'ús “Gestionar accions actualització”

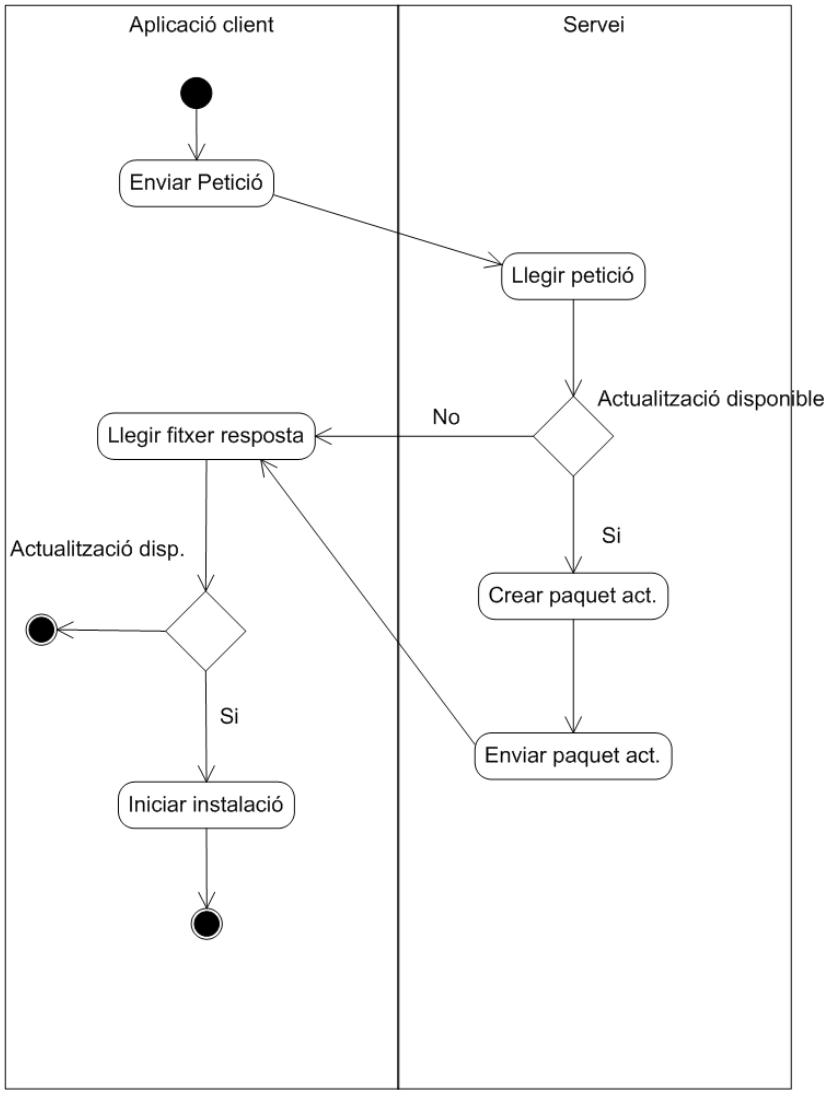


Resum de la funcionalitat: l'administrador afegeix o elimina noves accions a una actualització.
Paper dins el treball de l´usuari: es el cas d'ús necessari per mantenir el servei.
Actors: l'administrador del servei.
Precondició: no hi ha cap acció assignada a una actualització.
Postcondició: s'ha assignat un acció a una actualització.

Descripció:

L'administrador gestiona les accions que ha de dur a terme una actualització. D'un costat, pot afegir una acció seleccionant una actualització existent, a continuació, crea l'acció, introdueix el fitxer que s'ha d'incloure dins el paquet, el nom que ha de tenir quan s'instal·li en la màquina client i el tipus d'acció a realitzar per part de l'instal·lador. D'un altre costat, l'usuari pot eliminar una acció existent quan aquesta ja no és necessària per l'actualització.

2.8 Diagrama d'activitats

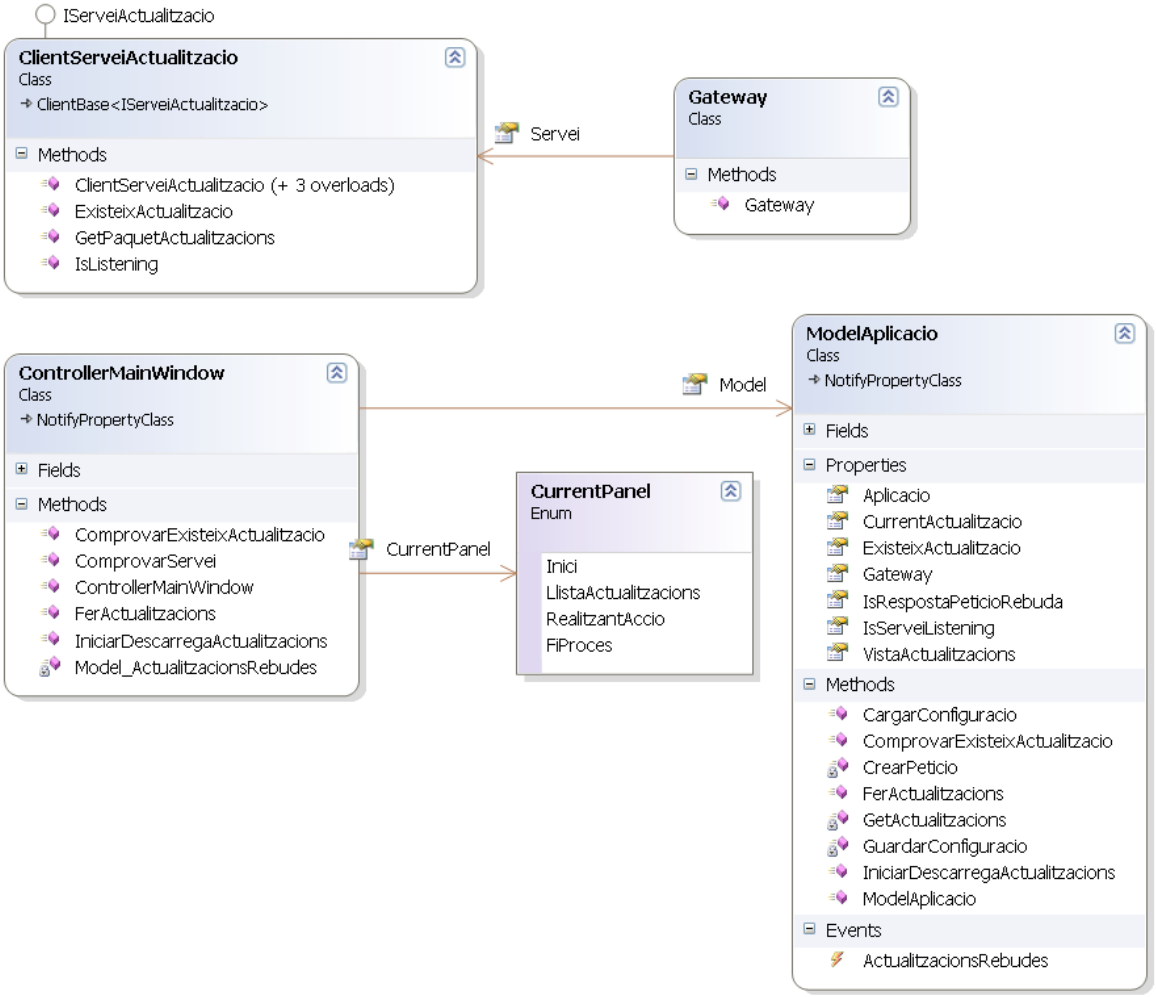


Descripció:

L'aplicació client envia la petició d'actualització, el servei la rep, comprova l'identificador i la versió de l'aplicació, amb aquestes dades es busca la base de dades si existeixen actualitzacions disponibles, en cas afirmatiu, es crea el paquet d'actualitzacions i s'envia a l'aplicació client, en cas contrari, es notifica al client de que no existeixen actualitzacions disponibles. Quan l'aplicació client rep el paquet resposta que ha enviat el servei, si el fitxer comté actualitzacions les mostra a l'usuari i s'inicia el procés d'instal·lació, en cas contrari, es notifica a l'usuari de que no existeixen actualitzacions disponibles.

2.9 Diagrames de classe

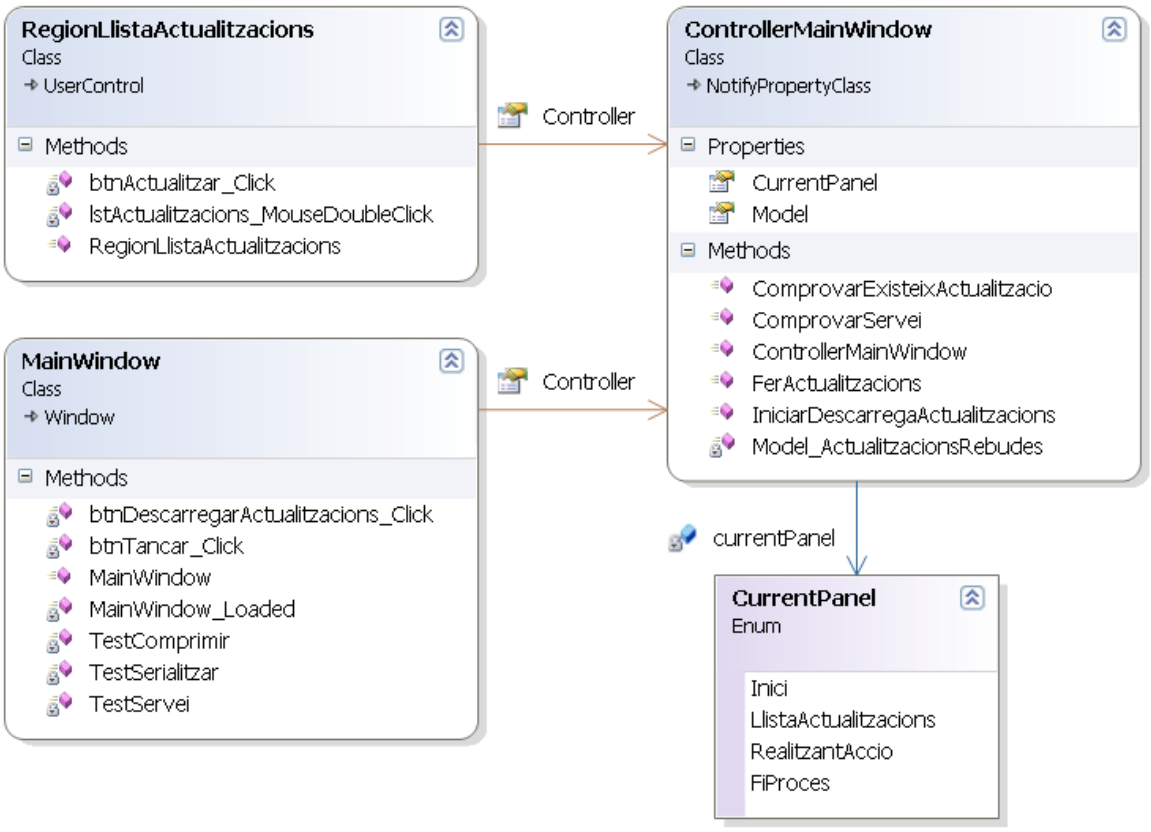
2.9.1 Model lògic de l'aplicació client



Aquest diagrama representa el model lògic de l'aplicació client, a continuació, es dona una breu descripció de cada classe:

- Gateway: és la porta d'enllaç entre l'aplicació client i les funcions que publica el servei.
- ClientServeiActualització: està associada com a propietat de la classe Gateway i dóna accés les funcions que publica el servei, ja que, implementa la interfície IServeiActualització.
- ControllerMainWindow: és la classe frontera entre la interfície d'usuari i el model lògic, té associada com a propietat la classe ModelAplicacio per a que el model de presentació pugui mostrar les dades d'interès a l'usuari.

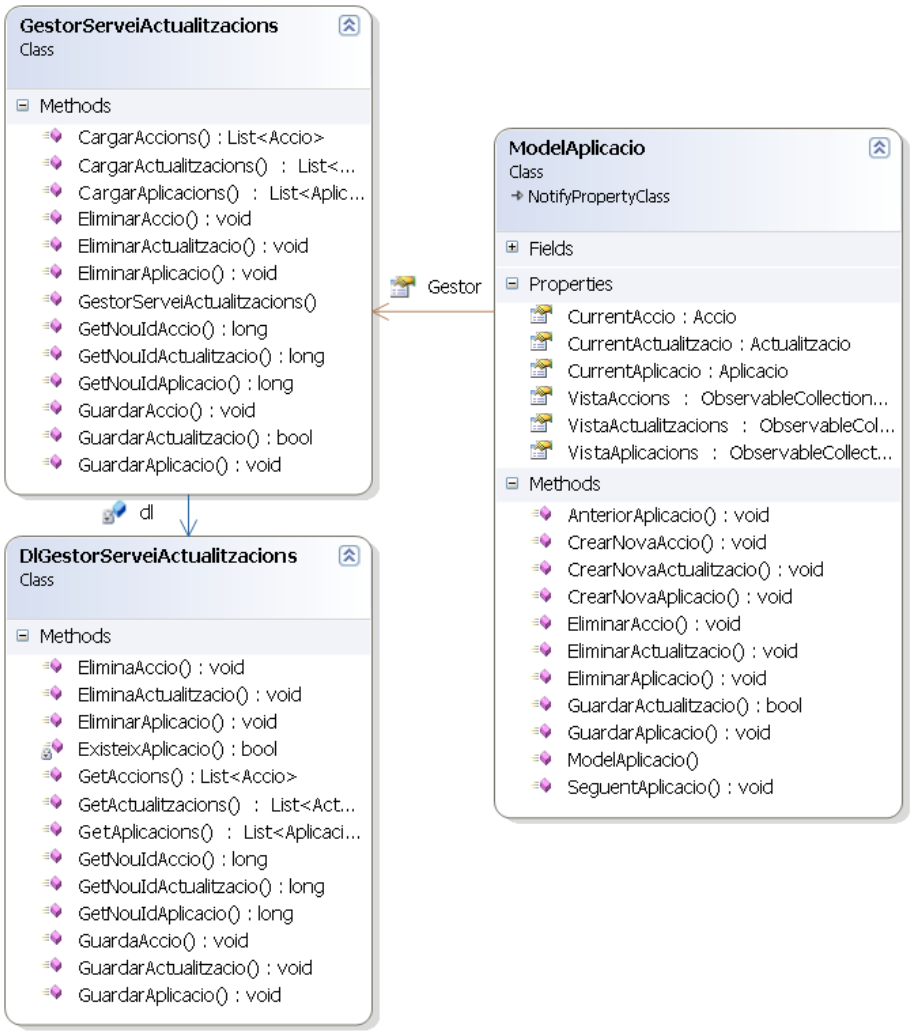
2.9.2 Model de presentació de l'aplicació client



Aquest diagrama representa el model de presentació de l'aplicació client, a continuació, es dona una breu descripció de cada classe:

- MainWindow: representa la finestra principal de l'aplicació, declara mitjançant codi XAML la classe regionLlistaActualitzacions.
- RegionLlistaActualitzacions: classe de tipus XAML UserControl que mostra el contingut del paquet de dades rebut amb un *ListBox* amb les actualitzacions disponibles.
- ControllerMainWindow: classe frontera que gestiona els objectes visuals de la finestra MainWindow i RegionLlistaActualitzacions, implementa la propietat CurrentPanel que determina quina finestra és visible actualment.

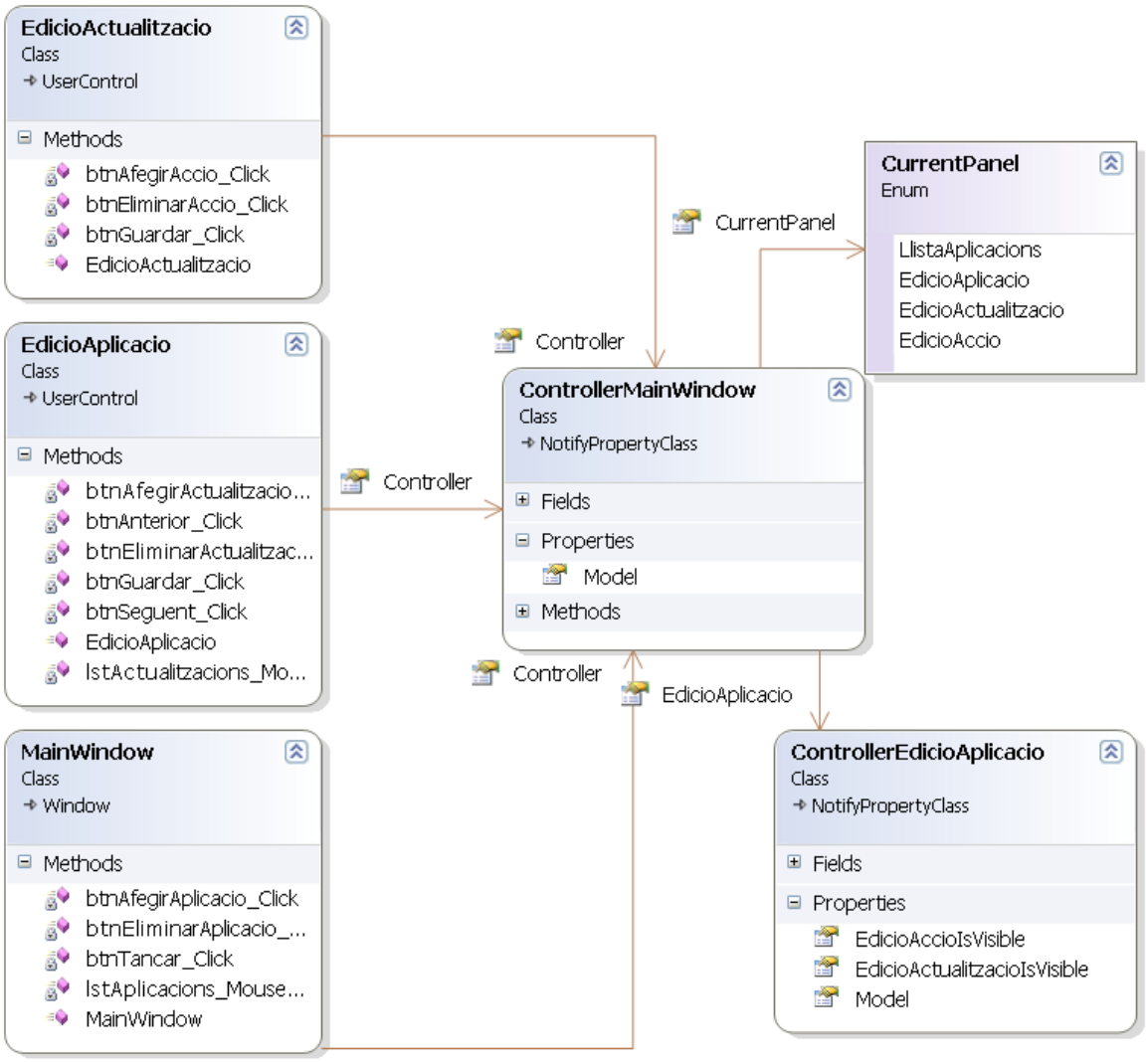
2.9.3 Model lògic de l'aplicació servidor



Aquest diagrama representa el model lògic de l'aplicació servidor, tot seguit, es dóna una breu descripció de cada classe:

- DlGestorServeiActualitzacions: implementa totes les funcions que necessiten accedir a la base de dades, a més a més, es l'única classe que hi té accés. Cadascun del mètodes implementats realitza una operació atòmica sobre la base de dades, o sigui, s'obre una nova connexió al SGBD, es realitza una operació en concret i es tanca la connexió alliberant tots el recursos utilitzats en el procés.
- GestorServeiActualitzacions: classe frontera entre les classes DlGestorServei i ModelAplicacio.
- ModelAplicació: representa el model lògic de l'aplicació, guarda en llistes indexades tota la informació referent a les aplicacions i actualitzacions.

2.9.4 Model de presentació de l'aplicació servidor

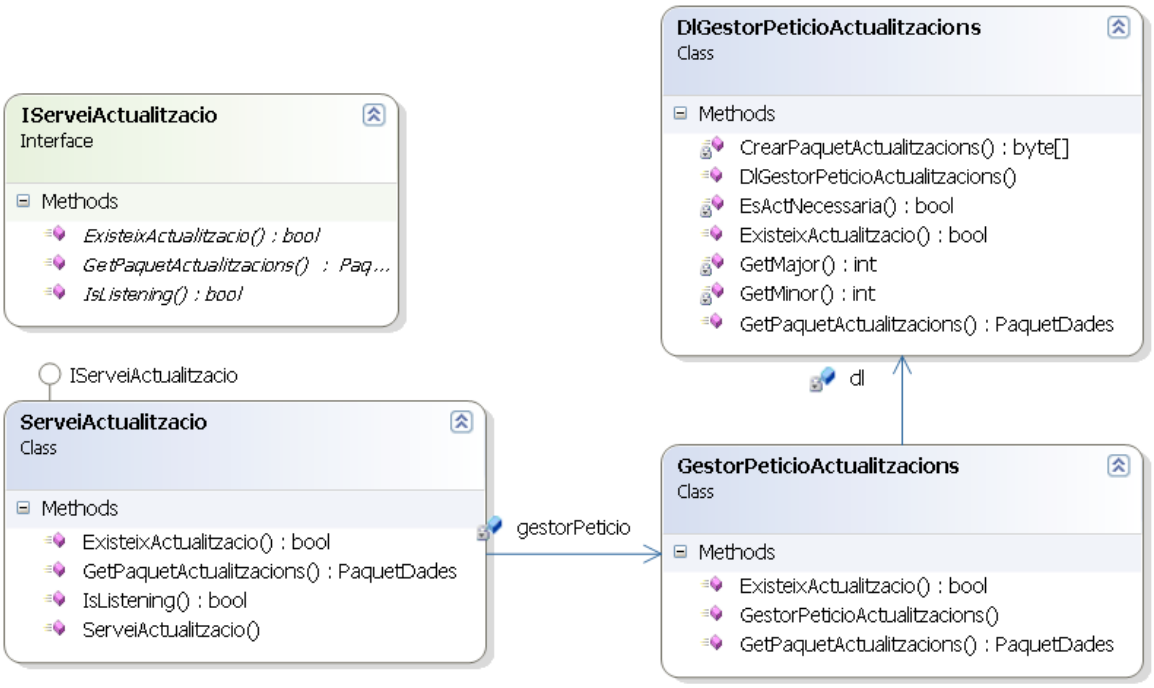


Aquest diagrama representa el model de presentació de l'aplicació servidor, a continuació, es dóna una breu descripció de cada classe:

- MainWindow: és la finestra principal de l'aplicació, declara mitjançant codi XAML les classess EdicioActualitzacio i EdicioAplicacio.
- EdicioAplicacio: classe de tipus XAML UserControl que representa la finestra d'edició de les aplicacions.
- EdicioActualitzacions: classe de tipus XAML UserControl que representa la finestra d'edició de les actualitzacions.
- ControllerMainWindow: classe frontera que comparteixen les classes d'interfície d'usuari, dóna accés al model lògic.

- ControllerEdicioAplicacio: classe frontera que gestiona la finestra EdicioAplicacio, mostra o amaga determinats objectes en funció de les accions de l'usuari.
- CurrentPanel: enumerat que determina quina és la finestra visible.

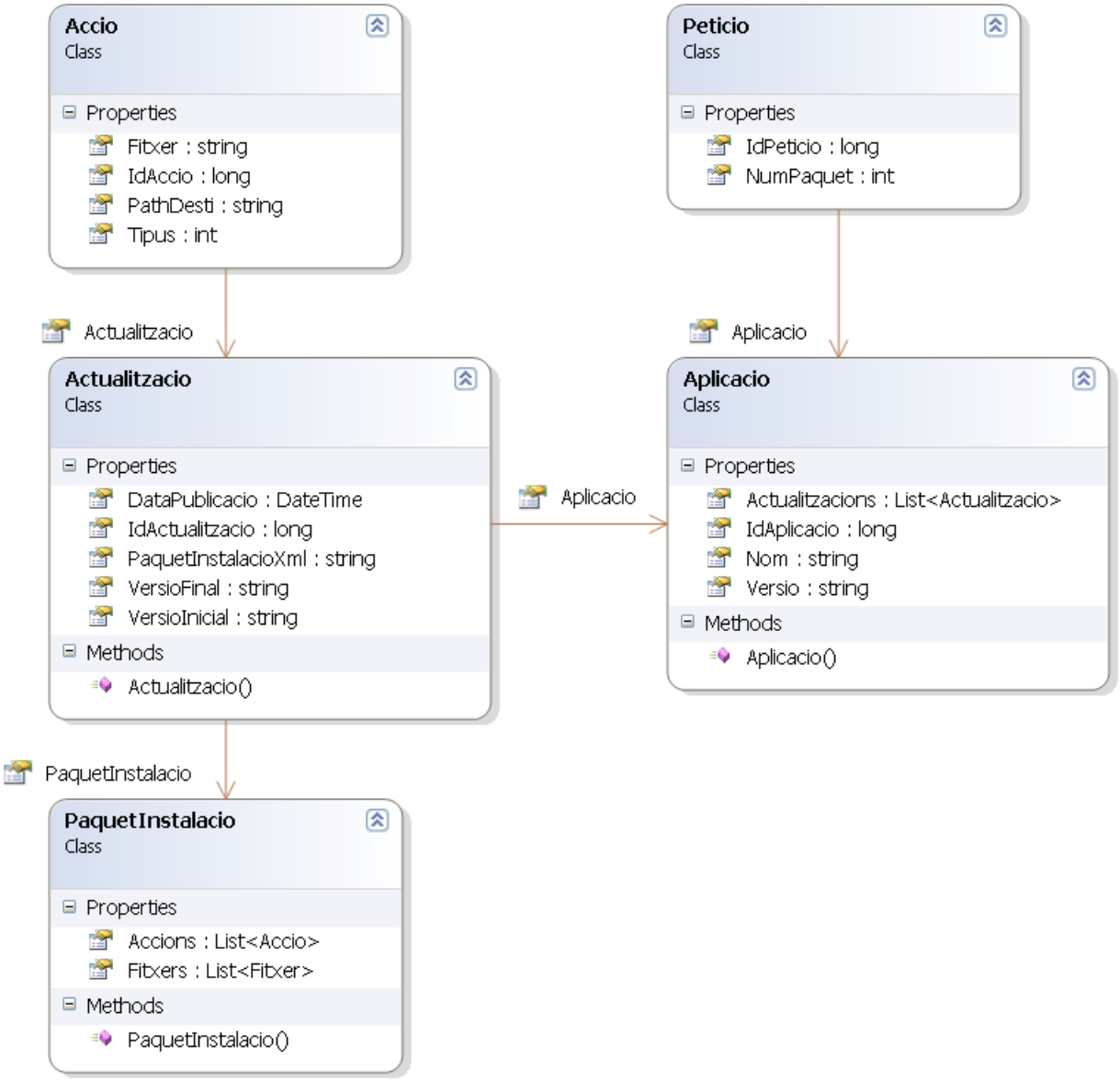
2.9.5 Model lògic del servei WCF



Aquest diagrama representa el model lògic del servei WCF, tot seguit, es dona una breu descripció de cada classe:

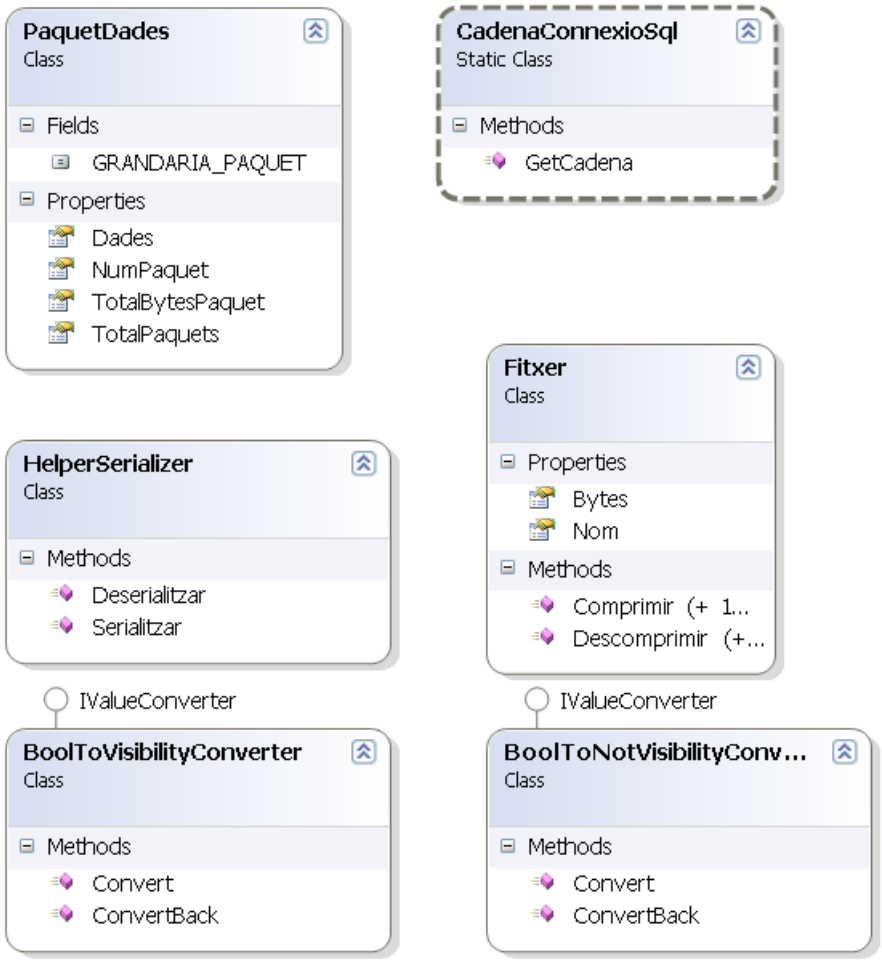
- DlGestorPeticioActualitzacions: implementa les funcions d'accés a la base de dades.
- GestorPeticioActualitzacions: classe frontera entre la classe serveiActualitzacio i l'accés a dades.
- ServeiActualitzacio: representa la classe que implementa les funcions bàsiques del servei, instancia l'objecte GestorPeticioActualitzacions per donar servei a les peticions de l'aplicació client.

2.9.6 Entitats de la llibreria base



- Accio: representa una acció a realitzar per una actualització.
- Peticio: representa una petició d'actualització feta per l'aplicació client.
- Actualitzacio: representa una actualització d'una aplicació.
- Aplicacio: representa una aplicació donada d'alta al sistema d'actualitzacions.
- PaquetInstalacio: comté els fitxers i instruccions necessàries per dur a terme una actualització.

2.9.7 Classes útils de la llibreria base



- PaquetDades: representa una paquet de dades d'una descàrrega d'actualitzacions, o sigui, durant la tramesa la informació es divideix en paquets d'una grandària determinada per poder optimitzar correctament la transacció.
- HelperSerializer: classe que implementa funcions per facilitar la serialització i la deserialització de fitxers.
- Fitxer: representa un fitxer en format binari, a més a més, implementa funcions estàtiques per comprimir i descomprimir fitxers.
- CadenaConexionSql: classe estàtica que facilita la creació de la cadena de connexió necessària per carregar i guardar informació de la base de dades.

3. Implementació

Una bona arquitectura en la fase de desenvolupament es fonamental per el correcte funcionament de l'aplicació, a més, facilita un futur manteniment. Es necessari separar en classes ben definides la lògica de negoci i la lògica de presentació, es a dir, les classes que gestionen la informació, que accedeixen a la base de dades, que controlen el sistema mai han de tenir accés directe als formularis de la interfície d'usuari. A continuació, es detalla l'arquitectura d'implementació que s'ha seguit per desenvolupar el model lògic del projecte.

3.1 Nivells

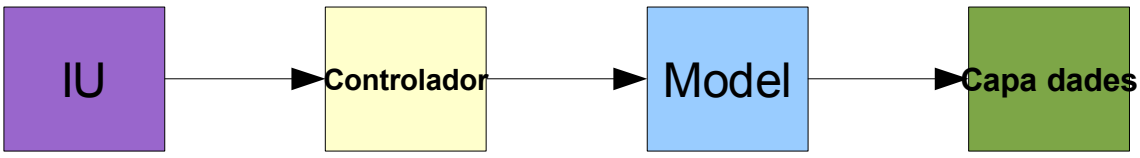
La implementació s'ha dividit en 4 nivells diferents:

- **Regió**: representa la interfície d'usuari en XAML, la informació es mostra mitjançant la tecnologia *Data Binding*. Cada regió té com a *DataContext* un controlador.

- **Controlador**: és la classe d'enllaç entre la regió i el model de l'aplicació, un controlador implementa propietats i funcions que permeten controlar d'interfície d'usuari, a més a més, dóna accés a la classe model.

- **Model**: implementa la lògica de l'aplicació, només un controlador té accés a un model.

- **Capa d'accés a dades:** és la classe que accedeix a la base de dades, només una classe model pot utilitzar-la.



3.2 Desenvolupament

El desenvolupament del projecte s'ha dut a terme en quatre fases, a continuació, es fa un resum de la implementació de cadascuna d'elles i es destaquen les parts més importants.

- **Creació de la llibreria de classes bàsiques**: es creen les entitats bàsiques peticio, accio, actualitzacio, aplicacio i paquetInstalacio que faran servir la resta d'aplicacions, cal destacar la importància de la classe HelperSerializer que s'utilitza per serialitzar i deserialitzar el paquets d'instal·lació, tot seguit, es mostra el codi font de la funció Serialitzar().

```
/// <summary>
/// Serialitza un objecte en xml.
///
/// Retorna una string que comté el document xml en format UTF8.
/// </summary>
/// <param name="obj">Objecte a serialitzar.</param>
/// <param name="fullPath">Path on es desarà el fitxer.</param>
public string Serialitzar(object obj)
{
    XmlSerializer ser = new XmlSerializer(obj.GetType());
    MemoryStream memStream = new MemoryStream();
    ser.Serialize(memStream, obj);

    byte[] bytes = new byte[memStream.Length];
    memStream.Seek(0, SeekOrigin.Begin);
    memStream.Read(bytes, 0, bytes.Length);

    memStream.Close();
    memStream.Dispose();

    return Encoding.UTF8.GetString(bytes);
}
```

- **Creació de l'aplicació servidor**: s'implementa la classe d'accés a la capa de dades dlGestorServeiActualitzacions, en totes les funcions es fa ús de la tecnologia ADO.NET per accedir a la base de dades, es crea la classe ModelAplicacio on part de la funcionalitat es basa en fer crides a les funcions públiques de dlGestorServeiActualitzacions, s'afegeixen els objectes XAML d'interfície d'usuari, es creen els diccionaris de recursos per integrar les imatges i les plantilles de disseny, per acabar, es creen les classes frontera per enllaçar la lògica de presentació amb la lògica d'aplicació. A continuació, es mostra el codi font de la funció GuardarAplicacio() de la classe dlGestorServeiActualitzacions on es pot apreciar l'us d'objectes ADO.NET i de consultes SQL.

```
public void GuardarAplicacio(Aplicacio aplicacio)
{
    SqlConnection conn = new SqlConnection(CadenaConnexioSql.GetCadena());
    using (conn)
    {
        conn.Open();

        //Si l'aplicació existeix es modifica sino es crea.
        SqlCommand command = conn.CreateCommand();
        command.CommandText = "SELECT COUNT(*) FROM Aplicacio WHERE IdAplicacio=@IdAplicacio";
        command.Parameters.Add(new SqlParameter("IdAplicacio", aplicacio.IdAplicacio));
        int result = (int)command.ExecuteScalar();
        if (result > 0)
        {
            command = conn.CreateCommand();
            StringBuilder sb = new StringBuilder();
            sb.AppendLine("UPDATE Aplicacio ");
            sb.AppendLine("SET Nom=@Nom, Versio=@Versio ");
            sb.AppendLine("WHERE IdAplicacio=@IdAplicacio ");
            command.CommandText = sb.ToString();
            command.Parameters.Add(new SqlParameter("Nom", aplicacio.Nom));
            command.Parameters.Add(new SqlParameter("Versio", aplicacio.Versio));
            command.Parameters.Add(new SqlParameter("IdAplicacio", aplicacio.IdAplicacio));
            command.ExecuteNonQuery();
        }
        else
        {
            StringBuilder sb = new StringBuilder();
            sb.AppendLine("INSERT INTO Aplicacio(IdAplicacio, Nom, Versio) VALUES (");
            sb.Append(aplicacio.IdAplicacio + ",'");
            sb.Append(aplicacio.Nom + "','");
            sb.Append(aplicacio.Versio + "')");

            SqlCommand command2 = conn.CreateCommand();
            command2.CommandText = sb.ToString();
            command2.ExecuteNonQuery();
            command2.Dispose();
        }
        conn.Close();
    }
}
```

- **Creació del servei:** es crea la classe ServeiActualització que publica les funcions que donen accés al servei i s'implementa la classe d'accés a dades dlGestorPeticioActualitzacions, cal destacar la implementació del sistema gestor de descàrregues que s'encarrega d'optimitzar el tràfic de paquets quan s'inicia la tramesa de dades desde el servei fins a l'aplicació client. El sistema es divideix en dos parts, una part és la funció GetPaquetActualitzacions() que es executada des del servidor, l'altra part és la funció ModelAplicacio.GetActualitzacions() que s' executa a l'aplicació client, ambdues funcions treballen de forma sincronitzada tal i com es detalla en el següent paràgraf.

  Quan es rep la primera petició es calcula el número de paquets que fan falta enviar a l'aplicació client, això va en funció del nombre de dades ha enviar. El client rep el primer paquet i comprova la propietat TotalPaquets per saber quantes peticions ha de fer per completar la tramesa, incrementa la propietat

peticio.NumPaquet i torna ha enviar una nova petició al servidor, quan aquest la rep comprova la propietat NumPaquet i retorna els següents 5000 bytes, el procés continua fins que el client ha rebut tots els bytes de la descàrrega. Tot seguit, es mostra el codi font de les dues funcions.

```
public PaquetDades GetPaquetActualitzacions(Peticio peticio)
{
    byte[] bytes = CrearPaquetActualitzacions(peticio);

    //Total de paquets ha enviar.
    int TotalPaquets = bytes.Length / PaquetDades.GRANDARIA_PAQUET;
    if ((bytes.Length % PaquetDades.GRANDARIA_PAQUET) > 0)
        TotalPaquets++;

    PaquetDades paquet = new PaquetDades();
    paquet.TotalPaquets = TotalPaquets;
    paquet.TotalBytesPaquet = bytes.Length;

    int sourceIndex = peticio.NumPaquet * PaquetDades.GRANDARIA_PAQUET;
    int longitud = 0;

    if ((sourceIndex + PaquetDades.GRANDARIA_PAQUET) > bytes.Length)
        longitud = bytes.Length - sourceIndex;
    else
        longitud = PaquetDades.GRANDARIA_PAQUET;

    paquet.Dades = new byte[longitud];
    Array.Copy(bytes, sourceIndex, paquet.Dades, 0, longitud);
    return paquet;
}


private void GetActualitzacions()
{
    PaquetDades paquet= Gateway.Servei.GetPaquetActualitzacions(peticio);
    byte[] bytes = new byte[paquet.TotalBytesPaquet];

    int i = 0;
    int index = i * PaquetDades.GRANDARIA_PAQUET;
    paquet.Dades.CopyTo(bytes, index);
    peticio.NumPaquet++;

    for (i = 1; i < paquet.TotalPaquets; i++)
    {
        paquet = Gateway.Servei.GetPaquetActualitzacions(peticio);
        index = i * PaquetDades.GRANDARIA_PAQUET;
        paquet.Dades.CopyTo(bytes, index);
        peticio.NumPaquet++;
    }

    HelperSerializer helper = new HelperSerializer();

    //Es descomprimeixen els bytes rebuts
    string stringXml= Encoding.Default.GetString(Fitxer.Descomprimir(bytes));

    //Es deserialitzen les actualitzacions en format xml.
    Aplicacio.Actualitzacions = (List<Actualitzacio>)helper.Deserialitzar
                                (typeof(List<Actualitzacio>), stringXml);

    VistaActualitzacions = Aplicacio.Actualitzacions;

    if (Aplicacio.Actualitzacions.Count>0)
        CurrentActualitzacio = Aplicacio.Actualitzacions[0];

    if (ActualitzacionsRebudes != null)
        ActualitzacionsRebudes(this, new EventArgs());
}
```

- **Creació de l'aplicació client**: es crea la classe Gateway i ClientServeiActualitzacio que donen accés a les funcions que publica el servei WCF, s'implementa la classe ModelAplicacio que incorpora part del sistema gestor de descàrregues, s'afegeixen els controls XAML d'interfície d'usuari creats amb l'eina Expression Studio, es creen els diccionaris de recursos per emmagatzemar components gràfics, com ara imatges i plantilles, finalment, es crea la classe frontera ControllerMainWindow, tot seguit, es mostra el codi font en XAML de la finestra on es mostra el fitxer XML que representa el paquet d'instal·lació. La primera imatge és la plantilla que defineix l'aspecte visual d'una actualització al ser mostrada mitjançant l'objecte *ListBox*, la segona imatge representa la finestra en si mateixa.

```
<UserControl.Resources>
    <DataTemplate x:Key="TemplateActualitzacio">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="50"/>
                <ColumnDefinition Width="150"/>
                <ColumnDefinition Width="60"/>
                <ColumnDefinition Width="40"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="20"/>
            </Grid.RowDefinitions>
            <TextBlock Grid.Column="0" Text="{Binding Path=IdActualitzacio}"/>
            <TextBlock Grid.Column="1" Text="{Binding Path=DataPublicacio}"/>
            <TextBlock Grid.Column="2" Text="{Binding Path=VersioInicial}"/>
            <TextBlock Grid.Column="3" Text="{Binding Path=VersioFinal}"/>
        </Grid>
    </DataTemplate>
</UserControl.Resources>
```

```
<StackPanel>
    <TextBlock Margin="5,0,0,0" Text="Llista d'actualitzacions pendents" Style="{StaticResource StyleText_12}"/>
    <Grid Margin="0,10,0,0">
        <Grid.ColumnDefinitions...>
        <!-- Listbox amb les actualitzacions pendents -->
        <Grid Grid.Column="0" Margin="10,0,0,0" >
            <Grid.ColumnDefinitions...>
            <Grid.RowDefinitions...>

            <TextBlock Grid.Column="0" Grid.Row="0" Text="Id" Style="{StaticResource StyleText_10}"></TextBlock>
            <TextBlock Grid.Column="1" Grid.Row="0" Text="Data de publicació" Style="{StaticResource StyleText_10}"/>
            <TextBlock Grid.Column="2" Grid.Row="0" Text="Ver. inicial" Style="{StaticResource StyleText_10}"/>
            <TextBlock Grid.Column="3" Grid.Row="0" Text="Ver. final" Style="{StaticResource StyleText_10}"/>

            <ListBox Grid.ColumnSpan="4" Grid.Row="1" Grid.Column="0" Name="lstActualitzacions"
                     ItemTemplate="{StaticResource TemplateActualitzacio}"
                     SelectedItem="{Binding Path=Model.CurrentActualitzacio}"
                     ItemsSource="{Binding P MS.Internal.Xaml.ReflectionTypeNode zacions}"
                     MouseDoubleClick="lstActualitzacions_MouseDoubleClick"/>
        </Grid>
        <!-- Mostra el fitxer xml -->
        <Grid Grid.Column="1" >
            <ScrollViewer HorizontalAlignment="Left" VerticalScrollBarVisibility="Auto" Margin="5,0,0,0"
                          HorizontalScrollBarVisibility="Auto" Width="320" Height="280" >
                <StackPanel>
                    <TextBlock Style="{StaticResource StyleText_10}"
                               Text="{Binding Path=Model.CurrentActualitzacio.PaquetInstalacioXml}"/>
                </StackPanel>
            </ScrollViewer>
        </Grid>
    </Grid>
    <Button Name="btnActualitzar" Margin="0,10,0,0" Content="Actualitzar" Click="btnActualitzar_Click"
            Width="150" Height="25" Template="{StaticResource btn_aceptar_iluminat}"></Button>
</StackPanel>
```

4. Manual d'ús

4.1 Aplicació servidor

- Crear una nova aplicació:

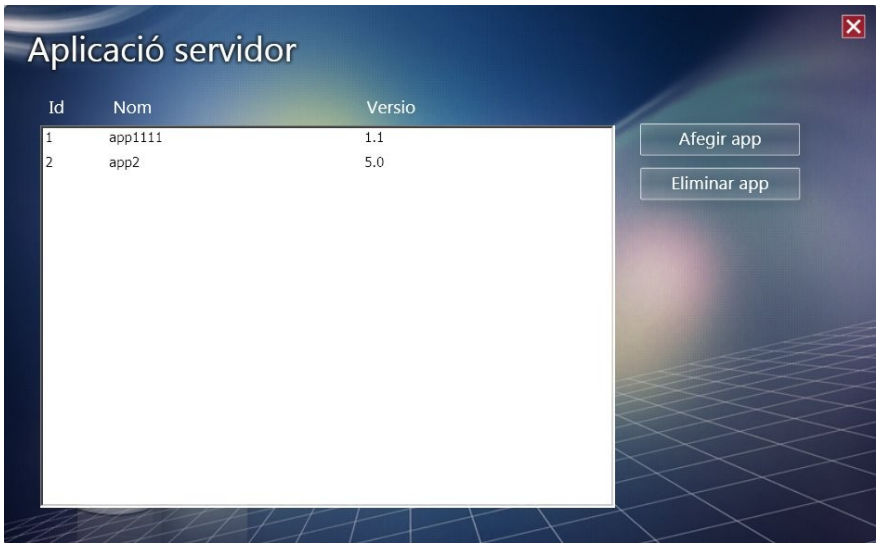
  Botó *Afegir app*.

- Eliminar una aplicació:

  Botó *Eliminar app*.

- Veure o modificar una aplicació:

  Fer doble click sobre una aplicació seleccionada.



- Tornar a la llista d'aplicacions

  Botó *Guardar apps*:

- Modificar el nom o la versió d'una aplicació:

  Editar el contingut del textbox del nom o de la versió i fer click a Llista apps.

- Afegir una actualització

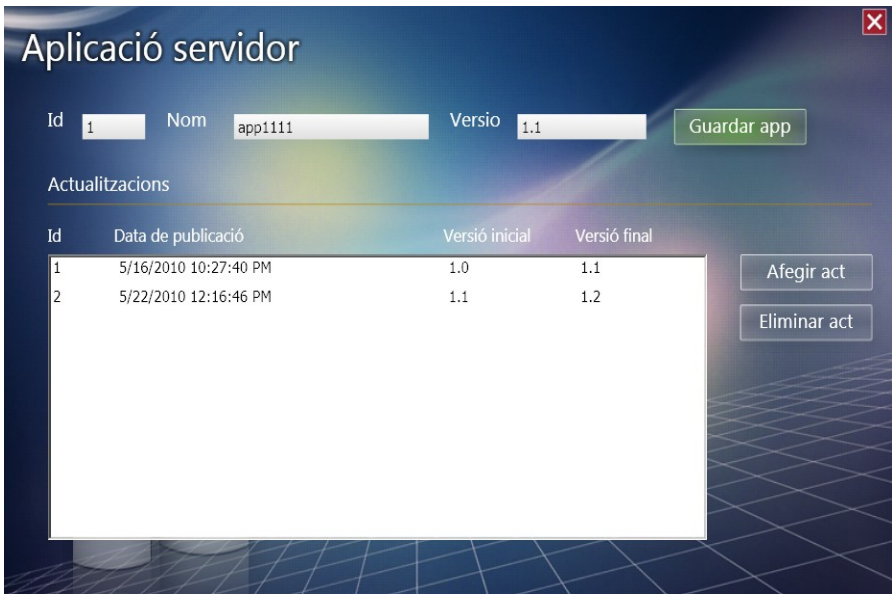
  Botó *Afegir actualització*

- Eliminar una actualització:

  Botó *Eliminar actualització*

- Veure o modificar una actualització:

  Fer doble click sobre una actualització seleccionada.



- Afegir una acció:

  Botó *Afegir acció*. S'han d'omplir el següents camps:

  - Fitxer: path complet del fitxer origen.
  - PathDesti: path complet on desarà el fitxer en l'equip client.
  - Tipus: tipus d'acció a realitzar.

  Nota: Quan es guardi l'acció, si aquesta es de tipus afegir o modificar i el path del camp fitxer es incorrecte l'aplicació ho notificarà a l'usuari.

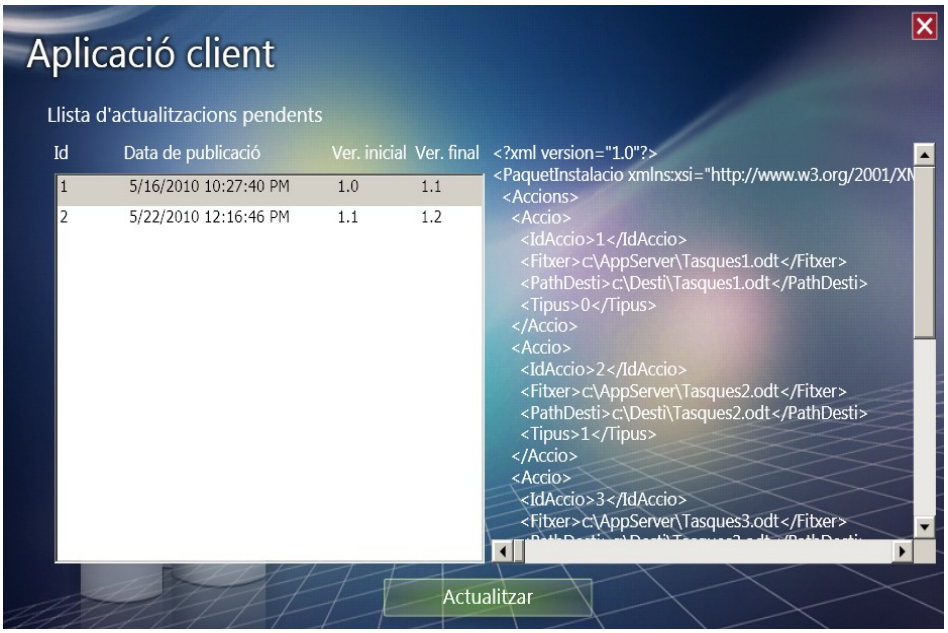
- Eliminar una acció

  Seleccionar una acció i fer click a "*Elim. Acció"*

4.2 Aplicació client

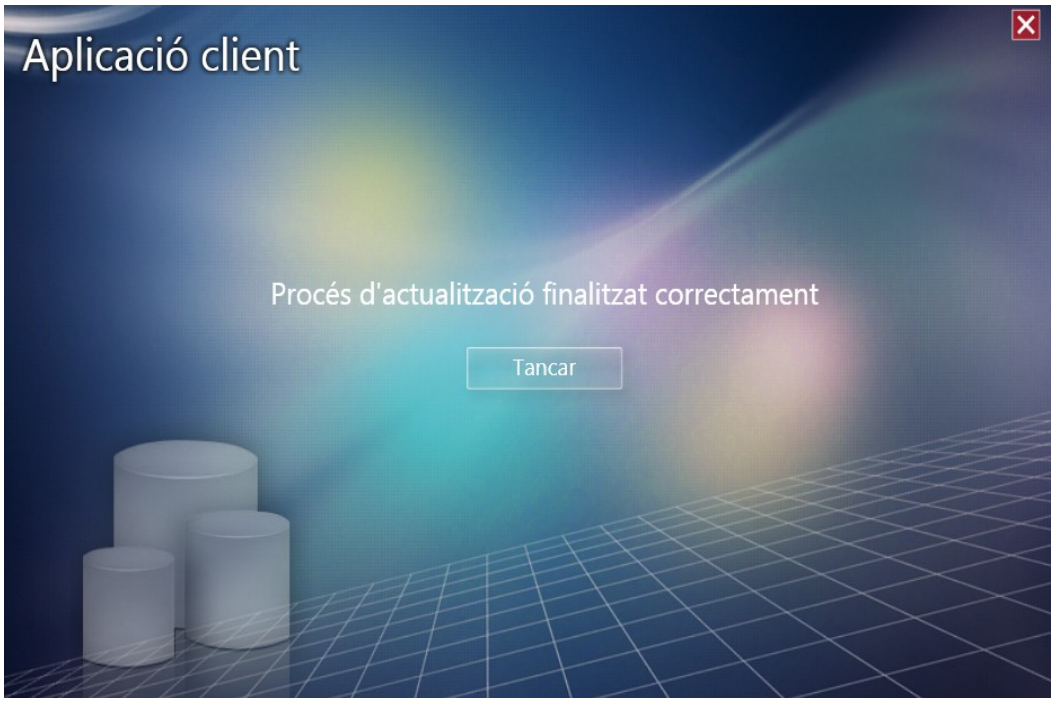
Quan s'inicia l'aplicació es comprova si existeixen actualitzacions disponibles, en cas afirmatiu apareix el botó *Descarregar Actualitzacions*.

Quan finalitza la descàrrega es mostra a la part esquerra les actualitzacions pendents d'instal·lar i a la part dreta el paquet d'instal·lació en XML de l'actualització seleccionada, això inclou: l'acció a dur a terme, el path origen i destí de cada fitxer i el fitxer en sí mateix en format binari.

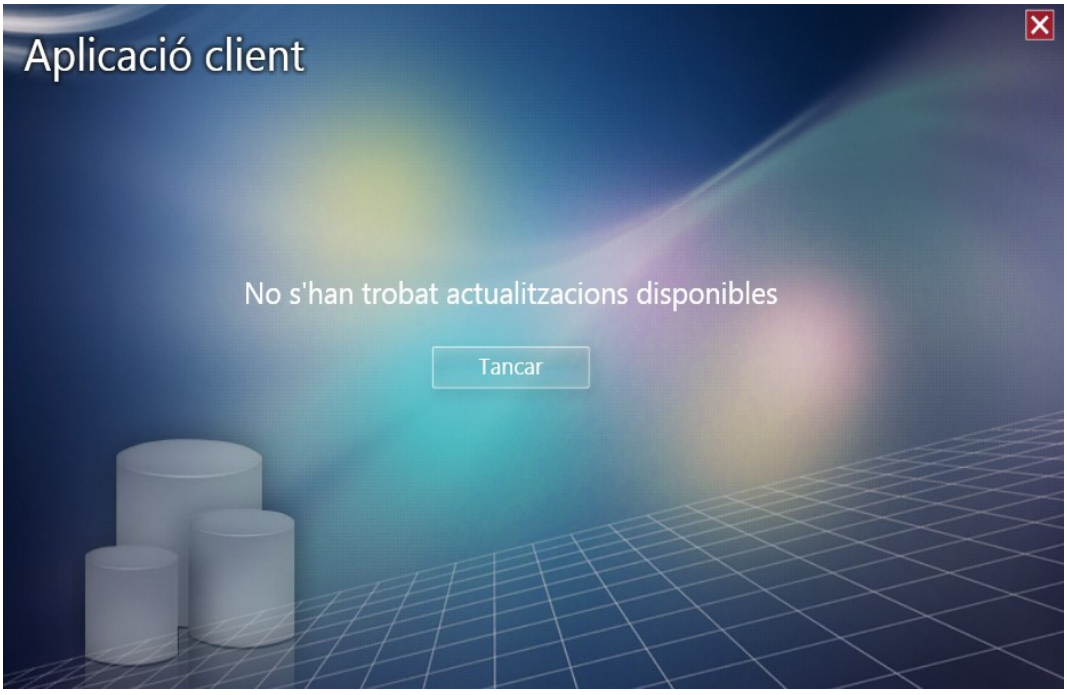


Per instal·lar les actualitzacions es prem el botó *Actualitzar.*

Un cop ha finalitzat la instal·lació, es fa click al botó *Tancar* per finalitzar l'aplicació.



En el cas de no haver-hi actualitzacions es mostra la següent pantalla, quan es fa click al botó *tancar* finalitza l'aplicació.



5. Conclusions

Desenvolupar programari es tot un disciplina, fa pocs anys la principal tasca d'un programador era implementar adequadament el model de negoci, ara, a més a més, cal tenir especial cura en la interfície d'usuari, l'aspecte exterior del programari ha de ser bonic i sobretot intuïtiu, definir quin pot ser el disseny més adequat, de vegades, pot convertir-se en una tasca molt complexa, definir la interfície d'usuari es tota una nova disciplina com s'explica en l'assignatura Interacció Humana amb els Ordinadors.

Les tecnologies de desenvolupament de programi com ara Windows Presentation Foundation obren nous conceptes sobre el que fins ara es considerava desenvolupament de *software*, abans, un control de formulari com per exemple el *command button* estava vinculat a un aspecte gràfic predefinit, aquest limita molt les possibilitats de canviar el disseny de l'aplicació, però, a partir de la versió 3.0 del Framework .NET de Microsoft i mitjançant el llenguatge d'etiquetes XAML (basat en XML), un control gràfic pot tenir qualsevol aparença externa, ja que, se li pot definir un *template* amb el disseny que es vulgui.

En conclusió, el disseny ha set un dels grans reptes d'aquest projecte, l'objectiu era donar un aire diferent, innovador i al mateix temps intuïtiu, en efecte, la tecnologia WPF dona pas a una nova forma de veure i entendre el programari de gestió i control de sistemes, com ja fa anys que passa en el món dels videojocs, per desenvolupar un producte ben acabat d'un caire modern i innovador fan falta dos perfils professionals ben diferenciats: el programador que ha d'implementar la lògica de l'aplicació i el dissenyador gràfic que ha de definir l'aspecte visual de l'aplicació.

5.1 Avaluació de costos

Per fer una valoració real del cost del projecte s'han tingut en compte dues característiques, el nombre de persones implicades i el tipus de valoració: per hores, per dies, per objectius, etc. En aquest cas, el projecte l'ha fet una persona amb una dedicació de 5 hores al dia i el cost per hora s'ha establert en 10 euros. L'avaluació s'ha dividit en funció de les fases de desenvolupament del projecte:

Fase	Hores	Preu (10 €/h)
Anàlisi, disseny de les interfícies gràfiques i elaboració de la documentació	20 dies x 5h/dia = 100 h	1.000,00 €
Implementació		
Implementació del servei	7 dies x 5h/dia = 35 h	350,00 €
Implementació gestió del servei	10 dies x 5h/dia = 50 h	500,00 €
Implementació aplicació client	10 dies x 5h/dia = 50 h	500,00 €
Total		2.350,00 €

Per tant, el cost total de desenvolupament del projecte és de 2.350,00 Euros.

5.2 Treball futur

Tot i que la solució es funcional i s'adapta als requisits inicials seria convenient implementar una sèrie de noves funcionalitats abans de posar el sistema en producció. El treball pendent seria el següent:

Aplicació client:

- Implementar un control que mostri el progrés de descàrrega de cada actualització.
- En la finestra “Llista d'actualitzacions pendents” es mostra el fitxer XML amb el contingut de cada actualització, seria convenient mostrar la informació d'una forma més entenedora per al client final.
- Substituir el títol “Aplicació client” per un nom més comercial com per exemple: SoftUpdater Client

Aplicació servidor:

- Crear un *Template* d'un estil semblant al disseny actual de l'aplicació per als controls “*Listbox*” i “*ComboBox*“
- Controlar l'entrada de les caixes de text “versió inicial” i “versió final” per a que només es pugui introduir el número de versió en format x.xx.
- Quan es crea una nova acció s'hauria de mostrar un “common dialog” per seleccionar el fitxer origen.
- Substituir el títol “Aplicació servidor” per un nom més comercial com per exemple SoftUpdater Admin.

5.3 Objectius aconseguits

El principal objectiu assolit és que s'han posat en pràctica els coneixements adquirits al llarg de la carrera, a tall d'exemple, s'ha fet ús de la capacitat d'abstracció, planificació i anàlisi, al mateix temps, s'han consolidat els coneixements en enginyeria de programari i interacció humana amb els ordinadors. S'ha fet ús del les nocions obtingudes en programació, programació orientada a l'objecte i estructura de la informació per implementar la lògica de la solució, s'han posat en pràctica els coneixements assolits en bases de dades per fer un ús òptim del SGBD, ha set imprescindible conèixer les matèries de xarxes i estructura i tecnologia de computadors per desenvolupar el sistema gestor de descàrregues.

Els objectius secundaris assolits són els següents:

- S'ha fet ús d'un llenguatge de programació d'última generació com C# per implementar la solució.
- S'ha fet ús del llenguatge SQL i de la base de dades Microsoft SQL Server.
- S'han fet servir les tecnologies .NET Framework 3.0 com WPF i WCF.

Amb aquest projecte s'ha assolit l'objectiu de crear una solució que permeti actualitzar qualsevol tipus d'aplicació, a més a més, les actualitzacions es fan amb el mínim de canvis possibles per estalviar ample de banda. El servei s'ha fet amb WCF (Windows Communication Foundation) i les interfícies gràfiques de client i servidor amb WPF (Windows Presentation Foundation), per tant, els objectius aconseguits son:

- S'ha creat un servei amb WCF que publica les actualitzacions disponibles per una aplicació determinada.
- S'ha creat una aplicació amb WPF per al servidor que permet gestionar les actualitzacions del servei.
- S'ha creat una aplicació amb WPF per al client que li permet descarregar les actualitzacions disponibles.

6. Glossari

WPF: **W**indows **P**resentation **F**oundation, tecnologia de Microsoft que utilitza el llenguatge basat XAML per crear interfícies gràfiques.

WCF: **W**indows **C**ommunication **F**oundation, model de programació unificat de Microsoft per generar aplicacions orientades a serveis.

XML: e**X**tensible **M**arkup **L**anguage, metallenguatge d'etiquetes desenvolupat per el World Wide Consortium (W3C).

XAML: e**X**tensible **A**pplication **M**arkup **L**anguage, és el llenguatge d'etiquetes basat en XML que fa servir la tecnologia WPF.

ADO.NET: Conjunt de classes dins del Framework .NET que faciliten l'accés a serveis i a dades.

SQL: **S**tructured **Q**uery **L**anguage, llenguatge de consultes que fan servir els SGBD.

FTP: **F**ile **T**ransfer **P**rotocol, protocol de transferència de fitxers.

SGBD: **S**istema **G**estor de **B**ases de **D**ades, com per exemple Microsoft SQL Server.

IIS: **I**nternet **I**nformation **S**erver, servidor web de Microsoft.

Classe frontera: classes que fan d'enllaç entre la lògica de presentació i la lògica d'aplicació.