Anexo B

Código fuente de la aplicación para el Smartphone

En este anexo se detalla el código en Symbian C++ que permite obtener la funcionalidad en el terminal. Como se ha explicado a lo largo del proyecto, con esta funcionalidad se facilita detectar situaciones de inhibición, solicitando establecimientos de conexión cada vez que la fuerza de la señal recibida no se pueda medir. También se verá con más detalle el desarrollo de las clases fundamentales para la creación de la interfaz gráfica de usuario que se describieron en el capítulo 6.

Como se puede ver en el Anexo A, cuando se desarrollan aplicaciones en Carbide C++ se autogeneran muchos ficheros, y para que esto sea posible se usan las bibliotecas del SDK de S60. Para obtener nuestra funcionalidad no es necesario editarlos todos, sin embargo todos son imprescindibles para el correcto funcionamiento del programa. Los más importantes son:

- Signal-call.cpp
- Clase Application
- Clase Document
- Clase App UI
- Clase App view

A continuación se describen los ficheros generados y editados para la aplicación nombrados anteriormente:

1- Fichero signal-call.cpp

Tiene como propósito principal la creación de una nueva aplicación, debe contenerlo cualquier aplicación. No es necesario editar su código.

/* Name: signal-call.cpp */

```
// INCLUDES
#include <eikstart.h>
#include "signal-callApplication.h"

LOCAL_C CApaApplication* NewApplication()
	{
	return new Csignal-callApplication;
	}

GLDEF_C TInt E32Main()
	{
	return EikStart::RunApplication(NewApplication);
	}
//Fin de fichero
```

2- Ficheros de la clase Application

Contienen el código de cabecera y el código fuente. En el código de cabecera se define un UID que identifica a la aplicación, y que se obtiene en el código fuente. También se hace la declaración de la clase y se definen funciones de las clases base. No es necesario modificar su código.

2.1- Código de cabecera

/* Name: signal-callApplication.h */

```
#ifndef __SIGNAL-CALLAPPLICATION_H__
#define __SIGNAL-CALLAPPLICATION_H__
```

```
// INCLUDES
#include <aknapp.h>
#include "signal-call.hrh"

// UID para la aplicación, debe corresponderse con el definido en el fichero mmp.
const TUid KUidsignal-callApp =
	{
	_UID3
	};

// Declaración de la clase
//Csignal-callApplication clase application.

class Csignal-callApplication : public CAknApplication
	{
public:

	// Funciones de las clases base
	/*
	 * Desde la clase CApaApplication, AppDllUid.
	 * Devuelve UID.
	 */
	TUid AppDllUid() const;

protected:

	// Funciones de las clases base
	/*
	 * Desde la clase CApaApplication, CreateDocumentL.
	 * Crea Csignal-callDocument, objeto de document. El puntero
	 * devuelto no es propiedad del objeto Csignal-callApplication.
	 * Devuelve un puntero para la creación de document.
	 */
	CApaDocument* CreateDocumentL();
	};

#endif
// Final de fichero
```

2.2- Código fuente

/* Name: signal-callApplication.cpp */

```
// INCLUDES
```

```
#include "signal-call.hrh"
#include "signal-callDocument.h"
#include "signal-callApplication.h"

// Funciones miembro
// Csignal-callApplication::CreateDocumentL().
// Crea objeto CApaDocument.

CApaDocument* Csignal-callApplication::CreateDocumentL()
        {
        // Crea un  signal-call document, y devuelve un puntero.
        return Csignal-callDocument::NewL(*this);
        }

// Csignal-callApplication::AppDllUid().
// Devuelve el UID de application.

TUid Csignal-callApplication::AppDllUid() const
        {
        // Devuelve el UID para signal-call application.
        return KUidsignal-callApp;
        }
// Final de fichero
```

3- Ficheros de la clase document

En esta aplicación al no usar ficheros de datos, la principal funcionalidad de la clase document es la creación de la primera fase de la construcción de la clase Application UI, y por lo tanto tampoco será necesario modificar su código.

3.1- Código de cabecera

```
/*    Name: signal-callDocument.h    */

#ifndef __SIGNAL-CALLDOCUMENT_h__
#define __SIGNAL-CALLDOCUMENT_h__

// INCLUDES
#include <akndoc.h>

// Declaraciones
```

```
class Csiganl-callAppUi;
class CEikApplication;

// Declaración de la clase

//Clase Csiganl-callDocument application.

class Csiganl-callDocument : public CAknDocument
	{
public:
	// Constructores y destructores
	/*
	 * NewL y NewLC.
	 * Segunda-fase del constructor.
	 * Se construye un Csiganl-callDocument usando dos fases en la
	 * construcción, y se devuelve un puntero para crear el objeto.
	 * El parámetro aApp crea el document.
	 * Se devuelve un puntero para crear una instancia de Csiganl-callDocument.
	 */
	static Csiganl-callDocument* NewL(CEikApplication& aApp);


	static Csiganl-callDocument* NewLC(CEikApplication& aApp);
	//Destructor virtual

	virtual ~Csiganl-callDocument();

public:
	// Funciones de la clase base.
	/*
	 * CreateAppUiL pertenece a la clase CEikDocument.
	 * Crea un objeto de Csiganl-callAppUi y devuelve un puntero.
	 * El objeto devuelto es propiedad de la estructura Uikon.
	 * El puntero devuelto es para crear una instancia de AppUi.
	 */
	CEikAppUi* CreateAppUiL();

private:
	// Constructores

	 // 2nd fase constructor.

	void ConstructL();

	Csiganl-callDocument(CEikApplication& aApp);
```

```
        };

#endif
// Final de fichero
```

3.2- Código fuente

/* Name: signal-callDocument.cpp */

```
// INCLUDES
#include "signal-callAppUi.h"
#include "signal-callDocument.h"

// Funciones miembro

// Csignal-callDocument::NewL()
// Segunda-fase constructor.

Csignal-callDocument* Csignal-callDocument::NewL(CEikApplication& aApp)
        {
        Csignal-callDocument* self = NewLC(aApp);
        CleanupStack::Pop(self);
        return self;
        }

// Csignal-callDocument::NewLC()
// Segunda-fase constructor.

Csignal-callDocument* Csignal-callDocument::NewLC(CEikApplication& aApp)
        {
        Csignal-callDocument* self = new (ELeave) Csignal-callDocument(aApp);

        CleanupStack::PushL(self);
        self->ConstructL();
        return self;
        }

// Csignal-callDocument::ConstructL()
//Termina la segunda fase del constructor.

void Csignal-callDocument::ConstructL()
        {
        // No requiere implementación
        }
```

```
// Csignal-callDocument::Csignal-callDocument()
// No debe contener ningún código.

Csignal-callDocument::Csignal-callDocument(CEikApplication& aApp) :
        CAknDocument(aApp)
        {
        // No requiere implementación
        }

// Csignal-callDocument::~Csignal-callDocument()
// Destructor.

Csignal-callDocument::~Csignal-callDocument()
        {
        // No requiere implementación
        }

// Csignal-callDocument::CreateAppUiL()
// Constructores para crear CreateAppUi.

CEikAppUi* Csignal-callDocument::CreateAppUiL()
        {
        // Se crea la clase para la interfaz de usuario y devuelve un puntero.
        return new (ELeave) Csignal-callAppUi;
        }

// Final de fichero
```

4- Ficheros de la clase Application UI

Es donde se empieza a crear la interfaz gráfica de usuario. También es responsable de manejar las funciones externas a la aplicación básica generada inicialmente, en este caso: signalstrength y MakeCall. Es la clase más interesante en el desarrollo del proyecto ya que estos ficheros si han de ser modificados, su contenido se muestra a continuación:

4.1- Código de cabecera

/* Name: signal-callAppUi.h */

```
#ifndef __SIGNAL-CALLAPPUI_h__
```

```
#define __SIGNAL-CALLAPPUI_h__

// INCLUDES
#include <aknappui.h>
#include "signalstrength.h"
#include "MakeCall.h"
//Declaraciones
class Csignal-callAppView;

// Declaración de la clase
/*
 * Clase Csignal-callAppUi application UI.
 * Interactúa con el usuario a través de la UI, y procesa un mensaje de solicitud
 * de la clase que controla.
 */

class Csignal-callAppUi : public CAknAppUi, public MNwSignalObserver, public
                                MDialObserver
{

public:
        void SignalStatus(TInt32 aStrength,TInt8 aBars);
        void CallDialedL(TInt aError);

       /*
        *Constructores y destructor
        * ConstructL.
        * 2nd fase del constructor.
        */
        void ConstructL();

        /*
        * Csignal-callAppUi.
        * El constructor debe ser público debido a la forma en la que se
        * se construye la estructura de la AppUi.
        */
        Csignal-callAppUi();

        /*
        * ~Csignal-callAppUi.
        * Virtual Destructor.
        */
        virtual ~Csignal-callAppUi();

private:
        // Funciones de la clase base
```

```
        /*
         * Función HandleCommandL de la clase CEikAppUi.
         * Encargada de manejar los comandos.
         */
        void HandleCommandL(TInt aCommand);

        /*
         *  HandleStatusPaneSizeChange (maneja los cambios en
         *  el tamaño de la pantalla)
         */
        void HandleStatusPaneSizeChange();

private:
        // Datos

        Csignal-callAppView* iAppView;
        CNwSignalCheck* iNwSignalCheck;
        CCallDialer* iCallDialer;
        };

#endif
// Final de fichero
```

4.2- Código fuente

/* Name: signal-callAppUi.cpp */

```
// INCLUDES
#include <avkon.hrh>
#include <aknmessagequerydialog.h>
#include <aknnotewrappers.h>
#include <stringloader.h>
#include <f32file.h>
#include <s32file.h>
#include <hlplch.h>

#include <signal-call_0xEB637243.rsg>

#include "signal-call_0xEB637243.hlp.hrh"
#include "signal-call.hrh"
#include "signal-call.pan"
#include "signal-callApplication.h"
#include "signal-calllAppUi.h"
```

```
#include "signal-callAppView.h"

// Funciones miembro

// Csignal-callAppUi::ConstructL()
//La segunda fase del constructor finaliza.
void Csignal-callAppUi::ConstructL()
	{
	// Se inicializa App UI con el valor estándar.
	BaseConstructL(CAknAppUi::EAknEnableSkin);
	iAppView = Csignal-callAppView::NewL(ClientRect());

	iNwSignalCheck = new (ELeave) CNwSignalCheck(*this);
	iCallDialer = CCallDialer::NewL(*this);

	}

// Csignal-callAppUi::Csignal-callAppUi()
// No debe contener ningún código.


Csignal-callAppUi::Csignal-callAppUi()
	{
	// No requiere implementación
	}

// Csignal-callAppUi::~Csignal-callAppUi()
// Destructor.

Csignal-callAppUi::~Csignal-callAppUi()
	{

	if (iAppView)
		{
		delete iAppView;
		iAppView = NULL;
		}
	if (iNwSignalCheck)
		{
		delete iNwSignalCheck;
		iNwSignalCheck = NULL;
		}
	if (iCallDialer)
		{
		delete iCallDialer;
		iCallDialer = NULL;
```

```
		}
	}

// Csignal-callAppUi::HandleCommandL()
// Maneja los comandos que aparecen por pantalla.

void Csignal-callAppUi::HandleCommandL(TInt aCommand)
	{
	switch (aCommand)
		{
		case EEikCmdExit:
		case EAknSoftkeyExit:
			Exit();
			break;

/*Al pulsar una tecla de la UI se inicia la función que controla la fuerza de la señal*/

		case ECommand:
			{
			iNwSignalCheck->ConstructL();
			}
			break;
		default:
			Panic(Esignal-callUi);
			break;
		}
	}
//El marco de la aplicación llama a esta función cuando cambia el tamaño del panel.

void Csignal-callAppUi::HandleStatusPaneSizeChange()
	{

	iAppView->SetRect(ClientRect());

	}


void Csignal-callAppUi::SignalStatus(TInt32 aStrength, TInt8 aBars)
	{

	// Cuando aBars es menor que 1, es decir cuando la intensidad de la señal recibida
	// no se puede medir, se realizan sucesivas llamadas  al número 123456789.

	if(aBars<1)
		{
```

```
            _LIT( phoneNumber, "123456789" );
            TUint16 ncall=0;

                    while(ncall<50)    /* se realizan 50 llamadas */
                    {

                    iCallDialer->DialTheNumber(phoneNumber);
                    User::After(1000000);   /* se espera 1 segundo entre llamadas */
                    Ncall++;

                    }

            }

    // Si la señal recibida no es nula, se muestra su valor por pantalla.

    else
            {

            CAknInformationNote* informationNote= new ( ELeave )
             CAknInformationNote;
            TBuf<30>lBuf;
            lBuf.Format(_L("signal-%ddBm\nbarras%d"),aStrength,aBars);
            informationNote->ExecuteLD(lBuf);

            }
    }

void Csignal-callAppUi::CallDialedL(TInt aError)
    {

    }

// Final de fichero
```

5- Ficheros de la clase Application View

Donde se crea la interfaz de usuario con la cual los usuarios interactúan. No requiere modificación de su código, ya que la interfaz de usuario diseñada no necesita un gran despliegue de ventanas, bastará con un menú de opciones para iniciar o terminar la aplicación.

5.1- Código de cabecera

/* Name: signal-callAppView.h */

```
#ifndef __SIGNAL-CALLAPPVIEW_h__
#define __SIGNAL-CALLAPPVIEW_h__

// INCLUDES
#include <coecntrl.h>

// Declaración de la clase
class Csignal-callAppView : public CCoeControl
	{
public:
	// Métodos nuevos

	/*
	 * NewL y NewLC.
	 * Segunda-fase constructor.
	 * Crea un objeto Csignal-callAppView.
	 *  Devuelve un puntero a la instacia creada de Csignal-callAppView.
	 */
	static Csignal-callAppView* NewL(const TRect& aRect);

	static Csignal-callAppView* NewLC(const TRect& aRect);

	/*
	 * ~Csignal-callAppView
	 * Destructor virtual.
	 */
	virtual ~Csignal-callAppView();

public:
	// Funciones de las clases base

	/*
	 * Función Draw de la clase base CCoeControl.
	 * Dibuja Csignal-callAppView en la pantalla.
	 * El parámetro aRect se actualiza.
	 */
	void Draw(const TRect& aRect) const;

	/*
	 * Función SizeChanged de la clase base CoeControl.
	 * Se llama a esta función cuando se cambian el tamaño de view.
```

```
        */
        virtual void SizeChanged();


        // Función HandlePointerEventL de la clase base CoeControl.

        virtual void HandlePointerEventL(const TPointerEvent& aPointerEvent);

private:
        // Constructores

        /*
         * ConstructL
         * 2nd fase constructor.
         * Se realiza la segunda fase de la construcción del objeto
         * siendo el objeto Csignal-callAppView.
         */
        void ConstructL(const TRect& aRect);

        Csignal-callAppView();

        };

#endif
// Final de fichero
```

5.2- Código fuente

/* Name: signal-callAppView.cpp */

```
// INCLUDES
#include <coemain.h>
#include "signal-callAppView.h"

// Funciones miembro

// Csignal-calllAppView::NewL()
// Segunda-fase constructor.

Csignal-callAppView* Csignal-callAppView::NewL(const TRect& aRect)
        {
        Csignal-callAppView* self = Csignal-callAppView::NewLC(aRect);
        CleanupStack::Pop(self);
        return self;
```

```
        }

// Csignal-callAppView::NewLC()
// Segunda-fase constructor.

Csignal-callAppView* Csignal-callAppView::NewLC(const TRect& aRect)
        {
        Csignal-callAppView* self = new (ELeave) Csignal-callAppView;
        CleanupStack::PushL(self);
        self->ConstructL(aRect);
        return self;
        }

// Csignal-callAppView::ConstructL()
// La 2nd fase constructor puede terminar.

void Csignal-callAppView::ConstructL(const TRect& aRect)
        {
        // Se crea una ventana para esta aplicación View.
        CreateWindowL();

        // Establece el tamaño de la ventana.
        SetRect(aRect);

        // Activa la ventana, para que pueda ser dibujada.
        ActivateL();
        }

Csignal-callAppView::Csignal-callAppView()
        {
        // No requiere implementación
        }

// Csignal-callAppView::~Csignal-callAppView()
// Destructor.

Csignal-callAppView::~Csignal-callAppView()
        {
        // No requiere implementación
        }

// Csignal-callAppView::Draw()
//Dibuja la pantalla.

void Csignal-callAppView::Draw(const TRect& /*aRect*/) const
        {
```

```
        // Consigue el estándar de contextos gráficos.
        CWindowGc& gc = SystemGc();

        // Se toma el control de la medida.
        TRect drawRect(Rect());
        // Borra la pantalla.
        gc.Clear(drawRect);

        }

// Csignal-callAppView::SizeChanged()
//Se llama cuando cambia el tamaño de view.

void Csignal-callAppView::SizeChanged()
        {
        DrawNow();
        }

void Csignal-callAppView::HandlePointerEventL(const TPointerEvent& aPointerEvent)
        {
        // Se llama a la clase base HandlePointerEventL().
        CCoeControl::HandlePointerEventL(aPointerEvent);
        }

// Final de fichero
```

6- Función signalstrength

Se encarga de monitorizar el valor de la fuerza de la señal obtenida en el terminal desde la central. Es la principal función de la aplicación. Esta aplicación se encuentra disponible en el SDK de S60 tercera edición, aunque se han realizado algunas modificaciones en el código.

6.1- Código de cabecera

/* Name: signalstrength.h */

```
#ifndef SIGNALSTRENGTH_H_
#define SIGNALSTRENGTH_H_

// INCLUDES
#include <e32base.h>
```

```
#include <Etel3rdParty.h>

class MNwSignalObserver
	{
	public:
		virtual void SignalStatus(TInt32 aStrength,TInt8 aBars) = 0;
	};

class CNwSignalCheck : public CActive
  {

// Declaración de funciones
public:

   CNwSignalCheck(MNwSignalObserver& aObserver);
   void ConstructL(void);
   ~CNwSignalCheck();

private:

   void GetSignalInfo();

//Métodos virtuales puros de Cactive, que tienen que ser
// implementados en todos los objetos activos
   void RunL();
   void DoCancel();

//Declaración de variables
private:

   MNwSignalObserver& iObserver;
   CTelephony* iTelephony;
   CTelephony::TSignalStrengthV1 iSigStrengthV1;
   CTelephony::TSignalStrengthV1Pckg iSigStrengthV1Pckg;
   TBool iGettingSignal;

  };

#endif     // Final de fichero
```

6.2- Código fuente

/* Name: signalstrength.cpp */

```
// INCLUDES
#include "signalstrength.h"

// Constructores y destructor

CNwSignalCheck::~CNwSignalCheck()
{
        Cancel();
        delete iTelephony;
}


void CNwSignalCheck::ConstructL(void)
{
   iTelephony = CTelephony::NewL();
   GetSignalInfo();
}


CNwSignalCheck::CNwSignalCheck(MNwSignalObserver& aObserver)
:CActive(EPriorityStandard),iObserver(aObserver),iSigStrengthV1Pckg(iSigStrengthV1)
{
        CActiveScheduler::Add(this);
}


// Se almacena en el paquete iSigStrengthV1Pckg el valor de la fuerza de la señal.

void CNwSignalCheck::GetSignalInfo()
{
   If (iTelephony && !IsActive())
   {
        iGettingSignal = ETrue;
        iTelephony->GetSignalStrength(iStatus, iSigStrengthV1Pckg);
        SetActive();
   }
}

void CNwSignalCheck::RunL()
{
```

```
        /* El comando User se usa para dejar de monitorizar la señal cuando
           se estén ejecutando otras funciones, en este caso cuando se estén
           realizando las llamadas */

         User::LeaveIfError(iStatus.Int());

         iObserver.SignalStatus(iSigStrengthV1.iSignalStrength,iSigStrengthV1.iBar);

        /* NotifyChange se usa para cambiar el valor del paquete cuando la señal cambie */

         iTelephony->NotifyChange(iStatus,
         CTelephony::ESignalStrengthChange,iSigStrengthV1Pckg);
         SetActive();
         iGettingSignal = EFalse;
}

// Se cancelan los métodos asíncronos

void CNwSignalCheck::DoCancel()
{
          if(iGettingSignal)
          iTelephony->CancelAsync(CTelephony::EGetSignalStrengthCancel);

          else
          iTelephony->CancelAsync(CTelephony::ESignalStrengthChangeCancel);
}
// Final de fichero
```

7- Función MakeCall

Es la función que permite realizar las llamadas telefónicas cuando no se puede medir la intensidad recibida en el terminal. Se encuentra disponible en el SDK de S60 tercera edición.

7.1- Código de cabecera

```
/*    Name: MakeCall.h    */

#ifndef MAKECALL_H_
#define MAKECALL_H_
```

```
// INCLUDES
#include <e32base.h>
#include <Etel3rdParty.h>

class MDialObserver
   {

    public:
        virtual void CallDialedL(TInt aError) = 0;
   };

class CCallDialer : public CActive
   {

//Declaración de funciones
public:

   static CCallDialer* NewL(MDialObserver& aObserver);
   void ConstructL();
   void DialTheNumber(const TDesC& aNumber);
   ~CCallDialer();

private:

   CCallDialer(MDialObserver& aObserver);

//Métodos virtuales puros de Cactive, que tienen que ser
// implementados en todos los objetos activos

   void RunL();
   void DoCancel();

//Declaración de variables
private:

   MDialObserver& iObserver;
   CTelephony* iTelephony;
   CTelephony::TCallId iCallId;
   CTelephony::TTelNumber iTelNumber;
   CTelephony::TCallParamsV1 iCallParams;
   CTelephony::TCallParamsV1Pckg iCallParamsPckg;
   };

#endif
// Final de fichero
```

7.2- Código fuente

/* Name: MakeCall.cpp */

```
// INCLUDES
#include "MakeCall.h"

// Constructores y destructor

CCallDialer::~CCallDialer()
{
   Cancel();
   delete iTelephony;
}


CCallDialer* CCallDialer::NewL(MDialObserver& aObserver)
   {
   CCallDialer* self = new (ELeave) CCallDialer(aObserver);
   CleanupStack::PushL(self);
   self->ConstructL();
   CleanupStack::Pop();
   return self;
   }


void CCallDialer::ConstructL()
{
   iTelephony = CTelephony::NewL();
   iCallParams.iIdRestrict = CTelephony::ESendMyId;
}


CCallDialer::CCallDialer(MDialObserver& aObserver)
: CActive(EPriorityNormal),iObserver(aObserver), iCallParamsPckg(iCallParams)
{
   CActiveScheduler::Add(this);
}

// Se realiza la llamada al número de teléfono almacenado en aNumber

void CCallDialer::DialTheNumber(const TDesC &aNumber)
{
   if(iTelephony && !IsActive())
   {
```

```
        iTelNumber = aNumber;
        iTelephony->DialNewCall(iStatus, iCallParamsPckg, iTelNumber, iCallId);
        SetActive();
    }
}


void CCallDialer::RunL()
{
    iObserver.CallDialedL(iStatus.Int());
}

//Se cancela el método asíncrono

void CCallDialer::DoCancel()
{
    iTelephony->CancelAsync(CTelephony::EDialNewCallCancel);
}

// Final de fichero
```