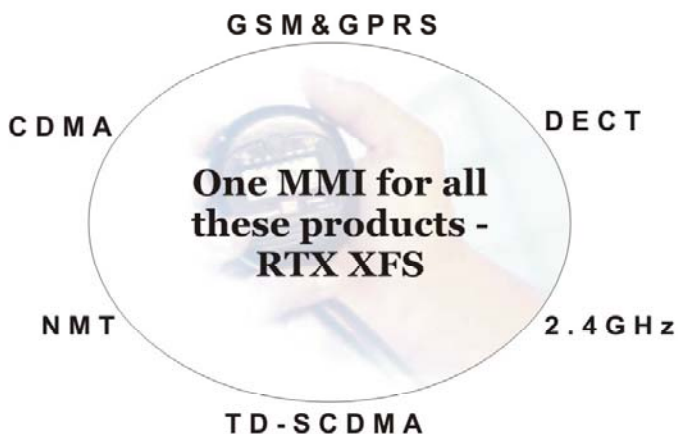


Cross Foundation Classes (XFC) for handset MMI application development

By Tage Lund, (tlu@rtx.dk) Software Coordinator at RTX.

Summary

RTX has developed a software framework that makes it possible to use the same Man Machine Interface (MMI) in various technologies and with various technological limitations.



The software framework gives a competitive advantage to a handset manufacturer because it is then able to use the same MMI in all its phones, from cordless to 3G to smart phones. To accomplish this versatility, we use a traditional Three Tier solution; thus the user applications are shielded against technology changes.

RTX is using the framework in our customer projects, and this gives our customers an advantage in terms of shorter development time and thus a faster return on investment.

Our MMI framework is based on a separation of the graphical user interface code and the application code. We also used a true component-based and extensible software framework to enable more reuse across various projects. An object-oriented programming language gave us the direct support for this.

The framework is based on well-known standards.



© 2002 vertical | products

Figure 1 RTX Consumer phone.



© 2002 vertical | products

Figure 2 RTX Business phone.

<i>Cross Foundation Classes (XFC) for handset MMI application development</i>	<u>1</u>
<i>Software reuse across various technologies</i>	<u>2</u>
<i>Advanced handsets can be based on one CPU running at a low speed</i>	<u>3</u>
Object-oriented programming language: reuse, quality and shorter time to market	<u>3</u>
<i>The RTX solution</i>	<u>4</u>
Object-oriented language is OOC; embedded C++ is not suitable	<u>4</u>
XFC windows	<u>7</u>
XFC telephony	<u>9</u>
XFC skins	<u>10</u>

Software reuse across various technologies

RTX (<http://www.rtx.dk>) has created a software framework suite that enables the development of advanced MMI applications across a wide range of devices. The software acts as a 100% reusable component-based framework that is scaleable to all the devices. Originally XFC was used in the first RTX CDMA phone projects. Like all true frameworks, it has been rewritten once to incorporate all experiences from the first projects. Later on, only minor design changes were made. It is still evolving, but rather than changing the software, we have now extended it with new features and tools.

XFC has now been used in two DECT cordless phones and in the RTX Turnkey GSM/GRPS phones.

Extremely versatile

These devices range from platforms such as GSM, GPRS, UMTS, CDMA W-CDMA and TD-SCDMA cellular phones, as well as DECT and 2.4GHz cordless phones. DECT phones typically have CPUs that are four times less powerful than cellular handset CPUs. The applications range from simple black-and-white display DECT phones with a phonebook to sophisticated large-screen, color-display cellular phones with Phonebooks, Calendar, Camera, WAP, Java and MMS applications.

Under these conditions, it is essential that the software can be reused to a high degree and that the needed scalability, productivity goals and quality can be obtained at the same time.

Hardware constraints

Developing reusable applications for handset devices can be quite challenging due to the various hardware and customer MMI look-and-feel requirements.

Hardware CPUs differ widely in respect to performance and thus put constraints on the kind of advanced user interface features that can be implemented. Another hardware issue is the various displays. They vary from black-and-white displays with a resolution of 110*64 pixels, for example, to large color TFT displays with 65000 colors and a resolution of 128*160 pixels or larger.

The software application is the differentiating module – hardware is becoming more and more alike

Customers usually have differing requirements for the user interface look and feel. One of the major keys for success for any product is to differentiate itself from other similar products. As hardware components get more and more standardized, similar and easier to use, it becomes more important that the software applications act as the differentiating factor. However, applications are also becoming more complex; it takes larger teams to implement and test them, and all this has to be accomplished within a shorter time to market. These are some of the challenges that we face when developing software for the wireless and cellular market.

Advanced handsets can be based on one CPU running at a low speed

In the recent years, the two platforms Microsoft Pocket PC and Symbian have been fighting to become the de facto standard for the Smartphone handset market. Choosing such a solution will make the various products very similar due to the common SW platform. These solutions are bundled with a lot of advanced applications such as Phonebook, MMS, Browser (WAP & HTML), Calendar, Media players for video and sound, Word, Excel viewers and so forth. Smartphone solutions all have to be based on a hardware platform which hosts a so-called application processor to speed up the execution of these applications. Examples of such processors are the Motorola DragonBall and Intel XScale running at least 200MHz. The actual phone protocol stack will be running on a so-called Baseband (CPU) processor based on an ARM core running about 50MHz, for example.

However, such solutions are very expensive to produce due to the large number of expensive hardware components, especially application processors, battery, flash and RAM.

This is why it is essential that handset solutions based on one CPU running at relative low speed (50-75MHz) can be exploited in such a way that advanced products still can be made. These solutions will enable cheaper and smaller devices that consume less power while still delivering the user experience that will keep this market the biggest for many years.

Hardware chip producers are adding features such as Java instruction sets to improve performance. But apart from performance, the user experience is also highly dependent on the MMI design and the stability of the product.

A full graphical multitasking windows system

The goals that we set for ourselves ranged from the very technical to goals oriented purely towards the software development process.

Looking at the target systems, we soon realized that a full graphical multitasking window system would help us implement advanced applications and systems. A clear separation of the graphical user interface code and the actual application code is a well-known property of good software, and we wanted direct support for this in the software.

Object-oriented programming language: reuse, quality and shorter time to market

A true component-based and extensible software framework would be needed. This would enable more reuse across various projects. An object-oriented programming language could

give us the direct support for this. By using an object-oriented programming language, we could more easily extend and aggregate components to other components and so forth. The long-term goal of reuse is shorter time to market and higher quality at the same time – two key factors, especially in the handset market.

XFC is based on defined standards

To ease the use and adoption of the software, we based the software on well-known standards and methodologies instead of proprietary solutions. This would mean that the window system should be just like any other window system but simpler and slimmer. Support tools and formats should be based on open standards such as XML (Extensible Markup Language) and XSL (XML Style sheet Language) together with graphical front ends.

No assumptions should be made about any specific RTOS or scheduling scheme (pre-emptive or non-pre-emptive). It could, however, be assumed that some kind of OS would be present and that memory management, a messaging mechanism (usually via a queue) and timers would be available.

Think in terms of interfaces – syntax and semantics are important

Whether one uses object-oriented programming or not, the important issue is to think in terms of interfaces. As long as the interfaces can be described in terms of syntax and semantics, the actual implementation is of no interest except to the ones who actually implement it. By using OO features such as inheritance and polymorphism, we add the possibility of extending interfaces in a consistent and simple way.

Describing systems and interfaces can be done in many ways. A popular notation these days is the use of UML as the descriptive language. UML supports interface description using object-oriented notations. To close the gap between an OO design and the actual implementation, an OOP language is needed as it directly supports design features such as classes, inheritance and polymorphism.

The RTX solution

Object-oriented language is OOC; embedded C++ is not suitable

XFC uses a proprietary object-oriented language implemented in ANSI C - OOC. The reason for not using embedded C++ is that not all of the platforms that we had in mind actually had a C++ compiler. Even if C++ were a possibility, there has been much bad experience with inefficient C++ compiler generated code, which results in bigger code size as well as degraded performance. We needed to be in total control of the OO language. Therefore we chose to describe the class interface in XML and then automatically generate ANSI C code by the use of XSL. When writing a class, the programmer has to write the class description and implement the function algorithms. The C interface file, class virtual table and class function stubs (optional) are automatically generated, typically using the native-make system. The class description also acts as hypertext documentation with embedded documentation tags that can contain free text or HTML. It can be opened in an Internet browser that supports XML and XSL. It provides hyperlinks within the class and between classes.

As OOC is modeled much like C++ both syntactically and semantically, any experienced C++ programmer will be familiar with this environment.

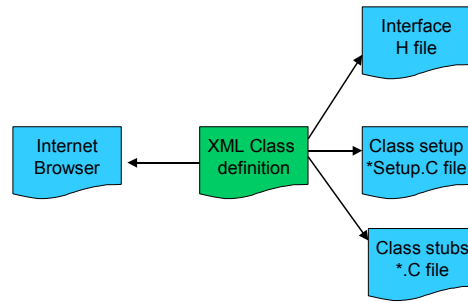


Figure 3 Class specification

XFC software components

XFC consists of a window and telephony framework that builds on XFC core software.

The core software provides APIs to handle general programming tasks related to MMI applications.

XFC core features:

- Unicode or 8 bit text support.
- Full Color support.
- Overlapping windows and window resources.
- Datamodels for data and application logic encapsulation.
- Images (proprietary, JPG, PNG). Easily extended with new formats.
- Flexible hardware key translation through Keymaps.
- Softkey support.
- Date and Time support with Timezone and daylight savings time (DST).
- Events and Filters.
- Flexible Graphics capabilities for displays and bitmaps.
- Fonts.

The window framework provides a general window API and specific window components such as single line and multiline edit fields, dialogs, message boxes, static and animated images, menu control, labels and so forth.

The telephony framework provides a general API for typical handset application logic. These applications are numerous, but some of the central applications can be listed here:

- Call Processing. Handles setting up calls, receiving calls, call waiting, conference and other call services.
- Phonebook. Handles a number of phonebook entries that each typically include name, phone numbers, number types, wallpaper, ring tones, e-mail and URL address.
- SMS, EMS. Handles SMS-related messages.
- Multimedia Messaging Service (MMS). Handles MMS messaging.
- Call history. Collects information about all calls.
- WAP. Provides Internet access and browsing.
- Battery. Provides battery status.

- Alert. Handles all the audio alerting concerned with various events such as incoming calls, messages and errors.
- Settings. Read-and-write access to all handset settings.
- Digital Rights Management (DRM). Content such as messages, games, melodies, pictures, video, Java applications and so forth can be protected by DRM objects. This component handles verification of content rights and content execution.
- Java. Handles application management such as downloads, deletions, executions and much more.

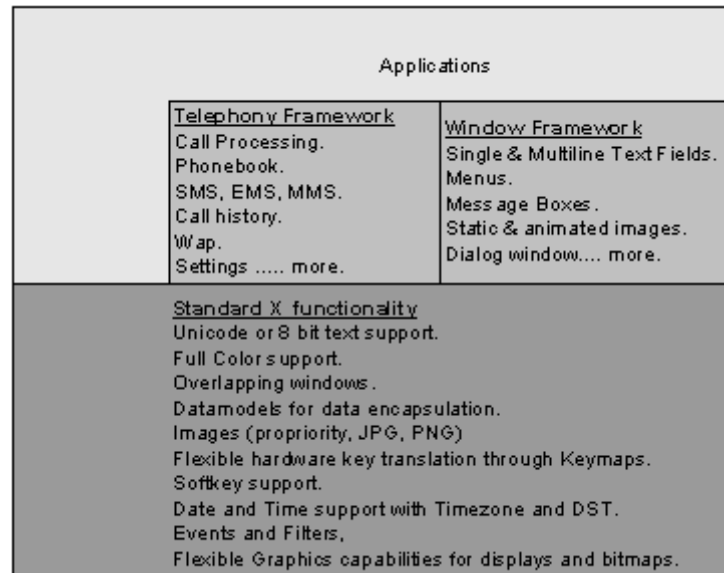


Figure 4 XFC Software components

XFC architecture and design

The architecture of XFC is composed of events, event manager, filters, datamodels, window manager and windows.

Messages are produced by the host system, where a number of tasks represent various subsystems such as the protocol stack (e.g. GSM/GPRS), Java, MMS, WAP and so forth. These messages are sent to the MMI task in which the XFC software is running. Messages are wrapped inside XFC events, which are then passed to the event manager. The event manager will pass the event to all installed filters. The role of the filters is to handle specific events. For example, an event representing a protocol message is sent to the needed datamodels; drawing events are sent to the window that needs redrawing; timeouts are sent to the object that started the timer; and so forth.

Datamodels process messages, collect data and notify windows about state changes. Windows use data from the datamodels and update the display accordingly. The main task of the window manager is to control window redrawing so that this process is done correctly and in the optimal way.

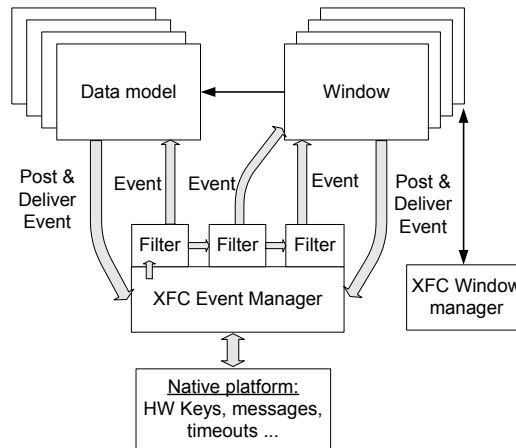


Figure 5 Architecture

Decoupling and little overhead

To facilitate a loose coupling between datamodels and windows, we use the design pattern “Subject Observer.” Briefly put, it provides a mechanism for observers (windows) to attach themselves to datamodels for the purpose of being notified by an event when data change. This mechanism ensures that no “polling” mechanisms are implemented and the overall system has as little overhead as possible.

Figure 6 shows an example of two windows that are attached to a datamodel. They show the same data in two different ways. If one of the three variables in the datamodel changes, the observers are notified one by one and they can update their display accordingly.

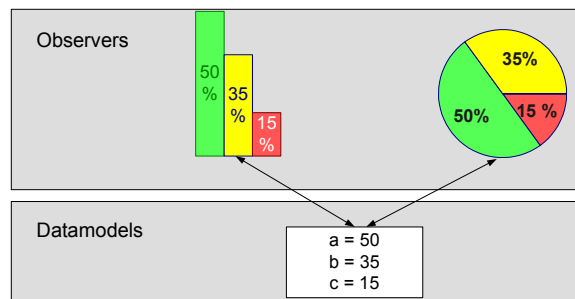


Figure 6 Subject Observer pattern

XFC windows

Windows come in two flavors: those that cannot contain any other windows and those that can. All windows have a parent window. The window system always has a desktop window, which acts as the root for all windows. So windows that do not explicitly specify a parent will be children of the desktop window.

Common properties and behavior for all windows are that they have an outer rectangle and an inner rectangle, which is used as the actual area in which they can draw. By default, these two areas are the same. If the inner rectangle is not the same as the outer, an area appears that is called the decoration area. This area is typically used for creating window frames and dialog titles.

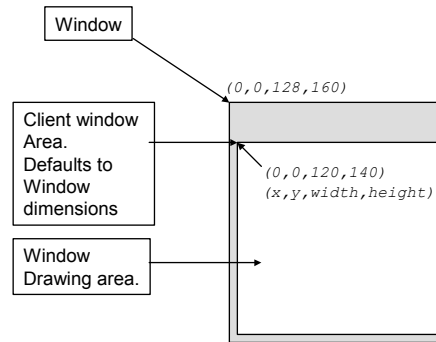


Figure 7 Window properties

Child windows – handled by z-order stack

Container windows can have a number of child windows. The windows are kept in a so-called z-order stack so that the window manager knows which windows are on top of other windows. This information is used for calculating clipping areas in case windows overlap. All graphics operations are clipped according to this information.

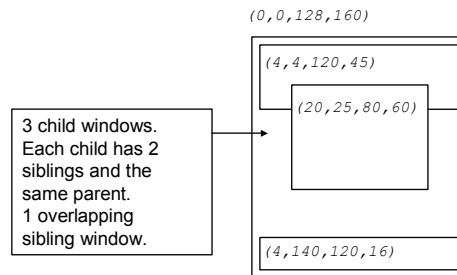


Figure 8 Container window

Capable of transparency

One of the more advanced window features is transparency – the window’s content below can be seen through the window. One can make windows transparent by using a specific window style when creating them. The programmer does not need to do anything specific to create this effect because it is taken care of by the window manager.

Class and window resources – common look and feel

XFC window resources are used for window configuration at compile and/or runtime. Window properties such as position, width, height, font, background/foreground color, keymap, title and so forth can be defined as resources. Resources can be overridden in any subclass, and new ones can be defined. They are automatically and transparently loaded into the window class instance upon creation. This way the look and feel can be configured according to the class – thus ensuring a common look and feel throughout the system where that class is used. At runtime it is possible to override the default class resources selectively if a specific look and feel are needed for various instances of the same class.

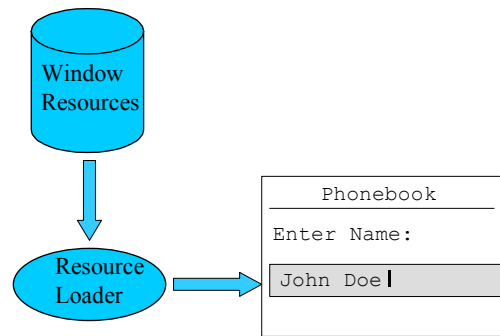


Figure 9 Window Resource loader

XFC telephony

The Telephony Framework is an abstraction of many different system services found in many wireless phones. This framework contains all the application logic and enables MMI applications to be ported across various wireless technologies without changes. Through the use of a traditional Three Tier solution, the user applications are shielded against technology changes. The goal is to be able to run the same MMI on very different protocols such as GSM/GPRS, CDMA, UMTS and TD-SCDMA.

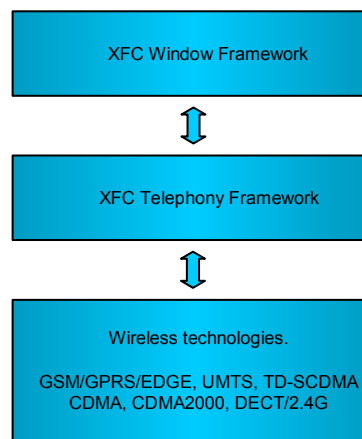


Figure 10 Three Tier solution

Through the use of class inheritance, all common behavior can be expressed in the interface that all the applications use – the Telephony Framework. In many cases this interface is abstract and is implemented in “Wireless Technology” subclasses. Specific cellular technology behavior for GSM or CDMA, for example, can be expressed in inherited classes, and the actual implementation must be done for each “Technology provider,” such as Motorola. MMI applications should use only the Telephony Framework API, but in rare cases they may use interfaces defined at the “Wireless Technology” level.

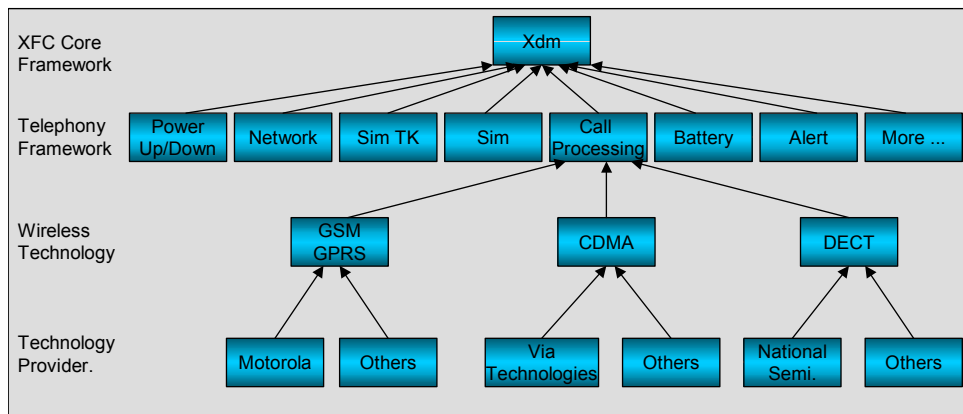


Figure 11 Common API but differing implementation

XFC skins

A skin can define window properties such as background and foreground colors, texts, bitmaps, fonts, positions, dimensions and much more. A skin definition is specific to the various MMI elements. For example, the standby screen of a typical phone includes MMI elements for showing the battery level, signal strength, new message indication, service provider name and much more. Each of these elements will have its own skin properties for specifying position, dimension, graphics, text, colors and more. For example, the battery element and signal strength indicator can have a number of images for showing the levels.

All applications will be defined in terms of various elements, and the handset implementation will have an interpreter for each element type in order to implement the correct behavior and look and feel for it.

Use skins to shorten development time

The reasons for the skin concept are many. First of all, it is important to shorten time to market for various solutions. Therefore coding everything by hand is not a long-term solution if it is basically just another skin that is being created. Thus developers need to have ways of easily creating variants of the same phone. Another purpose is providing the user with new user experiences – to be able to change the appearance dramatically on the fly by selecting predefined skins in a menu or by downloading them from a network provider.

Contact RTX to get more information on our MMI framework: sales@rtx.dk