AN APPROACH TO A COMPACT JTRS SCA CORE FRAMEWORK FOR HANDHELD RADIOS

Dr. John W. Cruz, Tony Davis,
Michael Mario, Gerald Rolon

ITT Industries Aerospace/Communications
Engineering Center
100 Kingsland Road
Clifton, New Jersey

Abstract

The JTRS Core Framework (CF) based on the Secure Communications Architecture Specification (SCA, currently in version 2.2) is a key enabler for the Software Defined Radio (SDR). By defining a set of common interfaces and standardized waveform control services the portability of radio software across multiple platforms is achievable. Portability and reuse are two of the key payoffs of the SDR concept implemented in JTRS. The handheld radio domain with its severe power, size and memory limitations present a challenge to the SDR developer who wants to gain the benefits of the CF common services and standard interfaces. The memory, storage and potential processing requirements of the CORBA ORB and services, an operating system robust enough to host CORBA, the CF software and the waveform itself must be addressed to create an effective solution in the handheld radio domain.

Introduction

The practical adoption of the SDR approach based on the Core Framework, shown in Figure 1, is potentially limited in handheld or smaller radios by the memory overhead required for the implementation of the complete CF. Typical designs provide 32 to 64 M per GPP. Allocation of this memory between OS, CORBA, the C++ library and waveform applications imposes a memory ceiling on the Core Framework. Sizing issues for other infrastructure components of the SDR (OS and ORB) are addressed through technology evolution of commercially available products. Optimization of executable sizes under Linux is addressed quite extensively in the literature. The approach to reduce the size of the common CF itself is largely in the hands of the SDR community. We cover some of the approaches considered and implemented in developing a Lightweight Core Framework targeted for the handheld SDR environment.

Core Framework Software Architecture Overview

Ease of technology insertion, Software Reuse and waveform software porting are feasible if the software architecture hides low level system details. The Software Communications Architecture, SCA[1], was created by radio developers under contract to the US government Joint Tactical Radio System (JTRS) Program Office (JPO). The SCA hides low level hardware and software details via:

- Encapsulation of hardware dependencies exclusively in SCA *Device* software which presents common interfaces defined in IDL (the CORBA Interface Design Language)
- The use of CORBA to hides details of the architecture, particularly the number and type of processors their operating systems and communication mechanisms
- A standard mechanism to describe system and application configuration, the SCA Domain Profile.

Core Framework Components

As defined in SCA 2.2 [1] , the Core Framework is a software architectural concept defining the essential set of open Interfaces that provide for the deployment, management, interconnection and intercommunication of software application components in embedded, distributed-computing communication systems. The Core Framework components are shown in Figure 1 labeled as the CF Services and Applications, a subset of which execute on each CORBA capable processor in the system.

The primary software components of the SCA 2.2 Core Framework that are candidates for optimization are:

- Operating Environment
    - POSIX Operating System (COTS)
    - CORBA Middleware (COTS)
    - CORBA Services
        - Naming Service (COTS/custom)
        - Event Service (COTS)
- Core Framework
    - Base Application Interfaces (App)
        - Port, Lifecycle, TestableObject, PortSupplier, PropertySet, Resource, Resource Factory
    - Framework Control Interfaces (App/CF)
        - Application, ApplicationFactory, DomainManager, Device, LoadableDevice, ExecutableDevice, AggregateDevice, DeviceManager
    - Framework Services Interfaces (CF)
        - File, FileSystem, FileManager, Timer

ITT has developed a Core Framework and has tested it extensively on Windows, x86 Linux and ARM Linux and has used this as a baseline for developing the Lightweight Core Framwork described in this paper.

Certain of the CF components shown above are potential candidates for size optimization to produce a Lightweight Core Framework. ITT chose the Linux OS for its customization potential and ORB Express for its size and real-time performance on the target processors. In a typical implementation of the CF, the Base Application Interfaces are uniquely implemented as part of the application thus do not contribute to the CF size while most of the CF operational software is part of the Framework Control Interfaces and Framework Services Interfaces. These areas are the focus of this investigation.

The ITT Lightweight Core Framework (LCF) is implemented as a set of dynamically loaded shared libraries, *.DLL* in Windows or *.so* files in UNIX. This approach avoids the duplication of memory images and also permits selection of the CF components as appropriate to each processor, minimizing memory use.

Approaches to Lightweight Core Framework Size Optimization

There are at least two conceptual approaches to reducing the size of the CF:
- **Elimination of Functionality**
- **Optimization of Components.**

Elimination of Functionality. The primary advantage of this approach is that it can result in substantial reductions in the memory footprint with the severe disadvantage that the result may be incompatible with the complete Core Framework and thus defeat the interoperability goal of the CF.

Optimization of Components. Conversely, the second approach may be stingier in providing a footprint reduction but preserves application portability among all CFs. We believe it is feasible to follow this second approach in creating a Lightweight Core Framework since preserving portability between both handheld and larger SDR platforms is a critical goal of JTRS. Near term technology developments will also facilitate development of smaller handheld radios with sufficient memory to permit the utilization of a complete interoperable core frameworks.

In order to select the most productive approach to CF footprint reduction, an initial measurement of the CF sizing was obtained and is shown in Table 1. The compiler optimizations used were -strip-debug and -strip-unneeded flags. The size optimization flag -Os were found to have a neglibible impact with the GNU x86 and ARM compilers used (<0.01%) and was not used. Note that the file size is a very close approximation of code size when symbols are stripped using the above compiler flags. An excellent overview of optimization in a Linux environment is presented in Reference 3.

XML Parser. Table 1 shows that the software footprint related to XML parsing is quite large, the two libraries (libxmllib.so and libxerces-c1_5_1.so) consume about 42 % of the memory required for the ARM Core Framework. Basically libxerces parses the XML Domain Profile and libxmllib allows CF components to extract the parsed information. Xerces is a widely used open source validating parser. Identification of a memory efficient approach to processing the XML Domain Profile has high payoff for the LCF.

Log Service. The 153 k footprint of the Log Service, optional in SCA 2.2, is quite small, thus omitting this optional component does not have high payoff.

Naming Service. The naming service shown in the Table is a fully functional custom lightweight implementation developed by ITT specifically for optimized performance with the smaller number of registered objects typical of an embedded Radio. It is fully compliant with the OMG specification. Its small size (600k) represents a substantial footprint saving over COTS naming services that can be up to several megabytes in size.

FileSystem and FileManager. The footprint of these two services is small, only about 2% of the total. Therefore, application use of the native file system will not contribute significantly to the CF footprint and would severely degrade application portability.

Selection of High Payoff CF Components for Optimization

Table 2 rates the approaches to developing a Lightweight Core Framework based on the initial size measurements. High payoff approaches are those which result in a substantial footprint reduction and are relatively easy to implement. All High payoff approaches were also required to maintain compatibility between LCF and complete CF functionality.

Compiler Optimization Flags

This obvious approach by itself resulted in a modest footprint reduction.

Naming Service

Implementation of naming service is straightforward and offers the reduction in footprint as shown, to about 600 k on the ARM and. The implementation is fully compliant with the OMG CORBA Naming Service Specification[3].

XML Parsing

Alternatives were examined to parse the Domain Profile offline or online. Offline parsing may offer more potential size reduction, but requires definition of an interface between offline parsing and online processing of the information. ITT had available a small XML parser which is non-validating. Taking this approach, validation is either done when XML is written using a tool like- xmlspy®[4], or in offline operation with a version of the radio built with a validating XML parser. In either case, XML introduced into the SDR must be pre-validated. XML files themselves are small compared to the executable sizes.

Future LCF Developments

Currently under investigation is the difference in the size of CF components between the ARM and x86 implementations. The impact of the C++ STL on the size of our LCF implementation another one of areas we wish to investigate further.

Conclusions

By optimizing some of the largest components including the NamingService and the XML Parser for the Domain Profile, we have developed and tested a Lightweight Core Framework compliant with SCA 2.2 with a total footprint under 10 MBytes including ORB on the ARM platform. The size of the LCF on the x86 platform is under 4 Mbytes. This memory footprint is and compatible with the memory availability in a handheld radio. The availability of a lightweight CF is a key enabler to achieve the JTRS goal of waveform portability between platforms with different memory resources, a key to the successful SDR.

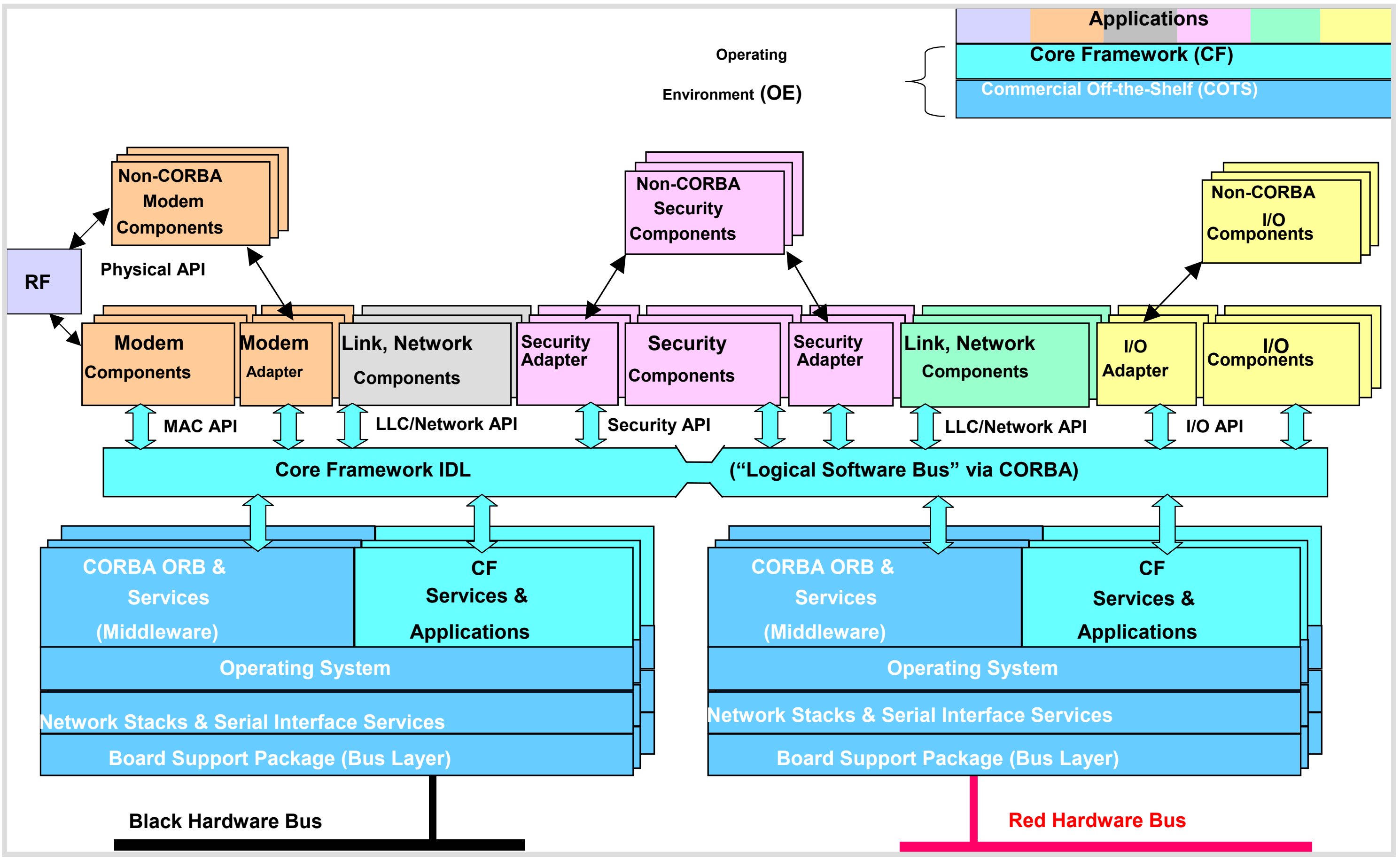
References

1. *Software Communications Architecture Specification, MSRC-5000SCA, V2.2 17 November 2001*
2. *Optimizaing Embedded Linux, Todd Fischer, Dr. Dobbs Journal, May, 2002*
3. *OMG Naming Service Specification, V2.2, see for this specificaiton. http://www.omg.org/technology/documents/formal/naming_service.htm*
4. *See http://www.xmlspy.com/ for information about this tool suite.*

Component	Linux x86 -strip -debug (in kbytes)	Linux x86 -strip -unneeded (in kbytes)	Linux ARM -strip -debug (in kbytes)	Linux ARM -strip -unneeded (in kbytes)	Notes
Gppdevice	213	166	393	354	
Libcfstubs.so	881	690	2864	2687	Output of IDL compiler for CF
Libdevicemanager.so	181	132	294	240	CF Device Manager
Libdomainmanager.so	462	356	678	585	CF Domain Manager
Libfilemanager.so	104	73	131	99	CF File Manager
Libfilesystem.so	115	90	154	131	CF File System
liblog.so	157	123	193	162	CF Log Service
libutils.so	267	213	314	271	Implementation specific utilities
libxmllib.so	1138	954	1420	1298	Interface to parser
Subtotal, Base CF (kbytes)	3518	2797	6441	5827	
XML Parsers					
libxerces-c1_5_1.so	2463	2331	2721	2445	Open source XERCES parser for Domain Profile
Libparserlite	140	124	953	953	Lightweight Parser
ORB					
libOEorb.so	580	580			ORB Express RT_2.3.5 config_rt_shared
libOEorb.so	-	-	2735	2527	ORB Express RT_2.5.0_ESC config_rt_full_shared
libOEorb.so	-	-	832	832	ORB Express RT_2.5.0_ESC config_rt_fast_shared
libOEtcp.so	154	154	90	75	
Naming Service					
Libcfnamingservice.so	179	179	445	399	Lightweight Naming Service
oenames_server	4076	3452	-	-	No available on ARM at the time
Totals					
"Standard" CF	10791	9314	12432	11273	Base CF, Full Parser, Naming Service, config_rt_full_shared
Lightweight CF	4571	3834	8761	8086	Base CF, Lightweight Parser, Lightweight Naming Service, config_rt_fast_shared

Table 1. Size of Core Framework Components

Figure 1 JTRS Core Framework



Applications
Core Framework (CF)
Commercial Off-the-Shelf (COTS)
Operating Environment (OE)
Non-CORBA Modem Components
Non-CORBA Security Components
Non-CORBA I/O Components
RF
Physical API
Modem Components
Modem Adapter
Link, Network Components
Security Adapter
Security Components
Security Adapter
Link, Network Components
I/O Adapter
I/O Components
MAC API
LLC/Network API
Security API
LLC/Network API
I/O API
Core Framework IDL
("Logical Software Bus" via CORBA)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
Red Hardware Bus

Approach	Expected Impact	Complexit y	Measured Impact	Notes
Lightweight Naming Service	**High**	**Low**	**4 M / 0.2 M**	**X86, only Naming Service not available on ARM**
XML Parsing				
Offline Parsing	High	Moderate	N/A	
Online Parsing				
Custom Parser	**High**	**Moderate**	**2.3 M / 0.12 M**	**X86**
			2.4 M / 0.95 M	**ARM**
COTS Parser	Moderate	Low		
Code Optimization Strategies				
Global Optimization	**High**	**Low**	**7 M / 5.5 M**	**X86 Compile flags**
			14 M / 12 M	**ARM Compile flags**
Module Optimization	Moderate	High		Although it does not appear to have high impact, this strategy should be part of maturation of the LCF
Unload components after startup	Moderate	Moderate	N/A	This dynamic strategy is easily made part of the waveform startup sequence.
Eliminate Optional CF Components	Low	Moderate	N/A	The size contribution of optional components is not a major contributor to the CF size.
Use Native File System	Low	Moderate	N/A	Very Undesireble Introduces serious Compatibility Problems
Eliminate Optional Features within CF Components	Low	Moderate	N/A	The size contribution of optional features is not a major contributor to the CF size.
Eliminate unneeded interfaces	Low	High	N/A	It is not possible to determine a priori which interfaces are not required, thus undesirable compatibility problems are introduced.
Notes: Impact: **Low**: < 100k **Moderate**: > 100k , <1M **High**: > 1M Complexity: **Low**: limited development **Moderate**: some development **High:** extensive analysis/SW development				

Table 2 Lightweight Core Framework Tradeoffs