Embedded Systems and SDR

Max Robert
Philip Balister
Jeffrey H. Reed

Wireless @ Virginia Tech
Mobile and Portable Radio Research Group (MPRG)
max.robert@gmail.com
philip@balister.com
Introduction

- SDR at different levels of reconfiguration has existed for several years
- Highly reconfigurable radio systems do exist
  - Generally limited to high-power systems
    - Base stations
    - PC-based prototyping systems
- Overlap between embedded systems and SDR of interest
SDR Architectures

- Provide infrastructure for the management of radio systems
  - Component reuse
  - Multi-mode support
  - Adaptive waveforms
- SCA selected as target architecture
  - Generally well defined
  - Significant industry-wide levels of activity
Presentation Layout

- Overview of SCA
- OSSIE
- SCA implementation on OMAP
- Analysis of SCA
The SCA

Overview of the Specifications
Overview

- Introduction
- The role of the SCA
- SCA Classes
- Configuration
- SCA API
Introduction

- SDR can be defined as one whose functionality is significantly implemented in software
- More than just a radio with some software
  - Generally extended to include additional features
    - Reusable software
    - Contains structure for the integration of disparate technologies from different developers
Contrast in Software Approaches

- **Traditional software for mobile radios**
  - Embedded software
    - C, assembly, VHDL
  - Rudimentary or no operating system
  - Relatively narrow scope of functionality

- **Software Defined Radio (SDR) extends this scope**
  - Broader functionality set
  - Reusable code more valuable
Software for SDR

- Still relies heavily on more traditional approaches
  - Will still need to carefully manage cycle count in DSP
  - Will still rely on tight timing tolerances for FPGAs
- Software for SDR does not replace traditional approaches
  - Instead, it extends them into broader scope
Partitioning Software

- Can be largely split into two categories
  - Functional software
    - All the software that performs the work
      - Filters, FEC, framing, other
    - Blends with the traditional development strategies
  - Management software
    - Control software and hardware
    - Is the principal concern of the SCA
Military SDR Needs

- DoD faced with operational and logistical challenges
  - Need to support advanced services
    - Every component on the battlefield has an IP address
  - Need to simplify procurement and maintenance
    - Reduce the number of disparate systems that need to be supported
JTRS

- To resolve issues, DoD (through JPO) established the Joint Tactical Radio System (JTRS) program
  - JTRS is “designed to provide a flexible new approach to meet diverse warfighter communications needs through software programmable radio technology”

- At the core of the JTRS program are the Software Communications Architecture (SCA) specifications
SCA

- Provides a framework for the establishment, maintenance, and tear-down of waveforms
  - SCA provides the “glue” that holds a radio system together
- Conceptually, SCA is fundamentally simple
  - Complexity arises from large number of details
Operating System

- SCA assumes that there is an underlying operating system (OS)
  - OS interface is POSIX compliant
    - PSE-52
- Waveform will need limited access to OS
  - Largely confined to process management
    - Threads
    - Processes
CORBA

- SCA requires use of CORBA in variety of pieces
- Common Object Request Broker Architecture
- Powerful set of capabilities
  - Allows two or more different programs in two different computers to exchange information and perform calls as if the two programs were co-located
  - The different programs can be written in different languages and be running in different operating systems
CORBA Capabilities

- Very powerful software
  - Used in several IT solutions
  - Similar capabilities to Jini and .NET
- Biggest issue is size
  - CORBA can take up significant amounts of memory
- Well suited for operating in an IP environment
  - Provide transparency between two systems connected by an IP network
CORBA in SDR

- Principal SDR benefits
  - Connectivity over multi-address systems
  - Support for components written in different languages
  - Support for components running over different operating systems
- SDR system must require all above characteristics for use of CORBA to be worthwhile
The Core Framework Classes

- Conceptually, software layout is fairly simple
  - Process for management of data flow similar to that used in PC today
- Details for management of waveform go beyond basic concepts
  - Significant infrastructure needed to perform basic waveform management
    - Replicate most functionality of OS in a generic fashion that maintains compatibility
  - Infrastructure is the class structure defined in the SCA
Classes

Legend:
- Implemented as Core Application Services
- Implemented as Non-Core Application Services
- Core Framework Interface

Base Components

Framework Control
- 0..*
  - AggregateDevice
  - LoadableDevice
  - ExecutableDevice
  - DeviceManager

Framework Services
- 0..1
  - File
  - FileSystem
  - FileManager

Modem, Repeater, RF, Waveform, Black, Utility, Router, Network, Bridge, Link, Security, Internet, Utility, Router, Network, Bridge, Link, HCI
Core Framework (CF)
Resources

● All components are derived from the Resource parent class
  ● All other classes intended to support these Resources

● A component is made up of one or more Resources
  ● For simple applications, there is only one Resource per component
Component/Resource Mapping

- A Component is made up of at least one Resource
- Component is defined by its interfaces and functionality
  - Particular content of Component largely irrelevant
    - Allows for Component scalability
Component Scalability

- Flexible nature of Component allows creation of waveform of waveforms
Component Needs

- Needs of Components can be largely parceled into four basic categories
  - Create or destroy Component
  - Connect to other Components
  - Configure Component
  - Test Component

- Beyond these basic needs is functionality
  - Waveform specific rather than intrinsic aspect of Component
Create or Destroy Component

- Creation/destruction supported by LifeCycle parent class
  - Two member methods
    - initialize()
      - Perform work needed to bring Resource to well-defined state for functioning
        - Component-specific functionality
    - releaseObject()
      - Bring Resource to well-defined state for termination
  - Not the same as the constructor or destructor
    - LifeCycle calls are handles for the framework to manage object's lifetime
Connect to other Components

- Connection to other Components performed through the PortSupplier parent class
  - Single member method
    - getPort
      - Provides a handle to the Resource’s Port(s)
  - Does not connect to other Resources
    - This set of methods provide the structure to exchange the CORBA pointers
    - Connection call performed on the actual Port
Configure Component

- Configuration supported through the PropertySet parent class
  - Manages unique properties of Resource
  - Four member methods
    - configure()
      - Set a Resource’s property value(s)
    - query()
      - Get a vector of properties from Resource
    - validate()
      - Make sure that a particular property exists
    - getPropertyValue()
      - Get a specific property’s value
Test Component

- Parent class TestableObject provides a standardized interface to perform Resource testing
  - Single member method
    - runTest()
      - Component-specific functionality
  - Note that TestableObject is an inherent part of the architecture
    - Deployed software can easily include self-test software
Abstractions and Reality

- Abstractions serve well for the creation of mental models, even roadmaps.
- However, to solidify concepts, it is important to include concrete examples.
  - This is the last part of the tutorial and is split into three sections:
    - How to build a node booter and a waveform
    - OSSIE
    - Modifying waveforms in Linux (practical)
Implementing Systems

- Classes and components were presented in this tutorial in stand-alone form
  - In reality, a shell is needed for these program constructs
  - In C++, this shell is a separate thread of execution
    - This thread can be either a separate executable or a thread of execution within a single executable
      - For the sake of clarity, the shell will be a separate executable
    - Each piece (node booter or component) has the structure seen in the following slide
Sample Shell Around Component

#include "DecodeData.h" // this header includes the prototype and other relevant headers

int main( int argc, char** argv )
{
    // create relevant object
    DecodeData_i *decoder;
    decoder=new DecodeData_i;
    decoder->start_running(); // this is not part of the SCA
    // it is a flag to exit the do while loop

    ACE_Time_Value orb_time_out = 1; // timeout is included to allow non-CORBA events

    do
    {
        ORB_WRAP::orb->perform_work(orb_time_out); // service CORBA requests
    } while( decoder->is_running() );

    delete decoder;
    return 0;
}
Component Code

- Beyond the shell, the component needs to inherit from CF’s Resource, so the declaration of the class should include

```c++
class DecodeData_i :
    public virtual POA_OSSIE::DecodeData,
    public Resource_impl
{
   ...
}
```

- Note the inheritance from POA_OSSIE::DecodeData – that denotes the inheritance from the IDL-generated code
  
  All the interfaces declared in the IDL are generated as virtual and hence must be populated

- Also, the component will need some code that will change little between implementations
  
  Constructor
  Destructor
Additional Modifications

- Some minor modifications to the component are also necessary for basic housekeeping
  - Start/stop
    - Start and Stop commands are received from the Assembly Controller and are used to start or stop the chain of CORBA events
      - The location and approach to modifying these methods are application-specific
  - Release Object
    - This is the termination call, which requires the program to terminate the main() do-while loop
      - In this example, the value of currently_running would be modified
Final Modifications to Component

- Beyond these generic modifications, the only aspect left is functionality.
- A couple of key aspects to remember:
  - You will probably have to service more than just CORBA events:
    - i.e.: A/D buffer full, user input
    - If following a structure like that shown here, the equivalent of `perform_work` in the do-while loop of `main()` should not be blocking.
  - You will have to extend the Resource class to perform additional functionality:
    - Three files need to be modified: the C++ file with the functional code, the header file with the prototypes, and the IDL file describing the interfaces:
      - You must include the IDL-generated files in the project or makefile.
  - Editing your own XML by hand using a simple editor is error-prone – most mistakes will be derived from this step.
Port

- Port is “used” by Resource
  - Resource will most likely create an instance of a Port within the Resource
- Port contains two member methods
  - connectPort()
    - This is the method that narrows the reference to the object to which data is pushed
  - disconnectPort()
    - Remove CORBA reference
  - Note that Port is a component-specific implementation
Resource Factory

- This is a class that is not necessarily used
  - Performs supporting functionality
- Its task is to create Resources
  - Single member method
    - `createResource()`
      - Can be used to deploy components with little or no knowledge about the host hardware
      - A more traditional way of creating Resources is to request the hardware that hosts the running code to spawn the Resource
        - More details on this later
Deployment of a waveform has two steps
- Create or assemble the waveform
  - Make sure all the pieces are available (or create them when not)
  - Connect all the pieces
- Start the waveform

The ApplicationFactory class is used to assemble a waveform
- Single member method
  - create()
    - Deceptively complex call
Application Factory

- Class that is used to create the waveform
  - Reads XML
  - Makes sure that required HW can run needed Resources
    - Executes waveform software on required hardware (not proxy)
  - Creates whatever Resources are needed (that have not already been created)
  - Connects all Ports
  - Informs any objects listening that it has successfully created waveform
  - Return Application object
Application

- An instance of this class is returned by the Application Factory after successful creation of the waveform
- This is the handle to the waveform
  - Contains information on the waveform
    - Application-specific information
  - Allows the user to terminate the waveform
    - Waveform termination is relatively complex
      - Reverse process of create() method in ApplicationFactory
Application Usage

- Provides convenient entry point into waveform
  - Single handle for runtime commands
    - Start and stop
  - Single handle for releaseObject
    - Waveform-level termination
      - Calls all dependent releaseObject methods in associated components
      - This is an example of where releaseObject is more than just a destructor
File System/File Manager/File

- File is the basic storage entity
  - Just like in an OS
- A File Manager is a File System of File Systems
  - Provides a scalable structure for the creation of a unified file system
- A File System is generally limited to one piece of hardware
  - i.e.: A hard drive, a block of memory
  - The File Manager can then be used to create an amalgamation of these File Systems
File System Usage

- A File System is generally associated with a single piece of hardware or single operating system
  - Provides common interface for the handling of storage on that specific medium
- Hybrid hardware best served through separate File Systems
  - All File Systems then collected onto a single File Manager
    - File Manager behaves as File System of File Systems
  - Single interface and structure for all storage
Domain Manager

- The central core of the SCA system
  - Maintains a comprehensive list
    - All possible waveforms
    - All associated HW
      - All capabilities of the associated HW
    - All installed waveforms
  - Maintains a File Manager
    - The root of the system-wide File System
  - Significant number of member methods and attributes
Domain Manager Interface

- Three general categories
  - Human Computer Interface (HCI)
    - Configure domain
    - Get domain capabilities
    - Initiate maintenance
  - Registration
    - Register/unregister Device Managers, Devices, Services, and Applications
  - CF administration
    - Access interfaces of registered Devices Managers and File Managers
User Interaction

- Ironically, Domain Manager does not control interfacing with the user
  - Instead, provides methods for an external user interface to call

- Flexibility well suited for deployment concept
  - Variety of usage models
    - Handheld, embedded, aircraft, ships, ground vehicles
      - Single user interface would not be appropriate
Starting Domain Manager

- Establishes all necessary part of the system
  - Naming context
    - For associating CORBA object references to a human readable string
  - File Manager
    - Central mounting point for all associated File Systems
  - Event Channels
    - To keep track of changes and inform different pieces of system updates
Hardware

- Hardware can be subdivided into three types
  - Hardware that may or may not be configured
    - NCO, filter, mixer
  - Hardware onto which an image can be loaded
    - FPGA
  - Hardware that can execute programs
    - DSP, GPP
- SCA includes family of classes to manage and control hardware
  - Structure mimics the structure described above
Hardware that may or may not be configured

- From a control standpoint, this hardware can only be configured (partially or in full) for specific use
  - NCO set to specific value
  - Filter tap values
  - A/D converter allocated for a waveform
- Device class used to support this functionality
Device Class

- Inherits from Resource class
  - Can be considered a component
    - Conceptually blur the line between hardware and software

- Two member methods
  - allocateCapacity()
    - Set aside a specific HW capacity
      - Used by ApplicationFactory in create()
  - deallocateCapacity()
    - Make specific HW capacity available for use
Hardware onto which an image can be loaded

- From a control standpoint, this type of hardware extends the needs of the Device class
  - Still has to allocate and deallocate capacity
  - Extension relates to its ability to load and unload images
- LoadableDevice class supports this functionality
LoadableDevice Class

- Inherits from Device Class
  - Leverages all capabilities associated with Device
- Three member methods
  - load()
    - Load a specific file onto HW
  - unload()
    - Unload a specific file from HW
  - isFileLoaded()
    - Check to see if a specific file has been loaded onto the HW
Hardware that can execute programs

- Extends LoadableDevice needs and capabilities
  - Must be able to execute loaded images
- ExecutableDevice supports this functionality
  - Two member methods
    - `execute()`
      - Run a specific program (returns a PID)
    - `terminate()`
      - Terminate a specific PID
Beyond simple hardware

- Aggregate Device is a class that represents a collection of Devices
  - i.e.: a board with many cores that the developer wants to treat like a single Device
- Can simplify the deployment of hardware
  - Reduce the relative complexity of the configuration software
DeviceManager

- Its basic role is to manage Devices
- However, role is more subtle
  - Creates and maintains a File System
    - Platform-specific storage
  - Installs and maintain associated hardware
- Once DeviceManager has processed the associated devices appropriately, it informs the DomainManager that its work is complete
DeviceManager and DomainManager

- DeviceManager mounts its File System (or File Manager) onto the DomainManager File Manager
- DeviceManager associates all of its Devices with the DomainManager
- User has access to HW under DeviceManager control through the DomainManager
Class Overview

- The classes defined so far are the extent of the SCA specifications for classes
  - Several details concerning functionality are not covered
- However, several characteristics become apparent
  - The classes support:
    - Integration of external hardware
    - Machine state information
    - Create/destroy applications from user input
    - Integral file system
  - The SCA classes basically supports all the functionality that an OS does with the exception of task and thread management
There is a special component whose sole job is to provide a start/stop point of entry into the Waveform, the Assembly Controller. This is just a Resource that knows which components it needs to tell to start and stop.

One of the XML files (_SAD.xml) describes the Assembly Controller:
- XML files are described in a later section.
Node Booters

- Class behavior has been described in fairly abstract ways
  - Classes interacting with other classes or installing different HW or SW
- In concrete terms, each of these classes resides in some executable code
  - Binaries that the operating system or local machine can run
  - These binaries are Node Booters
Configuration

- Classes describe the reaction to an event
  - Boot-up
  - User input
  - Error
- Information concerning the reaction to events is implied but not necessarily explicitly described
  - All configuration information in the system is contained within a series of XML (eXtensible Markup Language) files
XML

- eXtensible Markup Language
- Well-defined language for the description of information
  - Tag system used for defining values and properties
- XML used by SCA to describe waveforms, components, and overall system structure
XML Files in the SCA

- Provide variety of information
  - Description of the layout of the system
    - Necessary to bootup the appropriate HW and services
  - Description of the waveforms
    - Components making up Applications
    - Component placement
    - Component connections
    - Names of appropriate pieces
    - Location for relevant files
XML Relationships

- Different XML files describe different types of framework pieces
XML Usage

- Relationships between XML can be abstract and difficult to follow
- A more concrete way to look at it is with an example
  - Domain management configuration file
    - This example can be extended to the different types of configuration files
The Domain Manager needs configuration files.

There are generally found in the root directory of the File Manager.

Four files are found, namely:

- DomainManager_DMD.xml
  - This file will contain information about the Domain and a link to the software descriptor, _SPD.xml
Domain Manager Configuration (2/2)

- DomainManager_SPD.xml
  - Contains information about the software
    - Compiler, operating system, target processor
  - Also contains a pointer to the software component descriptor (_SCD.xml) and the property descriptor (_PRF.xml)

- DomainManager_SCD.xml
  - Contains a list of the interfaces making up the component

- DomainManager_PRF.xml
  - Contains the information that populates Property Set
    - Recall that Resource, Domain Manager, and Device Manager all inherit from Property Set
XML Pattern

- The same general pattern applies to all pieces
  - A software package descriptor (_SPD.xml) describes aspects of the piece of software and will hold pointers to the software component descriptor (_SCD.xml) and property descriptor (_PRF.xml)
XML Starting Points

- In the starting point resides the uniqueness of the different configuration files
  - The starting point for the Domain Manager is the Domain Manager Descriptor (_DMD.xml), as seen in the example
    - The DMD points to a SPD
  - The starting point for the Device Manager is the Device Manager Configuration Descriptor (_DCD.xml)
    - The DCD points to both an SPD and a DPD
      - The DPD has information on the general class of hardware
  - The starting point for a waveform is the Software Assembly Descriptor (_SAD.xml)
    - The SAD points to the SPD for all the components making up the waveform and all the connections necessary for the waveform
XML Ramifications

- Use of open standard allows for integration of pieces from different manufacturers
- Static nature of files limits dynamic behavior of radio system
  - Deployment structure and radio composition can be decoupled
- Single XML profile can be used to describe multiple deployment structures of the same basic radio
Interface Description

- CORBA, IDL, and HAL cover (most) of the mechanics of connecting components
- Technology falls short of establishing the semantics for the connection
  - Common naming guarantees that the correct method (function) is called on the relevant component
  - Increase portability
- API (Application Programming Interface) needed to achieve this level of portability
SCA API

- Multiple APIs specified
  - Generic Packet
  - Physical Real Time
  - Physical Non-Real Time
  - Medium Access Control (MAC)
  - Logical Link Control (LLC)
  - I/O
  - Network
  - Security

- APIs defined as building blocks
  - Allows the APIs to be scalable
    - Reduce the likelihood that anyone “re-invents the wheel”
SCA API Overview

A Data and Real-time Control

B Non-real-time Control, Setup and Initialization, from applications, other levels, user interface

Source: SCA 2.2 specifications
SCA API Boundary

- Interfaces described at the layer level
  - Granularity of components limited to layer
    - Finer granularity requires additional API

- Ramifications
  - Each deployed component is likely to be a full layer
    - MAC, LLC, PHY
  - Reuse capabilities are somewhat limited
    - API does not provide the ability to switch specific functional blocks like mixers, filters, or other low-level functional equipment
API Limitations

- Not only limitation at the small level
- API missing system-wide control structures
  - No wide-scale waveform control
  - No metadata
    - For the contextualization of data
      - Beyond data types
  - No power management interfaces
Sample API

- An example of BB for the API is “pushPacket”
  - Part of the Generic Packet API
  - Is the standard way for information to be exchanged between components
  - Attributes
    - maxPayloadSize
    - minPayloadSize
    - numOfPriorityQueues
  - Methods
    - pushPacket()
    - spaceAvailable()
    - enableFlowControlSignals()
    - enableEmptySignal()
    - setNumOfPriorityQueues()

- The API specifications are full of such building blocks
Security

- Radio system is split into a red and a black piece
- Beyond separation, there are three general security pieces
  - Limits on specific functionality for different parts of the specifications
  - Security interfaces (API)
    - Attempt to standardize the way to access security features
      - No attempt to standardize the security features themselves
        - i.e.: encryption, authentication
  - Functional specification
Putting It All Together

- Overview covered:
  - Components
  - HW placement
  - Link management
  - Configuration of system and waveforms
  - Core framework
  - API
  - Security

- Sufficient background to understand most abused image in JTRS program
Framework Structure
OSSIE

- Open Source SCA Implementation::Embedded (OSSIE)
  - Written in C++ for Linux
    - Has been tested on OS-X
    - May be used on Windows
  - Using open source supporting software
    - CORBA (omniORB)
    - XML Parser (Xerces)
  - First version released in July 2004
    - Available at: http://ossie.mprg.org
  - Development by team composed largely of volunteers
    - Undergraduate and graduate students
  - Designed to be simple yet functional

Available at: http://ossie.mprg.org
Development by team composed largely of volunteers
Undergraduate and graduate students
Designed to be simple yet functional
OSSIE Ramifications

- Initially designed for low-power research under IC Postdoctoral Research Fellowship
- Has been very well received by SDR community
  - Thousands of downloads to date
    - Downloads by government, industry, and academia
    - Released in July 2004
  - 15+ articles written about it in trade publications
    - Including front page in EE Times
  - Being evaluated by different entities for active use
  - Significantly lowers barrier to entry
- Has served as a catalyst for other SDR research
  - Rapid prototyping
  - Cognitive radio
  - Distributed signal processing
  - Integrated testing
Analysis of SCA
Analysis of SCA

- Specifications contain platform-specific aspects
  - Some are easy to override
  - Some are integral to the specifications
  - Two platform-specific aspects are key
    - CORBA
    - XML

- Specifications also contain platform-independent aspects
  - Drive the inherent behavior of the framework
Summary of Platform-Specific Ramifications

- To fully benefit from CORBA, SDR system must require:
  - Connectivity over multi-address systems
  - Support for components written in different languages
  - Support for components running over different operating systems

- XML
  - Parsing is cumbersome
    - Use of XML does not have to be limited to runtime
      - Limits runtime compatibility
  - XML is static
    - Completely dynamic waveform structure difficult to support
      - Additional structures needed
Analysis: Heavy tilt towards distributed system support

- Single point of access/control
  - Clear boundary for radio system, allowing multiple system to coexist in the same network
- Support for a distributed hardware support
  - Storage devices
  - Processing devices
- Self-contained, stand-alone operation for components
  - Components can be deployed as separate processes or threads in the system
- Allows the use of a deployed Application as a component in a larger Application
  - Suited for scalable applications, “waveforms of waveforms”
- CORBA ideally suited to operate in a multi-address distributed environment
Analysis: Expansions needed for large distributed systems

- No metadata descriptors
  - No standardized way of describing data beyond data type
- No distributed waveform control structures
  - No standardized deployment mechanisms
  - No easy way of performing load balancing
Analysis: Limited Support for Embedded Systems

- Reliance on CORBA assumes multi-address, multi-language, multi-OS space
  - Embedded system usually limited to single address
  - System likely to be developed using a single development environment
  - System likely to have a single core with an OS
- Use of Device structures assumes that underlying hardware is powerful enough to support multiple waveforms
  - Embedded system likely to support one waveform at any one time
- No power management structure
SDR Systems and the SCA

- Possible to outline the system that will benefit from the SCA:
  - Multi-address system
    - Multiple boards or cores
  - No stringent memory restrictions (10’s of MB+)
  - Multiple vendor’s hardware and software coexist in the same system
  - Dynamic behavior largely limited to switching waveforms
  - Hardware composition not known in detail by waveform developer at time of development
Conclusion

- SCA can be used on embedded platforms
  - Developer should expect significant expenditures of memory
    - Memory usage generally attributed to CORBA
  - CORBA is not necessarily a bad addition
    - Brings several benefits to system implementation
- Alternative architectures for embedded systems may bring benefits
  - Reductions can be achieved through the reduction of capabilities as compared to SCA
Acknowledgements

- The following entities provided significant levels of funding for Embedded SCA
  - IC Postdoctoral Research Fellowship
  - National Institute of Justice
  - National Science Foundation
  - SAIC