Software Communications Architecture (SCA)
and
Rapid Application Development

Presented by:
Steve Bernier and Hugues Latour

Communications Research Centre Canada
November 5, 2007
Outline

- SCA Overview
- SCA and Component-Based Design (CBD)
- Rapid Application Development (RAD)
- SCA Architect™ RAD Features
- Summary
Outline

- SCA Overview
- SCA and Component-Based Design (CBD)
- Rapid Application Development (RAD)
- SCA Architect™ RAD Features
- Summary
The SCA was created for the US DoD Joint Tactical Radio System (JTRS) program
- Created by the Modular Software–programmable Radio Consortium (MSRC): Raytheon, BAE Systems, Rockwell Collins, and ITT
- Assisted by the Communications Research Centre of Canada

The goal of the SCA is to facilitate the reuse of waveform applications across different radio sets
- Technology insertion and capability upgrades

The SCA defines a central piece of software that acts as the “SDR operating system”
- SCA Core Framework
The SCA is independent of the application domain

Different applications are supported by domain–specific APIs

SCA Core Framework

- Base Station APIs
- Automotive APIs
- JTRS Waveform Applications
- JTRS APIs
Outline

- SCA Overview
- SCA and Component-Based Design (CBD)
- Rapid Application Development (RAD)
- SCA Architect™ RAD Features
- Summary
The SCA is a Component-Based Design (CBD) architecture

What is Component-Based Development?

- **Definition**: an architecture which allows the creation, integration, and re-use of software components
- CBD is a development paradigm where the smallest unit of software is a **component**
- Using CBD, an application is ‘assembled’ using **software components** much like a board is populated with hardware components

Characteristics of a Software Component:

- A small, reusable module of **binary code** that performs a well-defined function (i.e. a black-box)
- Designed, implemented, and tested as a unit before it is used in an application
CBD promotes the COTS culture and is enabling the industrialization of software

The goal is to use the hardware development paradigm for software:

- Purchase software components from a catalog
  - Describe how to influence behavior (config properties)
  - Describe how to interface (ports)
  - Describe resource consumption (capacity properties)
  - Describe resource requirements (capability properties)

CBD is currently the most popular programming paradigm:

- Microsoft’s CBD is the “.NET” framework
- Sun Microsystems’s CBD is the “EJB” framework
- OMG’s CBD is the “CCM” framework
SCA – Component-Based Design

How do we build hardware?
To connect hardware components, appropriate connectors must be used:
Definitions; Back to the small board...
Software equivalent of the small board:
With the SCA, there are two types of constructs:

1. **Components:**
   - Some SCA components are provided with SCA Core Framework product
     - Ex: DomainManager, DeviceManager, Log service, File, FileSystem, FileManager, Event channels, etc.
   - Other components are created by platform providers and application developers
     - Ex: Resource, ResourceFactory, Device, LoadableDevice, ExecutableDevice, etc.

2. **Assemblies:**
   - Defined as a collection of application or node components
SCA – Component-Based Design

SCA components are described by 3 kinds of modeling elements:

1. **Ports**: used to get data to/from a component
2. **Properties**: used to alter the behaviour of a component
3. **Implementations**: used to describe which operating environments a component supports
SCA applications are described by 2 kinds of modeling elements:

1. **Component Instantiations:** which components are part of the application
2. **Connections:** how instantiations are interconnected
Outline

• SCA Overview
• SCA and Component-Based Design (CBD)
  • Rapid Application Development (RAD)
• SCA Architect™ RAD Features
• Summary
Rapid Application Development

What is Rapid Application Development (RAD)?
- Development process invented by James Martin in the 1980s
- Involves iterative development and use some form of Model Driven Development (MDD) tool

Rapid means *Fast*!
- The RAD process is optimized for speed and relies on two key concepts: Prototyping and Iteration
- Prototyping: creating a demonstrable result as early as possible
- Iteration: commitment to incremental development based on refinement
- Prototyping and Iteration go hand–in–hand
Rapid Application Development

Advantages of Rapid Application Development:

– **Clarity/precision:** Development starts at a higher level of abstraction

– **Portability:** High–level abstractions are translated into platform specific artifacts

– **Early visibility:** Can quickly create prototypes

– **Greater flexibility:** Developers can redesign almost at will

– **Fewer defects:** Because of modeling wizards and model translation which greatly reduce manual coding

– **Reduced cost:** Shorter development cycles, time is money!
RAD requires specialized tools that provide:

- **Graphical development/modeling**: to support a high level of abstraction
- **Creation of working prototypes**: for early visibility and greater flexibility
- **Multiple operating environments**: to support portability and greater flexibility
- **Teamwork/collaboration and version control**: because of early visibility and greater flexibility
- **Reusable artifacts**: to support shorter development cycles and reduced cost
Rapid Application Development

Concept of graphical development also known as Model–Driven Development (MDD):
Rapid Application Development

The development of a SCA assemblies is achieved by assembling a number of components together:
Development of a SCA applications can be performed using an iterative process

Iterative refinement happens at two levels:

1. **Component level example:**
   - Create a component with two ports and a couple of properties
   - Successively refine by adding business logic, ports and/or properties

2. **Assembly level example:**
   - Create an application made of a few components
   - Successively refine by adding more components, connections
   - Can also refine by requesting that some components be collocated or by overriding default values for component properties
Rapid Application Development

Typical iterations for development of a component:

1. Start here
2. Generate and Build Component 1
3. Test Component 1
4. Test and Evaluate
5. Understand Requirements
6. Design the System
7. Build Iteratively
8. Final Product Release
9. Add property to Component 1
10. Add Input port to Component 1
11. Generate, Specialize, Build Component 1
12. Model Component 1

Model Component 1
Add property to Component 1
Add Input port to Component 1
Generate, Specialize, Build Component 1
Build Iteratively
Test and Evaluate
Final Product Release
Understand Requirements
Start here
Rapid Application Development

Graphical view of the refinement process for a component:
Rapid Application Development

Typical iterations for development of an assembly:

Test and Evaluate

- Deploy and Run Application1
- Generate and Package Application1

Start here

Final Product Release

Understand Requirements

- Model Application1
- Add Component4, connections
- Change default value for a property of Component2

Design the System

Build Iteratively

- Generate and Package Application1
Rapid Application Development

Graphical view of the refinement process for an assembly:
The refinement process actually happens at both the component and assembly level simultaneously:

- Create Component1 with two ports and a couple of properties
- Create Application1 which includes Component1
- Deploy and run Application1
- Refine Component1 by adding business logic, ports, properties
- Refine the Application1 by adding more components, connections
- Deploy and run new revision of Application1
- Refine Application1a by collocating some components
- Refine Application1a by overriding default values for component properties
- Deploy and run Application1b
- Etc.
Rapid Application Development

RAD tools must support short cycles to promote refinement:

- Must be very simple to successively refine a model
- Must be easy to translate models into source code
- Translation must be flexible and generate as much functionality as possible
Outline

- SCA Overview
- SCA and Component-Based Design (CBD)
- Rapid Application Development (RAD)
- SCA Architect™ RAD Features
- Summary
CRC’s SCA modeling tool: SCA Architect™
SCA Architect™ Overview

SCA Architect™ main characteristics:

- Eclipse–based:
  - Platform independence, easy integration with third party tools, wealth of free plug–ins, etc.

- Supports modeling of every SCA concept graphically
  - *Application assemblies:* Resource instantiations, ResourceFactory, all types of connections, host–collocation, etc.
  - *Node assemblies:* Device instantiations, Device aggregations, use device relationships, all types of connections, etc.

- Translates models into source code, build files, documentation, etc.
- Supports multiple target Operating Environments (OEs)
- Provides real–time validation of models
- Provides reverse–engineering of SCA domain profile files
- Enables configuration management
- Etc.
Most importantly, SCA Architect™ is a RAD tool:

- Already supports several RAD features both at the component and at the assembly level

**Component-level RAD features:**

1. Flexible and Comprehensive Code Generation
2. Zero-Merge Code Generation
3. Model Refactoring
4. Quick-fixes

**Assembly-level RAD features:**

1. AssemblyController Modeling and Code Generation
2. ResourceFactory Modeling and Code Generation
1. Flexible and Comprehensive Code Generation:
   a. Generates a fully functional component out of the box
   b. Provides a Framework to handle component properties:
      • Type, Range and Enumeration validations are taken care of automatically
      • Transpareently handles SCA requirements:
        – Raises proper exceptions when validation problems occur
        – Supports empty queries
      • Abstract CORBA intricacies
        – Querying a property is mapped to a C++ getter
        – Changing a property is mapped to a C++ setter
        – ‘struct’ type of property is mapped to a C++ structure
        – ‘structsequence’ type of property is mapped to a C++ array of structures
1. Flexible and Comprehensive Code Generation (cont):
   c. Provides a framework to handle capacity properties:
      • Allocation and deallocation of capacity is automatically handled
      • Required Device state management is also automatically handled
         – 21 states and close to 70 transitions

Legend:
Operational State: E = ENABLED, D = DISABLED
Admin State: U = UNLOCKED, L = LOCKED, SD = SHUTTING DOWN, LKG=LOCKING
Usage State: I = IDLE, A = ACTIVE, B = BUSY

lock = adminState(LOCKED)
unlock = adminState(UNLOCKED)
enable = set the operationalState at ENABLED
disable = set the operationalState at DISABLED
[de]allocate - all = [de]allocation of the remaining capacity
[de]allocate - partial = [de]allocation of a portion of the remaining capacity
1. Flexible and Comprehensive Code Generation (cont):
   
   d. Provides a Framework to route packets from input ports to output ports:
      
      - Connection handling is done automatically
      - Data processing is controlled via the component start/stop
      - Data processing simply requires the implementation of one method
        - Default behavior is “pass through”
1. Flexible and Comprehensive Code Generation (cont):
   
e. Provides the option of generating a thread to pump data out:
   
   • Thread processing is controlled via the component start/stop
   • Data acquisition simply requires the implementation of one method
2. **Zero–Merge Code Generation:**
   
a. Supports iterative refinement without any merge tool

b. Merging source code is very error prone and cumbersome

![Diagram of Zero–Merge Code Generation]

- **Refined Model**
- **Generated Source Code**
- **Previously Modified Source Code**
- **Merged Source Code**
- **Add Business Logic**
- **Modified Source Code**
2. Zero–Merge Code Generation (cont):
   c. Is achieved by keeping the business logic separate from the model generated code
   d. **Base Code**: Generated from the model
   e. **Business Logic**: Specializes the base code
2. Zero–Merge Code Generation (cont):
   f. Model can be refined several ways without requiring a merge:
      • Can add/remove a property
      • Can edit a property to add/remove/change range or enumeration validations
      • Can add/remove a port
      • Can add/remove fields to a property of type structure
      • Generated code can always be specialized
3. Model Refactoring:
   a. Model can be refactored comprehensively:
      • The model of a Property being used by several components can be changed across a whole project
      • The same is true for Ports and Components

4. Quick Fixes:
   a. After reverse–engineering SCA domain profile files, validation may produce several errors and warnings
   b. Fixing errors/warning manually can be very tedious
   c. SCA Architect offers an automated way of fixing problems:
      • Don’t have to edit a form to repair the problem; choose from alternatives fixes
      • Can apply the same fix to all similar problems
1. **AssemblyController (AC) Modeling and Code Generation**
   
a. Using a wizard, SCA Architect™ can generate an AC model from an application assembly model:
   - Specify which component needs to be controlled
   - Specify which port / property needs to be exported

b. Code generation of an AC creates proxy ports and proxy properties

c. The AC is the main component of an application assembly

d. The AC is generally connected to every component of an application assembly in order to control them

e. Every time a new component is added in the application assembly, the AC must be changed. The same is true when a new property/port needs to be made external

f. Maintaining an AC can quickly become a nightmare
2. **ResourceFactory Modeling and Code Generation**
   
a. Using a wizard, SCA Architect™ can generate a ResourceFactory model from a list of application components:
   
   * Specify which component needs to be deployed by the ResourceFactory
   * Doesn’t require a single line code to be changed in the Resources

b. Can be used to optimize footprint and performance of several application components
Outline

• SCA Overview
• SCA and Component-Based Design (CBD)
• Rapid Application Development (RAD)
• SCA Architect™ RAD Features

• Summary
The SCA is a Component-Based Design architecture

Summary
Without any API supplement, the SCA is not radio nor military specific.
Summary

Using a RAD tool can definitely make it easier to use the SCA

Build Iteratively

Generate, Specialize, Build Component1

Add CodeRate property to Component1

Model Component1

Add Input port to Component1

Test and Evaluate

Test Component1

Generate and Build Component1

Start here

Final Product Release

Understand Requirements

Design the System

Start here
Questions?

Business: jeet.hothi@crc.ca

Technical: steve.bernier@crc.ca

Web Sites: http://www.crc.ca/rars
          http://www.crc.ca/scari