Communications Research Centre Canada
An Agency of Industry Canada

Centre de recherches sur les communications Canada
Un organisme d'Industrie Canada

Evolution of the SCA
Past, Present, and Future

Presented by: **Steve Bernier, M.Sc.**

Research Scientist

Advanced Radio Systems

Communications Research Centre (CRC)

Government of Canada

Canada

CRC

CENTRE DE RECHERCHES SUR LES COMMUNICATIONS RESEARCH CENTRE

Outline

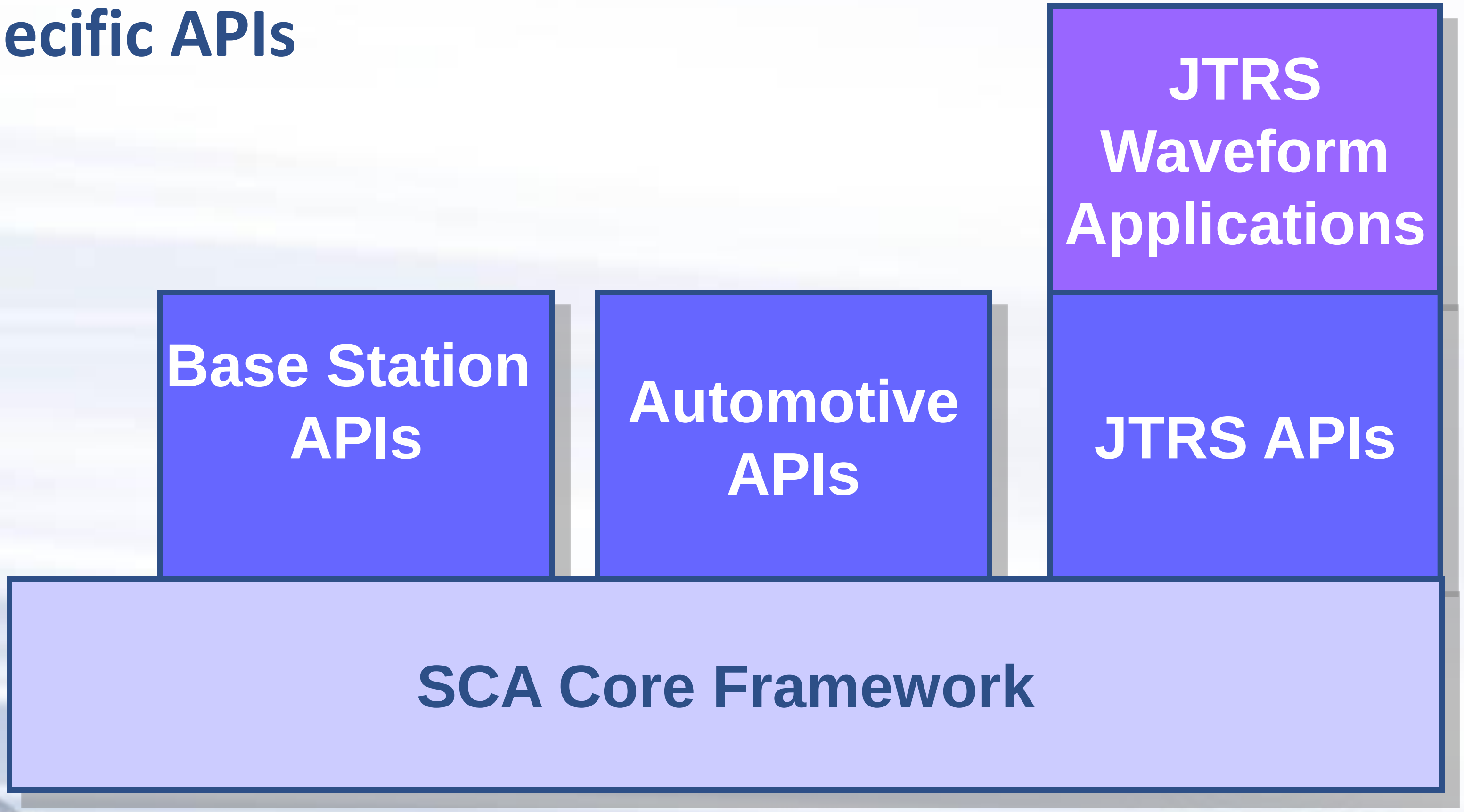
1. SCA Overview
2. Evolution of the SCA Specification
3. Evolution of SCA Tools
4. Evolution of SCA Core Framework
5. Future Core Frameworks
6. Summary

SCA Overview - JTRS

- **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
- **The goal of the SCA is to facilitate the reuse of waveform applications across different radio sets**
- **The SCA is not a system specification!**
  - Provides an implementation-independent set of rules that constrain the design of systems to achieve the above objectives

SCA Overview – Block Diagram

- The SCA is independent of the application domain
- Different applications are supported by domain-specific APIs



SCA Overview - CBD

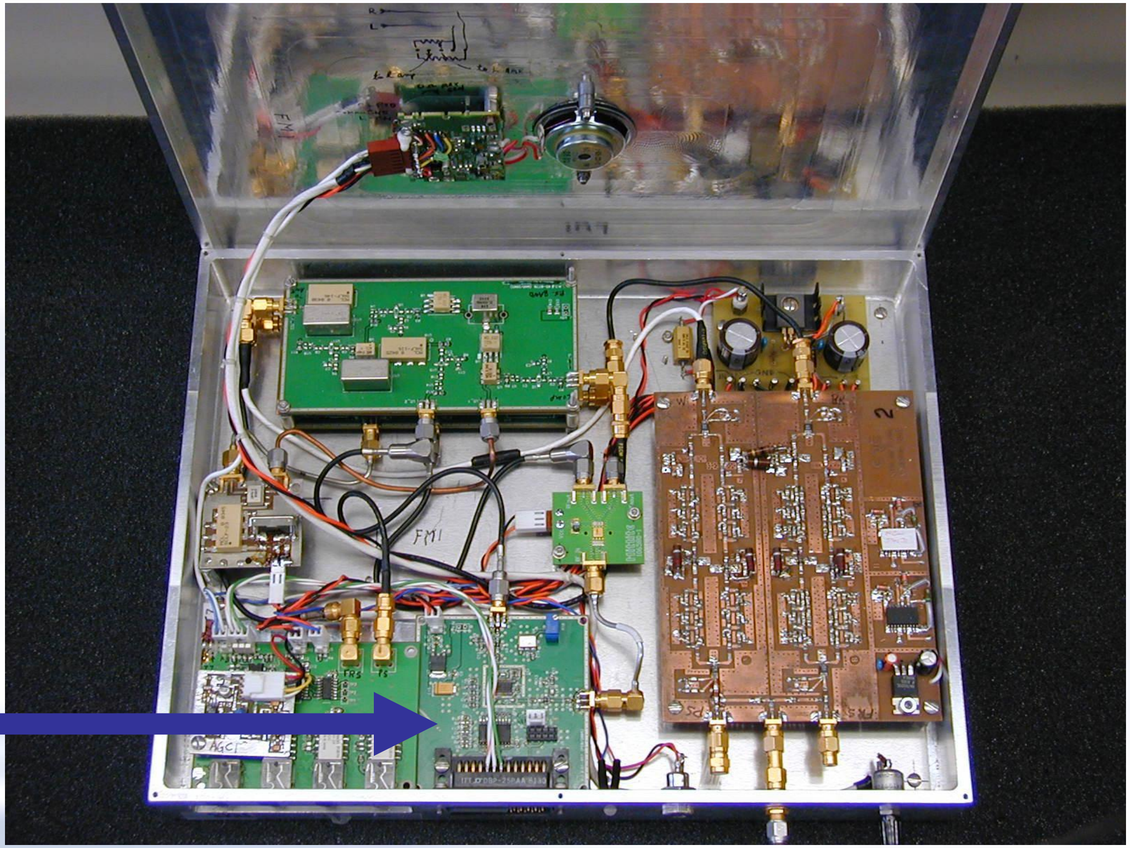
- **From a software development perspective, the SCA is a Component-Based Development (CBD) architecture**
- **What is Component-Based Development ?**
  - **Definition:** an architecture which allows the creation, integration, and re-use of software components
  - CBD is a new development paradigm where the smallest unit of software is a **component**
  - With 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

SCA Overview - CBD

- **CBD promotes the COTS culture and is enabling the industrialization of software**
- **The goal is to apply the hardware development paradigm to software**
  - Purchase software components from a ‘spec-sheet’ 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 Microsystem’s CBD is the “**EJB**” framework
  - OMG’s CBD is the “**CCM**” framework

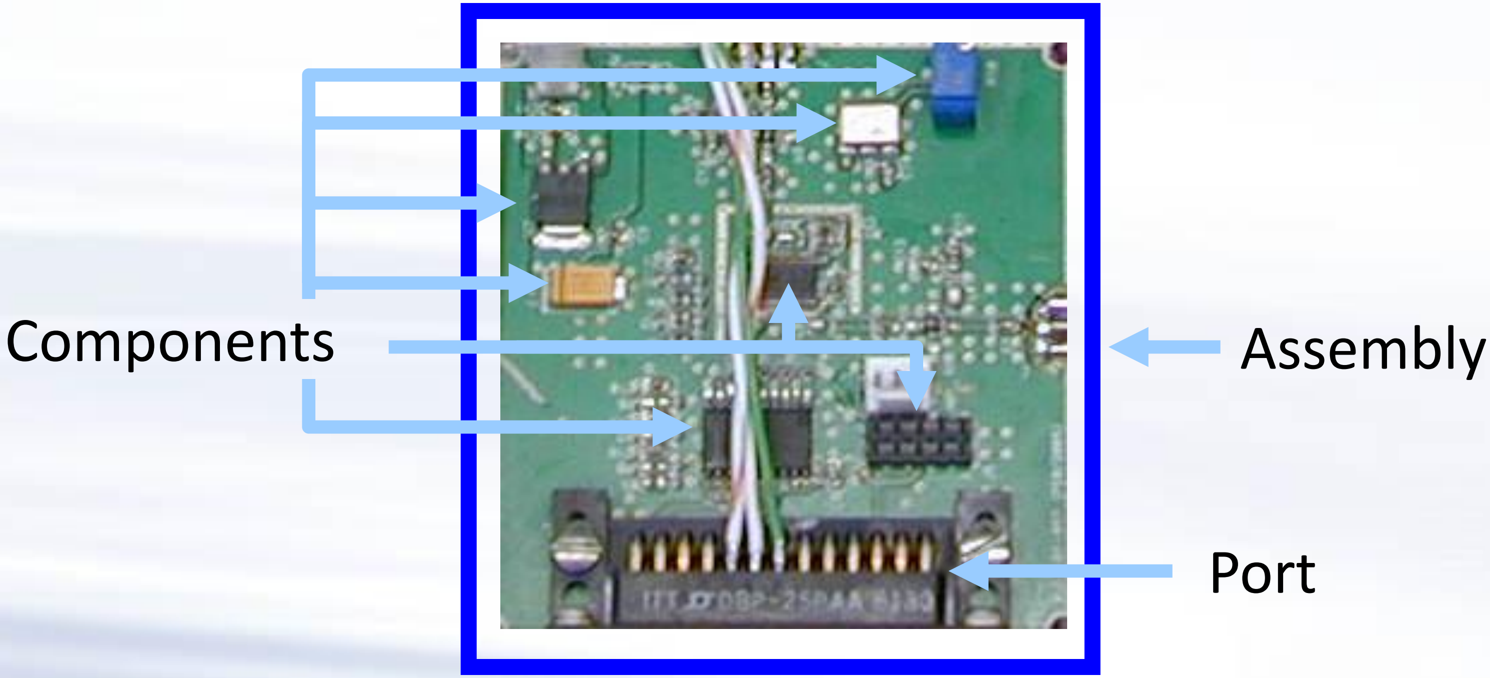
SCA Overview - CBD

- **How do we build hardware?**



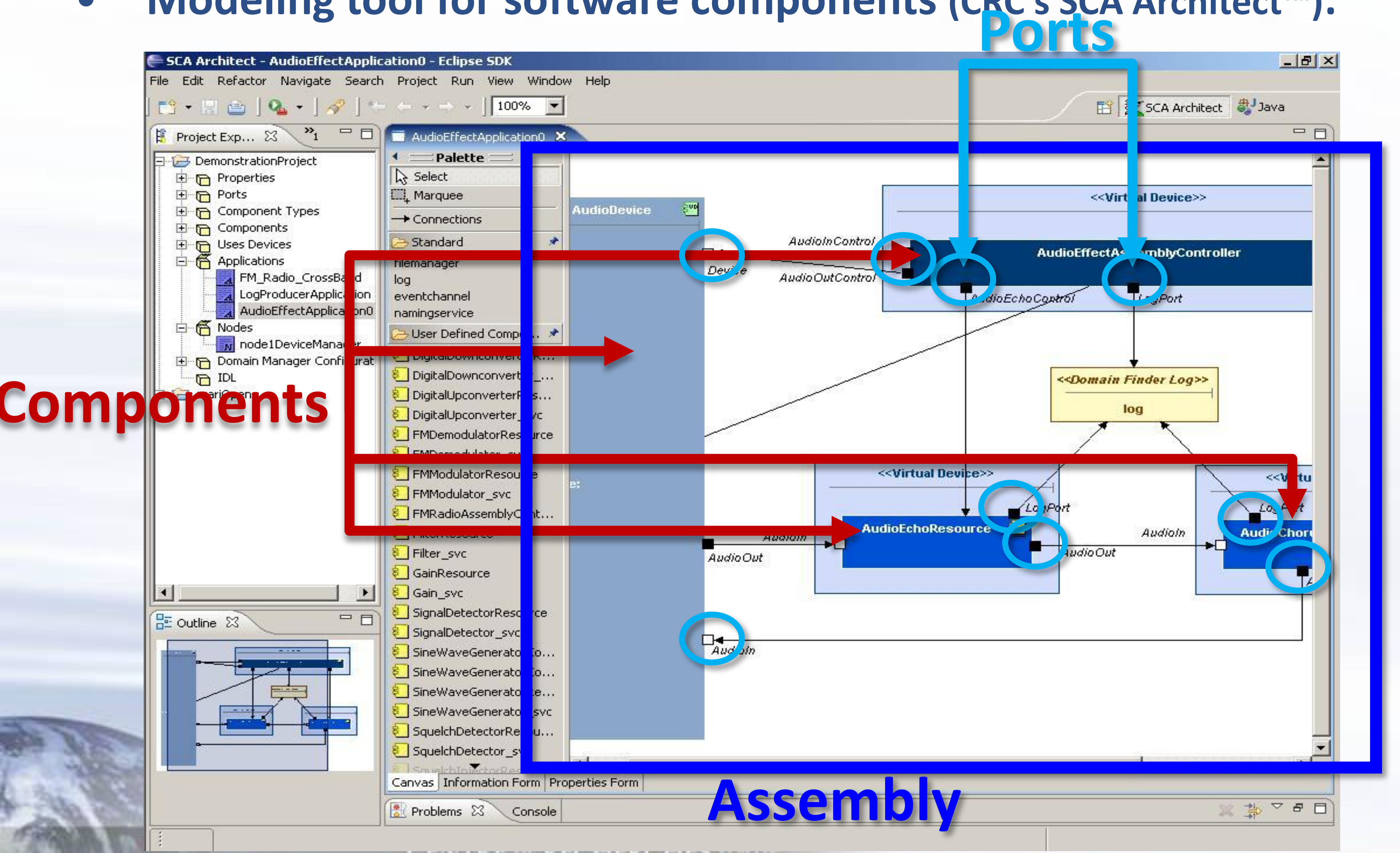
SCA Overview - CBD

- **Definitions; Back to the small board…**



SCA Overview - CBD

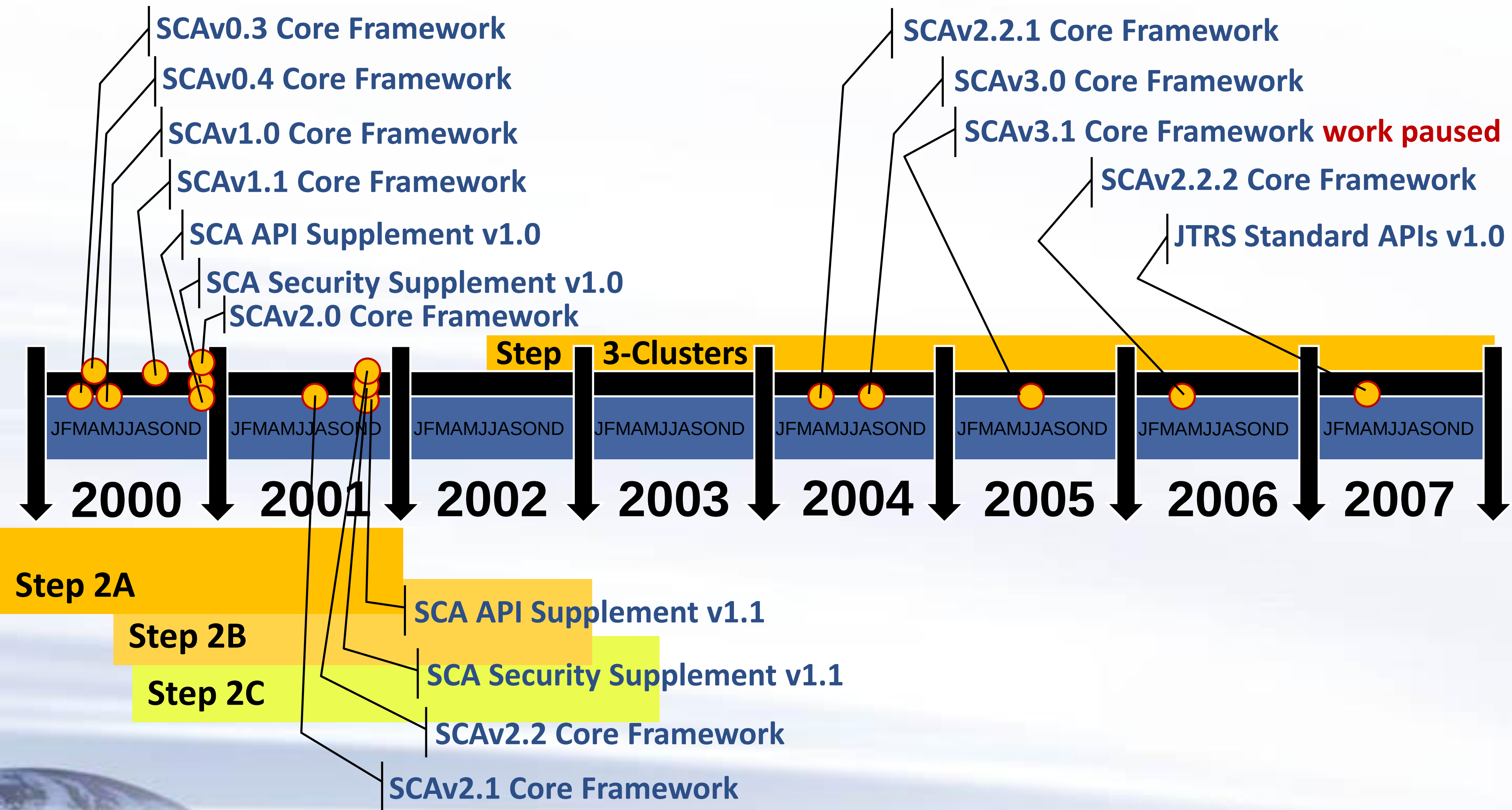
- **Modeling tool for software components (CRC's SCA Architect™):**



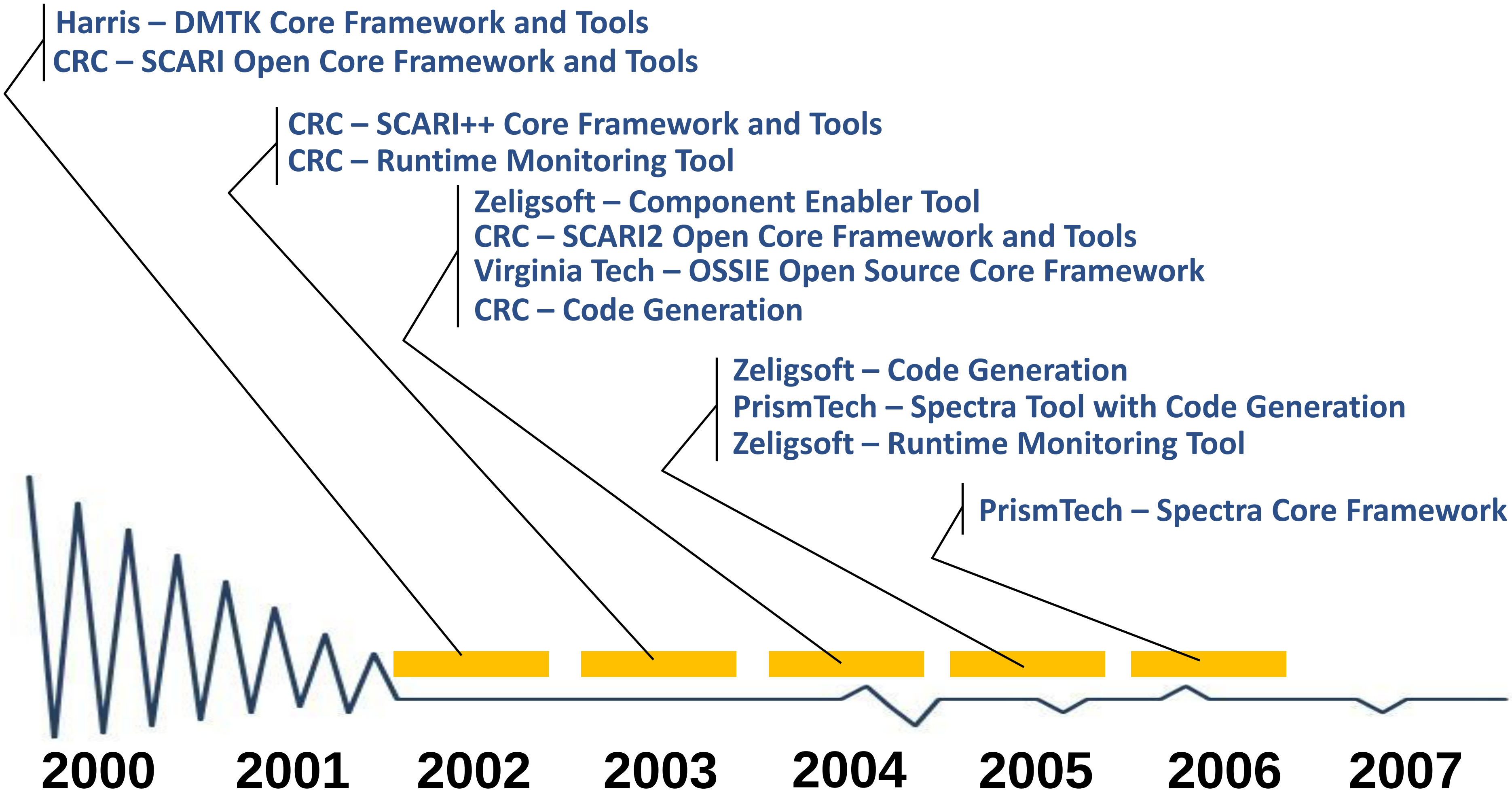
Outline

1. SCA Overview
2. Evolution of the SCA Specification
3. Evolution of SCA Tools
4. Evolution of SCA Core Framework
5. Future Core Frameworks
6. Summary

Evolution of the SCA Specification



Evolution of SCA Products



Magnitude of changes for the SCA Specification

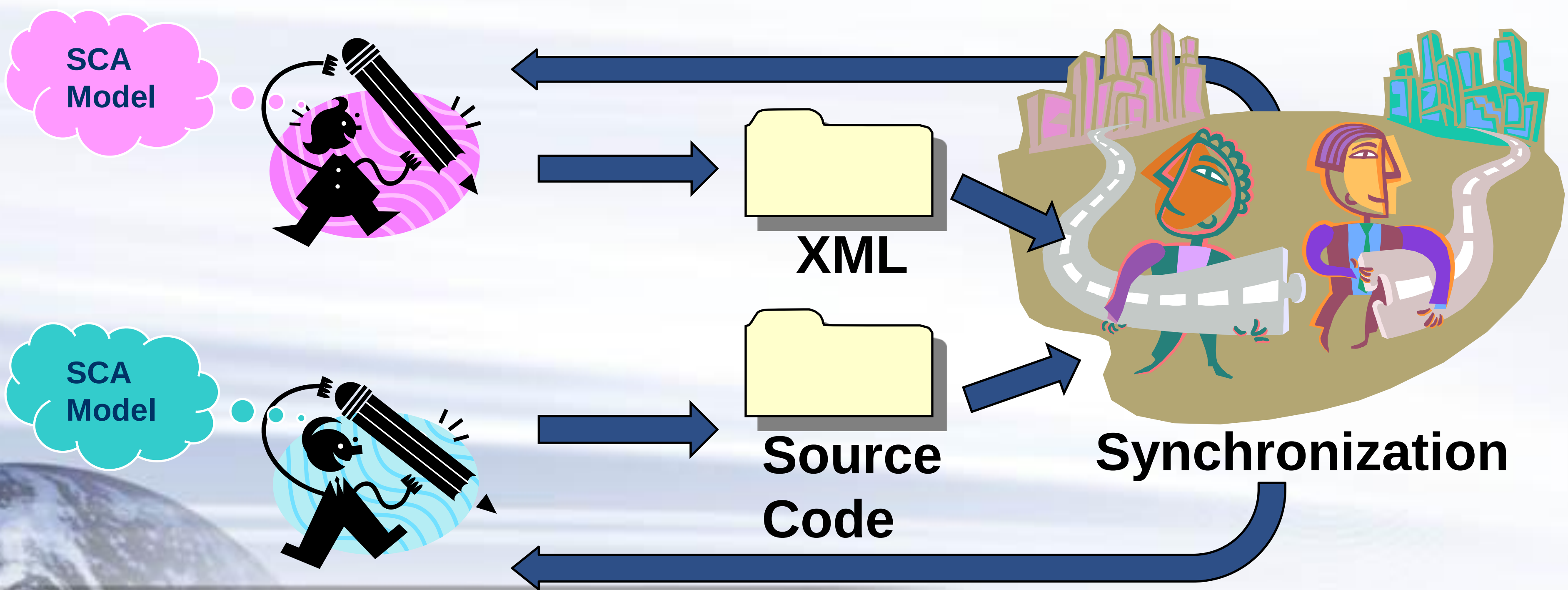
Outline

1. SCA Overview
2. Evolution of the SCA Specification
3. **Evolution of SCA Tools**
4. Evolution of SCA Core Framework
5. Future Core Frameworks
6. Summary

Evolution of SCA Tools

Before 2002

- The most popular SCA tool was a *text editor*!
- XML files were developed manually
- Synchronization between source code and XML was done manually
- SCA compliance validation was also manual
- No formal way of representing an SCA model



Evolution of SCA Tools

Since 2002

- There are two kinds of tools: Development and Runtime
- Development tools evolved from glamorized XML editors to graphical modeling tools
- Development tools now provide:
  - Graphical modeling environment
  - Model validation
  - Reverse-engineering
  - Automatic generation for source code, documentation, domain profile
  - Configuration Management
  - And more…
- Runtime tools provide:
  - Application installation, launching, and control
  - Platform monitoring and control

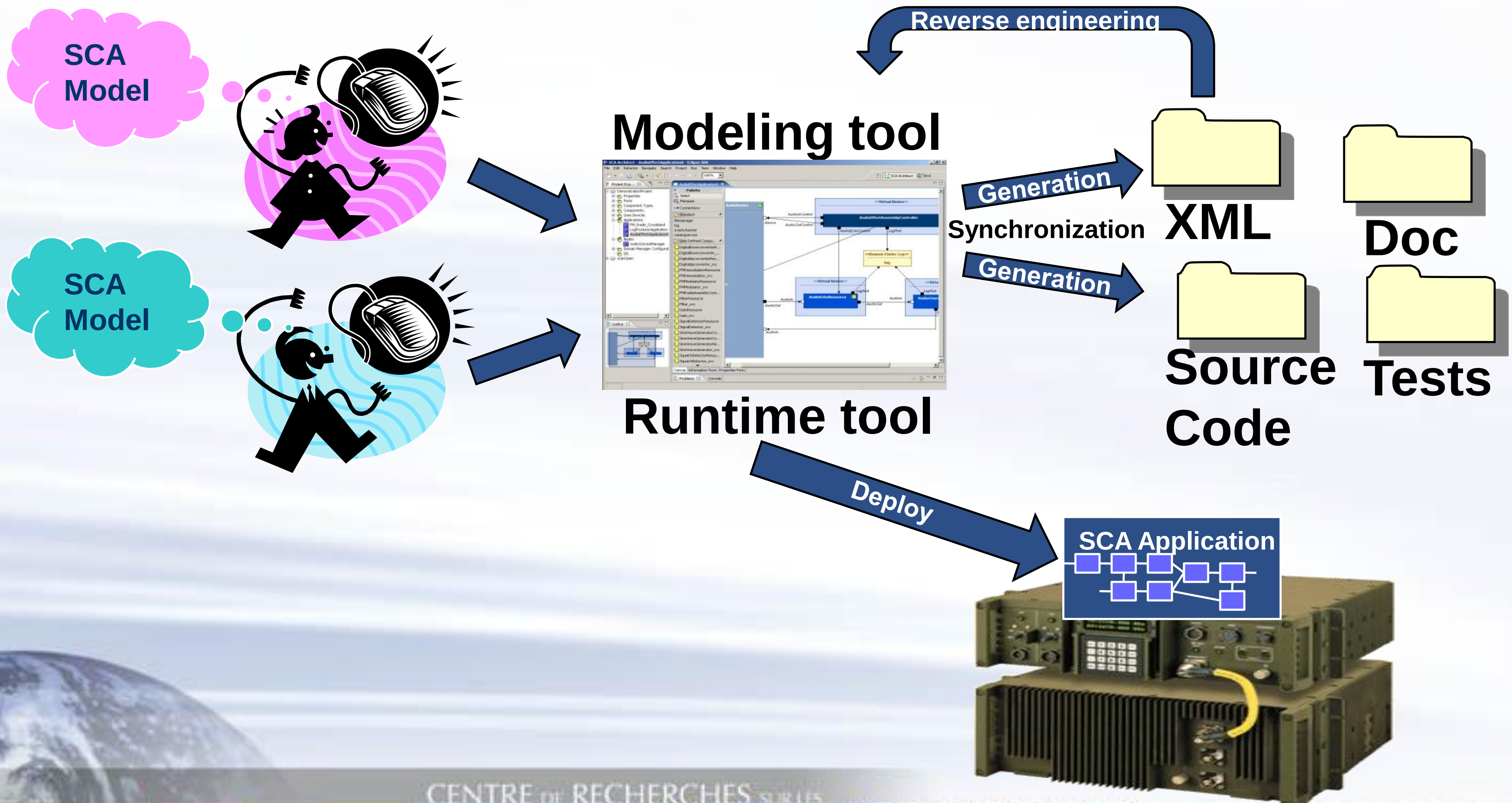
Evolution of SCA Tools

Today

- XML files are generated from a model; not manually created
- The same is true for Source Code, Documentation, etc.
- Synchronization between artifacts is handled by modeling tools
- Tools provide “validators” for SCA-compliance
- Tools can be used to import manually created XML
- Tools also provide a modeling environment which guides developers to avoid non-compliance issues
- SCA models can be represented graphically

Evolution of SCA Tools

Today



Outline

1. SCA Overview
2. Evolution of the SCA Specification
3. Evolution of SCA Tools
4. Evolution of SCA Core Framework
5. Future Core Frameworks
6. Summary

Evolution of SCA Core Frameworks

- **First generation Core Frameworks were implemented in the early days of the JTRS Program**
  - Many US DoD contractors implemented portions of the full SCA Core Framework between 2000 and 2002 (Steps 2A, 2B, and 2C)
  - Harris DMTK Core Framework was made available for licensing in 2002
  - CRC released the SCARI-Open publicly for free in 2002

- **First generation Core Frameworks were rather large and slow**
  - Mainly due to the use of XML parsers designed for desktop applications
  - Developers concentrated on understanding (and fixing) the SCA specification and making their Core Framework work
  - Used CORBA with TCP/IP transport which is slow for real-time systems

Evolution of SCA Core Frameworks

- **Second generation Core Frameworks are smaller and faster (2003 - 2004)**
  - Small Form Factors required for the JTRS program led US DoD contractors to take different implementation decisions
  - XML parsing is achieved with smaller/faster XML parsers
  - Some basic optimizations have been implemented
  - Use different transports for CORBA (no TCP/IP for local comms)

- **Since then, R&D from several sources has led to the identification of several optimization techniques with great potential**

1. **DESIGNING JTRS CORE FRAMEWORKS FOR BATTERY-POWERED PLATFORMS: 10 TECHNIQUES FOR SUCCESS,** C. A. Linn, Harris Corporation, SDRF'02 Conference
2. **JTRS SCA:CONNECTING SOFTWARE COMPONENTS**, S.Bernier and al., SDRF'03 Conference
3. **PUTTING IT ALL TOGETHER – OBJECTIVES AND CHALLENGES**, J. Belzile, SDRF'05 Conference
4. **TAKING THE SCA TO NEW FRONTIERS,** S. Bernier and C. Belisle, Communications Research Centre Canada, SDRF'06 Conference
5. **COMMENTS ON SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION VERSION 2.2.2**, SCA Working Group, SDR Forum, SDRF-06-W-0012-V0.01, October 2006

Evolution of SCA Core Frameworks

- **Core Framework optimizations fall into two categories:**
  - Task Optimizations
  - Static Deployment Optimizations
- **The main service provided by a Core Framework is the deployment and configuration of applications**
  - To do so, a Core Frameworks performs a number of tasks:
    1. Load Application XML files
    2. Read Application assembly files
    3. For each component of the application:
       a. Choose an implementation
       b. Deploy the implementation
       c. Configure the component
    4. Establish Component Inter-connections
    5. Etc.

Evolution of SCA Core Frameworks

- **Task Optimizations consists in making each task faster and/or use less memory**
- **Papers published regarding Core Framework optimizations concentrate on Task Optimizations:**
  1. Accelerate local file access
  2. Use a caching systems for connections
  3. Get all ports at once
  4. Perform all connections at once
  5. Allow full node registration in one call
  6. Provide support for remote Devices
  7. Provide a parser-free DeviceManager
  8. Allow co-location of Core Framework components
  9. Use a specialized XML parser
  10. Use digested profiles

Evolution of SCA Core Frameworks

1. Accelerate local file access

– Avoid copying a file through SCA FileSystems when they are running on the same native file system. Perform a native file copy instead.
  - Can be significant for large files

Test Scenario	File Size	Time without acceleration	Time with acceleration	Improvement
Linux Desktop 3Ghz Pentium without NFS	4 MB	355 ms	20 ms	~94%
INTEGRITY PPC405 SBC using NFS	1.5 MB	2.5 sec	1.5 sec	~40%

2. Use a caching system for connections

– Cache the components and ports involved in connections to avoid redundant lookups
  - Provides speed improvement when several connections involve the same port of a same component like in fan-in/fan-out scenarios

Evolution of SCA Core Frameworks

3. Get all ports at once

- Support a new way of obtaining all the ports of a single component in one call
  - Can be combined with the caching system for connections
  - Provides speed improvement for applications with several connections

4. Connect all ports at once

- Support a new way of connecting all the ports of a single component in one call
  - Can be combined with the caching system for connections
  - Can be combined with getting all ports at once
  - Provides speed improvement for applications with several connections

Evolution of SCA Core Frameworks

5. Allow full node registration in one call

- Allow a DeviceManager to register with DomainManager using digested information
  - Can be very significant for platform with slow file systems (avoids reading and interpreting XML files)
  - Can also be very significant for slower processors (saves 19 CORBA calls per registering Device)

Test Scenario	Standard Registration	One call Registration	Improvement
Linux Desktop, 1 Device	0.56 sec	0.19 sec	~ 66%
Linux Desktop, 4 Devices	1.53 sec	0.24 sec	~ 84%
LynxOS PPC405, 1 Device	0.86 sec	0.13 sec	~ 85%
LynxOS PPC405, 4 Devices	2.33 sec	0.22 sec	~ 91%

Note: Tests conducted in scenarios where DomainManager and DeviceManager run on same processor

Evolution of SCA Core Frameworks

6. Provide support for remote Devices

- Allow Devices started manually to register to a DeviceManager
  - Minimize number of DeviceManagers required
  - Allows Devices to be collocated in a single address space which provides significant middleware speed improvements and footprint savings
  - By using this optimization, ISR Technologies was able to lower transport latencies from 300 usec to 10 usec by co-locating remote devices and using the INTCONN ORBexpress pluggable transport available for INTEGRITY [reference paper #3]

7. Provide a parser-free DeviceManager

- Provide a DeviceManager with no deployment engine
  - Can be combined with the remote Device support
  - Allows Devices to be co-located in the address space of the DeviceManager
  - Provides both speed and footprint improvement

Evolution of SCA Core Frameworks

8. Allow colocation of Core Framework components

- Enable the DomainManager, DeviceManager, Services and Devices to be colocated in a single address space
  - Provides footprint savings
  - Combining the DomainManager and DeviceManager can save up to 50% of the total footprint

9. Use a specialized XML parser

- Use a hand-crafted XML parser. Avoid using DOM parsers
  - Performs in 13 to 21% of the memory* required for DOM parsing
  - Performs in 8 to 20% of the time required for DOM parsing

10. Use digested profiles

- Use XML files formatted as binary files and containing only the meta-data required for deployment and configuration
  - Performs in 7 to 13% of the memory* required for DOM parsing
  - Performs in 5 to 15% of the time required for DOM parsing

*** Dynamic memory (e.g. heap)**

Outline

1. SCA Overview
2. Evolution of the SCA Specification
3. Evolution of SCA Tools
4. Evolution of SCA Core Framework
5. **Future Core Frameworks**
6. Summary

Future Core Frameworks

- **Future Core Frameworks will need to be able to run on very Small Form Factors**
  - Small both in terms of memory available and speed of processors
- **To achieve this, Static Deployment Optimizations will have to be used**
- **This kind of optimization consists in skipping tasks a Core Framework normally performs for the deployment of components**
  - Avoid performing the same task twice for the deployment of the same application on the same platform
    - Remember where components have previously been deployed
    - Use a caching file system to avoid copying files previously loaded

Future Core Frameworks

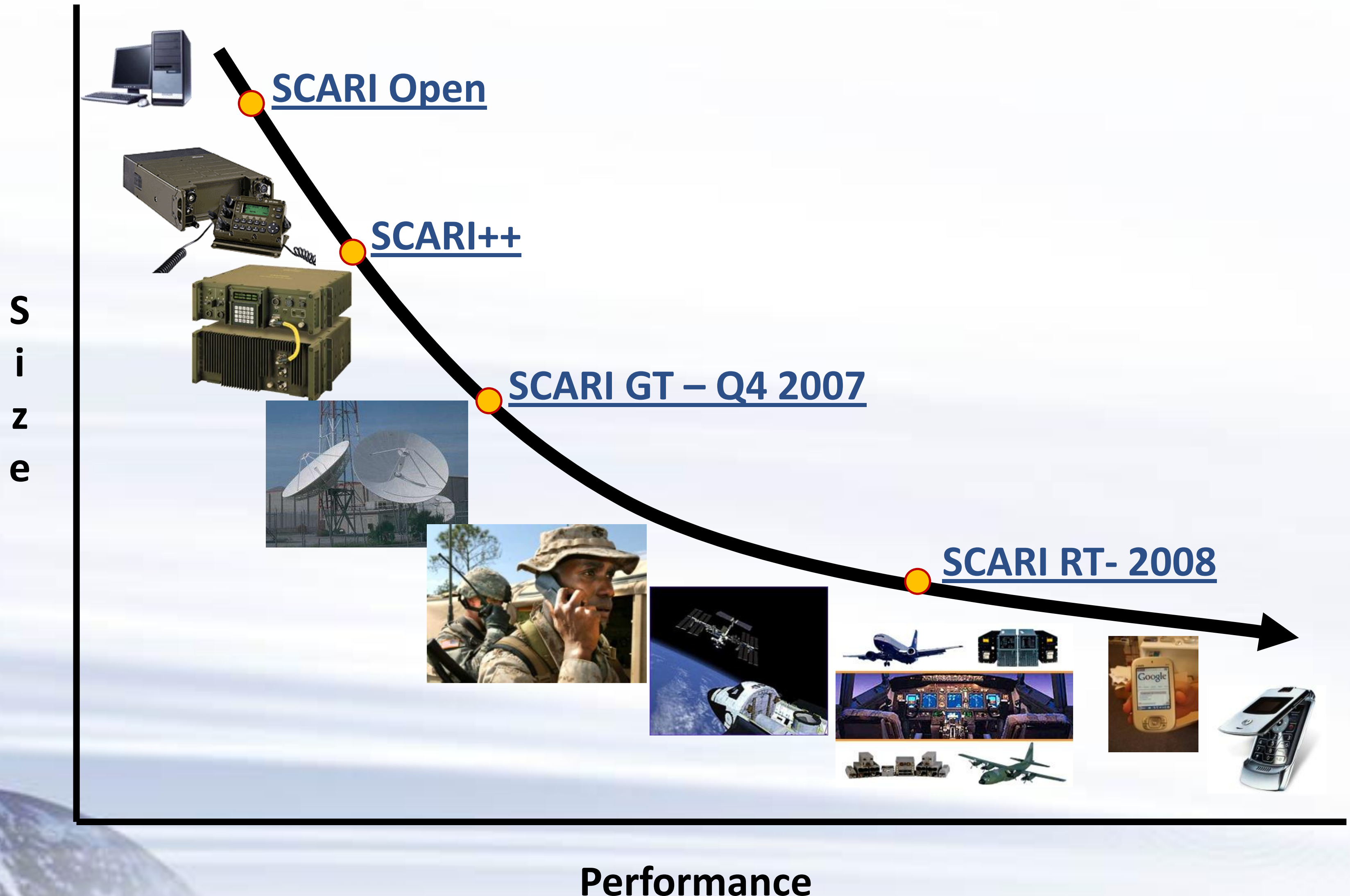
- **Use a caching FileSystem**
  - Avoid copying a file (whether remote or local) when most recent version is already present in cache
    - Requires persistent storage
    - Can be very significant for large files or if an SCA FileSystem uses small buffers to copy

Test Scenario	File Size	Cache miss W/O local file acceleration	Cache miss with local file acceleration	Cache hit	Improvement
Linux Desktop 3Ghz Pentium without NFS	4 MB	355 ms	20 ms	9 ms	~ 98%
INTEGRITY PPC405 SBC using NFS	1.5 MB	2.5 sec	1.5 sec	35 ms	~ 99%

CRC's Core Frameworks

- **CRC's current SCARI++ Core Framework already implements some of the task optimizations described in this presentation**
- **The upcoming SCARI-GT Core Framework will provide the optimizations introduced in this presentation**
  - Provides all of the many task optimization described in this presentation
  - Also implements the caching file system optimization
  - Will be available in Q4 2007
- **The SCARI-RT Core Framework will enable full static deployment of standard SCA applications**
  - Release in 2008

CRC's Core Frameworks



Outline

1. SCA Overview
2. Evolution of the SCA Specification
3. Evolution of SCA Tools
4. Evolution of SCA Core Framework
5. Future Core Frameworks
6. **Summary**

Summary

- **The SCA is a Component Based Development architecture**
  - Not specific to military SDR; Can be used for any embedded application
- **The SCA specification has reached an acceptable level of maturity**
  - There is an eco-system of COTS SCA products
- **SCA Core Frameworks are becoming smaller, faster, more deterministic and can still deploy standard SCA applications unchanged**

Summary

Questions?

steve.bernier@crc.ca

www.crc.ca/scari