JTRS SCA: CONNECTING SOFTWARE COMPONENTS

François Lévesque, Charles Auger, Steve Bernier, Hugues Latour

Communications Research Centre (CRC), Canada

Presented by Steve Bernier at
Software Defined Radio Technical Conference
17th – 19th November 2003
Outline

• Software Component Portability
• SCA Connection Model
• Connection Portability Issues
• Connection Portability Guidelines
• Inter-Application Connections
Software Component Portability

- In general, software component portability can be obtained using one of three approaches:
  - **Interpreter**: source code of a portable component is sent to an interpreter program that behaves appropriately for the host platform.
  - **Virtual machine**: source code of a portable component is compiled for a specific platform and is executed by a virtual machine that behaves appropriately for the host platform.
  - **Multiple compiles**: source code of a portable component is compiled for each different host platforms and is executed natively.
Software Component Portability (con’t)

- The SCA uses the multiple-compiles model to achieve portability:
  
  - SCA components (e.g. Devices and Resources) are compiled for the different platforms in which they are intended to be used
  
  - Components provide a description of their requirements and capabilities, which are compared during the process of software deployment
Software Component Portability (con’t)

- **SCA connection portability**
  - This portability model doesn’t deal with connection portability
  - If it is not used properly, the SCA connection model can lead to portability problems
  - Even though SCA components were compiled for their target SCA platforms, connection portability problems may preclude their execution
SCA Connection Model

- **SCA connections**
  - Connections are used to provide references to components for communication and control purposes
  - Connections are unidirectional
  - Orientation of a connection doesn’t indicate data flow
  - Connections are used in DCD (node) and SAD (application)

![Diagram of SCA connections](Image)
SCA Connection Model (con’t)

• **Connection source:** to receive a reference to the destination component of a connection, the connection source must provide a special API named *Port*

<table>
<thead>
<tr>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Img" alt="connectPort" /> <code>connectPort(connection : in Object, connectionId : in string)</code></td>
</tr>
<tr>
<td><img src="Img" alt="disconnectPort" /> <code>disconnectPort(connectionId : in string)</code></td>
</tr>
</tbody>
</table>
SCA Connection Model (con’t)

- Two types of SCA connection:
  - **Port-to-component:** Connection destination inherits the interface needed by the source; connection is established directly to the component
  - **Port-to-port:** Connection destination implements the needed interface by aggregation; connection has to be established to the sub-component implementing the interface
SCA Connection Model (con’t)

- Port-to-component connection

```python
portA1 = component_A.getPort("Port1")
portA1.connectPort(component_B, "toB")
```

- Disconnection:

```python
portA1.disconnectPort("toB")
```
SCA Connection Model (con’t)

- **Port-to-port connection**

  - **Connection:**
    
    ```
    portA1 = component_A.getPort("Port1")
    portB1 = component_B.getPort("Port1")
    portA1.connectPort(portB1, "toB1")
    ```

  - **Disconnection:**
    
    ```
    portA1.disconnectPort("toB1")
    ```
SCA Connection Model (con’t)

- The SCA offers five mechanisms for obtaining/identifying the source or destination component of a connection

- Can be categorized into “direct” and “indirect” identification mechanisms
  - **Direct identification mechanisms**: source or destination component is identified using pre-defined (static) information
  - **Indirect identification mechanisms**: source or destination component is identified using runtime information
SCA Connection Model (con’t)

• Direct identification mechanisms:

  – **Naming service name**: CORBA naming service is queried using a name to obtain a reference to a component. Naming service registration is only mandatory for *Resources*

  – **Component instantiation reference**: in an assembly descriptor (SAD, DCD), each component instance is associated to a unique identifier which can be used to establish connections
Indirect identification mechanisms:

- **Domain finder**: used to establish connections to radio services (e.g. log, naming service). Services from all nodes register to the *DomainManager* using a name and a type.

- **Device that loaded a component**: allows a connection with a *Device* that was used to load a specific component (e.g. FPGA *Device* that was used to load a specific algorithm).

- **Device used by a component**: allows a connection with a *Device* that is being used by a specific component. Usage relationship are declared using capabilities and capacities requirements.
SCA Connection Model (con’t)

• Restrictions for node connections (DCD):
  – Components of a node are launched by a DeviceManager while the connections are established by the DomainManager
  – Because of a lack of API between the DeviceManager and DomainManager, the information gathered by the DeviceManager when launching the components cannot be provided to the DomainManager
  – Therefore, the following indirect identification mechanisms cannot be used:
    • Device that loaded a component
    • Device used by a component
Connection Portability for Applications

• Issues that can preclude connection establishment for an application:
  – Reference to a specific *Device* name
  – Reference to a specific *Port* name
  – Reference to a specific *Service* name
  – Different Radio Frequency (RF) chain implementations
  – Association between a component and a *Device*
  – External connections
Connection Portability for Applications (con’t)

• **Reference to a specific Device name**
  
  – The name of a *Device* is chosen by a radio integrator; it may differ in each radio

  – Therefore using a direct identification mechanism is not portable

  – The Device involved in a connection should be identified using its characteristics (capabilities and capacities)

• The SCA will have to standardize more capabilities/capacities
Connection Portability for Applications (con’t)

• Reference to a specific Port name
  – To connect to *Devices*, applications may use port names which are defined by the *Device* developer
  – Port names for SCA components provided by a platform and used by an application should be standardized
Reference to a specific *Service* name

- Services register to the *DomainManager* using a name and a type
  
- Components connect to radio services using domain finder connections

- Since the component that implements a radio service may be different in each radio, connections to services should not be identified using a name

- Connections to radio services should be identified using a service type (e.g. filemanager, filesystem, logger, namingservice)
Different Radio Frequency (RF) chain implementations

- The steps that need to be performed for the conversion of RF signals to baseband data sample can be implemented in various ways

- Different radios could provide different groups of RF Devices and still offer an equivalent service

- To configure a Device, an application needs to be connected to it

- Since the number of Devices used by an application can vary, it is impossible to define a portable application assembly descriptor because of the varying requirements in terms of connections
Connection Portability for Applications (con’t)

- Different Radio Frequency (RF) chain implementations
  - RF Chain 1
    - High-Speed ADC
    - Digital Down Converter
    - Waveform Application
  - RF Chain 2
    - Down Converter
    - ADC
    - Waveform Application
Different Radio Frequency (RF) chain implementations

- One way to address this problem is to abstract the RF device chain

- Application components would not connect to individual RF devices but only to a high-level abstraction artifact

- The OMG SWRadio group has introduced the concept of an RF_Channel component that could be used as a basis for a solution in the SCA
Connection Portability for Applications (con’t)

• Association between a component and a *Device* (ie: *usesdevice*)
  – The declaration of an association can be defined:
    • globally for a component (component level) or
    • for a specific implementation of a component (implementation level)
  – Connections referencing an association not defined for all implementations could fail depending on the chosen implementation
  – Associations should be defined at the component level rather than the implementation level
    • For more portability, associations to devices should only be made with the assembly controller component
Application Connections Portability (con’t)

**External connections**

- For an Application to be controlled (ex: from a console GUI), connections must be established with it. There are two options:
  - Connections to sub-components
  - Connections to external ports

- Connections to sub-components of an Application break the encapsulation concept and may create portability problems
  - Modifications to sub-components of an application may render external connections unusable
  - Even tough the application is portable and may be executed, it cannot be controlled
Application Connections Portability (con’t)

• **External connections (con’t)**
  - External entities should connect to the external ports of an application (i.e. defined in the application’s SAD)
  - The external ports should be mapped to the ports of the assembly controller to allow connections without breaking the encapsulation
Inter-Application Connections

• Problem
  – In many scenarios, multiple applications must communicate with each other in order to provide a single aggregated functionality to the radio user
  – The current version of the SCA (2.2) doesn’t specify how applications may be connected with each other
  – An application can potentially be connected to another application since it implements the PortSupplier interface
Inter-Application Connections (con’t)

• Problem (con’t)
  – However, there are problems preventing portable connections between applications:
    • The order in which applications are created is usually controlled by the radio operator. It is difficult to automate the launch of two (or more) applications in a specific sequence suitable for connection establishment
    • The name of an application component is always appended to the name of the application (chosen at run time by the radio operator). Pre-defined application names have to be used
Inter-Application Connections (con’t)

• Potential Solution 1
  – One of the applications involved in a connection registers as a radio service
  – The connection could identify this application using the domainfinder identification mechanism
  – Advantage:
    • This solution would preserve application encapsulation which is good for maintenance and portability
  – Disadvantages:
    • Requires a new type of service called “application”
    • The name of the second application (i.e. service) would have to remain unchanged
Inter-Application Connections (con’t)

• Potential Solution 2
  – Add support for the concept of aggregate applications to the SCA
  – An aggregate application would be composed of two or more applications
  – Advantages:
    • The radio operator does not need to launch many individual applications with a specific name to obtain the aggregate application behavior
    • Connections would be established between applications, thus preserving encapsulation
Inter-Application Connections (con’t)

• Potential Solution 2 (con’t)
  – Disadvantages:
    • Requires modifications to the SCA standard
      – *ApplicationFactory* behavior has to be altered to launch aggregate applications in addition to the standard applications
    – A new assembly descriptor is needed to indicate which applications compose the aggregate application and how they must be inter-connected
Questions ?