COMPONENT TECHNOLOGY NEUTRAL IMPLEMENTATION

Jerry Bickle (Raytheon, Fort Wayne, IN, USA, Gerald.l.Bickle@raytheon.com)
Vincent Kovarik (PrismTech, vince.kovarik@prismtech.com)

ABSTRACT

This paper and demonstration describes a component-based, technology neutral implementation approach based upon the Joint Tactical Networking Center (JTNC) Software Communications Architecture (SCA) [1]. SCA defines specific technologies profiles (e.g., Common Object Request Broker Architect (CORBA®) [2], Portable Operating System Interface (POSIX™) [3], etc.) that provide the features for portability and reuse of a component technology implementation. SCA allows other middleware communication technologies besides CORBA such as Data Distributions Service (DDS) [4] and POSIX Inter-Process Communications (IPC) (e.g., shared memory, queue, etc.). The direct usage of middleware communication and component frameworks (e.g., SCA 2.2.2 [5] versus SCA 4.X) technologies in a component's implementation makes the implementation portable and reusable only for those technologies. The paper and demonstration will describe the component technology neutral implementation design patterns for middleware communication and component frameworks.

1. INTRODUCTION

The initial focus of the SCA is to promote portable and reusable application code across real-time operating systems, thus the usage of POSIX profile and minimum CORBA profile on General Purpose Processor (GPP). Starting with SCA version 4, the concepts of components were introduced along with allowing other technologies besides CORBA as the middleware for control and data distribution. SCA 4 introduced additional POSIX and CORBA profiles to address portability and reuse beyond the GPP and into the signal processing elements. The usage of POSIX profiles in a component's implementation promotes technology neutral implementations but even the usage of POSIX IPC mechanisms can hinder the implementation portability and reuse on another platform. Likewise, component's implementation using CORBA is tied to a specific middleware technology but the implementation is still portable and reusable for that technology as specified by the SCA 4 CORBA profiles. The remainder of the paper explains a component implementation design pattern for a component's implementation to be technology neutral and abstracts the IPC technology being used for communication by the implementation.

2. COMPONENT IMPLEMENTATION DESIGN PATTERN

The Component Implementation Design Pattern, as shown in figure 1 below, can be viewed as containing three essential elements of a component as described below.

1. Component Container that is technology specific,
2. Component Implementation that is technology neutral, and
3. Component Uses and Provides Ports (SCA terminology) that is technology specific, which relates to UML™ [6] [7] Required and Provided Ports.

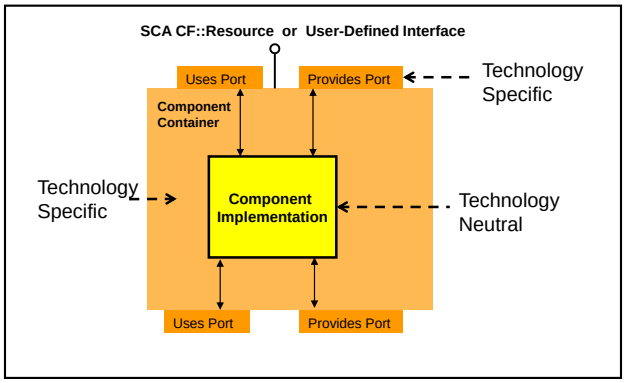

Figure-1. Component Implementation Design Pattern Illustration

There can be many different component containers that the same component implementation can be plugged into, as shown in figure 2 below, such as SCA 2.2.2 Resource Component Container, SCA 4.0.1 Resource (V222 equivalent) component container and SCA 4.1 User-Defined Resource (V222 equivalent) component container. V222 equivalent means offers similar Resource interface capabilities. In all cases, the same component implementation can be plugged into each of the above component containers.

Also in each of the above component containers (SCA 2.2.2, SCA 4.0.1 and SCA 4.1) there can be different technologies (e.g., CORBA, DDS, POSIX IPC, etc.) for the component ports that same component implementation is plugged into.

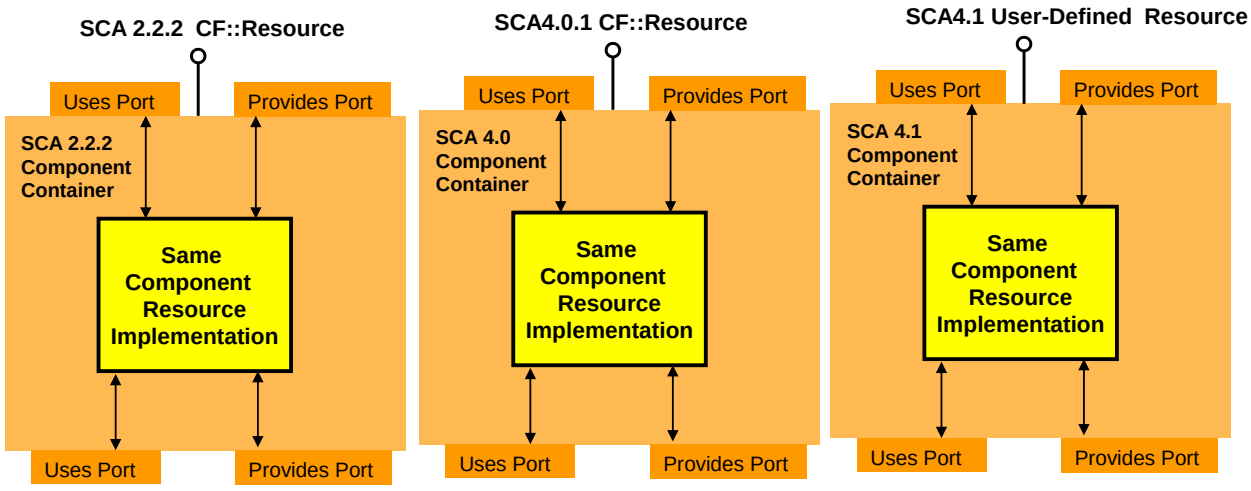

Figure 2. Component Containers Illustration

The following sections give more details on each of the elements of the Component Implementation Design Pattern.

3. COMPONENT CONTAINER

The component container is technology specific with respect to: 1) the component framework (as defined by the standard, e.g. SCA 2.2.2, SCA 4.0, 4.1, etc.) and 2) the middleware technology (e.g., CORBA, DDS, POSIX IPC (queue, shared memory, etc.), etc.). Note that a component may utilize more than one middleware technology. The Component Container Design Pattern from a UML perspective can be viewed as depicted in figure 3 below.

The component container contains a component implementation and component ports. The component container setups the component ports and associates the component ports with the component implementation.

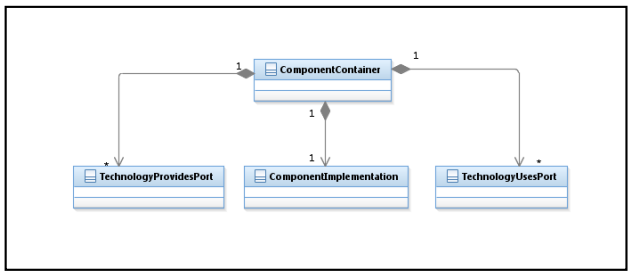

Figure 3. Component Container Design Pattern UML Illustration

The component container also implements CF's interfaces for:

- Life cycle management that relates to SCA CF::LifeCycle interface
- Configuration management that relates to SCA CF::PropertySer interface
- Port management that relates to SCA 4.x CF::PortAccessor interface and SCA 2.2.2 CF::Port and CF::PortSupplier.
- Component identification that relates to SCA 4.x CF::ComponentIdentifier interface and SCA 2.2.2 CF::Resource interface
- Control management that relates to SCA 4 CF::ControllerComponent interface, SCA 4.1 ControllableInterface, and SCA 2.2.2 CF::Resource interface
- Test management that relates to SCA 2.2.2 and 4 CF::TestableObject interface and SCA 4.1 Testable Interface.

The component container transforms CF interface operations into implementation interface operations as described in Component Implementation section 5.

4. COMPONENT PORTS

In order to obtain a component technology neutral implementation, one must place constraints on port interfaces. Object Management Group (OMG) Interface Definition Language (IDL™) [2] is used to define port interfaces. The reason to use IDL to define a port interface is twofold: 1) IDL is an industry standard for specifying an interface and 2) standard mapping of IDL to an implementation language (e.g., C[8], C++[9], Java, Ada, etc.). UML interface definition is not considered since there is a minimum set of UML primitive types defined and there are no standard UML language profiles for translating an interface into code.

In using IDL, one must place constraints on IDL interface and type definitions to avoid the use or reference of CORBA name space in code that adheres to the IDL standard language mappings. These constraints are:

- No CORBA types in interface operations, structs and exception.
- No usage of SCA CF::Properties or DataType since this contains any type.

In order to adhere to these constraints, typedefs for IDL primitive types and primitive sequence types must be used. For example, invalid would be "void setFrequency (in long freq)". The valid interface operation for this would be "void setFrequency (in FrequencyHzType freq)" where FrequencyHzType freq is typedef to unsigned long IDL primitive type.

SCA 4.1 specification appears to be adding typedefs to IDL primitive types and bringing back the primitive sequence types.

The design patterns for the provides and uses ports are described in the following subsections.

4.1. Provides Port Design Pattern

A provides port design pattern, as shown in figure 4 below, consists of:

- Abstract Provides Port Handler is the class that provides interface operations that adheres to IDL standard language mappings and is where the provides interface requests are sent to.
- Abstract Provides Port provides abstraction for all technology specific provides ports and is associated with the Abstract Provides Port Handler.
- Technology Specific Provides Port is middleware technology specific class that handles the incoming technology requests and delegates the request to the Abstract Provides Port Handler.

4.2. Uses Port Design Pattern

A uses port design pattern, as shown in figure 5 below, consists of:

- A Uses Port Base class (Abstract Uses Port) that is technology neutral following CORBA IDL standard language mappings for an interface.
- A Technology Specific Uses Port (e.g., CORBA, DDS, Queues, Device Driver, etc.) class handles outgoing requests and receives requests from a component's implementation class.

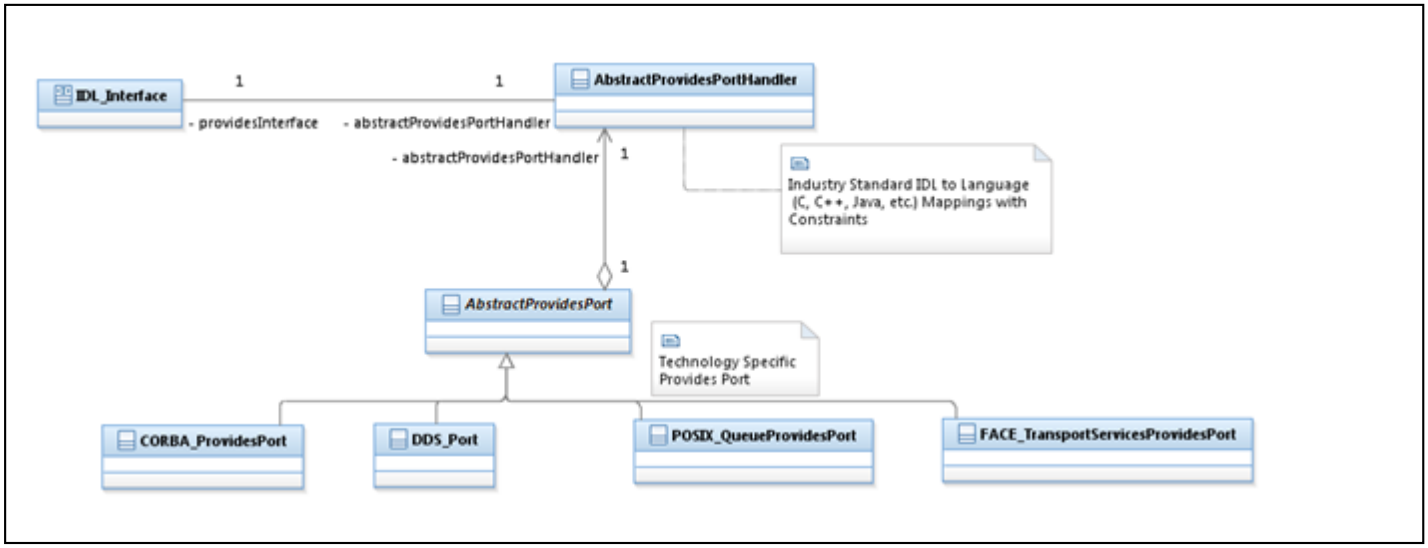

Figure 4. Provides Port Design Pattern

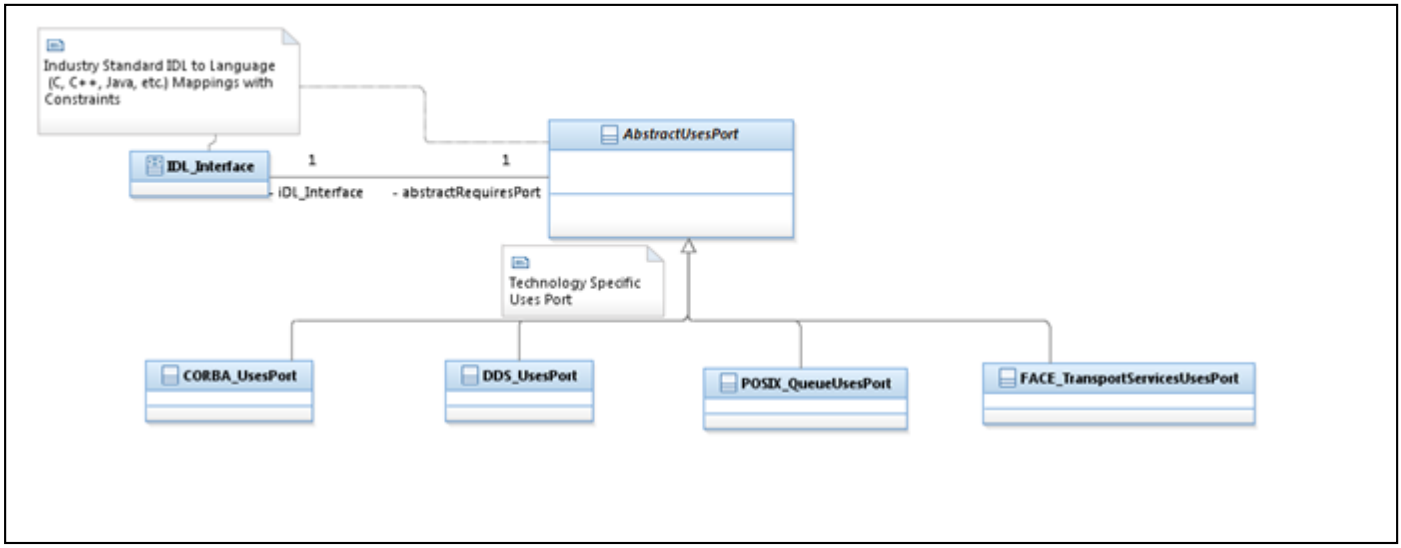

Figure 5. Uses Port Design Pattern

5. COMPONENT IMPLEMENTATION

A component implementation is technology neutral by constraining the usage of specific middleware technology in its business logic. This is accomplished by:

- Implementation Design Pattern
- SCA interface Restrictions
- Middleware Technology usage restriction

The implementation design pattern in conjunction with ports design pattern, as shown in figure 6 below, consists of:

- Abstract Provides Port Handler is inherited by a component implementation. Component implementation handles provides interface requests by its Abstract Provides Port Handler operations that are implemented by component implementation.
- Abstract Uses Port is an attribute of a component implementation. A component implementation sends requests to another component by its Abstract Uses Ports.
- Component Implementation, which is the technology neutral implementation of component business logic.

Component implementation restrictions on the usage of SCA CF interfaces are as follows:

- SCA CF Port interfaces (SCA 2.2.2 CF::Port and CF::PortSupplier, SCA 4.X CF::PortAccessor). Port connections are managed at the component container.
- SCA CF::PropertySet interface. The component implementation contains configure and query property operations that the component container uses for transforming the CF::Properties. The component implementation's configure and query properties operations are based upon POSIX primitive types, component's structure and structure sequence types. The CORBA string type is converted into a programming language string type.
- SCA CF::TestableObject or TestableInterface. The component implementation contains test property operations that the component container uses for transforming the test Properties. The component implementation's test properties operations are based upon POSIX primitive types.

Component implementation restrictions on the usage of middleware technology such as CORBA are as follows:

- CORBA name space
- CORBA operations and types (poa, poa manager, CORBA base object operations).

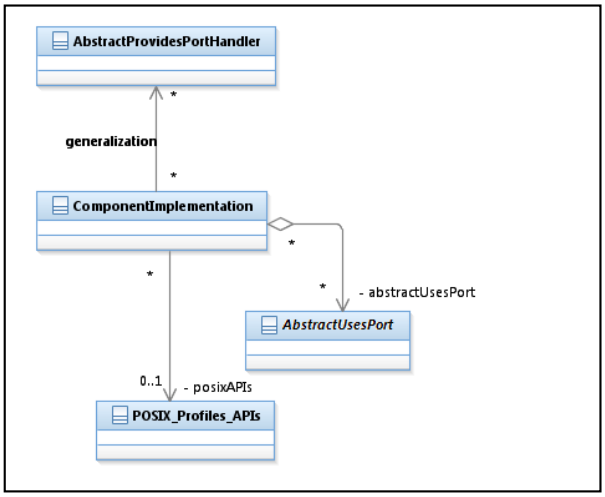

Figure 6. Implementation Design Pattern

7. CONCULSION

This paper described an example component implementation design pattern that shows how a component's implementation can be decoupled from middleware and framework technologies (e.g. utilizing CORBA, DDS, SCA, etc.). This enables a more optimized implementation through the selection of a specific middleware technology, the flexibility to employ multiple technologies, for example the use of CORBA or the control plane and DDS for the data plane, and decrease the cost for code reuse by decoupling implementation from the middleware-specific interfaces and protocols.

The approach is based upon industry standards such as IDL and standard IDL language mappings, and having middleware port abstractions along with restrictions on the usage of middleware and SCA interfaces in implementation logic. This approach allows a technology independent implementation and, therefore, is portable and reusable and still SCA compliant.

8. REFERENCES

[1] Software Communications Architecture (SCA), Version 4.0.1, October 1, 2012.

[2] Common Object Request Broker Architecture (CORBA) Specification, Version 3.1.1 Part 1: CORBA Interfaces, Version 3.2 formal/2011-11-01, November 2011

[3] The Open Group Standard for Information Technology—Portable Operating System Interface (POSIX®) Base Specifications Issue 7 IEEE Std 1003.1, 2013 Edition Copyright © 2001-2013 The IEEE and The Open Group

[4] OMG Data Distribution Service (DDS) for Real-time Systems, Version 1.2, formal/07-01-01, January 2007

[5] Software Communications Architecture (SCA), Version 2.2.2, May 15, 2006

[6] OMG™ Unified Modeling Language™ (OMG UML), Infrastructure, Version 2.4.1 formal/2011-08-05, August 2011.

[7] OMG™ Unified Modeling Language™ (OMG UML), Superstructure, Version 2.4.1 formal/2011-08-06, August 2011.

[8] C Language Mapping Specification, Version 1.0, formal/1999-07-35, July 1999

[9] C++ Language Mapping, Version 1.2 formal/2008-01-09, January 2008