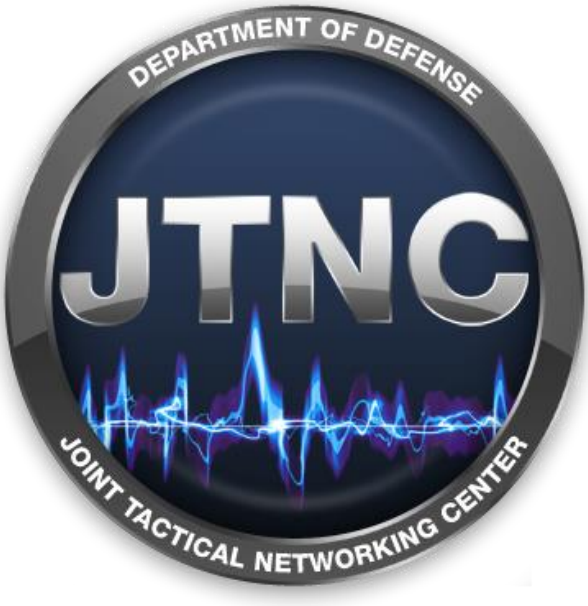
SOFTWARE COMMUNICATIONS ARCHITECTURE V 2.2.2 PRODUCT MIGRATION GUIDE



26 August 2016
Version: 0.1

Prepared by:

Joint Tactical Networking Center
33000 Nixie Way
San Diego, CA 92147-5110

REVISION SUMMARY

Version	Revision
0.1	Initial Release

TABLE OF CONTENTS

TABLE OF FIGURES

1 SCOPE

This Product Migration Guide is an engineering focused document intended to provide practical guidance and suggestions for migrating Software Communications Architecture (SCA) compliant products from version 2.2.2 to 4.1 compliance. It is not a substitute for the SCA specification, but a companion document to highlight items that should be taken into consideration when modernizing an existing implementation.

1.1 INFORMATIVE REFERENCES

The following documents are referenced within this specification or used as reference or guidance material in its development.

[1] Software Communications Architecture Specification, Version 4.1, 20 August 2015.

[2] Software Communications Architecture Specification Version 2.2.2, 15 May 2006.

2 OVERVIEW

SCA 4.1 was published in August of 2015 [1]. The specification incorporates a host of features that facilitate the development and deployment of better performing radio products that are more secure, capable, and cost effective. The current SCA release provides an upgrade of the widely deployed SCA 2.2.2 which was released in May 2006 [2].

A topic of interest associated with SCA 4.1 relates to the question of what differences exist between the specification versions and what steps would be required to migrate SCA 2.2.2 compliant versions to the current SCA version. This document highlights the interface and requirements differences between the specifications and provides general guidance related to the steps that would be required to transition an SCA 2.2.2 product. Since SCA products can be developed using several approaches it is likely that the suggestions contained within this document will not provide a detailed roadmap of all of the steps required to perform the migration of any particular implementation. However, the guidance should identify the majority of conceptual items that will be applicable to products that are migration candidates.

3 SCA 4.1 STRUCTURE

SCA 4.1 has a very different appearance than SCA 2.2.2, but at its core the specification includes the same elements and addresses similar issues. The primary driver behind the cosmetic changes was the introduction of the Component Model within SCA 4.1. Components represent "autonomous units within a system or subsystem" which have the following characteristics:

- Provide one or more Interfaces which users may access, and
- Hide the internal representation and make it inaccessible other than as provided by the Interfaces.

Component definitions reference interface definitions (which may not be component-unique) and specify required behaviors, constraints or associations that must be adhered to when their corresponding products are built.

At a functional level, component specifications differ from their incorporated interfaces because they include the dynamic behavior and semantics that must be provided by the containing entity. SCA 2.2.2 also contained those functional requirements, but no distinction was made between its

"static" and "dynamic" requirements. In some instances, the lack of separation made the specification more difficult to comprehend. Table 1 below highlights the similarities between the SCA 2.2.2 and 4.1 interfaces.

Table 1: Comparison between SCA 4.1 and SCA 2.2.2 Interfaces

SCA 4.1 Interface	SCA 2.2.2 Interface	Similar Across Versions**	Portion of 2.2.2 Interface***	Identical Content*
AdministratableInterface	N/A		X	
AggregateDevice	AggregateDevice	X		
AggregateDeviceAttributes	N/A		X	
ApplicationFactory	ApplicationFactory	X		
ApplicationManager	Application	X		
CapacityManagement	N/A			
ComponentFactory	ResourceFactory	X		
ComponentIdentifier	N/A		X	
ComponentRegistry	N/A			
ControllableInterface	N/A		X	
DeploymentAttributes	N/A		X	
DeviceAttributes	N/A		X	
DomainInstallation	N/A		X	
DomainManager	DomainManager	X		
EventChannelRegistry	N/A		X	
ExecutableInterface	N/A		X	
File	File			X
FileManager	FileManager			X
FileSystem	FileSystem			X
FullComponentRegistry	N/A			
LifeCycle	LifeCycle			X
LoadableInterface	N/A		X	
PortAccessor	Port, PortSupplier	X		
PropertySet	PropertySet			X

ReleasableManager	N/A			
TestableInterface	TestableObject			X
N/A	Device			
N/A	Resource			
N/A	DeviceManager			
N/A	LoadableDevice			
N/A	ExecutableDevice			

*Identical interfaces – self-explanatory, the SCA 2.2.2 implementation can be reused in SCA 4.1.

**Similar interfaces – much of the SCA 2.2.2 implementation can be reused in an SCA 4.1 product, however some elements will need to be developed or removed to account for the SCA 4.1 feature set.

***Portion of SCA 2.2.2 interface – Can be either a similar or identical interface, these interfaces represent an extraction of SCA 2.2.2 concepts within a new, SCA 4.1 interface definition.

4 MIGRATION OF 2.2.2 PRODUCTS

SCA 2.2.2 systems are comprised of Waveforms, Operating Environment components and a Core Framework implementation. In SCA 4.1 terminology this equates to a collection of Base Application Components, Base Device Components, Framework Control Components, Framework Service Components and the underlying services provided by any middleware implementations or the Real-time Operating System. The primary components within these categories are ManageableApplicationComponents, Device Components, ApplicationManagerComponents, ApplicationFactoryComponents, DeviceManagerComponents and DomainManagerComponents.

When migrating a component from SCA 2.2.2 to SCA 4.1 a development team must account for interface changes, requirements changes and design changes. The subsequent text will focus on the interface and requirements changes. There may be some references to design changes, but at best they will be high level because the SCA requirements can be fulfilled using a wide variety of implementation approaches.

4.1 SCA 4.1 COMMON CONSTRUCT – BASECOMPONENT

The BaseComponent construct is reused across many of the SCA 4.1 Components. In large part BaseComponent is equivalent to the composition of the SCA 2.2.2 *Resource* (including all of its inherited interfaces). Consequently, many of the same elements and techniques are involved in the migration process. This section captures the activities required to migrate the “BaseComponent” portion of an SCA 2.2.2 component.

The SCA 2.2.2 base entity is the *Resource* interface that is shown in Figure 1.

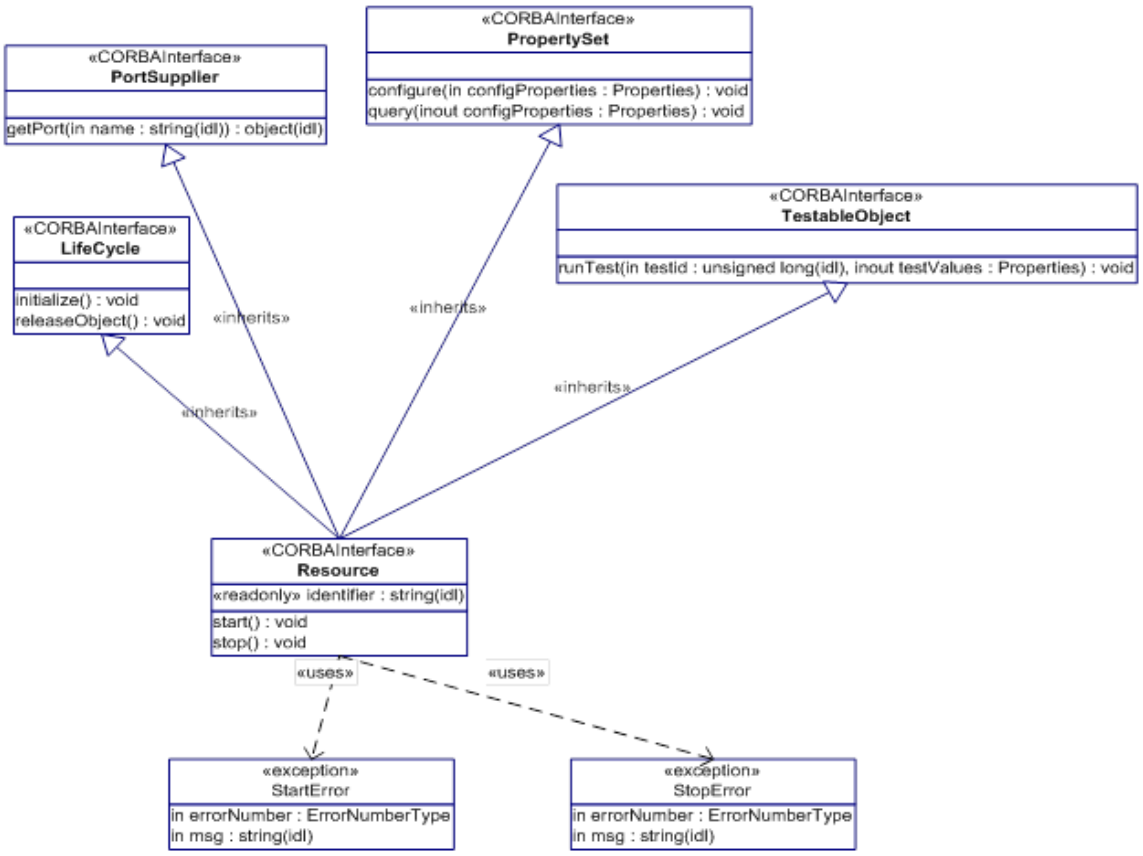


Figure 1: SCA 2.2.2 *Resource* Interface

In SCA 4.1 this functionality was componentized and encapsulated within the BaseComponent construct. The original *Resource* interface inherited from several smaller interfaces whereas the new BaseComponent, Figure 2, is an aggregation of optional interfaces. This design pattern is extended throughout SCA 4.1. In SCA 2.2.2, more complex interfaces such as *Device* inherit from *Resource*. In SCA 4.1, there is no *Device* interface, but the DeviceComponent inherits the decomposed interfaces previously encapsulated by *Resource*.

When porting from SCA 2.2.2 to SCA 4.1, the opportunity exists to delete unnecessary interfaces to reduce code complexity. As an example, many SCA 2.2.2 implementations merely stubbed out the *TestableObject* interface. Depending upon the system design, <<TESTABLE>> may or may not be supported, and thus represents an opportunity to eliminate the interface.

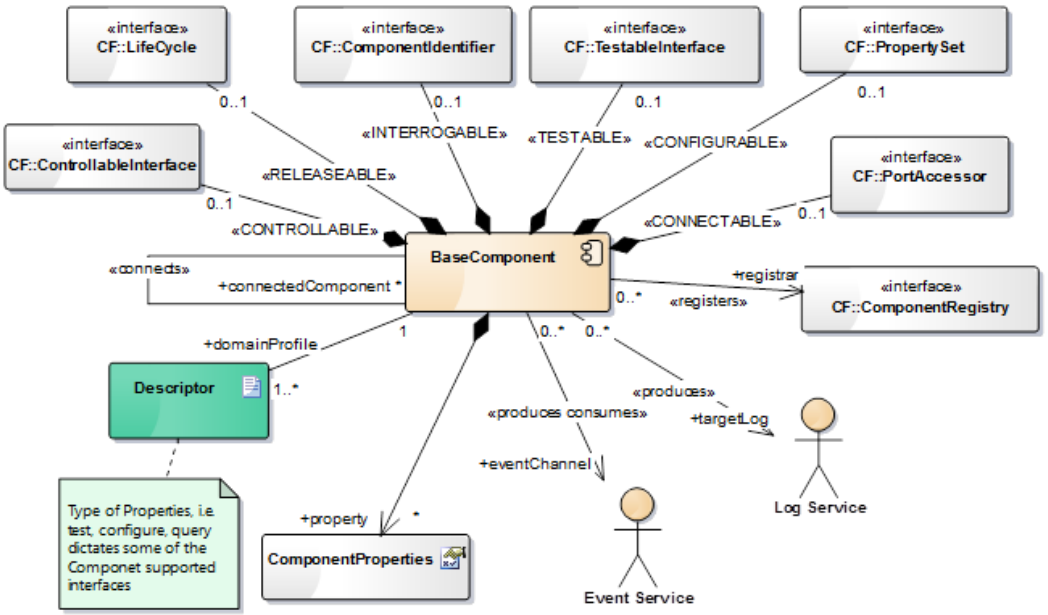


Figure 2: SCA 4.1 Base Component

The following interfaces are part of the *Resource* interface: *Resource*, *LifeCycle*, *PropertySet*, *PortSupplier* and *TestableObject*. The following sections will provide a comparison of those interfaces.

4.1.1 Interface Changes

4.1.1.1 Resource

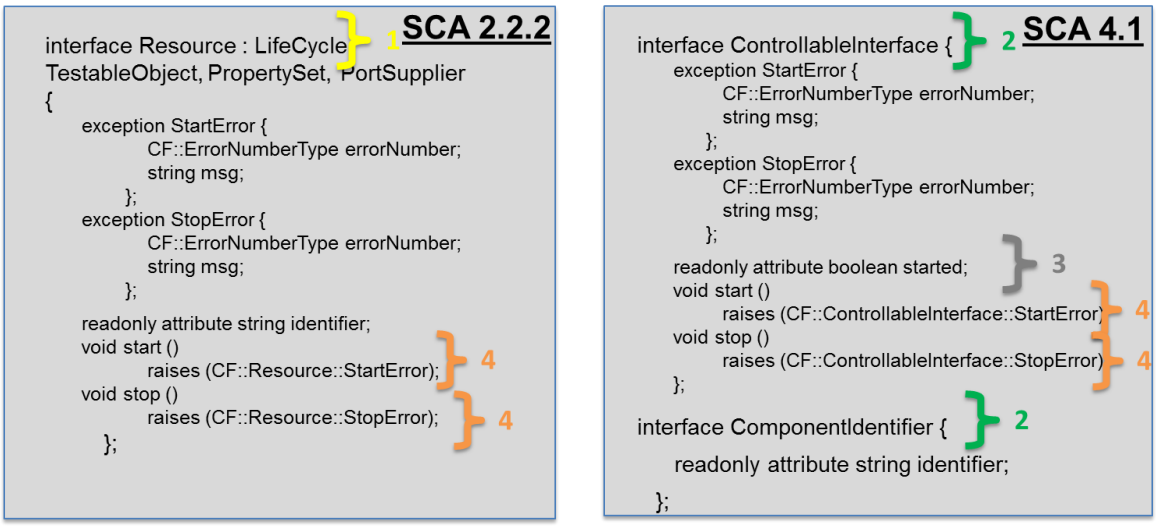


Figure 3: *Resource* Interface Comparison

1. SCA 4.1 removes the *Resource* interface.
2. SCA 4.1 introduces the new *ControllableInterface* and *ComponentIdentifier* interfaces in place of Resource.

3. SCA 4.1 introduces the new started attribute within the *ControllableInterface* interface.
4. SCA 4.1 re-scopes the exceptions from the Resource interface to the *ControllableInterface* interface.

4.1.1.2 LifeCycle

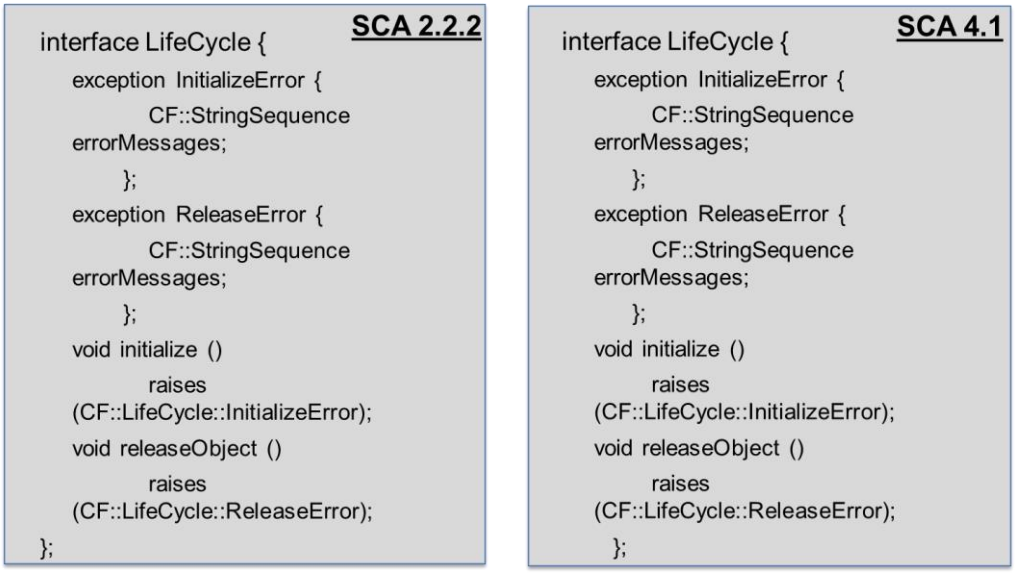


Figure 4: *Lifecycle* Interface Comparison

The interfaces are identical.

4.1.1.3 PropertySet

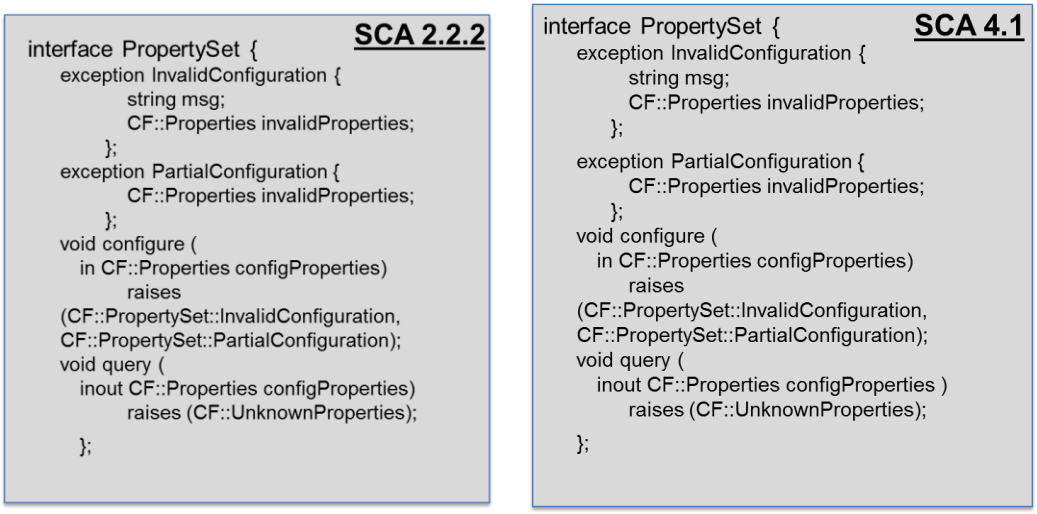


Figure 5: *PropertySet* Interface Comparison

The interfaces are identical.

4.1.1.4 PortSupplier

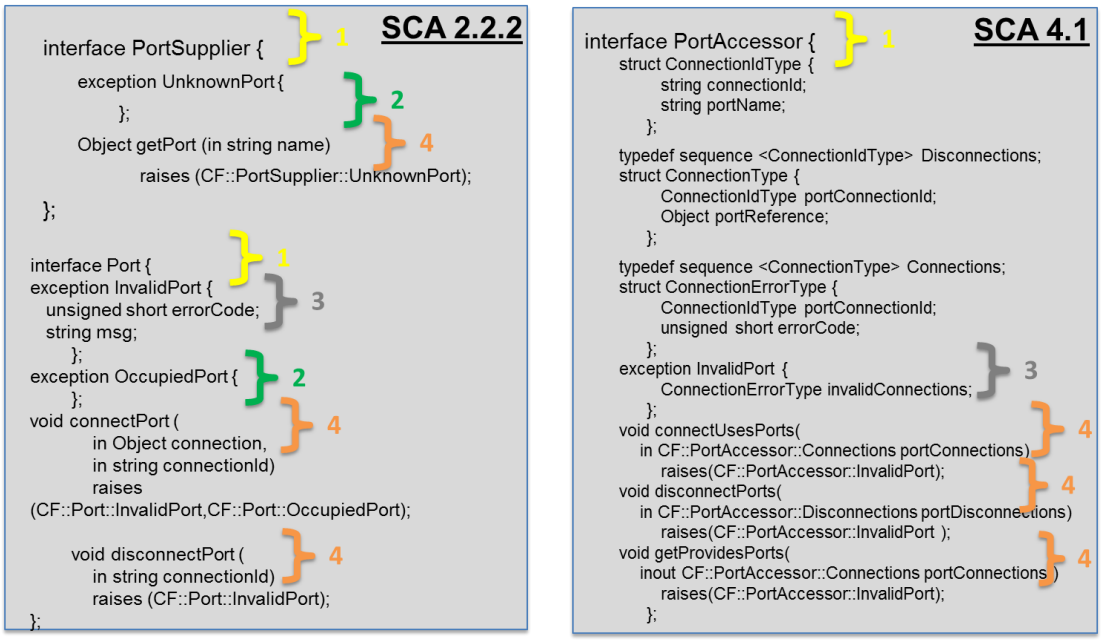


Figure 6: Port Interfaces Comparison

1. SCA 4.1 collapses the functionality of the *Port* and *PortSupplier* interfaces and combines it within the *PortAccessor* interface.
2. SCA 4.1 eliminates the OccupiedPort exception and reflects its semantics within the InvalidPort exception.
3. SCA 4.1 refactors the composition of the InvalidPort exception variable to a ConnectionErrorType.
4. SCA 4.1 refactors port operations to enable multiple connections to be managed on a single call (e.g. connectUsesPorts rather than connectPort).

4.1.1.5 TestableObject

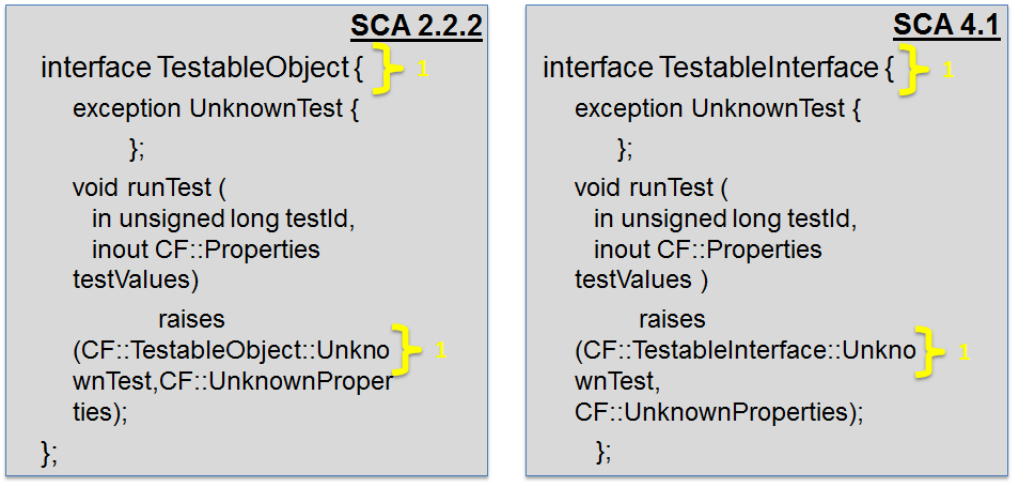


Figure 7: Test Interface Comparison

1. SCA 4.1 renames the *TestableObject* interface to *TestableInterface.*

4.1.2 Implementation Changes

4.1.2.1 Requirements Driven

4.1.2.1.1 Resource

SCA 4.1 introduces three new requirements, SCA32, SCA33 and SCA36 that are associated with the new started attribute. The implementation requirements associated with this change should be minimal.

4.1.2.1.2 PortSupplier

The 4.1 specification includes three new requirements SCA11, SCA14 and SCA519 that are a result of the port restructure, two of which are a byproduct of the fact that the operations need to accommodate multiple ports. This change will likely result in a moderate change to an existing implementation.

4.1.2.2 Structural

An SCA 2.2.2 component that uses the *Resource* interfaces will require the following changes in order for it to be migrated to SCA 4.1 compliance:

1. Change any scoped or qualified *TestableObject* references should be to *TestableInterface.*
2. Change any scoped or qualified *Port* or *PortSupplier* references to *PortAccessor.*
3. Change any use of the InvalidPort exception to represent the exception's new format.
4. Update any use of the OccupiedPort or UnknownPort exceptions to become InvalidPort exceptions.
5. Rename the *PortSupplier::getPort* operation to *PortAccessor::getProvidesPorts* and update its implementation to support requests for multiple provides ports.

6. Update the *PortAccessor* connectPort and disconnectPort operations to delegate calls to the converted (references and servant classes) non-CORBA (e.g. cpp) *Port* class.
7. Rename the connectPort operation to connectUsesPorts and update it to support requests for multiple ports.
8. Rename the disconnectPort operation, will need to be renamed to disconnectPorts and update it to support requests for multiple ports.
9. Re-scope any use of the StopError or StartError exceptions to a definition within the *ControllableInterface* interface.
10. Update any use of the stop or start operations to reflect a location within the *ControllableInterface* interface.
11. Extend the implementation to include the *ControllableInterface* started attribute, a Boolean attribute that is set and unset when stop and start are called.

4.2 SCA 4.1 MANAGEABLEAPPLICATIONCOMPONENT

SCA 2.2.2 applications or waveforms (ApplicationComponents) realize the *Resource* and optionally *ResourceFactory* interfaces that are illustrated in Figure 8. Application components, within both SCA 2.2.2 and SCA 4.1 support the same core capabilities of:

- Configuration management
- Operations management
- Life cycle support
- Connectivity management
- Test management

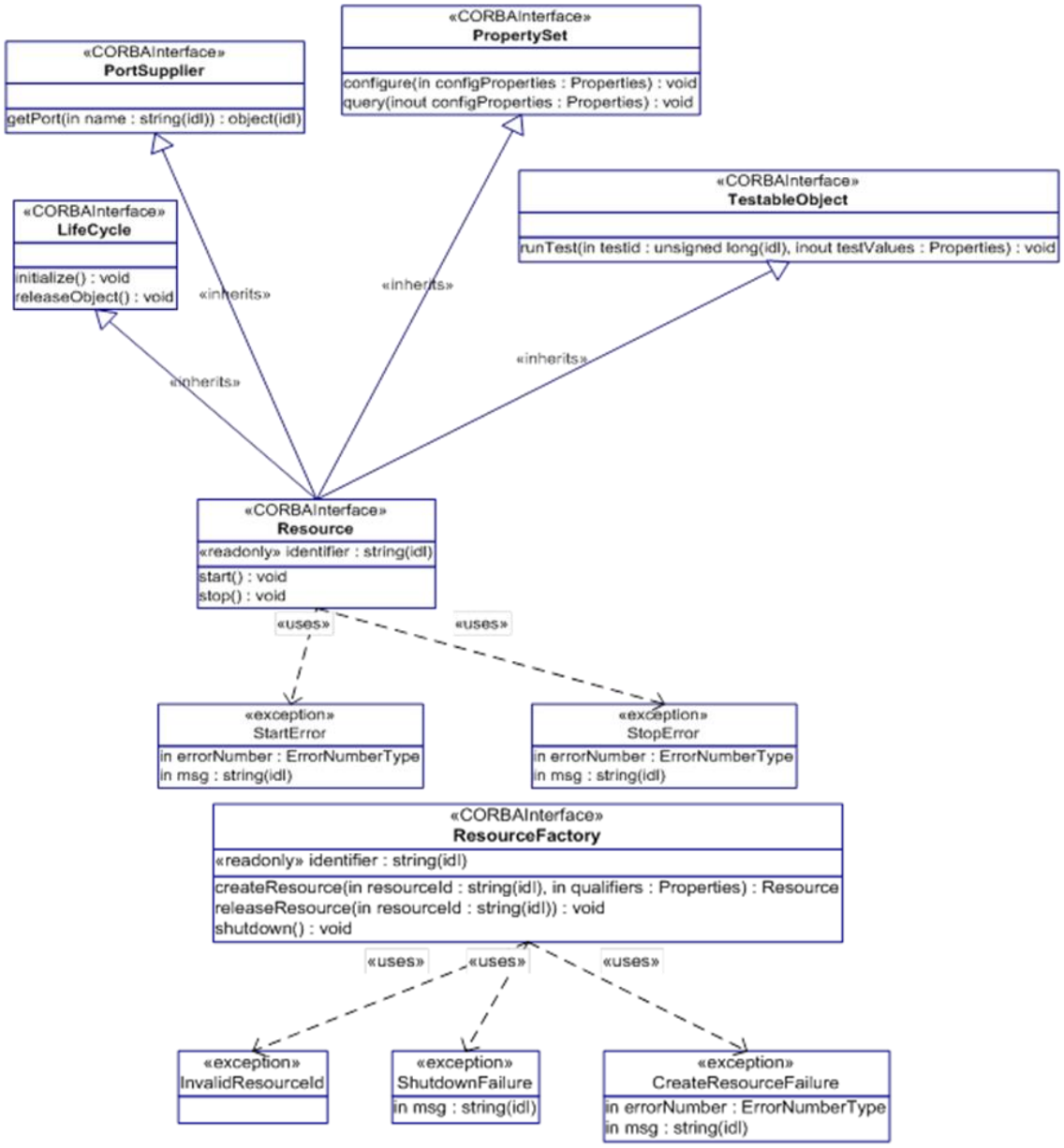


Figure 8: SCA 2.2.2 *Resource* and *ResourceFactory* Interfaces

As illustrated earlier, *Resource* is a monolithic interface that incorporates several lower level interfaces. An SCA 2.2.2 waveform is composed of multiple application components that utilize the capabilities and services provided by the platform's operating environment.

Within SCA 4.1 componentization the *Resource* interface was removed and the developer has the responsibility of defining Component interfaces, which when realized provides equivalent functionality. In other words, the developer must build the component (as an example, the application component) with the BaseComponent interfaces. Unlike the paradigm of SCA 2.2.2, there is no BaseComponent interface (or corresponding *.idl) to inherit.

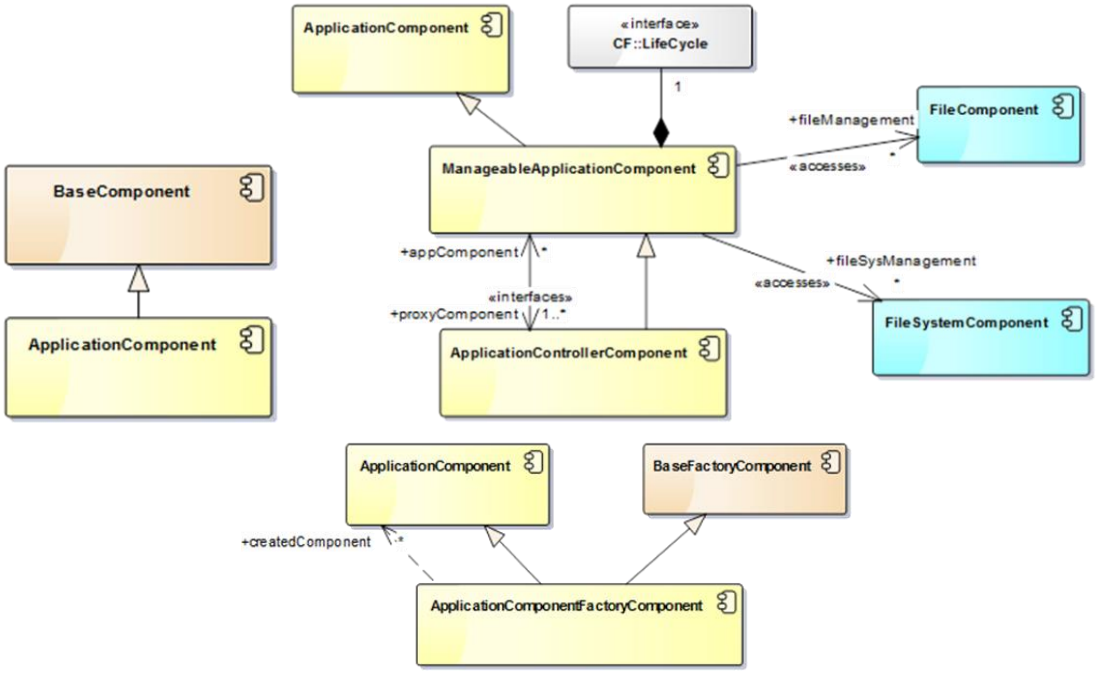


Figure 9: SCA 4.1 ManageableApplicationComponent

An SCA 4.1 developer could define the following interface which effectively mimics the 2.2.2 *Resource* (there is a difference in the identifier definition which will be accounted for):

```
interface BaseResource : LifeCycle, TestableInterface,
PropertySet, PortAccessor, ControllableInterface
```

Utilizing the SCA 4.1 "Resource", which is equivalent to a 2.2.2 *Resource* (*LifeCycle* is picked up through BaseResource), as the basis for an SCA 4.1 ApplicationComponent definition, as shown in Figure 9, is an approach that may be employed to minimize the changes required to migrate an application.

Similar to SCA 2.2.2, an ApplicationComponentFactoryComponent (*ResourceFactory*), can be incorporated optionally as part of an application if desired.

4.2.1 Interface Changes

The SCA 4.1 ApplicationComponent does not introduce additional interface changes beyond those introduced by the BaseComponent.

4.2.2 Implementation Changes

4.2.2.1 Requirements Driven

SCA 2.2.2 applications contain approximately 75 requirements and that count drops to about 70 in SCA 4.1. In actuality there is a greater disparity as there are structural and modeling oriented requirements such as SCA550 (a ManageableApplicationComponent shall realize the *LifeCycle* interface) that are included within that count. Once those are removed, there are about 58 requirements allocated to each component.

SCA 4.1 introduces one new requirement, SCA82, at the application level beyond those of the BaseComponent. SCA82 requires an application component to register with a component registry

rather than a naming service. The mechanics of this functionality are similar, the component is provided with a reference that it uses to perform the registration. When the component registers with the *ComponentRegistry* it needs to provide a populated ComponentType structure to utilize the new push model registration. The SCA 4.1 implementation will also need to remove any code that was associated with the Naming Service. Thus the level of effort and associated with this change should be moderate.

4.2.2.2 Structural

Another instance where the SCA 4.1 Component Model differs from an SCA 2.2.2 "component" is that in SCA 4.1 a Waveform developer will need to define their own interface(s) to represent their waveform components because they are not provided by the framework. As an example, the developer might choose to extend the *BaseResource* interface described earlier to create a Waveform specific interface as follows:

```
interface MyWaveform1 : BaseResource
```

An SCA 2.2.2 application component that uses the *Resource* interfaces will require the following changes beyond those required of a BaseComponent in order to be migrated to SCA 4.1 compliance:

1. Modify any interfaces associated with an SCA component to inherit from a non-CORBA *CF::Port* equivalent in order to minimize changes to an existing implementation.
2. Update any use of the identifier attribute to its new location within *CF::ComponentIdentifier*.
3. Review the use of AEP operations to ensure that they are all still in accordance with the selected profile.

4.3 SCA 4.1 DEVICE COMPONENT

SCA 2.2.2 devices (DeviceComponents) realize the *Device* interface, see Figure 10, which inherits the *Resource* interface. SCA 2.2.2 and 4.1 devices support the same basic capabilities:

- Capacity management
- Configuration management
- Operations management
- Life cycle support
- Connectivity management
- Test management

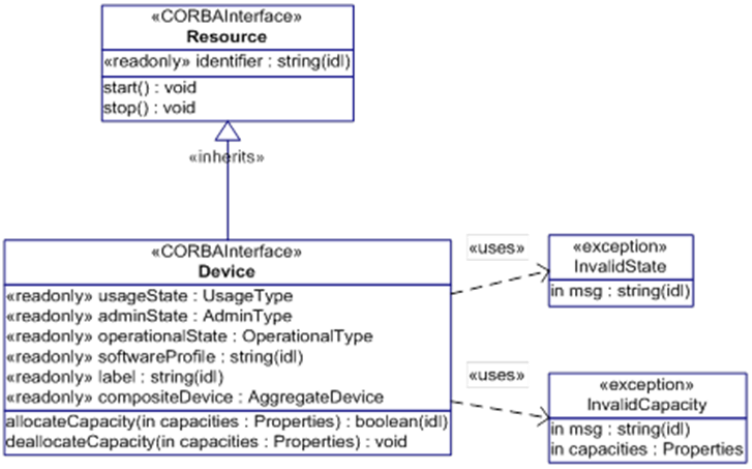


Figure 10: SCA 2.2.2 *Device* Interface

Within SCA 4.1 componentization the *SCA 2.2.2 hierarchy of Device, Loadable Device, and Executable Device was removed.* Instead, SCA 4.1 introduces the DeviceComponent, LoadableDeviceComponent and ExecutableDeviceComponent. As with BaseComponent, there is no encompassing interface for DeviceComponent, Figure 11.

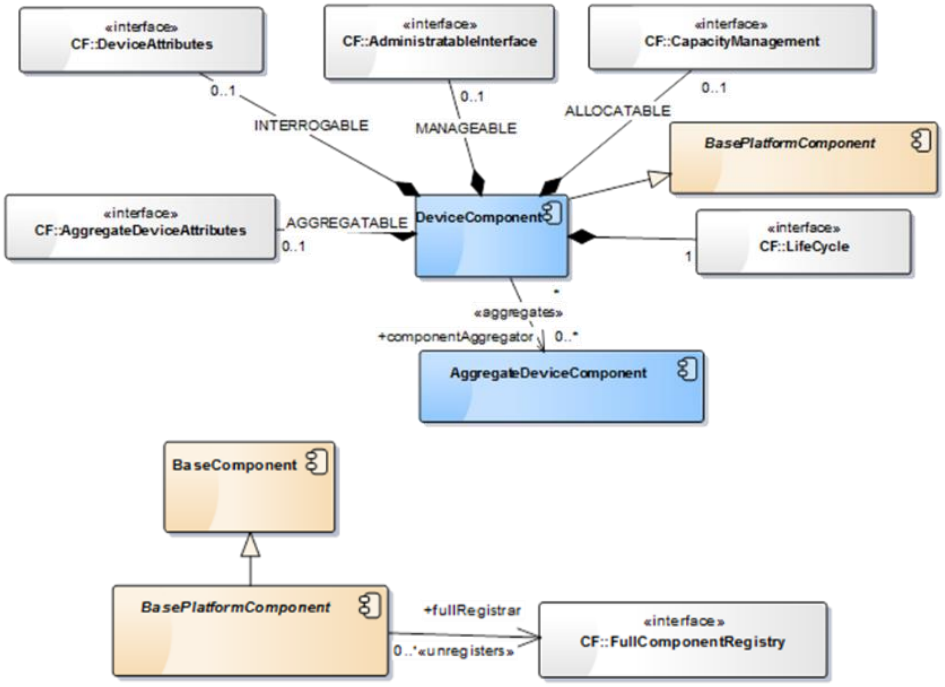


Figure 11: SCA 4.1 DeviceComponent

A developer could leverage the SCA 4.1 "Resource" (BaseResource) to create a compliant interface which is equivalent to a 2.2.2 *Device* (*LifeCycle* is picked up through BaseResource):

```
interface BaseDevice: BaseResouce, DeviceAttributes,
AdministrableInterface, CapacityManagement
```

The *AggregateDevice* association can be provided at the component level if needed.

4.3.1 Interface Changes

4.3.1.1 Device

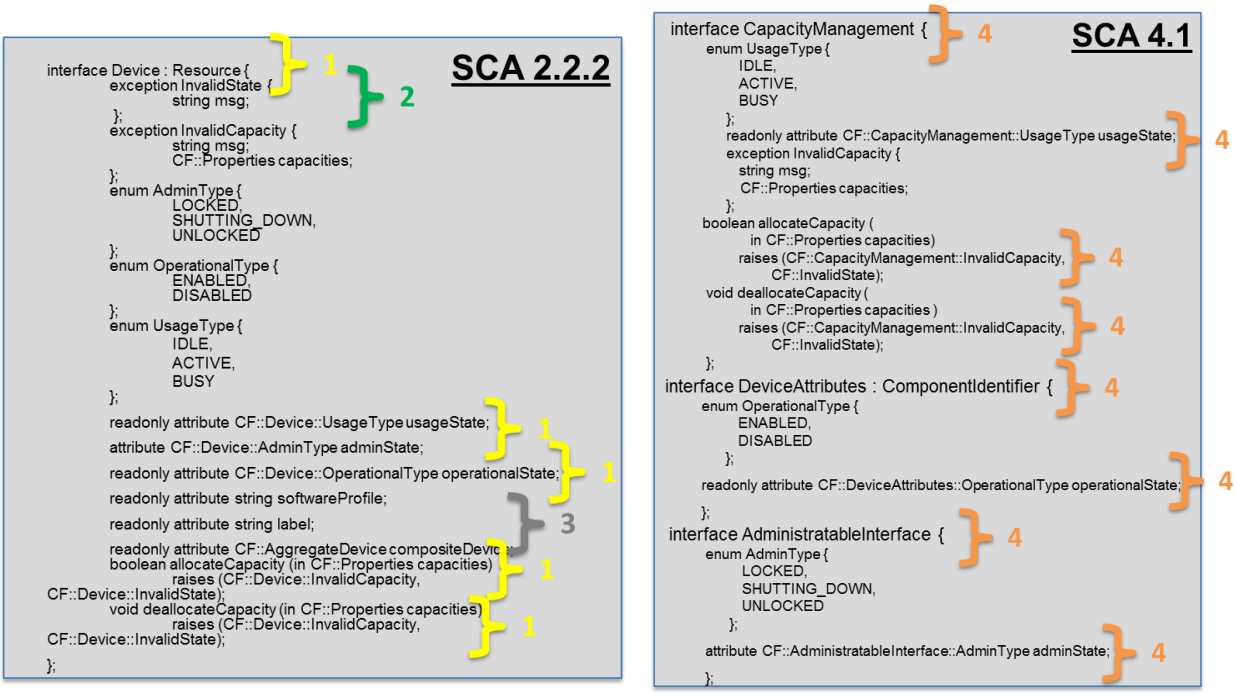


Figure 12: *Device* Interface Comparison

1. SCA 4.1 removes the *Device* interface and its *Device* scoped attributes and exceptions.
2. SCA 4.1 relocates the InvalidState exception to the CF:: name scope.
3. SCA 4.1 eliminates the Device interface softwareProfile and label attributes, softwareProfile moves to the ComponentType structure and label is removed.
4. SCA 4.1 refactors *Device* interface into three new interfaces *AdministrableInterface*, *CapacityManagement* and *DeviceAttributes*, and replaces and of the corresponding scoped names for attributes, exceptions or data types.

4.3.2 Implementation Changes

4.3.2.1 Requirements Driven

SCA 2.2.2 devices contain approximately 98 requirements, that count increases to 99 in SCA 4.1. However, when the structural and modeling oriented requirements are removed, there are roughly 84 requirements allocated to each device component.

SCA 4.1 introduces one new requirement, SCA298, at the device level beyond those of the BaseComponent. SCA298 requires a DeviceComponent to register with a component registry rather than a naming service. The mechanics of this functionality are similar, the component is provided with a reference that it uses to perform the registration. When the component registers with the *ComponentRegistry* it needs to provide a populated ComponentType structure to utilize

the new push model registration. The SCA 4.1 implementation will also need to remove any code that was associated with the Naming Service. Thus the level of effort and associated with this change should be moderate.

4.3.2.2 Structural

An SCA 4.1 device developer will need to define their own interface(s) to represent their device components because they are not provided by the framework. As an example, the developer might choose to extend the *BaseDevice* interface to create a Platform Operating Environment specific interface as follows:

```
interface MyDecoderDevice : BaseDevice
```

An SCA 2.2.2 component that uses the *Device* interface will require the following changes beyond those required of a BaseComponent to be migrated to SCA 4.1 compliance:

1. Modify any interfaces associated with an SCA component could be modified to inherit from a non-CORBA *CF::Port* equivalent in order to minimize changes to an existing implementation.
2. Update any use of the identifier attribute to its new location within *CF::ComponentIdentifier*.
3. Revise any use of the InvalidState exception to reflect its new location within the CF module.
4. Update the use of the adminState, usageState or operationalState attributes to reflect their location within the new CF interfaces.
5. Re-scope any use of the InvalidCapacity exception to reflect its definition within *Device::CapacityManagement.*
6. Relocate the implementation of the allocateCapacity and deallocateCapacity operations to the *Device::CapacityManagement* interface.
7. Rename any use of the softwareProfile attribute (to profile) and update its scoping in accordance with its location within the ComponentType structure.
8. Remove any use of the label attribute.
9. Integrate any implementation of an *AggregateDevice* at the component level, and incorporate the necessary changes, e.g. forming associations, to represent its new location.

Similar distinctions exist within the comparison of SCA 4.1 LoadableDeviceComponents and ExecutableDeviceComponents with their SCA 2.2.2 counterparts and their migration should be able to be performed with a comparable level of effort.

4.4 SCA 4.1APPLICATIONMANAGERCOMPONENT

SCA 2.2.2 applications (ApplicationManagerComponents) realize the *Application* interface, shown in Figure 13, which inherits the SCA 2.2.2 *Resource* interface. Both SCA 2.2.2 and 4.1 application managers support the same basic capability:

- Provides the Core Frameworks proxy to access an independently developed SCA application

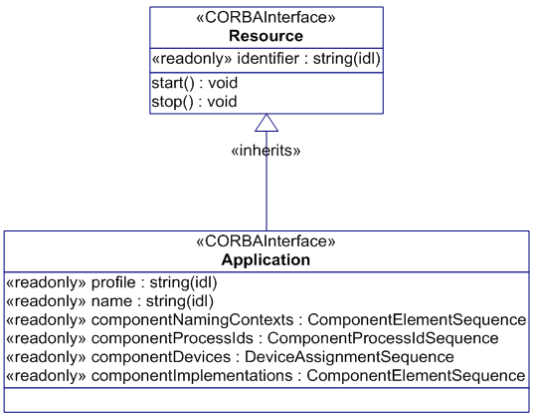


Figure 13: SCA 2.2.2 *Application* Interface

Application is a monolithic interface which incorporates several lower level interfaces via its inheritance of the *Resource* interface.

Following the pattern of application (i.e. waveform) migration from SCA 2.2.2., an SCA 4.1 developer has the responsibility of defining the Component interface, which when realized provides the specified functionality.

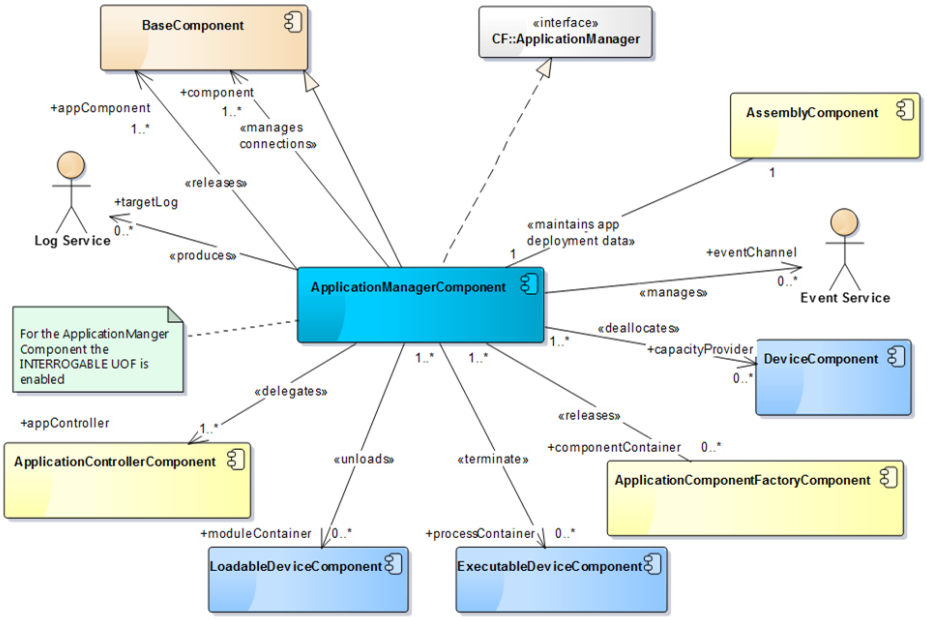


Figure 14: SCA 4.1 ApplicationManagerComponent

An SCA developer could define the following interface which could be utilized to manage applications:

```
interface myApplicationManager : CF::ApplicationManager
```

An ApplicationManagerComponent, which is illustrated in Figure 14, inherits the functions and capabilities of a BaseComponent and can be managed as such. It is worth noting that the *ApplicationManager* interface, in its role as a proxy, is also a monolithic interface as it must be able to support the delegation of operations to any of the instantiated applications that it manages.

4.4.1 Interface Changes

4.4.1.1 Application

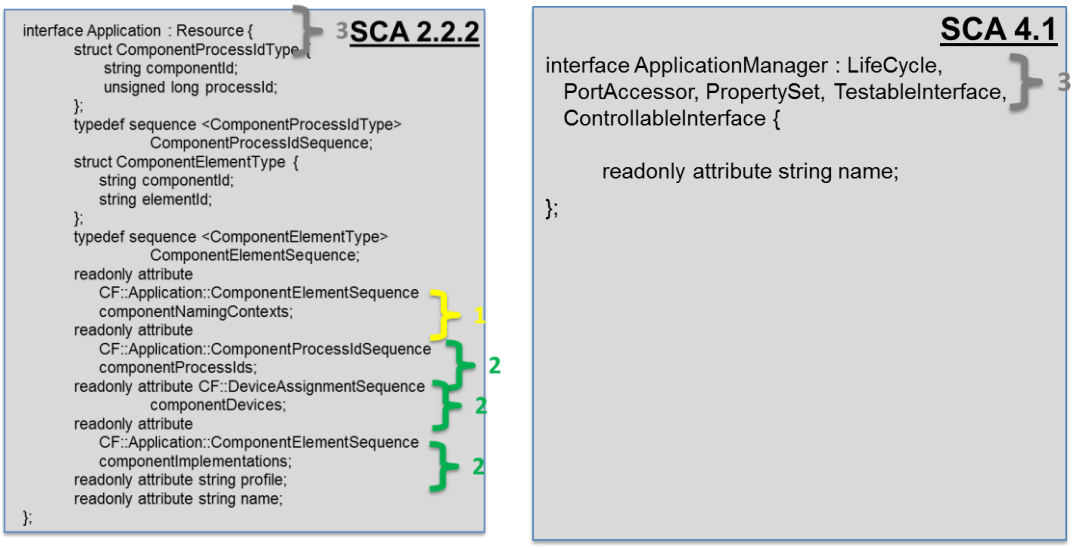


Figure 15: *Application* Interface Comparison

1. SCA 4.1 removes componentNamingContexts interface which was associated with the Naming Service.
2. SCA 4.1 removes the componentImplementations attribute and relocates the profile, componentDevices and componentProcessIds attribute information within the ComponentType structure.
3. SCA 4.1 renames the *Application* interface to *ApplicationManager* and modifies its inheritance to reflect the removal of the Resource interface.

4.4.2 Implementation Changes

4.4.2.1 Requirements Driven

SCA 2.2.2 applications contain approximately 114 requirements, and the count decreases to 83 in SCA 4.1. However, when the structural and modeling oriented requirements are removed, the count is diminished even more and there are approximately 68 requirements allocated.

SCA 4.1 introduces eight new *ApplicationManager* requirements beyond those of the BaseComponent. Four of the new requirements, SCA55, SCA58, SCA59 and SCA523 are associated with an ApplicationManagerComponent's role in establishing and destroying connections to external components. The impact of these changes should be minimal, as the behavior should mimic the connection logic required for a BaseComponent. The other four requirements SCA161, SCA162, SCA163 and SCA543 provide clarification of the ApplicationManagerComponent's role in delegating operations to the application components that it manages. The implementation of the requirements should result in a minimal to moderate level of effort as they introduce new, although not too complex, logic for features such as multiple assembly controllers.

4.4.2.2 Structural

An SCA 4.1 Core Framework developer that realizes the *ApplicationManager* interface will need to define their own interface(s) to represent the ApplicationManagerComponent, for example the *myApplicationManager* interface described earlier.

An SCA 2.2.2 component that implements the *Application* interface will require the following changes beyond those required of a BaseComponent to be migrated to SCA 4.1 compliance:

1. Update any use of the *Application* interface to *ApplicationManager.*
2. Eliminate any use of the *Resource* name will and have the name go directly to one of the Base Application interface names.
3. Modify any interfaces associated with an SCA component to inherit from a non-CORBA *CF::Port* equivalent in order to minimize changes to an existing implementation.
4. Update any use of the identifier attribute to its new location within *CF::ComponentIdentifier.*
5. Remove any reference to the namingContext attribute, or other naming service related concept (if the implementation is going to implement backwards compatibility then this logic should be preserved).
6. Any usage of the componentProcessIds, componentDevices or componentImplementations attributes is integrated within the ApplicationFactoryComponent's population of the ApplicationManagerComponent's ComponentType representation.

4.5 SCA 4.1 APPLICATIONFACTORYCOMPONENT

SCA 2.2.2 application factories (ApplicationFactoryComponents) realize the *ApplicationFactory* interface, which is shown in Figure 16. SCA 2.2.2 and 4.1 application factories support the same basic capabilities:

- Application deployment
- Application component connection, initialization and configuration

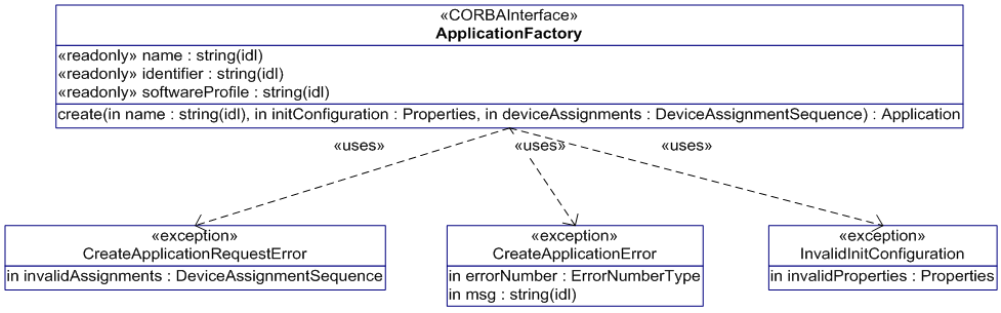


Figure 16: SCA 2.2.2 *ApplicationFactory* Interface

From a functional perspective, SCA 4.1 ApplicationFactoryComponents, see Figure 17, are very similar to their SCA 2.2.2 counterparts. The primary distinction is that the ApplicationFactoryComponent has an associated registry, *ComponentRegistry*, with which its deployed ApplicationComponents register, as opposed to registering with a Naming Service.

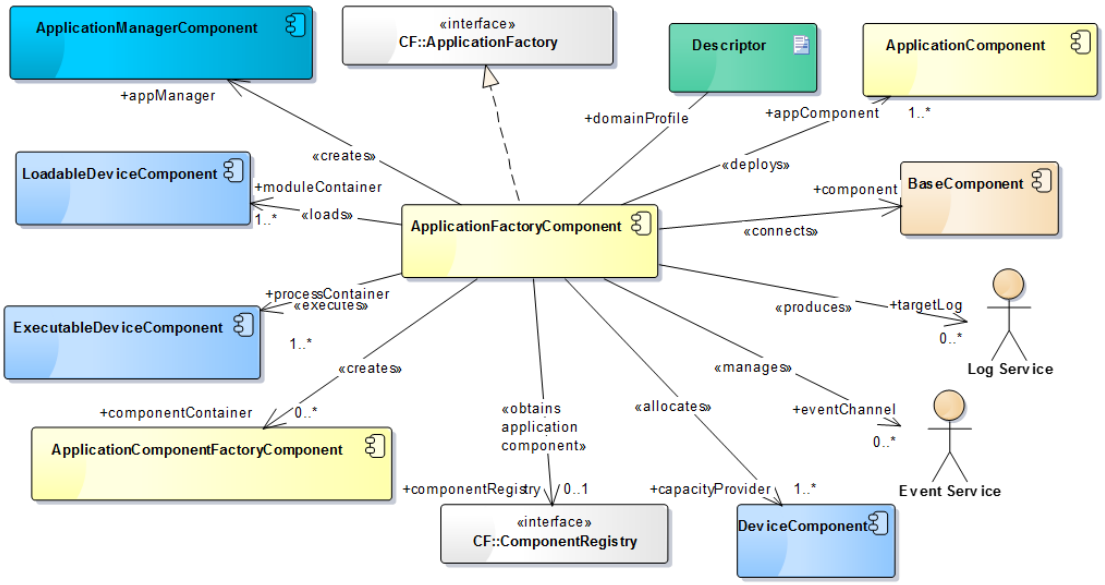


Figure 17: SCA 4.1 ApplicationFactoryComponent

The ApplicationFactoryComponent is unique in that it is not a BaseComponent, a fact that serves to minimize some of the differences that would be encountered when migrating an SCA 2.2.2 implementation.

4.5.1 Interface Changes

4.5.1.1 ApplicationFactory

SCA 2.2.2

```
interface ApplicationFactory {

        exception CreateApplicationRequestError {
                CF::DeviceAssignmentSequence
                        invalidAssignments; };
        exception CreateApplicationError {
                CF::ErrorNumberType errorNumber;
                 string msg; };
        exception InvalidInitConfiguration {
                 CF::Properties invalidProperties; };

readonly attribute string name;
readonly attribute string identifier;
readonly attribute string softwareProfile;
};
```

1

SCA 4.1

```
interface ApplicationFactory {
        exception CreateApplicationRequestError {
                CF::DeviceAssignmentSequence
                  invalidAssignments;  };
        exception CreateApplicationError {
                CF::ErrorNumberType errorNumber;
                  string msg; };
        exception InvalidInitConfiguration {
                CF::Properties invalidProperties; };

        readonly attribute string name;
        struct ExecutionAffinityType
        {
                string componentId;
                string processCollocation;
                CF::ULongSeq coreAffinities;
        };
        typedef sequence <ExecutionAffinityType>
        ExecutionAffinitySequence;
```

2

Figure 18: *ApplicationFactory* Interface Comparison

1. SCA 4.1 eliminates the identifier attribute and moves the softwareProfile to the ComponentType structure.
2. SCA 4.1 introduces new data types for multi-core processor support.

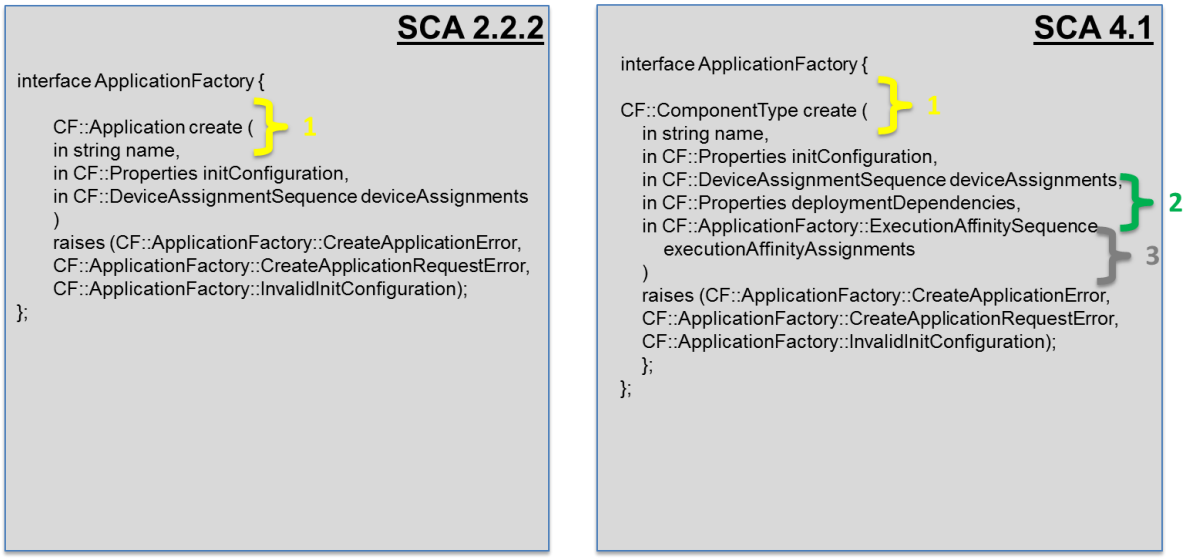


Figure 19: *ApplicationFactory* Interface Operation Comparison

1. SCA 4.1 modifies the create operation's return value from an *Application* interface to a ComponentType structure.
3. SCA 4.1 adds the deploymentDependencies parameter to enable enhanced deployment support.
4. SCA 4.1 introduces the executionAffinityAssignments parameter for multi-core processor support.

4.5.2 Implementation Changes

4.5.2.1 Requirements Driven

The SCA 2.2.2 application factory contains approximately 35 requirements. The SCA 4.1 ApplicationFactoryComponent appears to have many more requirements with 64, or when the structural and modeling oriented requirements are removed 63.

However, many of the 35 new *ApplicationFactory* requirements would not need to be implemented in a scenario where an SCA 2.2.2 implementation was being migrated to SCA 4.1 because they are associated with features that were not available within the older specification. 16 requirements, SCA84*, SCA68*, SCA71*, SCA72*, SCA73*, SCA76*, SCA81*, SCA83*, SCA85*, SCA69*, SCA70*, SCA77*, SCA86*, SCA87*, SCA98*, SCA524*, are associated with Application Backwards Compatibility – an SCA 4.1 Core Framework managing SCA 2.2.2 Applications; SCA70 was introduced to support sub-applications within an application, nested deployment; SCA575 was introduced to in support to allow an ApplicationFactoryComponent to deploy operations and utilize the capabilities of multi-core processors, Core Affinity; and nine requirements, SCA92, SCA93, SCA94, SCA95, SCA96, SCA97, SCA105, SCA106, SCA98 are associated with the SCA 2.2.2 Channel Extension and could be reused if the 2.2.2 product implemented the extension. Therefore, the SCA 4.1 ApplicationFactoryComponent effectively introduces eight new requirements.

Three of the new requirements, SCA77, SCA86 and SCA87 are associated with component identifiers and should result in a minimal change for a CF developer as identifier because the logic to create an identifier exists at other locations within the Core Framework and can be reused. SCA69 instructs the developer on how to handle the deploymentDependencies parameter. This change should also be relatively straightforward as it can reuse or leverage other code which accommodates property precedents. SCA576 dictates how an ApplicationFactoryComponent should store information about its deployed components and this should be a trivial extension to the ComponentType structure. SCA570 requires the ApplicationFactoryComponent to throw an exception if the ApplicationManagerComponent already exits. This should be a simple extension to throw the exception, as it is likely that logic already exists to check the value. SCA542 modifies the parameters passed to an executable device to include a reference to the *ComponentRegistry* instance, which should be an easy modification to the existing execute call. Lastly, SCA555 introduces a check that instructs the ApplicationFactoryComponent on when it should instantiate an SCA 2.2.2 application. This final change should also be relatively simple because most Core Framework implementations know how to extract and process domain profile information.

4.5.2.2 Structural

An SCA 4.1 Core Framework developer that realizes the *ApplicationFactory* interface will need to define their own interface(s) to represent the ApplicationFactoryComponent, for example the myApplicationFactory interface defined below, because one is not provided by the framework.

```
interface myApplicationFactory : CF::ApplicationFactory
```

If the objective of the migration is to transition the existing implementation to SCA 4.1 and minimize the resources required, then the new SCA 4.1 ApplicationFactoryComponent features of Channel Extension, Nested Deployment, Multicore Support and Application Backwards Compatible will not be implemented.

1. Eliminate the identifier and softwareProfile ApplicationFactory interface attributes and reconstitute them as fields within the ApplicationFactoryComponent's ComponentType representation.
2. Refactor the application's proxy object that is instantiated by the ApplicationFactory from a realized Application to *ApplicationManager* interface.
3. Refactor the ApplicationFactoryComponent to construct and populate a ComponentType structure.
4. Modify the create operation's return type from an Application object to a ComponentType structure which represents the *ApplicationManager*.
5. Modify the create operation implementation to accommodate the existence of the deploymentDependencies parameter.
6. Modify the create operation implementation to accommodate the existence of the executionAffinityAssignments parameter.
7. Convert the application factory's association with a Naming Service implementation to an association with a *ComponentRegistry*. The *ComponentRegistry* will serve as the repository with which deployed components will register (may require implementation of *ComponentRegistry*).

8. Store the components deployed by the ApplicationFactoryComponent within the ComponentType's specializedInfo.
9. Update the ApplicationFactoryComponent's call to the platform's execution operation to pass a reference to a *ComponentRegistry*.
10. Update any use of the *ResourceFactory* interface to refer to a ComponentFactoryComponent reference.
11. Update any use of the *ExecutableDevice* interface to refer to a ExecutableDeviceComponent reference.
12. Update any use of the *LoadableDevice* interface to refer to a LoadableDeviceComponent reference.
13. Update any use of the *Device* interface to refer to a DeviceComponent reference.
14. Update any use of the *Resource* interface to refer to a ManageableApplicationComponent reference.
15. Modify any use of the DomainManagementObjectAddedEventType to use a ComponentChangeEventType.
16. Extend the ApplicationFactoryComponent to create a unique connection identifier when none is provided.

4.6 SCA 4.1 DEVICEMANAGERCOMPONENT

SCA 2.2.2 device managers (DeviceManagerComponents) realize the *DeviceManager* interface, illustrated in Figure 20, which inherits the SCA 2.2.2 *PortSupplier* and *PropertySet* interfaces. SCA 2.2.2 and 4.1 device managers support the same basic capabilities:

- Device and Service deployment
- Node management

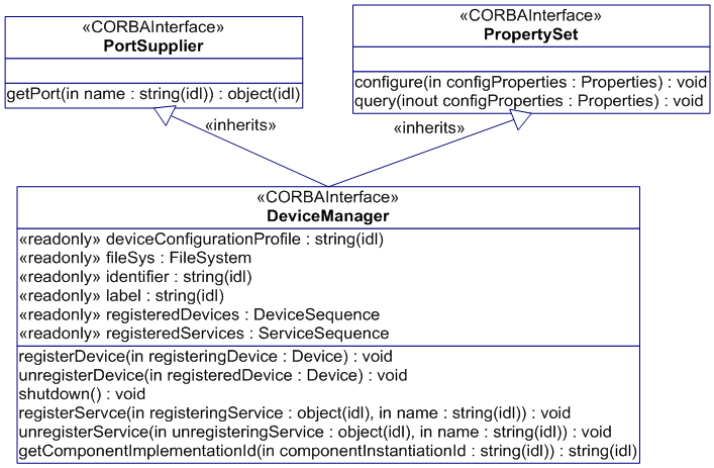


Figure 20: SCA 2.2.2 *DeviceManager* Interface

SCA 4.1 removed the *DeviceManager* interface and modifies the DeviceManagerComponent to have an associated registry, *ComponentRegistry*, with which the components it deploys register.

The DeviceManagerComponent, which is shown in Figure 21, inherits the functions and capabilities of a BaseComponent and consequently can be managed as such.

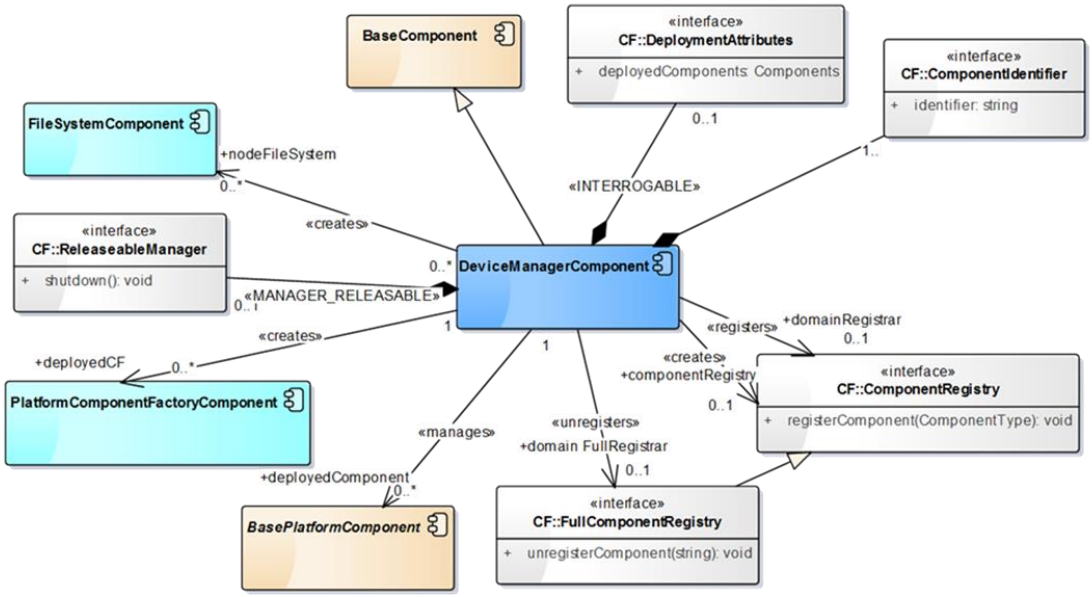


Figure 21: SCA 4.1 DeviceManagerComponent

A developer could define the following SCA 4.1 compliant interface:

```
interface myDeviceManager : CF::DeploymentAttributes,
ComponentIdentifier
```

The inheritance of the *CF::DeploymentAttributes* interfaces provides external clients with the ability to interrogate the DeviceManagerComponent regarding the platform components it deployed.

4.6.1 Interface Changes

4.6.1.1 DeviceManager Attributes

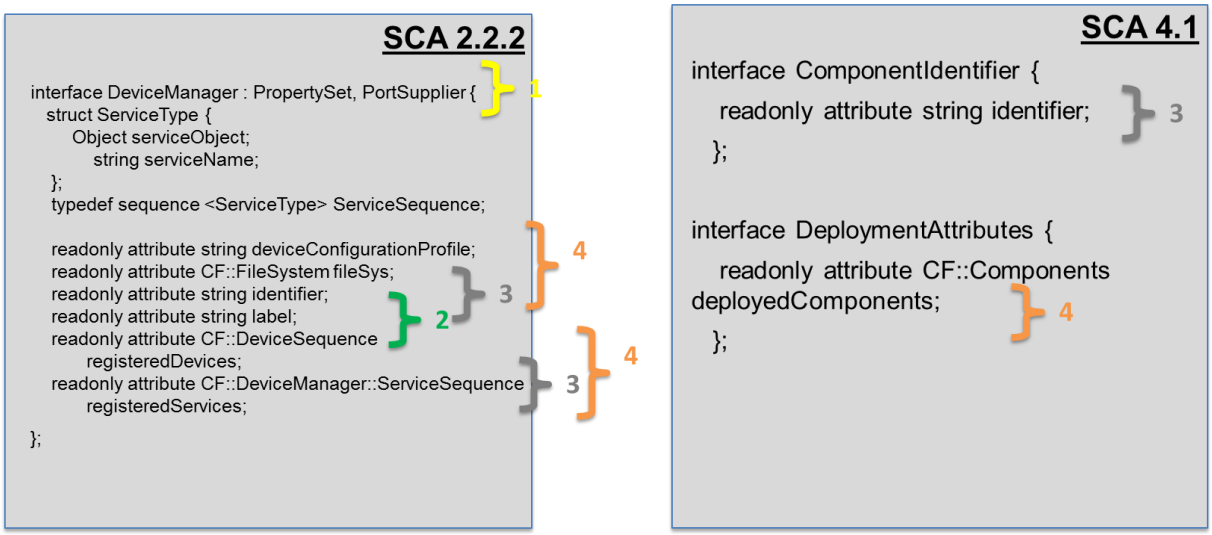


Figure 22: *DeviceManager* Interface Comparison

1. SCA 4.1 removes the *DeviceManager* interface.
2. SCA 4.1 removes the label attribute.
3. SCA 4.1 moves the identifier attribute to the *ComponentIdentifier* interface and collapses the registeredDevices and registeredServices attributes to the deployedComponents attribute within the *DeploymentAttributes* interface.
4. SCA 4.1 relocates the deviceConfigurationProfile, fileSys, registeredComponents and registeredServices (deployedComponents) attributes to the ComponentType structure.

4.6.1.2 DeviceManager Operations

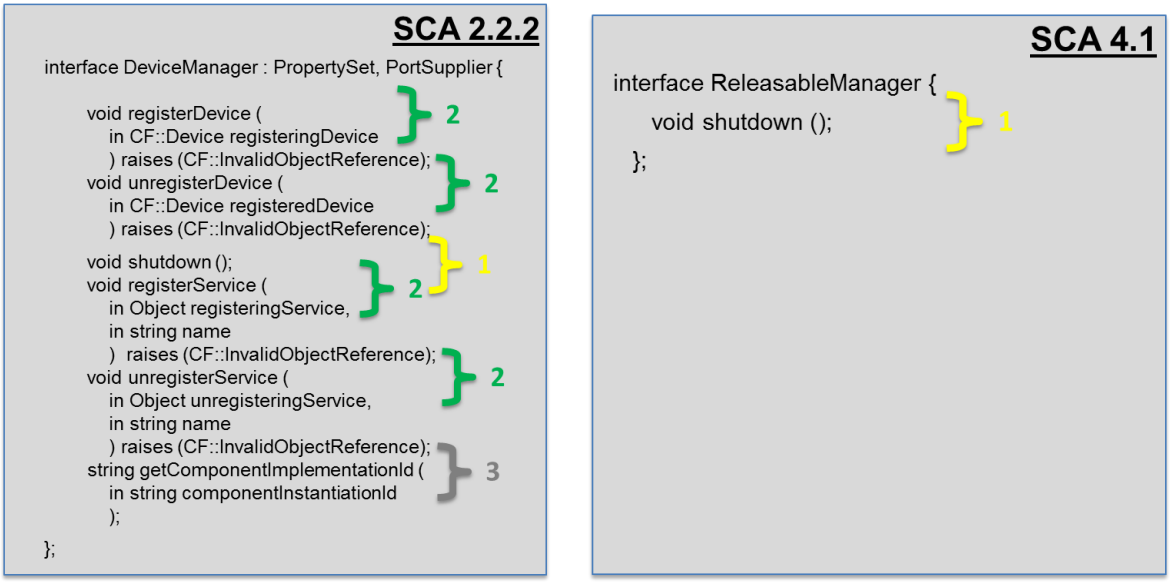


Figure 23: *DeviceManager* Interface Operation Comparison

1. SCA 4.1 relocates the shutdown operation within the *ReleasableManager* interface.
2. SCA 4.1 abstracts the registerService and registerDevice operations to registerComponent; the unregisterService and unregisterDevice operations to unregisterComponent SCA 4.1; removes registration and unregistration operations from the device manager and places them in independent registry components.
3. SCA 4.1 removes the getComponentImplementationId operation and maintains implementation properties within the ComponentType structure.

4.6.2 Implementation Changes

4.6.2.1 Requirements Driven

SCA 2.2.2 device managers contain approximately 56 requirements. The SCA 4.1 DeviceManagerComponent contains approximately 91 requirements and when the structural and modeling oriented requirements are removed there are about 70. Many of the 33 new requirements introduced in SCA 4.1 would not need to be implemented if an SCA 2.2 device manager was being migrated. 25 of the new requirements are a result of the DeviceManagerComponent's inheritance of BaseComponent and which would not need to be fully implemented in order to provide SCA 2.2.2 functionality. Once the BaseComponent requirements are removed there are eight new requirements that would need to be implemented.

Two requirements, SCA429 and SCA153 provide text clarifications from SCA 2.2.2 and may already be implemented. If they require a change, the effort should be minimal. SCA 4.1 introduces four requirements SCA438, SCA439, SCA449, SCA573 which are associated with support for the PlatformComponentFactory. The introduction of the PlatformComponentFactory represents a new capability within SCA 4.1 and would require a moderate change within a DeviceManagerComponent as it introduces new logic, but the code should be similar to an

ApplicationFactoryComponent's use of the ComponentFactory. One new requirement, SCA572, is associated with saving component allocation properties and requires a minimal change to store the property information within the ComponentType structure. The final new requirement is SCA133 and it should require a minimal change as it introduces a new exception case.

4.6.2.2 Structural

An SCA 4.1 Core Framework developer will need to define their own interface(s) to represent a DeviceManagerComponents because it is not provided by the framework.

An SCA 2.2.2 component that uses the *DeviceManager* interface will require the following changes beyond those required of a BaseComponent to be migrated to SCA 4.1 compliance:

1. Remove the *DeviceManager* interface in lieu of a new, user-defined interface.
2. Modify any interfaces associated with an SCA component could be modified to inherit from a non-CORBA CF::Port equivalent in order to minimize changes to an existing implementation.
3. Construct a container of type ComponentType for the *DeviceManager* component.
4. Relocate the deviceConfigurationProfile attribute information within the ComponentType container.
5. Relocate the fileSys attribute information within the ComponentType container.
6. Relocate the identifier attribute information within the ComponentType container.
7. Relocate the shutdown operation implementation to the *ReleasableManager* interface.
8. Copy the identifier attribute information within the *ComponentIdentifier* interface.
9. Remove the label attribute.
10. Remove the Device and Service registration and unregistration operations in favor of a *ComponentRegistry* implementation (if needed those operations could provide the basis of the registry implementation).
11. Migrate the logic which stored data within the registeredDevices and registeredServices to be associated with the component registry.
12. Store the information about the registered (deployed) components within the *DeploymentAttributes* interface.
13. Store the information about the registered (deployed) components within the ComponentType container.
14. Remove the *getComponentImplementationId* interface and ensure that the data that would have be retrieved through that interface is stored within the ComponentType container.

4.7 SCA 4.1 DOMAINMANAGERCOMPONENT

SCA 2.2.2 domain managers (DomainManagerComponents) realize the *DomainManager* interface, that is shown in Figure 24, which inherits the *PropertySet* interface. SCA 2.2.2 and 4.1 domain managers support the same basic capabilities:

- Application installation
- Component registration and unregistration
- Management of applications, application factories and device managers within the domain

- Event channel registration for external consumers

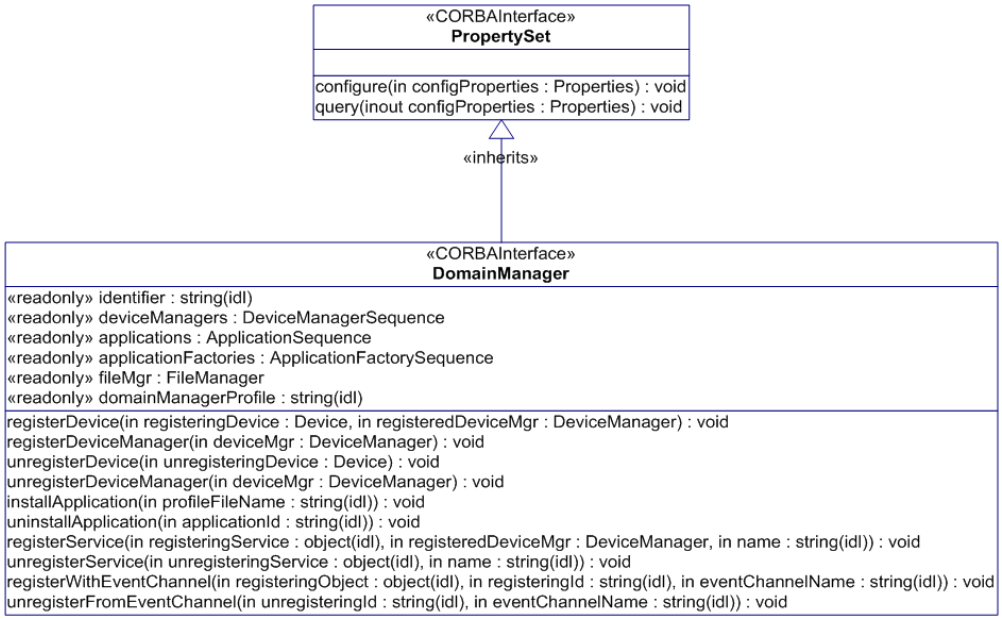


Figure 24: SCA 2.2.2 *DomainManager* Interface

The SCA 4.1 DomainManagerComponent has an associated registry, ComponentRegistry, with which the components it managers register. The DomainManagerComponent, illustrated in Figure 25, inherits the functions and capabilities of a BaseComponent and consequently can be managed as such.

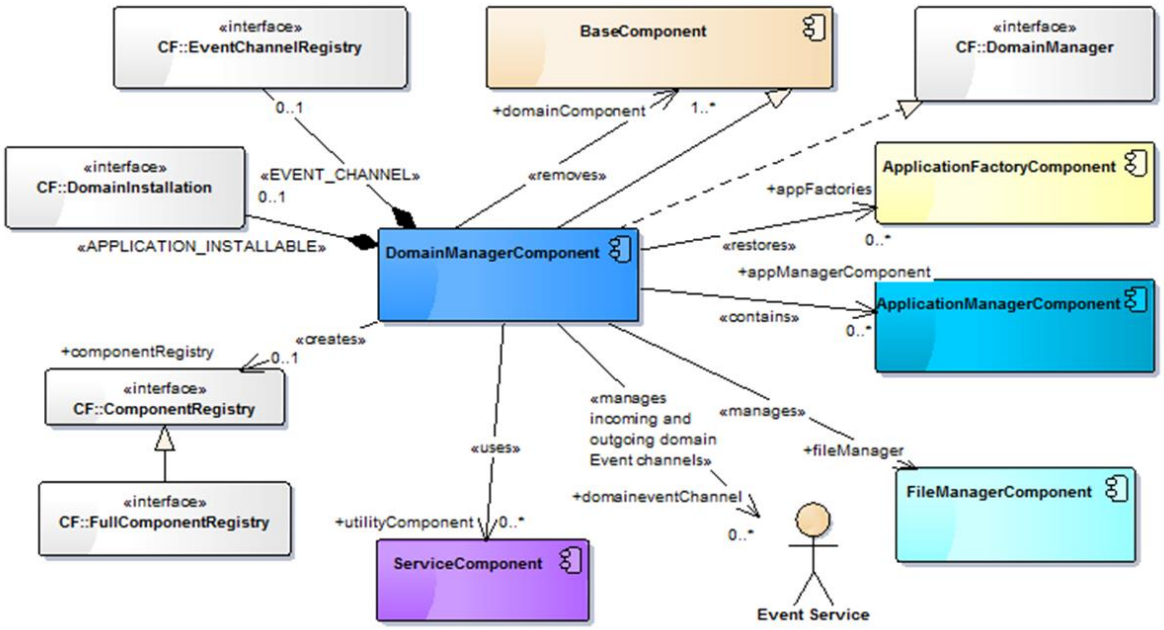


Figure 25: SCA 4.1 DomainManagerComponent

A developer could define the following SCA 4.1 compliant interface:

```
interface myDomainManager : CF::DomainManager,
CF::DomainInstallation
```

Where the inheritance of the *CF::DomainInstallation* interfaces provides the DomainManagerComponent with the ability to install applications.

4.7.1 Interface Changes

4.7.1.1 DomainManager Types and Exceptions

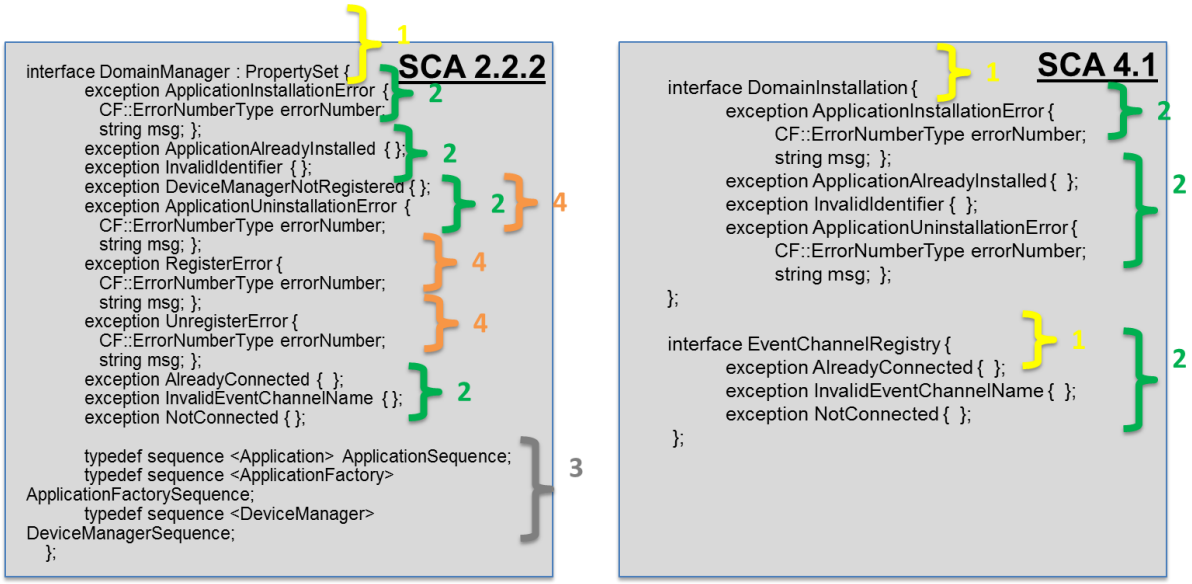


Figure 26: *DomainManager* Interface Comparison

1. SCA 4.1 preserves the *DomainManager* interface but decomposes it to create two new interfaces, *DomainInstallation* and *EventChannelRegistry*.
2. SCA 4.1 relocates exceptions to the new interfaces.
3. SCA 4.1 removes specialized type definitions.
4. SCA 4.1 relocates registration exceptions to component registry interfaces (distinct from the *DomainManager* interface).

4.7.1.2 DomainManager Attributes

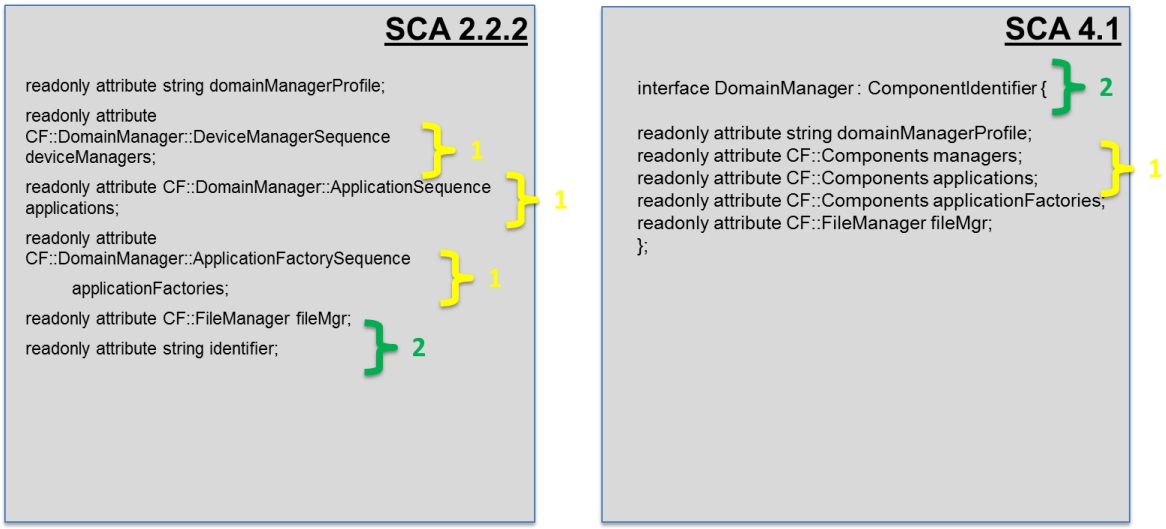


Figure 27: *DomainManager* Interface Attribute Comparison

1. SCA 4.1 uses a common type definition, CF::Components for managed elements.
2. SCA 4.1 relocates the identifier attribute to the *ComponentIdentifier* interface.

4.7.1.3 DomainManager Registration Operations

Figure 28: *DomainManager* Interface Registration Operation Comparison

1. SCA 4.1 removes registration and unregistration operations from *DomainManager* and places them in the *ComponentRegistry* interface for registration and *FullComponentRegistry* interface for unregistration.

4.7.2 Implementation Changes

4.7.2.1 Requirements Driven

SCA 2.2.2 domain managers contain approximately 130 requirements, and the count decreases to 124 in SCA 4.1. However, when the structural and modeling oriented requirements are removed, the count is decreased to about 112 allocated 68 requirements.

Similar to the DeviceManagerComponent, several new DomainManagerComponent requirements were introduced with its BaseComponent inheritance which do not need to be implemented in the migration scenario. Therefore, there are nine new DomainManagerComponent requirements beyond those of the BaseComponent. SCA518 establishes the domain manager as a safety valve to disconnect components that are being torn down. This enhancement should be a minimal change, as it should reuse other releaseObject logic. SCA571 introduces a requirement for the installApplication operation to return a ComponentType structure, which should be a minimal change that requires an implementation to reorganize most of the information that is maintained within the code. Six requirements, SCA132, SCA135, SCA149, SCA194, SCA198 and SCA199 are associated with component registration. The change should be a minimal impact, as it will be a refactoring of logic from the preexisting registration and unregistration operations. One requirement, SCA552, is associated with backwards compatibility and should require a minimal change, to check for the presence of an SCA 2.2.2 application and throw an exception when they are not handled by the Core Framework.

4.7.2.2 Structural

An SCA 4.1 Core Framework developer that realizes the *DomainManager* interface will need to define their own interface(s) to represent the DomainManagerComponent, for example the myDomainManager interface described earlier.

An SCA 2.2.2 component that implements the *DomainManager* interface will require the following changes beyond those required of a BaseComponent to be migrated to SCA 4.1 compliance:

1. Refactor any use of the *PropertySet* interface operations to reflect its location within the DomainManagerComponent.
2. Implement the new *DomainInstallation* interface which will be inherited by the *DomainManager* interface.
3. Implement the new *EventChannelRegistry* interface which will be inherited by the *DomainManager* interface.
4. Remove the device, service and device manager registration and unregistration operations in favor of a *ComponentRegistry* implementation (if needed those operations could provide the basis of the registry implementation).
5. Relocate the installation related operations to the new *DomainInstallation* interface.
6. Relocate the event channel registration related operations to the new *EventChannelRegistry* interface.
7. Relocate the registration and unregistration exceptions to the *ComponentRegistry* implementation.
8. Relocate the installation and uninstallation exceptions to the *DomainInstallation* interface.
9. Relocate the event channel registration exceptions to the *EventChannelRegistry* interface.

10. Remove the type definition of the specialized ApplicationSequence type.
11. Remove the type definition of the specialized ApplicationFactorySequence type.
12. Remove the type definition of the specialized DeviceManagerSequence type.
13. Rename the deviceManagers attribute to managers and change its type to be ComponentType.
14. Modify the applications attribute to be type ComponentType.
15. Modify the applicationFactories attribute to be type ComponentType.
16. The implementation will need to introduce the new *ComponentIdentifier* interface which will be inherited by the *DomainManager* interface.
17. Relocate the identifier attribute to the new *ComponentIdentifier* interface.
18. Modify the installApplication interface to return a ComponentType rather than a void.