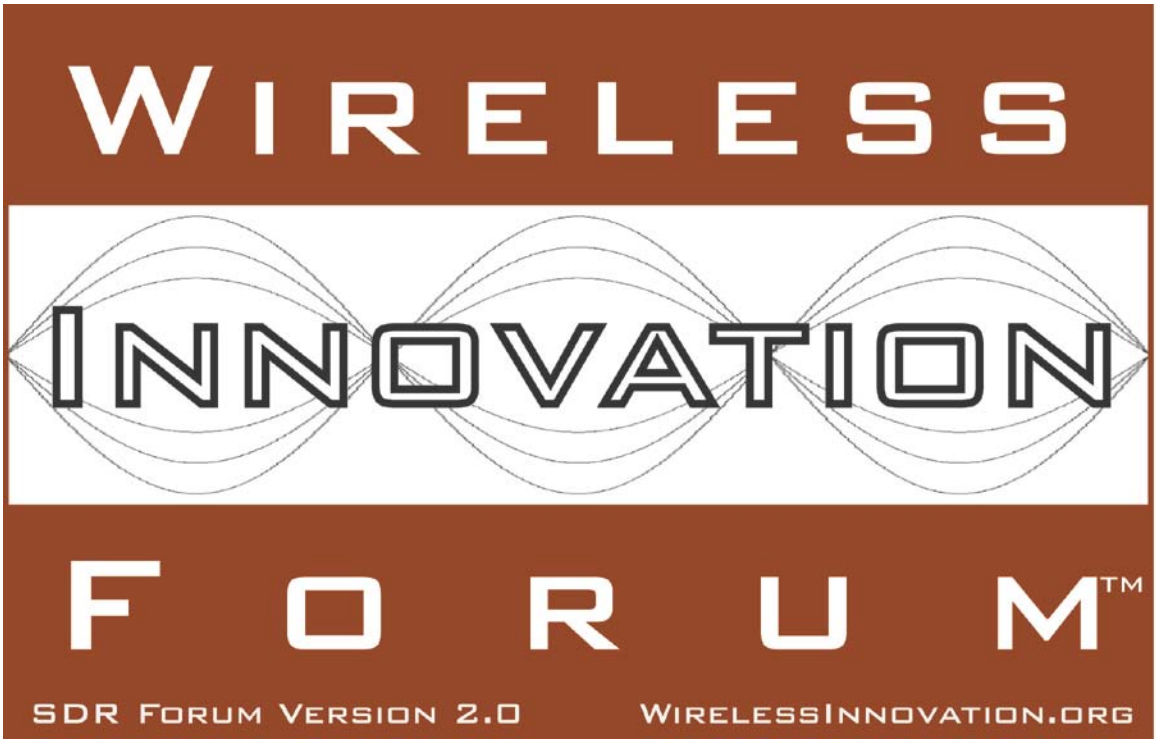


Request for Comment on Generalization of the Resource Factory Concept

Working Document WINNF-10-RFI-0005

Version V2.0.0

24 January 2011

TERMS, CONDITIONS & NOTICES

This document has been prepared by the SCA Next Work Group to assist The Software Defined Radio Forum Inc. (or its successors or assigns, hereafter "the Forum"). It may be amended or withdrawn at a later time and it is not binding on any member of the Forum or of the SCA Next Work Group.

Contributors to this document that have submitted copyrighted materials (the Submission) to the Forum for use in this document retain copyright ownership of their original work, while at the same time granting the Forum a non-exclusive, irrevocable, worldwide, perpetual, royalty-free license under the Submitter's copyrights in the Submission to reproduce, distribute, publish, display, perform, and create derivative works of the Submission based on that original work for the purpose of developing this document under the Forum's own copyright.

Permission is granted to the Forum's participants to copy any portion of this document for legitimate purposes of the Forum. Copying for monetary gain or for other non-Forum related purposes is prohibited.

THIS DOCUMENT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS DOCUMENT.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

This document was developed following the Forum's policy on restricted or controlled information (Policy 009) to ensure that that the document can be shared openly with other member organizations around the world. Additional Information on this policy can be found here: http://www.wirelessinnovation.org/page/Policies_and_Procedures

Although this document contains no restricted or controlled information, the specific implementation of concepts contain herein may be controlled under the laws of the country of origin for that implementation. Readers are encouraged, therefore, to consult with a cognizant authority prior to any further development.

Wireless Innovation Forum ™ and SDR Forum ™ are trademarks of the Software Defined Radio Forum Inc.

Table of Contents

Preface

In August 2009, the JPEO and its JTRS SCA Next Working Panel, invited the WINNF to assist it in developing the specification of a new release of the SCA whose working title is "SCA Next." The WINNF created a "SCA Next Work Group" to coordinate this work and informed the JTRS SCA Next Working Panel that the WINNF wished to take the lead on developing solutions for two of the previously defined Change Proposals: S047 "Develop CORBA/e and CORBA Services wording" and S013 "SCA Architectural Consistency", as well as offer comments and suggestions for many of the other change proposals.

As part of the consideration of architectural consistency, the WINNF explored the idea of generalizing the Resource Factory concept to allow it to be used to create platform components, such as, Devices and Services, and concluded that this was a useful option for many of the same reasons Resource Factories are useful, especially to allow co-locating platform components into a single address space.

The WINNF SCA Next Work Group is pleased to contribute the attached **document describing the concept and the required changes to the SCA** specification. This document is being send to you at this time for your consideration and comments. However, the WINNF intends to continue the work and suggest specific wording changes to the SCA specification and modified DTDs to accomplish the changes suggested here. This additional document will be sent to you when we complete that task. Note that this relates to Change Proposal S013. The WINNF requests your consideration and invite your comments.

Please respond with your comments to the WINNF SCA Next Work Group Chair: Terry Anderson, terry.anderson@itt.com.

Request for Comment on Generalization of Resource Factory Concept

1 Description of Enhancement

The ComponentFactory provides architectural consistency since it now allows both the platform and application components to be created using a factory. The ComponentFactory can also be used as a platform independent means of collocating several components into a single address space. The ResourceFactory currently provides this capability but only for application Resources.

2 Rationale for Change

The *ComponentFactory* provides architectural consistency since it now allows both the platform and application components to be created using a factory. There is no obvious reason why the factory concept has not been available for devices and services.

The *ComponentFactory* can also be used as a platform independent means of collocating several components into a single address space. The *ResourceFactory* currently provides this capability but only for application *Resources*. Collocation of several components into a single address space provides significant footprint savings, improves real-time performances, and reduces deployment time of components. Similar savings can be achieved using implementation techniques such as the use of shared libraries. But these techniques are specific to operating environments, thus it affects portability of components. The use of a component factory is the only platform-independent technique which provides the benefits of address space collocation.

3 Impacts

On Existing CF/Applications	Existing applications will need to be updated only if they use Resource factories. The required modifications would be minimal. Modifications would consist in transforming the Resource factories into Component factories which are very similar in APIs. Minor impact for Core Framework that does not support this optional feature. The impact is related to the new optional tag in the DCD and to a new component type in the SCD.
On in-dev CF/Applications:	none.
On existing CP:	none.

4 Recommended Changes

Change the specification to allow the DeviceManager to instantiate Device and service components using a generic ComponentFactory. The ComponentFactory creates CORBA Objects that can be narrowed to the proper component type. The ComponentFactory shall replace the ResourceFactory. This involves changing the SCA Specification document to describe the ComponentFactory and to describe how the ApplicationFactory and the DeviceManager may use a ComponentFactory.

The use of a ComponentFactory is optional. Platforms are neither required to use them or to support their use, but if a ComponentFactory is provided it can be used to create any types of component that can be launched via a DCD or a SAD. For instance the ComponentFactory could create a Resource, a Device or a service.

Here is the list of changes required to the Specification version 2.2.2. Note that highlighting is used in many places to emphasize the parts that should change.

Change #0: Section 2.4
Amend Figure 2-2 to show the ComponentFactory under the DeviceManager and change the ResourceFactory box for the ComponentFactory.

Change #1: Section 2.2.2
From:
Base Application Interfaces: *Port*, *LifeCycle*, *TestableObject*, *PropertySet*, *PortSupplier*, *ResourceFactory*, and *Resource*), which provide the management and control interfaces for all system software components.

To:
Base Application Interfaces: *Port*, *LifeCycle*, *TestableObject*, *PropertySet*, *PortSupplier*, *ComponentFactory*, and *Resource*), which provide the management and control interfaces for all system software components.

Change #2: Section 3.1.2.2.1
From:
A log producer is a CF component (e.g., *DomainManager*, *Application*, *ApplicationFactory*, *DeviceManager*, *Device*) or an application's CORBA capable component (e.g., *Resource*, *ResourceFactory*) that produces log records using the Lightweight Log Service *CosLwLog::LogProducer* interface. Log records are of type *CosLwLog::ProducerLogRecord.*

To:
A log producer is a CF component (e.g., *DomainManager*, *Application*, *ApplicationFactory*, *DeviceManager*, *Device*) or an application's CORBA capable component (e.g., *Resource*, *ComponentFactory*) that produces log records using the Lightweight Log Service *CosLwLog::LogProducer* interface. Log records are of type *CosLwLog::ProducerLogRecord.*

Change #3: Section 3.1.3

From:

Figure 3-2 depicts the key elements of the CF and the IDL relationships between these elements. A *DomainManager* component manages the software applications, application factories, hardware devices (represented by software devices) and device managers within the system. Some software components may directly control the system's internal hardware devices; these components are logical devices, which implement the *Device*, *LoadableDevice*, or *ExecutableDevice* interfaces. Other software components have no direct relationship with a hardware device, but perform application services for the user and implement the *Resource* interface. This interface provides a consistent way of configuring and tearing down these components. Each resource can potentially communicate with other resources. An application is a specific collection of one or more resources which provides a specified service or function and which is managed through the *Application* interface. The resources of an application are allocated to one or more hardware devices by the application factory based upon various factors including the current availability of hardware devices, the behavior rules of a resource, and the loading requirements of each resource. The resources may then be created by using the *ResourceFactory* interface or through the *Device* interfaces (*Device*, *LoadableDevice, or ExecutableDevice*) an connected to other resources or devices resident on the system.

To:

Figure 3-2 depicts the key elements of the CF and the IDL relationships between these elements. A *DomainManager* component manages the software applications, application factories, hardware devices (represented by software devices) and device managers within the system. Some software components may directly control the system's internal hardware devices; these components are logical devices, which implement the *Device*, *LoadableDevice*, or *ExecutableDevice* interfaces. Other software components have no direct relationship with a hardware device, but perform application services for the user and implement the *Resource* interface. This interface provides a consistent way of configuring and tearing down these components. Each resource can potentially communicate with other resources. An application is a specific collection of one or more resources which provides a specified service or function and which is managed through the *Application* interface. The resources of an application are allocated to one or more hardware devices by the application factory based upon various factors including the current availability of hardware devices, the behavior rules of a resource, and the loading requirements of each resource. The resources may then be created by using the *ComponentFactory* interface or through the *Device* interfaces (*Device*, *LoadableDevice, or ExecutableDevice*) and connected to other resources or devices resident on the system.

Change #4: Section 3.1.3.1.7

Section "3.1.3.1.7 ResourceFactory" should be renamed to "ComponentFactory" and moved out of the section "3.1.3.1 Base Application Interfaces". The content of the section should be as follows:

3.1.3.1.7 *ComponentFactory*

3.1.3.1.7.1 Description

A component factory is used to create and tear down resources, devices, and services. The *ComponentFactory* interface is designed after the Factory Design Patterns. The *ComponentFactory*

interface UML is depicted in Figure 3-9. The factory mechanism provides client-server isolation among components and provides a standard mechanism of obtaining a component without knowing its identity. An application is not required to use component factories to obtain, create, or tear down resources. Similarly, a *DeviceManager* is not required to use component factories to obtain, create, or tear down devices and services. Software profiles specify which component factories are to be used by the *ApplicationFactory* and the *DeviceManager*.

When the *ComponentFactory* is used to launch application components, it is limited to using the OS services that are designated as mandatory in the SCA AEP. However, when the *ComponentFactory* is not used to launch application components, it is not limited to using the OS services designated as mandatory by the SCA AEP.

When used to create platform components, the *ComponentFactory* shall only create *Devices* or *services*. When used to create application components, the *ComponentFactory* shall only create *Resources*. The *ComponentFactory* shall not be used to create a *ComponentFactory* .

3.1.3.1.7.2 UML

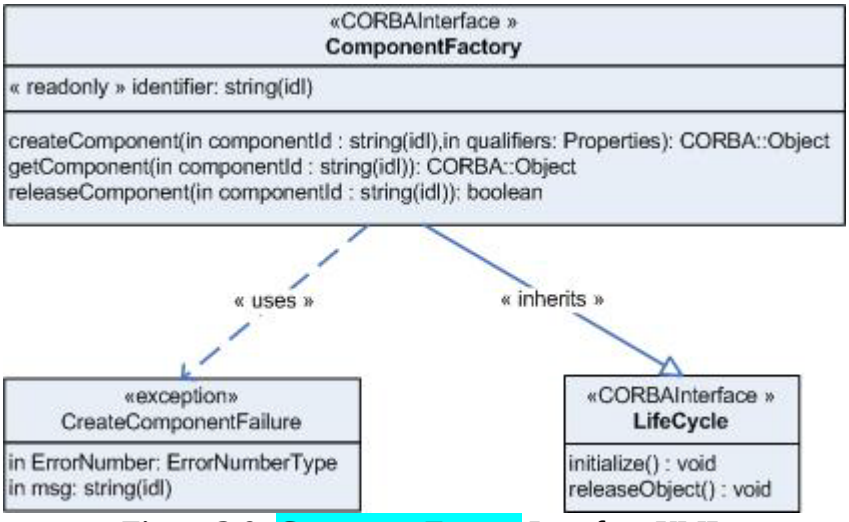


Figure 3-9: *ComponentFactory* Interface UML

3.1.3.1.7.3 Types

3.1.3.1.7.3.1 CreateComponentFailure

The CreateComponentFailure exception indicates that the *createComponent* operation failed to create the component. The error number shall indicate a CF ErrorNumberType value. The message is component-dependent, providing additional information describing the reason for the error.

```
exception CreateComponentFailure { ErrorNumberType errorNumber;
string msg; };
```

3.1.3.1.7.4 Attributes

3.1.3.1.7.4.1 identifier

The readonly identifier attribute shall contain the unique identifier for a *ComponentFactory* instance.

```
readonly attribute string identifier;
```

3.1.3.1.7.5 Operations

3.1.3.1.7.5.1 *createComponent*

3.1.3.1.7.5.1.1 Brief Rationale

The *createComponent* operation provides the capability to create components in the same process space as the component factory. This behavior is an alternative approach to the *Device::execute* operation for creating a component.

3.1.3.1.7.5.1.2 Synopsis

```
CORBA::Object createComponent (in string componentId, in Properties
qualifiers) raises (CreateComponentFailure);
```

3.1.3.1.7.5.1.3 Behavior

The componentId parameter is the identifier for a component. The qualifiers parameter contains values used by the factory in creation of the component. The *ApplicationFactory* and the *DeviceManager* determine the values to be supplied for the qualifiers from the description in the component factory's software profile. The qualifiers may be used to identify, for example, specific subtypes of components created by a component factory.

The *createComponent* operation shall create a component if no component exists for the given *componentId* and shall assign the given *componentId* to the new component. The *createComponent* operation shall set a reference count to one, when the component is created.

3.1.3.1.7.5.1.4 Returns

The *createComponent* operation shall return a reference to the created component.

3.1.3.1.7.5.1.5 Exceptions/Errors

The *createComponent* operation shall raise the CreateComponentFailure exception when it cannot create the component or when the component already exists.

3.1.3.1.7.5.2 *releaseComponent*

3.1.3.1.7.5.2.1 Brief Rationale

In CORBA there is client side and server side representation of a component. The *releaseComponent* operation provides the mechanism of releasing the component in the CORBA environment on the server side when all clients are through with a specific component. The client still has to release its client side reference of the component.

3.1.3.1.7.5.2.2 Synopsis

```
boolean releaseComponent (in string componentId);
```

3.1.3.1.7.5.2.3 Behavior

The *releaseComponent* operation shall decrement the reference count for the specified component, as indicated by the *componentId* parameter. The *releaseComponent* operation shall release the component from the CORBA environment and make the component no longer available when the component's reference count is zero.

3.1.3.1.7.5.2.4 Returns

This operation returns true if the release was successful. False will be returned if an invalid *componentId* is specified.

3.1.3.1.7.5.3 *getComponent*

3.1.3.1.7.5.3.1 Brief Rationale

The *getComponent* operation provides the capability to return a reference to a component that has already been created.

3.1.3.1.7.5.3.2 Synopsis

```
CORBA::Object getComponent(in string componentId);
```

3.1.3.1.7.5.3.3 Behavior

The *componentId* parameter is the identifier for a component. The *getComponent* operation shall return a reference to an existing component identified by the *componentId* parameter and shall increment the reference count by one for the specified component. The reference count is used to indicate the number of times that a specific component reference has been given to requesting clients. This ensures that the component factory does not release a component that has a reference count greater than zero (0). When multiple clients have obtained a reference to the same component, each client requests release of the component when it is through with the component. However, the component is not released until the release request comes from the last client in existence.

3.1.3.1.7.5.3.4 Returns

The *getComponent* operation shall return a reference to the existing component.

3.1.3.1.7.5.3.5 Exceptions/Errors

The *getComponent* operation shall return a nil CORBA object reference when the component does not exists.

Change #5: Section 3.1.3.2.1.6.1.3

From:

The *Application::releaseObject* operation shall release each application component not created by a resource factory by utilizing the component's *Resource::releaseObject* operation. The *Application::releaseObject* operation shall release each component created by a resource factory via the *ResourceFactory::releaseResource* operation. The *Application::releaseObject* operation shall terminate a resource factory when no more resources are managed by the resource factory via the *ResourceFactory::shutdown* operation. The *Application::releaseObject* operation shall terminate the processes / tasks on allocated executable devices belonging to each application component by utilizing the *ExecutableDevice:terminate* operation.

To:

The *Application::releaseObject* operation shall release each application component not created by a component factory by utilizing the component's *Resource::releaseObject* operation. The *Application::releaseObject* operation shall release each component created by a component factory via the *ComponentFactory::releaseComponent* operation. The *Application::releaseObject* operation shall terminate a component factory when no more components are managed by the component factory via the *ComponentFactory::releaseObject* operation. The *Application::releaseObject* operation shall terminate the processes / tasks on allocated executable devices belonging to each application component by utilizing the *ExecutableDevice:terminate* operation.

From:

For components (e.g., *Resource*, *ResourceFactory*) that are registered with Naming Service, the *releaseObject* operation shall unbind those components and destroy the associated naming contexts as necessary from the Naming Service.

To:

For components (e.g., *Resource*, *ComponentFactory*) that are registered with Naming Service, the *releaseObject* operation shall unbind those components and destroy the associated naming contexts as necessary from the Naming Service.

From:

The following steps demonstrate one scenario of the application's behavior for the release of an application that contains *ResourceFactory* behavior:

1. Client invokes *releaseObject* operation.
2. Disconnect *Ports*.
3. Release the *ResourceFactory* components.
4. Shutdown the *ResourceFactory* components.
5. Release the *Resource* components.
6. Terminate the components' processes.
7. Unload the components' executable images.

8. Change the state of the associated devices to be available, along with device(s) memory utilization availability and processor utilization availability based upon the Device Profile and software profile.

9. Unbind application components from Naming Service.

10. Log an Event indicating that the application was either successfully or unsuccessfully released.

11. Remove the application reference from the applications attribute and generate an event to indicate the application has been removed from the domain.

To:
The following steps demonstrate one scenario of the application's behavior for the release of an application that contains *ResourceFactory* behavior:

1. Client invokes *releaseObject* operation.

2. Disconnect *Ports*.

3. Release the *ComponentFactory* components.

4. Release the *ComponentFactory*.

5. Release the *Resource* components.

6. Terminate the components and ComponentFactories' processes.

7. Unload the components and ComponentFactories' executable images.

8. Change the state of the associated devices to be available, along with device(s) memory utilization availability and processor utilization availability based upon the Device Profile and software profile.

9. Unbind application components from Naming Service.

10. Log an Event indicating that the application was either successfully or unsuccessfully released.

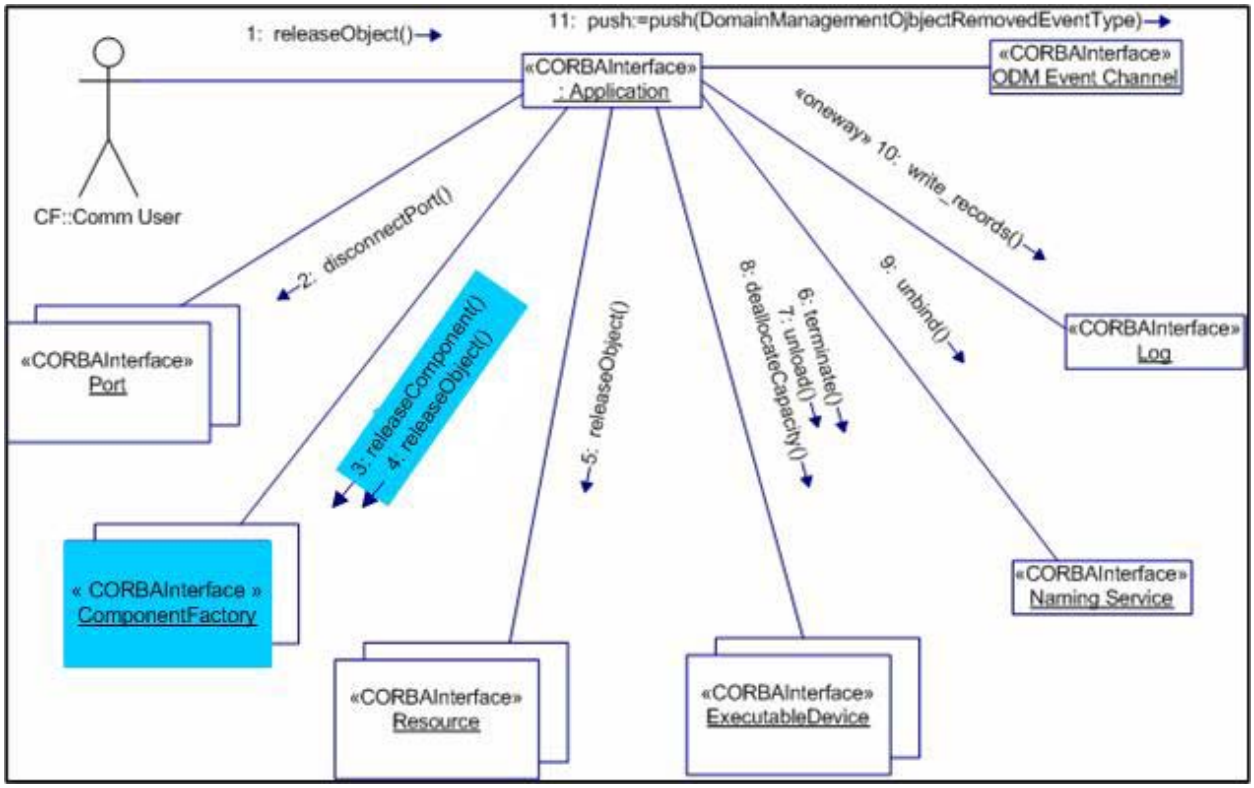
11. Remove the application reference from the applications attribute and generate an event to indicate the application has been removed from the domain.

Change #6: Figure 3-11 Application Behavior

Change the messages number 3 and 4 to the following:

3: releaseComponent()

4: releaseObject()



Change #7: Section 3.1.3.2.2.5.1.3

From:
The *create* operation validates all component-device associations in the input deviceAssignments parameter by verifying that the device indicated by the assignedDeviceId element provides the necessary capacities and properties required by the component indicated by the componentId element. Device assignments should not be given for resources created via a resource factory since instantiation of these *Resources* is controlled by the creating *ResourceFactory*.

To:
The *create* operation validates all component-device associations in the input deviceAssignments parameter by verifying that the device indicated by the assignedDeviceId element provides the necessary capacities and properties required by the component indicated by the componentId element. Device assignments should not be given for resources created via a *ComponentFactory* since instantiation of these resources is controlled by the creating *ComponentFactory*.

From:
Upon execution of a software module by the *create* operation, a *Resource* or a *ResourceFactory* component shall register with the Naming Service. The *create* operation uses "ComponentName_UniqueIdentifier" to retrieve the component's CORBA object reference from the Naming Context IOR.

To:
Upon execution of a software module by the *create* operation, a *ComponentFactory* or a *Resource* created via an *ExecutableDevice* shall register with the Naming Service. The *create* operation uses "ComponentName_UniqueIdentifier" to retrieve the component's CORBA object reference from the Naming Context IOR.

From:
The *create* operation shall include the mandatory execute parameters Naming Context IOR, Name Binding, and Component Identifier, as described in this section, in the parameters parameter of the *ExecutableDevice::execute* operation when the CORBA instance's *componentinstantiation* element of the SAD contains a *findcomponent* element with a *namingservice* sub-element.

The execute parameter for the Naming Context IOR shall be a CF *Properties* type with an id element set to "NAMING_CONTEXT_IOR" and a value element set to the stringified IOR of the naming context to which the component will bind. The *create* operation shall create any naming contexts that do not exist but which are required for successful binding to the Naming Context IOR. The structure of the naming context path shall be "/ DomainName / [optional naming context sequences]". In the naming context path, each "slash" (/) represents a separate naming context.

The Name Binding execute parameter shall be a CF *Properties* type with an id element set to "NAME_BINDING" and a value element set to a string in the format of "ComponentName_UniqueIdentifier". The ComponentName value is the SAD *componentinstantiation findcomponent namingservice* element's *name* attribute. The UniqueIdentifier is determined by the implementation. The Name Binding parameter is used by the component to bind its object reference to the Naming Context IOR parameter.

The Component Identifier execute parameter shall be a CF *Properties* type with an id element set to "COMPONENT_IDENTIFIER" and a value element set to a string in the format of “Component_Instantiation_Identifier: Application_Name”. The Component_Instantiation_Identifier is the *componentinstantiation* element *id* attribute for the component in the application’s SAD file. The Application_Name field shall be identical to the *create* operation’s input name parameter. The Application_Name field provides a specific instance qualifier for executed components.

To:
When the *create* operation creates a component via an *ExecutableDevice*, it shall include the three mandatory execute parameters Naming Context IOR, Name Binding, and Component Identifier, as described in this section, in the parameters parameter of the *ExecutableDevice::execute* operation when the CORBA instance’s *componentinstantiation* element of the SAD contains a *findcomponent* element with a *namingservice* sub-element. However, when the *create* operation creates a component via a *ComponentFactory*, it shall provide the Component Identifier parameter but not the Naming Context IOR and Name Binding parameters.

When a *ComponentFactory* is created or when a *Resource* is created via an *ExecutableDevice*, the execute parameter for the Naming Context IOR shall be a CF *Properties* type with an id element set to "NAMING_CONTEXT_IOR" and a value element set to the stringified IOR of the naming context to which the component will bind. The *create* operation shall create any naming contexts that do not exist but which are required for successful binding to the Naming Context IOR. The structure of the naming context path shall be "/ DomainName / [optional naming context sequences]". In the naming context path, each "slash" (/) represents a separate naming context.

When a *ComponentFactory* is created or when a *Resource* is created via an *ExecutableDevice*, the Name Binding execute parameter shall be a CF *Properties* type with an id element set to "NAME_BINDING" and a value element set to a string in the format of "ComponentName_UniqueIdentifier". The ComponentName value is the SAD *componentinstantiation findcomponent namingservice* element’s *name* attribute. The UniqueIdentifier is determined by the implementation. The Name Binding parameter is used by the component to bind its object reference to the Naming Context IOR parameter.

When a *ComponentFactory* is created or when a *Resource* is created via an *ExecutableDevice*, the Component Identifier execute parameter shall be a CF *Properties* type with an id element set to "COMPONENT_IDENTIFIER" and a value element set to a string in the format of “Component_Instantiation_Identifier: Application_Name”. The Component_Instantiation_Identifier is the *componentinstantiation* element *id* attribute for the component in the application’s SAD file. The Application_Name field shall be identical to the *create* operation’s input name parameter. The Application_Name field provides a specific instance qualifier for executed components. When a *Resource* is created via a *ComponentFactory*, the Component Identifier shall be passed as the *qualifiers* parameter to the referenced *ComponentFactory* component’s *createComponent* operation.

From:
The *create* operation shall pass the values of the “execparam” properties of the *componentinstantiation componentproperties* element contained in the SAD, as parameters to the *execute* operation. The *create* operation passes “execparam” parameters values as string values.

To:
When a *Resource* is created via an *ExecutableDevice*, the *create* operation shall pass the values of the "execparam" properties of the *componentinstantiation componentproperties* element contained in the SAD, as parameters to the *execute* operation. The *create* operation passes "execparam" parameters values as string values.

From:
The *create* operation obtains a resource in accordance with the SAD via the CORBA Naming Service or a resource factory. The *ResourceFactory* object reference is obtained by using the CORBA Naming Service. The *create* operation, when creating a resource from a resource factory, shall pass the *componentinstantiation componentresourcefactoryref* element properties whose *kindtype* element is *factoryparam* as the *qualifiers* parameter to the referenced *ResourceFactory* component's *createResource* operation.

To:
The *create* operation obtains a resource in accordance with the SAD via the CORBA Naming Service or a *ComponentFactory*. The *ComponentFactory* object reference is obtained by using the CORBA Naming Service. The *create* operation, when creating a resource from a *ComponentFactory*, shall pass the *componentinstantiation componentfactoryref* element properties whose *kindtype* element is *factoryparam* as the *qualifiers* parameter to the referenced *ComponentFactory* component's *createComponent* operation.

From:

6. Obtain the object reference (*Resource* or *ResourceFactory*) as described by the SAD.

7. If the component obtained from the CORBA Naming Service is a resource factory as indicated by the SAD, then narrow the object reference to be a *ResourceFactory* component.

8. If the component is a *ResourceFactory*, then create a resource using the *ResourceFactory* interface.

To:

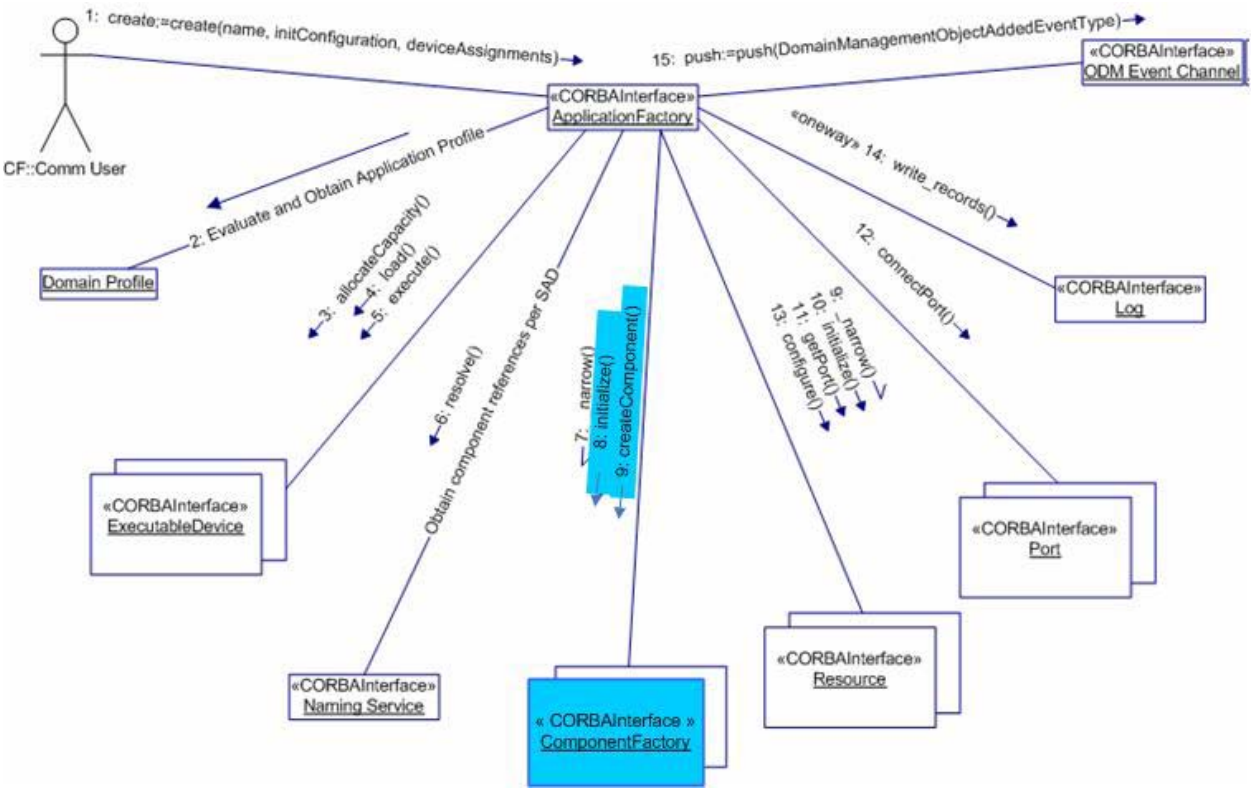
6. Obtain the object reference (*Resource* or *ComponentFactory*) as described by the SAD.

7. If the component obtained from the CORBA Naming Service is a *ComponentFactory* as indicated by the SAD, then narrow the object reference to be a *ComponentFactory* component.

8. If the component is a *ComponentFactory*, then create a resource using the *ComponentFactory* interface.

Change #8: Figure 3-13 ApplicationFactory Behavior

Firgure 3-13 needs to be updated. The object instance called “ResourceFactory” must be renamed to “ComponentFactory”. Also, the ComponentFactory must be initialized before it is used by the ApplicationFactory. This means the figure must have a new message inserted between messages 7 and 8. The new message must be labeled as “initialize”. Finally, the message number 8 must be relabeled as ”createComponent”



Change #9: Section: 3.1.3.2.4.5 General Behavior

From:

The device manager shall supply execute operation parameters for a device consisting of:

1. Device manager IOR – The ID is “DEVICE_MGR_IOR” and the value is a string that is the *DeviceManager* stringified IOR.

2. Profile Name – The ID is “PROFILE_NAME” and the value is a CORBA string that is the full mounted file system file path name.

3. Device Identifier – The ID is “DEVICE_ID” and the value is a string that corresponds to the DCD *componentinstantiation id* attribute.

4. Device Label – The ID is “DEVICE_LABEL” and the value is a string that corresponds to the DCD *componentinstantiation usage* element. This parameter is only used when the DCD *componentinstantiation usage* element is specified.

5. Composite Device IOR - The ID is “Composite_DEVICE_IOR” and the value is a string that is an *AggregateDevice* stringified IOR. This parameter is only used when the DCD *componentinstantiation* element represents the child device of another *componentinstantiation* element.

6. The execute (“execparam”) properties as specified in the DCD for a *componentinstantiation* element. The device manager shall pass the componentinstantiation element “execparam” properties that have values as parameters. The device manager shall pass “execparam” parameters’ IDs and values as string values.

The device manager shall use the *componentinstantiation* element’s SPD *implementation* code’s *stacksize* and *priority* elements, when specified, for the *execute* operation options parameters. The device manager shall initialize and then configure logical devices that are started by the device manager, after they have successfully registered with the device manager. The device manager shall configure a DCD’s *componentinstantiation* element provided the *componentinstantiation* element has “configure” readwrite or writeonly properties with values. Figure 3-19 depicts a device manager startup scenario.

If a service is deployed by the device manager, the device manager shall supply execute operation parameters consisting of:

1. Device manager IOR – The ID is “DEVICE_MGR_IOR” and the value is a string that is the *DeviceManager* stringified IOR.

2. Service Name – The ID is “SERVICE_NAME” and the value is a string that corresponds to the DCD *componentinstantiation usagename* element.

3. The execute (“execparam”) properties as specified in the DCD for a *componentinstantiation* element. The device manager shall pass the componentinstantiation element “execparam” properties that have values as

parameters. The device manager shall pass "execparam" parameters' IDs and values as string values.

To:

The *DeviceManager* can launch *Devices, ComponentFactories* and *services* directly (e.g. *thread, posix_spawn*) or by using an *ExecutableDevice*. The *Devices* deployed this way shall register with the launching *DeviceManager* via *registerDevice* while *ComponentFactories* and *services* shall register via *registerService*. Upon successful registration, the *Devices* shall be added to the registeredDevices attribute of the *DeviceManager* while the *ComponentFactories* and *services* shall be added to the registeredServices attributes of the *DeviceManager*. *Devices* and *services* deployed using a *ComponentFactory* shall not register with the launching *DeviceManager*, however the *DeviceManager* shall add the *Devices* and services to the attributes registredDevices and registeredServices respectively.

3.1.3.2.4.5.1 Direct Launch

When a device is deployed directly (e.g. *thread*, *posix_spawn*) or by using an *ExecutableDevice*, the *DeviceManager* shall supply execute operation parameters for a device consisting of:

1. Device manager IOR – The ID is "DEVICE_MGR_IOR" and the value is a string that is the *DeviceManager* stringified IOR.

2. Profile Name – The ID is "PROFILE_NAME" and the value is a CORBA string that is the full mounted file system file path name.

3. Device Identifier – The ID is "DEVICE_ID" and the value is a string that corresponds to the DCD *componentinstantiation id* attribute.

4. Device Label – The ID is "DEVICE_LABEL" and the value is a string that corresponds to the DCD *componentinstantiation usage* element. This parameter is only used when the DCD *componentinstantiation usage* element is specified.

5. Composite Device IOR - The ID is "Composite_DEVICE_IOR" and the value is a string that is an *AggregateDevice* stringified IOR. This parameter is only used when the DCD *componentinstantiation* element represents the child device of another *componentinstantiation* element.

6. The execute ("execparam") properties as specified in the DCD for a *componentinstantiation* element. The device manager shall pass the componentinstantiation element "execparam" properties that have values as parameters. The device manager shall pass "execparam" parameters' IDs and values as string values.

The device manager shall use the *componentinstantiation* element's SPD *implementation* code's *stacksize* and *priority* elements, when specified, for the *execute* operation options parameters.

The device manager shall initialize and then configure logical devices that are launched by the device manager, after they have successfully registered with the device manager. The device manager shall configure a DCD's *componentinstantiation* element provided the *componentinstantiation* element has "configure" readwrite or writeonly properties with values. Figure 3-19 depicts a device manager startup scenario.

If a *ComponentFactory* or a service is laucnhed by the device manager, the device manager shall supply execute operation parameters consisting of:

1. Device manager IOR – The ID is "DEVICE_MGR_IOR" and the value is a string that is the *DeviceManager* stringified IOR.

2. Service Name – The ID is "SERVICE_NAME" and the value is a string that corresponds to the DCD *componentinstantiation usagename* element.

3. The execute ("execparam") properties as specified in the DCD for a *componentinstantiation* element. The device manager shall pass the componentinstantiation element "execparam" properties that have values as parameters. The device manager shall pass "execparam" parameters' IDs and values as string values.

3.1.3.2.4.5.2 ComponentFactory Launch

When a device is deployed via a ComponentFactory, the *DeviceManager* shall supply the following properties as the *qualifiers* parameter to the referenced *ComponentFactory createComponent* operation.

1. Profile Name – The ID is "PROFILE_NAME" and the value is a CORBA string that is the full mounted file system file path name.

2. Device Identifier – The ID is "DEVICE_ID" and the value is a string that corresponds to the DCD *componentinstantiation id* attribute.

3. Device Label – The ID is "DEVICE_LABEL" and the value is a string that corresponds to the DCD *componentinstantiation usage* element. This parameter is only used when the DCD *componentinstantiation usage* element is specified.

4. Composite Device IOR - The ID is "Composite_DEVICE_IOR" and the value is a string that is an *AggregateDevice* stringified IOR. This parameter is only used when the DCD *componentinstantiation* element represents the child device of another *componentinstantiation* element.

5. The *componentinstantiation componentfactoryref* element properties whose *kindtype* element is *factoryparam*

The device manager shall use the *componentinstantiation* element's SPD *implementation* code's *stacksize* and *priority* elements, when specified, as qualifiers parameters for the *createComponent* operation.

When a service is deployed via a ComponentFactory, the device manager shall supply the following properties as the *qualifiers* parameter to the referenced *ComponentFactory* component's *createComponent* operation.

1. Service Name – The ID is "SERVICE_NAME" and the value is a string that corresponds to the DCD *componentinstantiation usagename* element.

2. The *componentinstantiation componentfactoryref* element properties whose *kindtype* element is *factoryparam*

Change #10: Section 3.1.3.5.2
From:
A Software Component Descriptor (SCD) contains information about a specific SCA software component (*Resource, ResourceFactory, Device*). A Software Component Descriptor file shall have a ".scd.xml" extension. A Software Component Descriptor file contains information about the interfaces that a component provides and/or uses. A Software Component Descriptor for a *Device* type has a reference to Device Package Descriptor file.

To:
A Software Component Descriptor (SCD) contains information about a specific SCA software component (*Resource, ComponentFactory, Device*). A Software Component Descriptor file shall have a ".scd.xml" extension. A Software Component Descriptor file contains information about the interfaces that a component provides and/or uses. A Software Component Descriptor for a *Device* type has a reference to Device Package Descriptor file.

Change #11: Section 3.2.1.3
From:
Applications shall implement the Base Application Interfaces as specified in section 3.1.3.1 using the corresponding IDL in Appendix C. Use of the *ResourceFactory* interface per section 3.1.3.1.7 is optional.

To:
Applications shall implement the Base Application Interfaces as specified in section 3.1.3.1 using the corresponding IDL in Appendix C. Use of the *ComponentFactory* interface per section 3.1.3.1.7 by the ApplicationFactory is optional. Use of the *ComponentFactory* interface per section 3.1.3.1.7 by the *DeviceManager* is optional.

Appendix D

Change #12: Section D.2.1 Software Package *(last paragraph)*
From:
Any duplicate properties having the same ID are ignored. Duplicated properties must be the same property type, only the value can be over-ridden. The implementation properties are only used for the initial configuration and creation of a component by the CF ApplicationFactory and cannot be referenced by a SAD componentinstantiation, componentproperties or resourcefactoryproperties element.

To:
Any duplicate properties having the same ID are ignored. Duplicated properties must be the same property type, only the value can be over-ridden. The implementation properties are only used for the initial configuration and creation of a component by the CF ApplicationFactory and cannot be referenced by a SAD componentinstantiation, componentproperties or componentfactoryproperties element.

Change #13: D.4.1.1.6 kind
From:
1. configure, which is used in the configure and query operations of the CF Resource interface. The application factory will use the configure kind of properties to build the CF Properties input parameter to the configure operation that is invoked on the assemblycontroller component during application creation. The device manager will use the configure kind of properties to build the CF Properties input parameter to the configure operation that is invoked on components implementing the Device interface, during device creation. The application factory will also use the configure kind of properties for CF ResourceFactory create options parameters. When the mode is readonly, only the query behavior is supported. When the mode is writeonly, only the configure behavior is supported. When the mode is readwrite, both configure and query are supported.

To:
1. configure, which is used in the configure and query operations of the CF Resource interface. The application factory will use the configure kind of properties to build the CF Properties input parameter to the configure operation that is invoked on the assemblycontroller component during application creation. The device manager will use the configure kind of properties to build the CF Properties input parameter to the configure operation that is invoked on components implementing the Device interface, during device creation. The application factory will also use the configure kind of properties for CF ComponentFactory create options parameters. When the mode is readonly, only the query behavior is supported. When the mode is writeonly, only the configure behavior is supported. When the mode is readwrite, both configure and query are supported.

Change #14: Section D.4.1.1.6 kind
From:
5. factoryparam, which is used in the createResource operations of the CF ResourceFactory interface. The CF ApplicationFactory will use the factoryparam type of properties to build the CF Properties input parameter to the createResource operation.

To:
5. factoryparam, which is used in the createComponent operations of the CF ComponentFactory interface. The CF ApplicationFactory will use the factoryparam type of properties to build the CF Properties input parameter to the createComponent operation.

Change #15: Section D.4.1.4.1 configurationkind
From:
2. factoryparam, which is used in the createResource operations of the CF ResourceFactory interface. The CF ApplicationFactory will use the factoryparam kind of properties to build the CF Properties input parameter to the createResource() operation. A property can have multiple configurationkind elements and their default kindtype is “configure”.

To:
2. factoryparam, which is used in the createComponent operations of the CF ComponentFactory interface. The CF ApplicationFactory will use the factoryparam kind of properties to build the CF Properties input parameter to the createComponent() operation. A property can have multiple configurationkind elements and their default kindtype is “configure”.

Change #16: Section D.5 SOFTWARE COMPONENT DESCRIPTOR
From:
This descriptor file is based on the CORBA Component Descriptor specification. The SCA components CF Resource, CF Device, and CF ResourceFactory that are described by the software component descriptor are based on the SCA CF specification, and the following specification concentrates on definition of the elements necessary for describing the ports and interfaces of these components.

To:
This descriptor file is based on the CORBA Component Descriptor specification. The SCA components CF Resource, CF Device, and CF componentFactory that are described by the software component descriptor are based on the SCA CF specification, and the following specification concentrates on definition of the elements necessary for describing the ports and interfaces of these components.

Change #17: Section D.5.1.2 componentrepid
From:
The componentrepid uniquely identifies the interface that the component is implementing. The componentrepid may be referred to by the componentfeatures element. The componentrepid is derived from the CF Resource, CF Device, or CF ResourceFactory.

To:
The componentrepid uniquely identifies the interface that the component is implementing. The componentrepid may be referred to by the componentfeatures element. The componentrepid is derived from the CF Resource, CF Device, or CF ComponentFactory.

Change #18: Section D.5.1.3 componenttype
From:
The componenttype describes properties of the component. For SCA components, the component types include resource, device, resourcefactory, domainmanager, log, filesystem, filemanager, devicemanager, namingservice and eventservice.

To:
The componenttype describes properties of the component. For SCA components, the component types include resource, device, componentfactory, domainmanager, log, filesystem, filemanager, devicemanager, namingservice and eventservice.

Change #19: Section D.5.1.4 componentfeatures
From:
The componentfeatures element (see Figure D-18) is used to describe a component with respect to the components that it inherits from, the interfaces the component supports, and its provides and uses ports. At a minimum, the component interface has to be a CF Resource, CF ResourceFactory, or CF Device interface. If a component extends the CF Resource or CF Device interface then all the inherited interfaces (e.g., CF Resource) are depicted as supportsinterface elements.

To:
The componentfeatures element (see Figure D-18) is used to describe a component with respect to the components that it inherits from, the interfaces the component supports, and its provides and uses ports. At a minimum, the component interface has to be a CF Resource, CF ComponentFactory, or CF Device interface. If a component extends the CF Resource or CF

Device interface then all the inherited interfaces (e.g., CF Resource) are depicted as supportsinterface elements.

Change #20: Section D.6.1.3.1 componentplacement
From:
The componentplacement element (see Figure D-22) defines a particular deployment of a component. The component can be deployed either directly or by using a CF ResourceFactory. .

To:
The componentplacement element (see Figure D-22) defines a particular deployment of a component. The component can be deployed either directly or by using a CF ComponentFactory.

Change #21: Section D.6.1.3.3 componentinstantiation *(second paragraph)*
From:
1. The SAD partitioning : componentplacement : componentinstantiation : findcomponent : componentresourcefactoryref : resourcefactoryproperties element,

To:
1. The SAD partitioning : componentplacement : componentinstantiation : findcomponent : componentfactoryref : componentfactoryproperties element,

Change #22: Section D.6.1.3.3 componentinstantiation *(forth paragraph)*
From:
1. The componentresourcefactoryref element, which refers to a particular CF ResourceFactory componentinstantiation element found in the SAD, which is used to obtain a CF Resource instance for this componentinstantiation element. The refid attribute refers to a unique componentinstantiation id attribute. The componentresourcefactoryref element contains an optional resourcefactoryproperties element (see Figure D-26), which specifies the properties "qualifiers", for the CF ResourceFactory create call.

To:
1. The componentfactoryref element, which refers to a particular CF ComponentFactory componentinstantiation element found in the SAD, which is used to obtain a CF Resource instance for this componentinstantiation element. The refid attribute refers to a unique componentinstantiation id attribute. The componentfactoryref element contains an optional componentfactoryproperties element (see Figure D-26), which specifies the properties "qualifiers", for the CF ComponentFactory create call.

Change #23: Section D.6.1.3.3 componentinstantiation *(figure D-25)*
From:

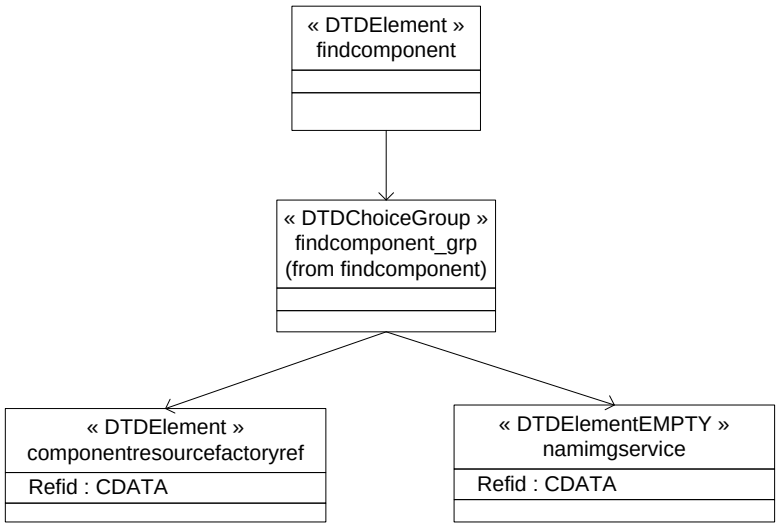


Figure D-25. *findcomponent* Element Relationships

```
<!ELEMENT findcomponent
( componentresourcefactoryref | namingservice )>
```

To:

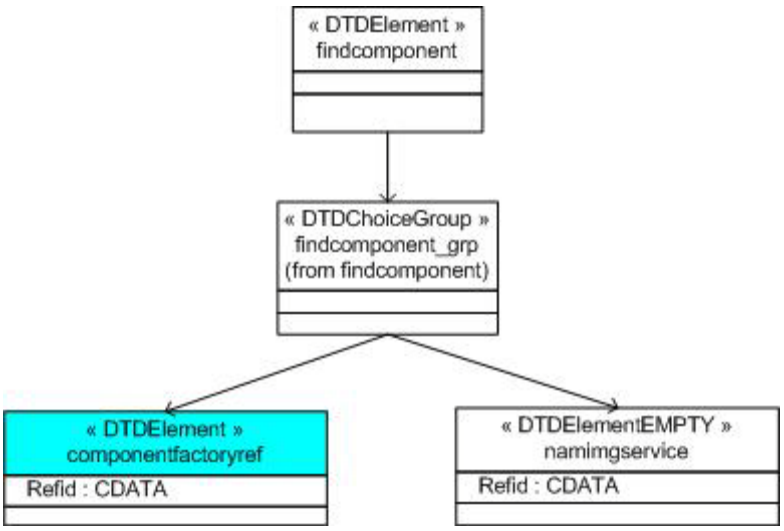


Figure D-25. *findcomponent* Element Relationships

```
<!ELEMENT findcomponent
( componentfactoryref | namingservice )>
```

Change #24: Section D.6.1.3.3 componentinstantiation *(figureD-26)*
From:

```
<!ELEMENT componentresourcefactoryref
     ( resourcefactoryproperties? )>
<!ATTLIST componentresourcefactoryref
refid CDATA #REQUIRED>
```

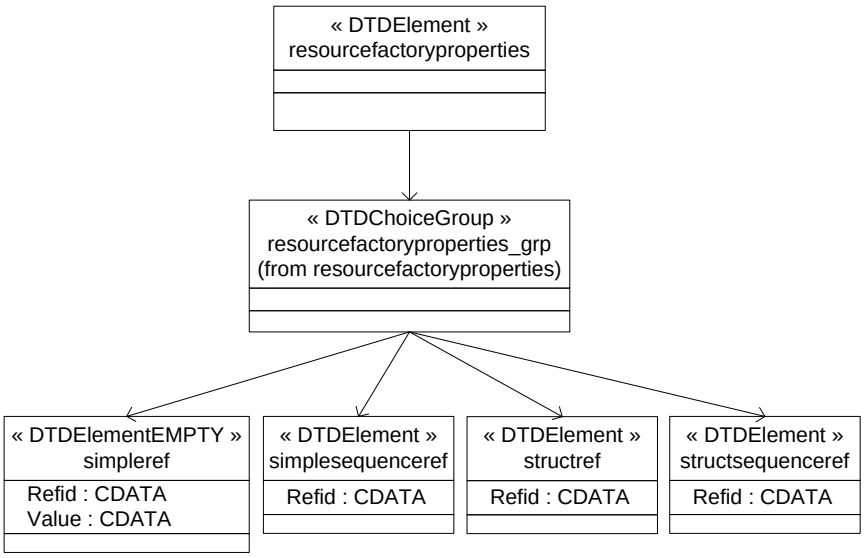


Figure D-26. *resourcefactoryproperties* Element Relationships

```
<!ELEMENT resourcefactoryproperties
( simpleref | simplesequenceref | structref | structsequenceref )+
>
```

To:

```
<!ELEMENT componentfactoryref
     ( componentfactoryproperties? )>
<!ATTLIST componentfactoryref
                        refid CDATA #REQUIRED>
```

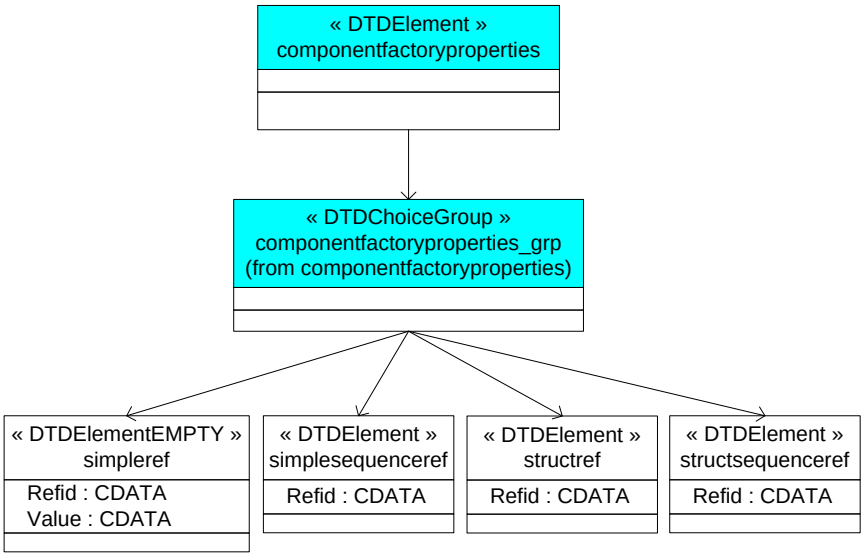


Figure D-26. *componentfactoryproperties* Element Relationships

```
<!ELEMENT componentfactoryproperties
( simpleref | simplesequenceref | structref | structsequenceref )+
>
```

Change 25: Section D.7.1.4.1.6 componentinstantiation (*after first paragraph***)**
From:

The *componentinstantiation* element (see Figure D-35) is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation*‘s id attribute is a DCE UUID that uniquely identifier the component. The id is a DCE UUID value as specified in section D.2.1. The *componentinstantiation* contains a *usagename* element that is intended for an applicable name for the component. The optional *componentproperties* element (see Figure D-36) is a list of property values that are used in configuring the component. D.6.1.3.3 defines the property list for the *componentinstantiation* element, which contains initial properties values. For a component service type (e.g,, Log), the *usagename* element is not optional and needs to be unique for each service type.

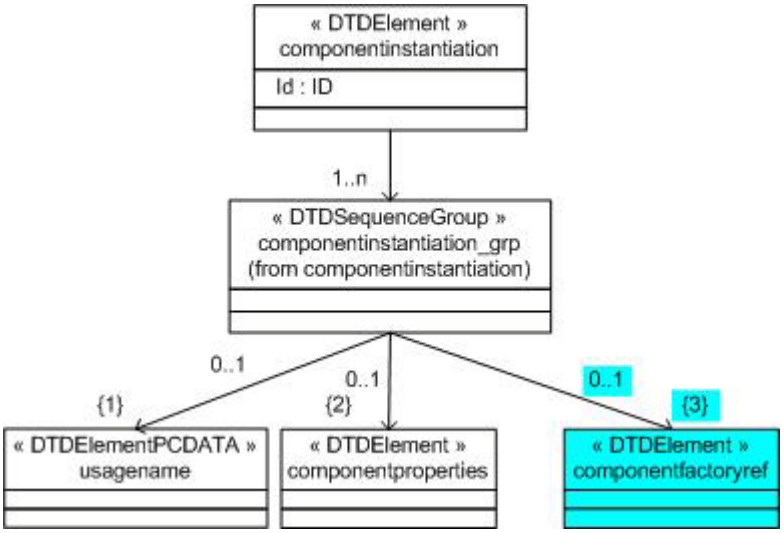


Figure D-35. *componentinstantiation* Element Relationships

```
<!ELEMENT componentinstantiation
( usagename?
,componentproperties?
)>


<!ATTLIST componentinstantiation
id ID #REQUIRED>
<!ELEMENT usagename (#PCDATA)>
```

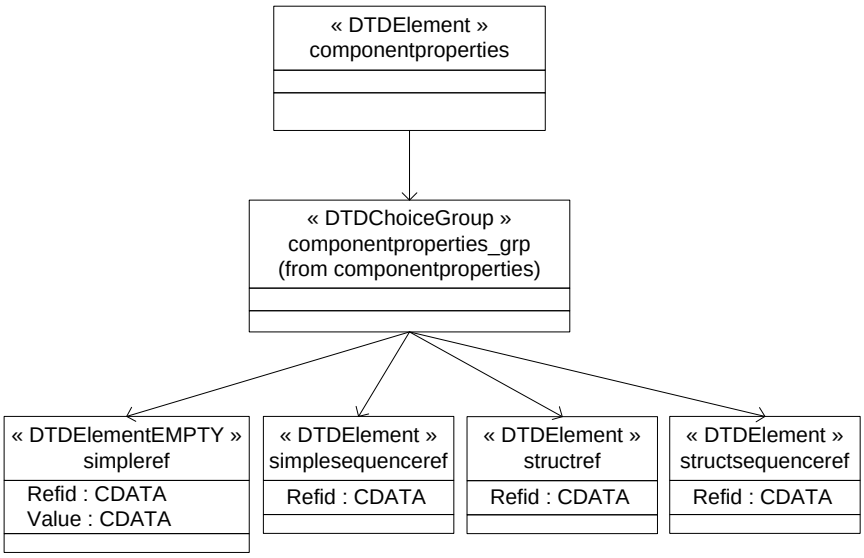


Figure D-36. *componentproperties* Element Relationships

```
<!ELEMENT componentproperties
( simpleref | simplesequenceref | structref |
structsequenceref )+ >

<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
refid CDATA #REQUIRED
value CDATA #REQUIRED>
<!ELEMENT simplesequenceref
( values )>
<!ATTLIST simplesequenceref
refid CDATA #REQUIRED>
<!ELEMENT structref
( simpleref+ )>
<!ATTLIST structref
refid CDATA #REQUIRED>
<!ELEMENT structsequenceref
( structvalue+ )>
<!ATTLIST structsequenceref
refid CDATA #REQUIRED>
<!ELEMENT structvalue
( simpleref+ )>

<!ELEMENT values
( value+ )>
```

```
<!ELEMENT value (#PCDATA)>
```

To:

The *componentinstantiation* element (see Figure D-35) is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation*‘s id attribute is a DCE UUID that uniquely identifier the component. The id is a DCE UUID value as specified in section D.2.1. The *componentinstantiation* contains a *usagename* element that is intended for an applicable name for the component. The optional *componentproperties* element (see Figure D-36) is a list of property values that are used in configuring the component. D.6.1.3.3 defines the property list for the *componentinstantiation* element, which contains initial properties values. For a component service type (e.g,, Log), the *usagename* element is not optional and needs to be unique for each service type.

The optional *componentfactoryref* element (see Figure D-36) refers to a particular CF *ComponentFactory componentinstantiation* element found in the DCD, which is used to obtain a CF *Device* or a CF *Service* instance for this *componentinstantiation* element. The refid attribute refers to a unique componentinstantiation id attribute. The *componentfactoryref* element contains an optional *componentfactoryproperties* element (see Figure D-37’’), which specifies the properties “qualifiers”, for the CF *ComponentFactory create* call. The optional *componentfactoryref* element should be specified only when a *ComponentFactory* is used to create *Device* or *Service* components.

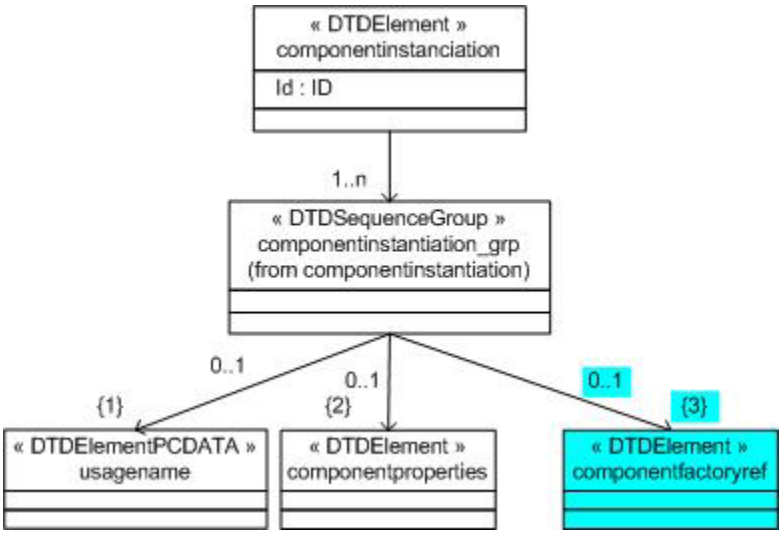


Figure D-36. *componentproperties* Element Relationships

```
<!ELEMENT componentinstantiation
      ( usagename?
      , componentproperties?
      , componentfactoryref?
)>
<!ATTLIST componentinstantiation
```

```
id ID #REQUIRED>

<!ELEMENT usagename (#PCDATA)>
```

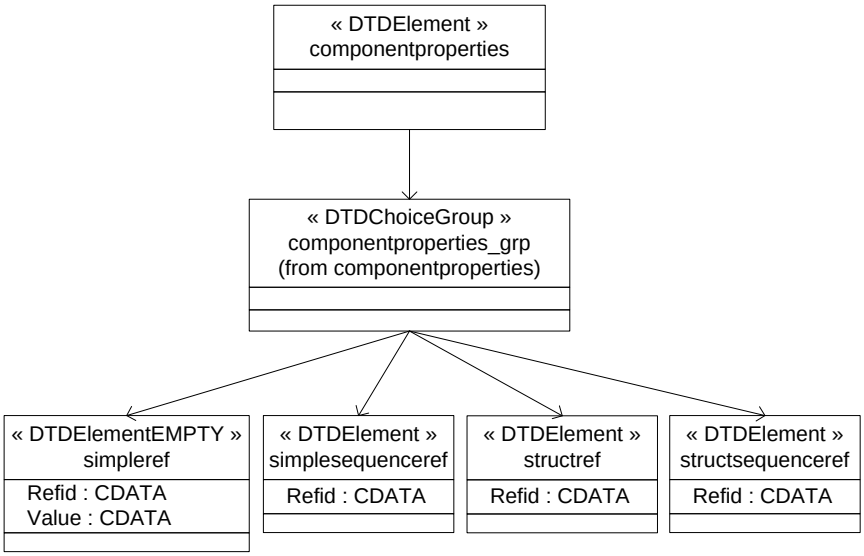


Figure D-36. *componentproperties* Element Relationships

```
<!ELEMENT componentproperties
( simpleref | simplesequenceref | structref |
structsequenceref )+ >

<!ELEMENT componentfactoryref
      ( componentfactoryproperties? )>
<!ATTLIST componentfactoryref
                        refid CDATA #REQUIRED>
```

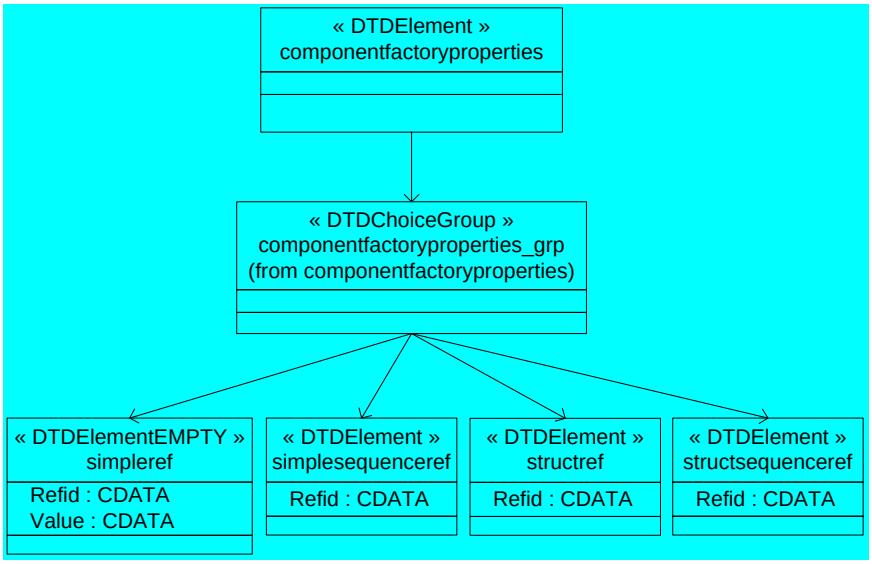


Figure D-36'. *componentfactoryproperties* Element Relationships

```
<!ELEMENT componentfactoryproperties
( simpleref | simplesequenceref | structref | structsequenceref )+
>
<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
refid CDATA #REQUIRED
value CDATA #REQUIRED>
<!ELEMENT simplesequenceref
( values )>
<!ATTLIST simplesequenceref
refid CDATA #REQUIRED>
<!ELEMENT structref
( simpleref+ )>
<!ATTLIST structref
refid CDATA #REQUIRED>
<!ELEMENT structsequenceref
( structvalue+ )>
<!ATTLIST structsequenceref
refid CDATA #REQUIRED>
<!ELEMENT structvalue
( simpleref+ )>

<!ELEMENT values
( value+ )>
<!ELEMENT value (#PCDATA)>
```

5 Appendix A – Interface Definition Language

```
/* A ComponentFactory can be used to create and destroy a
Component. */

interface ComponentFactory extends LifeCycle

{

/* The CreateComponentFailure exception indicates that the
createComponent operation failed to create the Component. The
message is component-dependent, providing additional information
describing the reason for the error. */

exception CreateComponentFailure {

CF::ErrorNumberType errorNumber;

string msg;

};


/* The readonly identifier attribute contains the unique identifier
for a ComponentFactory instance. */


readonly attribute string identifier;


/* The createComponent operation provides the capability to create
Components. */


CORBA::Object createComponent(in string componentId,
                              in CF::Properties qualifiers)
raises (CF::ComponentFactory::CreateComponentFailure);


/* The getComponent operation provides the capability to obtain a
component that was previously created. When the component does not
exists a nil CORBA object reference is returned */


CORBA::Object getComponent(in string componentId);


/* In CORBA there is client side and server side representation of
a Component. This operation provides the mechanism of releasing the
component in the CORBA environment on the server side. This method
```

```
should only be called once since the server will be destroyed. The
client still has to release its client side reference of the
Component. true is returned indicating that the component server
has been successfully released, otherwise false is returned
indicating that no component was released i.e. server did not
exists*/


boolean releaseComponent(in string ComponentId);
};
```