Communications Research Centre Canada
An Agency of Industry Canada

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

Software Communications Architecture v2.2 Reference Implementation Project

SDRF Forum Contract

SDRF-04-A-0002-V0.00

Deliverable 3

Software Communications Architecture Reference Implementation (SCARI) Project

Presented to:

Software Defined Radio Forum
Denver, Colorado
USA

Prepared by:

Communications Research Centre Canada
Advanced Radio System Research Group
Ottawa, Ontario

10 February 2005

Document Prepared By	S. Bernier, CRC / VPSAT / RARS H. Latour, CRC / VPSAT / RARS F. Lévesque, CRC / VPSAT / RARS N. Le, CRC / VPSAT / RARS
Approved By	C. Bélisle Research Management Authority Research Manager CRC / VPSAT / RARS
Submitted To	Software Defined Radio Forum
Date	February 10, 2005

TABLE OF CONTENTS

List of Figures

List of Tables

1. Application Waveform and Related Tools

Section 1.1 contains a description of all use cases illustrating the most important features of interfaces. For reasons of simplicity, these scenarios are executed on a single-node radio since it requires only one computer. Section 1.2 contains a description of the implemented interfaces. Section 2 describes the components of the Demonstration Application which performs two different effects on an audio stream: echo and chorus. Also described in that section is how to use two simplex voice applications to create a full duplex link between two nodes of a single radio.

1.1 Use Case Diagram

This section contains a description of the use cases. This set of use cases illustrates the most important features of the CF interfaces associated with the creation and execution of waveform applications. Each use case is described by at least one scenario. The scenario is described using a textual description and some UML diagrams (ex: sequence diagrams). In order to execute a scenario, follow the instructions described in the section 'Test Procedure' for each scenario

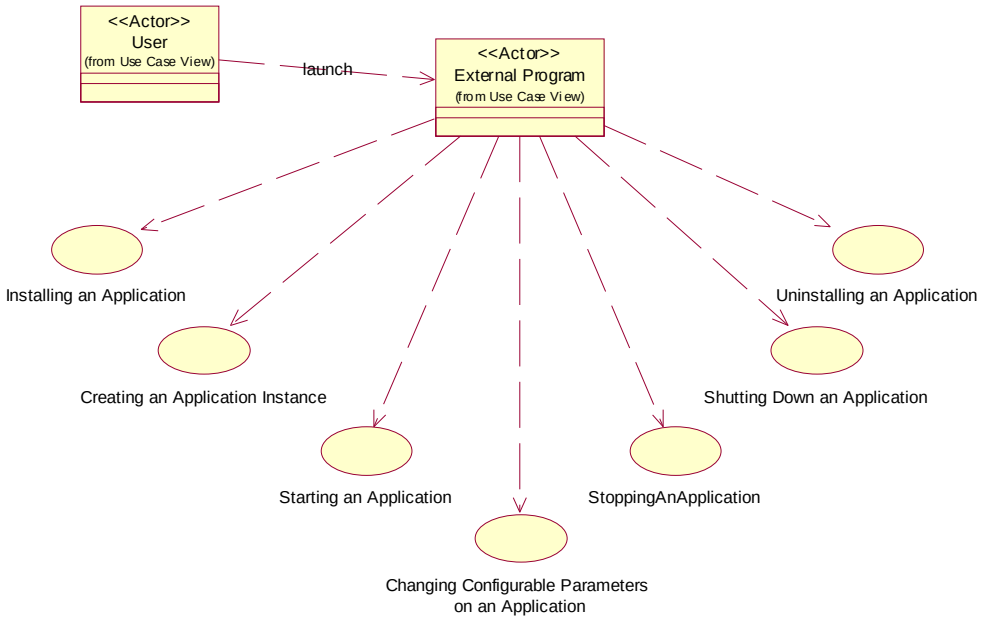

Figure 1 – Use Case Diagram

Use Case	Scenario ID
Install an Application	CF.InstallApplication
Create an Application Instance	CF.CreateApplicationInstance CF.ApplicationFactoryCreateApplication
Start an Application	CF.StartApplication
Change Configurable Parameters on an Application	CF.ChangeApplicationConfigurableParameters
Stop an Application	CF.StopApplication
Shut Down an Application	CF.ApplicationRelease
Uninstall an Application	CF.DomainManagerUninstallApplication

Table 1 – Use Case Name and Scenario Relationship

Figure 1 contains the use case diagram and Table 1 defines the correspondence between each use case and its scenarios. Each scenario is described in the following sections.

1.1.1 CF.InstallApplication Scenario

1.1.1.1 Scenario Textual Description

ID: CF. InstallApplication

Description: Illustrates how a waveform application can be installed on the radio (remotely or not) using the GUI called *PackagedApplicationInstaller*.

Actor(s): User

Precondition: The Naming Service accepts request and a DomainManager is registered in an active Naming Server.

Responsibilities:

1. The *User* starts a GUI *PackagedApplicationInstaller*
2. The *PackagedApplicationInstaller* gets the *domainManager* from the *NamingService*
3. The *NamingService* returns the *domainManager*
4. The *PackagedApplicationInstaller* gets the *fileManager* of the *domainManager*
5. The *domainManager* returns the *fileManager*
6. The *User* specifies the name of an application file packaged as a JAR that contains all the files of an application
7. The *User* clicks on the Install button
8. The *PackagedApplicationInstaller* creates a directory named after the application name in the *fileManager*

For each file included in the packaged application file the steps 9 to 14 are performed

9. The *PackagedApplicationInstaller* reads the content of an application source file in the native file system
10. The *PackagedApplicationInstaller* creates a destination file in the *fileManager* that has the same name as the native source file
11. The *fileManager* creates the *destinationFile*
12. The *fileManager* returns the *destinationFile*
13. The *PackagedApplicationInstaller* writes the content of the source file in the *destinationFile*
14. The *PackagedApplicationInstaller* closes the *destinationFile*

15. The *PackagedApplicationInstaller* uses the *domainManager* to install the application
16. The *domainManager* creates an *applicationFactory* for the application

17. The *domainManager* logs that the installation has succeeded
18. The *PackagedApplicationInstaller* gets the list of application factories from the *domainManager*
19. The *domainManager* returns the list of application factories
20. The *PackagedApplicationInstaller* displays the application factories in a list

Results: Results are displayed in the *PackagedApplicationInstaller*.

Services Used DomainManager.fileManager(), DomainManager.installApplication(…), DomainManager.applicationFactories(), FileManager.mkdir(…), FileManager.create(…), File.write(…), File.close(), Log.writeRecords(…)

Alternative Scenarios: None

1.1.1.2 UML Sequence Diagram

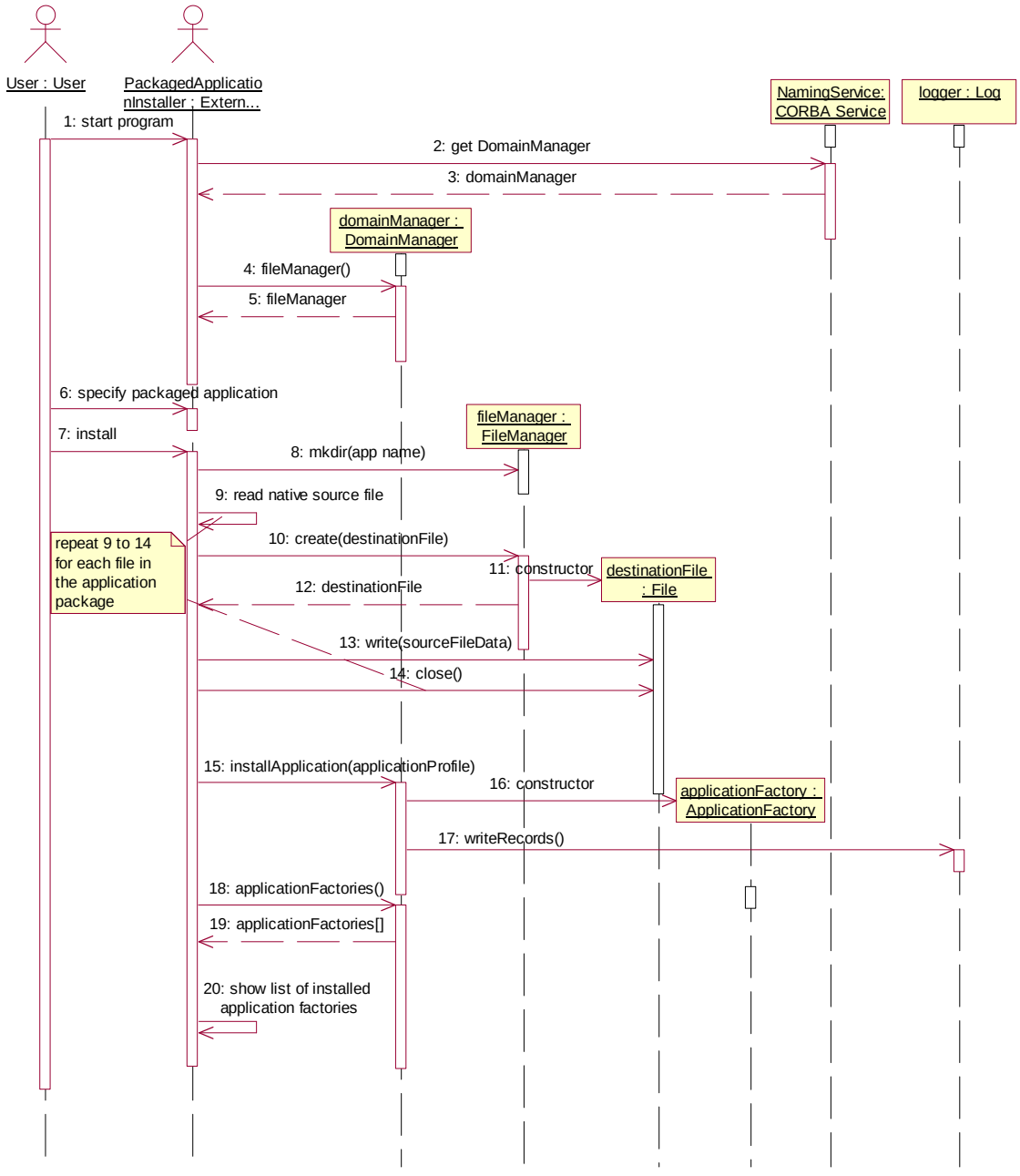

Figure 2 - CF.InstallApplication Sequence Diagram

1.1.2 CF.CreateApplicationInstance Scenario

1.1.2.1 Scenario Textual Description

ID: CF.CreateAnApplicationInstance

Description: Illustrates how an installed waveform application is started.

Actor(s): User

Precondition: The Naming Server accepts request, a DomainManager is registered with the active Naming Server, and the application AudioEffect0 is installed.

Responsibilities:

1. The *User* starts a GUI *ApplicationManager*
2. The *ApplicationManager* gets the *domainManager* Object reference from the *NamingService*.
3. The *NamingService* returns the *domainManager* Object reference.
4. The *ApplicationManager* gets the *domainManager's* application factories.
5. The *domainManager* returns an array of applicationFactories.
6. The *ApplicationManager* displays the applicationFactories in the GUI.
7. The *ApplicationManager* gets the *domainManager's* applications.
8. The *domainManager* returns an array of applications.
9. The *ApplicationManager* displays the applications in the GUI.
10. The *User* selects the *AudioEffect0* application factory in the GUI.
11. The *User* clicks on the Create Application button.
12. The *ApplicationManager* shows an input dialog.
13. The *User* enters the application name and presses the Ok button
14. The *ApplicationManager* creates an *AudioEffect0's* application.
15. The *AudioEffect0* factory instantiates an application.
16. The *AudioEffect0* registers the application with the *domainManager*.
17. The *AudioEffect0* returns the application to the *ApplicationManager*.
18. The *ApplicationManager* displays the newly created application in the Applications list.

Results: Results are displayed in the *ApplicationManager*.

Services Used: DomainManager.applicationFactories(…), DomainManager.applications(…), ApplicationFactory.create(…)

Alternative Scenarios: None

1.1.2.2 UML Sequence Diagram

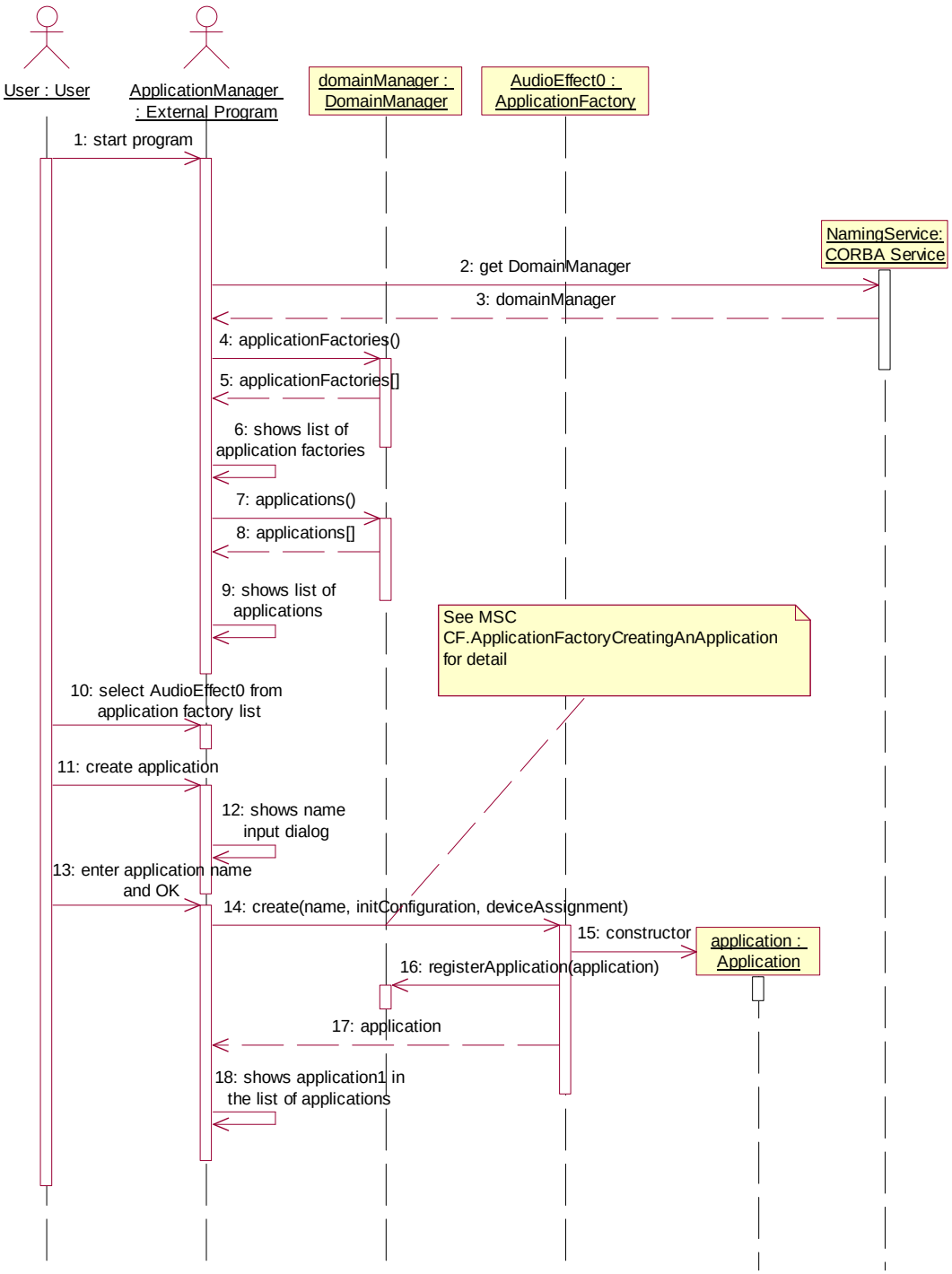

Figure 3 - CF. CreateApplicationInstance Sequence Diagram

1.1.3 CF.ApplicationFactoryCreateApplication Scenario

1.1.3.1 Scenario Textual Description

ID: CF.ApplicationFactoryCreatingAnApplication

Description: The ApplicationManager creates an application using an ApplicationFactory that is installed in the DomainManager.

Actor(s): ApplicationManager (GUI)

Precondition The Naming Server accepts request, a DomainManager is registered with the active Naming Server, and the application AudioEffect0 is installed.

Responsibilities:

1. The *ApplicationManager* invokes the *ApplicationFactoryImpl* to create an *Application* Object.
2. The *ApplicationFactoryImpl* creates the *Application* object.
3. The *ApplicationFactoryImpl* gets a list of devices that have registered with the *domainManager*.
4. The *domainManager* returns an array of devices with their matching properties.
5. The *ApplicationFactoryImpl* gets the list of components from the *SADParser*.
6. The *SADParser* returns an array of SADComponentPlacement.
7. The *ApplicationFactoryImpl* creates a SADDeployedComponentLauncher for each component of the *application*.

Steps 8 to 11 are performed if a device assignment greater than 0 has been given to the create operation.

8. The *ApplicationFactoryImpl* queries the *launcher* to see if it is responsible to launch an instance.
9. The *launcher* returns a Boolean to the *ApplicationFactoryImpl* indicating if the instance specified by the device assignment belongs to the *launcher*.
10. The *ApplicationFactoryImpl* looks for the device object reference that matched the device identifier specified in the device assignment.
11. The *ApplicationFactoryImpl* assigns a device to the *launcher*.

12. The *ApplicationFactoryImpl* launches the application using the *launcher*.
13. The *launcher* looks for the next device capable of loading the component if a device has not been assigned.

14. The *launcher* chooses an implementation that matches the device selected.
15. The *launcher* creates a launchTransactionManager with the selected device.
16. The *launcher* allocates capacity required by the all components' dependencies.
17. The *ltm* returns true since it has successfully allocated all capacity requested.
18. The *launcher* allocates capacity required by the component itself.
19. The *ltm* returns true since it has successfully allocated all capacity requested.
20. The *launcher* loads the components' code files.
21. The *launcher* executes the components' code file on the executable device via the *ltm*.
22. The *ltm* returns the process id that was returned by the executable device.
23. The *ApplicationFactoryImpl* builds the naming context association list.
24. The *ApplicationFactoryImpl* builds the device assignment association list.
25. The *ApplicationFactoryImpl* builds the process id association list.
26. The *ApplicationFactoryImpl* builds the implementation association list.
27. The *ApplicationFactoryImpl* looks for the resources object reference using the *NamingService*.
28. The *NamingService* returns the object reference.
29. The *ApplicationFactoryImpl* initializes the resources.
30. The *ApplicationFactoryImpl* gets the application's connections from the *SADParser*.
31. The *SADParser* returns an array of connections.
32. The *ApplicationFactoryImpl* adds all the application's connections to be established to the *connectionManager*.
33. The *ApplicationFactoryImpl* configures the resources.
34. The *ApplicationFactoryImpl* configures the assembly controller with the initConfiguration properties provided by the actor.
35. The *ApplicationFactoryImpl* registers the application with the *domainManager*.
36. The *ApplicationFactoryImpl* writes a log record for the successful creation of the application.
37. The *ApplicationFactoryImpl* returns the application to the *ApplicationManager*.

Results: Results are displayed in the *ApplicationManager*.

Services Used ApplicationFactory.create(…), Device.allocateCapacity(…), LoadableDevice.load(…), ExecutableDevice.execute(…), LifeCycle.intialize(…), PropertySet.configure(…), Log.writeRecord(..)

Alternative Scenarios: None

1.1.3.2 UML Sequence Diagram

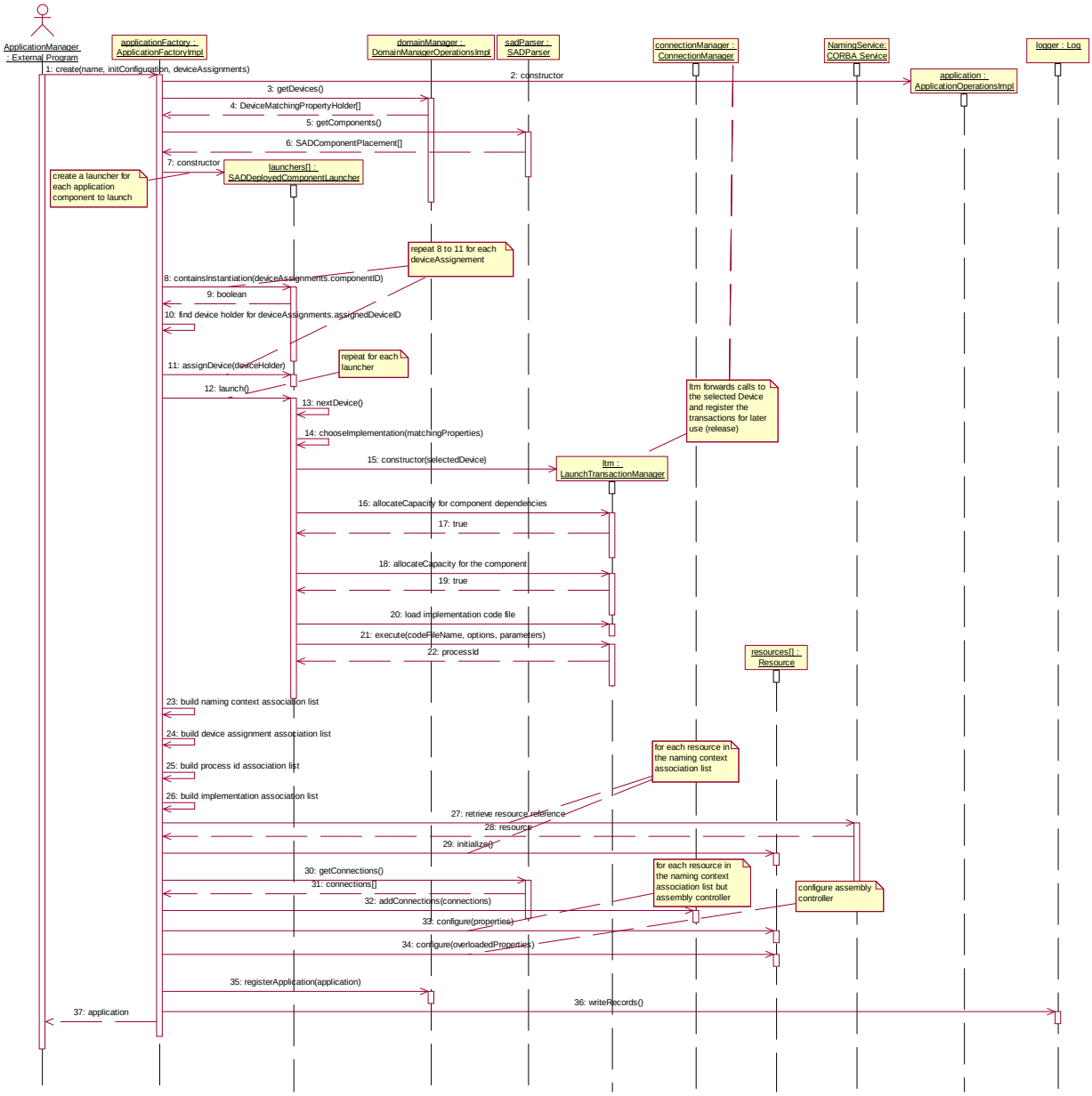

Figure 4 - CF. ApplicationFactoryCreateApplication Sequence Diagram

1.1.4 CF. StartApplication Scenario

1.1.4.1 Scenario Textual Description

ID: CF.StartApplication

Description: Illustrates how to start a waveform application such that it will begin to process data.

Actor(s): User

Precondition: The Naming Server accepts request, a DomainManager is registered with the active Naming Server, and the application AudioEffect0 is installed. An instance of the application AudioEffect0, named ‘application’ is created. The User has started the GUI ApplicationManager, selected the application AudioEffect0 from the list of application type, and selected the application instance named ‘application’ from the list of applications.

Responsibilities:

1. The *User* presses the Start button of the *ApplicationManager*
2. The *ApplicationManager* starts the a*pplication*
3. The *application* starts its *assemblyController*
4. The *assemblyController* starts the *node1AudioDevice*
5. The *assemblyController* starts the *echoResource* of the application
6. The *assemblyController* starts the chorusResource of the application

Results: The *User* can speak in the microphone and eared its voice modified with echo and chorus.

Services Used Resource.start()

Alternative Scenarios: None

1.1.4.2 UML Sequence Diagram

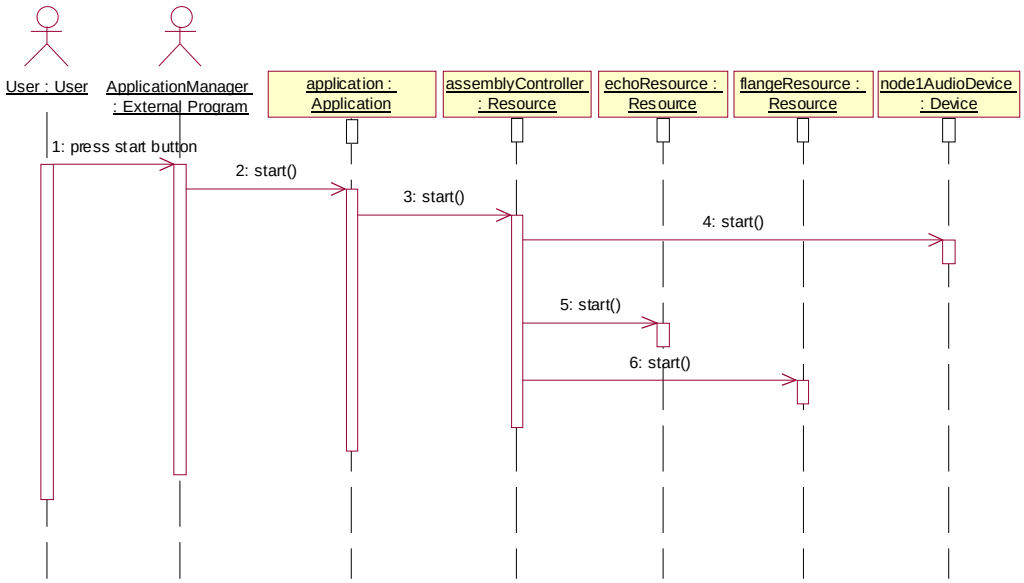

Figure 5 - CF. StartApplication Sequence Diagram

1.1.5 CF.ChangeApplicationConfigurableParameters Scenario

1.1.5.1 Scenario Textual Description

ID: CF.ChangeApplicationConfigurableParameters

Description: Illustrates how a waveform application can be controlled by configuring its properties with the ApplicationManager GUI.

Actor(s): User

Precondition: The Naming Server accepts request, a DomainManager is registered with the active Naming Server, and the application AudioEffect0 is installed. An instance of the application AudioEffect0, named 'application' is created and started. The User has started the GUI ApplicationManager, selected the application AudioEffect0 from the list of application type, and selected the application instance named 'application' from the list of applications.

Responsibilities:

1. The *User* changes the ECHO_GAIN parameter value in the *ApplicationManager*
2. The *ApplicationManager* configures the parameter ECHO_GAIN of the *application*
3. The *application* configures the parameter ECHO_GAIN of its *assemblyController*
4. The *assemblyController* configures the parameter ECHO_GAIN of its *echoResource*
5. The *User* speaks in the microphone and hears its voice affected by the new value of the ECHO_GAIN parameter
6. The *User* double clicks on the NUMBER_OF_VOICES parameter value in property table of the *ApplicationManager*, modifies the value and presses enter
7. The *ApplicationManager* configures the parameter NUMBER_OF_VOICES of the *application*
8. The *application* configures the parameter NUMBER_OF_VOICES of its *assemblyController*
9. The *assemblyController* configures the parameter NUMBER_OF_VOICES of its *chorusResource*
10. The *User* speaks in the microphone and hears its voice affected by the new value of the NUMBER_OF_VOICES parameter

Results: The *User* can speak in the microphone and heard its voice modified with the new value of the parameters.

Services Used: Resource.configure(…)

Alternative Scenarios: None

1.1.5.2 UML Sequence Diagram

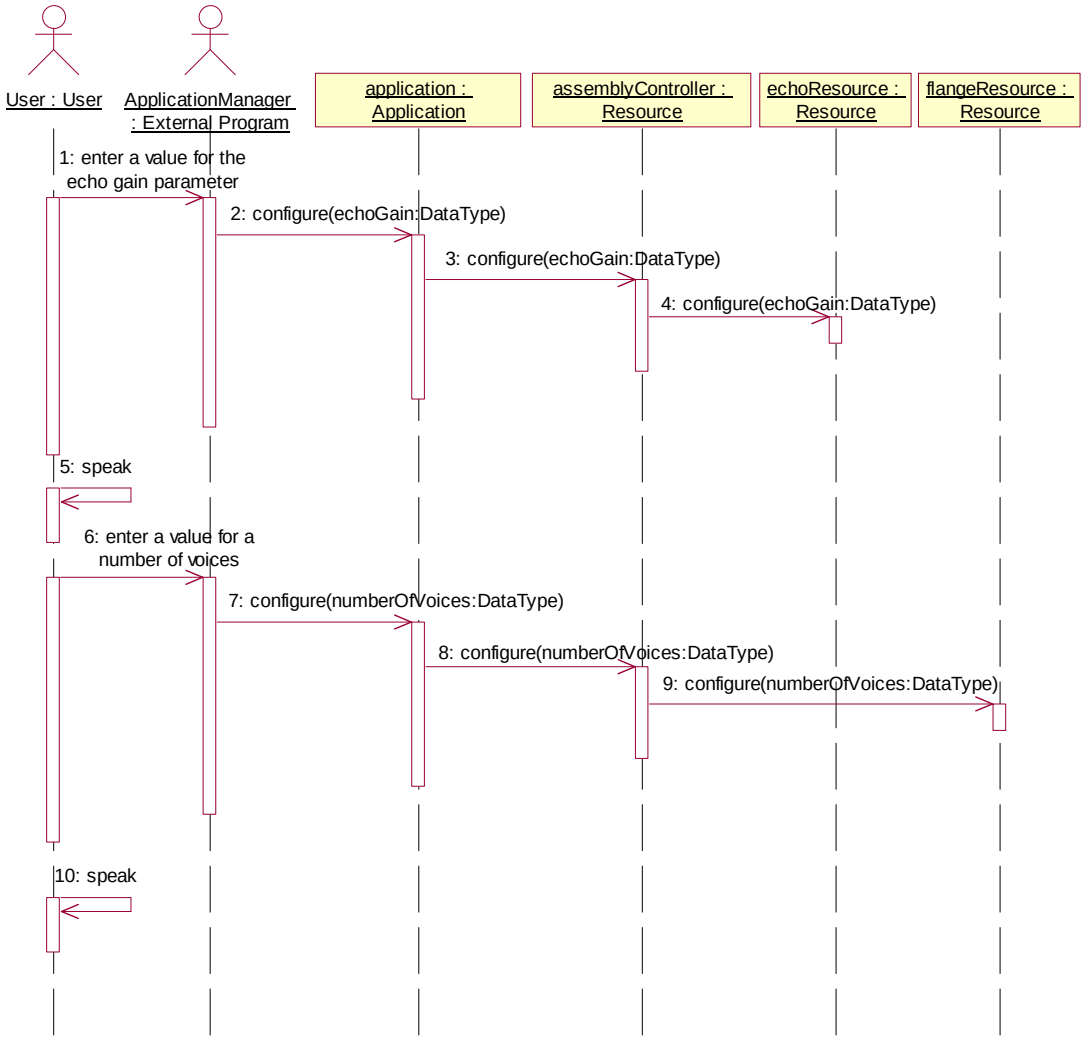

Figure 6 - CF. ChangeApplicationConfigurableParameters Sequence Diagram

1.1.6 CF.StopApplication Scenario

1.1.6.1 Scenario Textual Description

ID: CF.StopApplication

Description: Illustrates how to stop the execution of a waveform application such that it will interrupt the processing of data.

Actor(s): User

Precondition: The Naming Server accepts request, a DomainManager is registered with the active Naming Server, and the application AudioEffect0 is installed. An instance of the application AudioEffect0, named 'application' is created and started. The User has started the GUI ApplicationManager, selected the application AudioEffect0 from the list of application type, and selected the application instance named 'application' from the list of applications.

Responsibilities:

1. The *User* presses the Stop button of the *ApplicationManager*
2. The *ApplicationManager* stops the a*pplication*
3. The *application* stops its *assemblyController*
4. The *assemblyController* stops the *node1AudioDevice*
   Note: At this point, the waveform application stops receving new data from the AudioDevice. Therefore, its execution continues until the current data if flushed out of the waveform *Resource*s.
5. The *assemblyController* stops the *echoResource* of the application
6. The *assemblyController* stops the chorusResource of the application

Results: The *User* cannot speak in the microphone and hear its voice in the speaker.

Services Used: Resource.stop()

Alternative Scenarios: None

1.1.6.2 UML Sequence Diagram

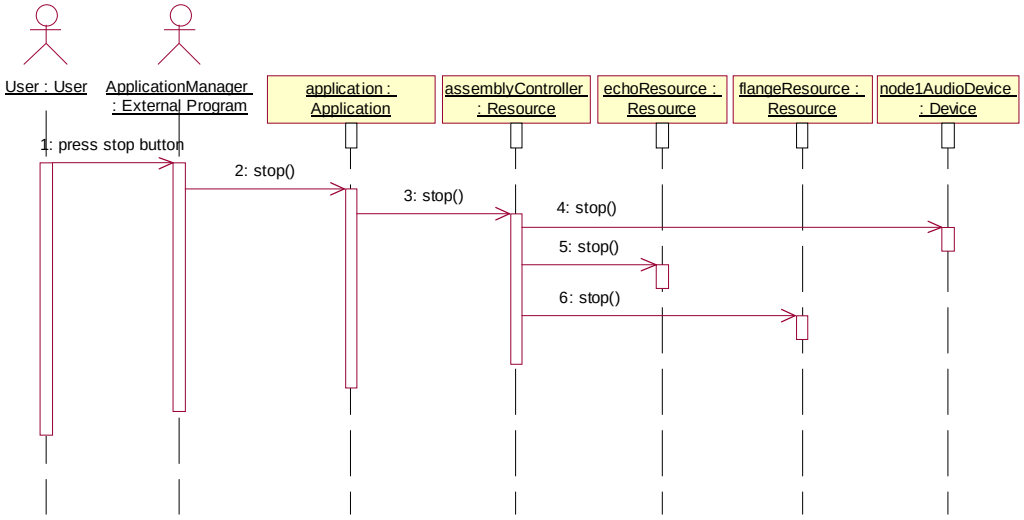

Figure 7 - CF.StopApplication Sequence Diagram

1.1.7 CF.ApplicationRelease Scenario

1.1.7.1 Scenario Textual Description

ID: CF.ApplicationRelease

Description: Illustrates how a waveform application is released from a radio. After the release operation is performed, the waveform application is not available for being started anymore.

Actor(s): ApplicationManager (GUI)

Precondition: The Naming Server accepts request, a DomainManager is registered with the active Naming Server, and the application AudioEffect0 is installed. An instance of the application AudioEffect0, named 'application' is created and started. The User has started the GUI ApplicationManager, selected the application AudioEffect0 from the list of application type, selected the application instance named 'application' from the list of applications, and pressed the Shutdown button.

Responsibilities:

1. The *ApplicationManager* releases the *application* by calling the releaseObject method.
2. The *application* informs *AudioEffect0* ApplicationFactory that it is being release.
3. The *AudioEffect0* retrieves the component launchers for the application for later use.
4. The *AudioEffect0* unregisters the application from *DomainManager's*.
5. The *AudioEffect0* retrieves all the object reference specified in the naming context association list from the *NamingService*.
6. The NamingService returns the component object references.
7. The *AudioEffect0* informs the *connectionManager* that a component is leaving.
8. The *AudioEffect0* releases each component resource.
9. The *AudioEffect0* uses each launcher to free the target device of processes, code files and allocated capacity.
10. The *launcher* asks the *ltm* to free the target device using all the transactions that have been performed on the target device.
11. The *AudioEffect0* unbinds all the *application*'s components from the *NamingService*.
12. The *AudioEffect0* releases all the launchers that were used for the *application*.

13. The *application* writes a log record for the successful release of the application.

Results: The waveform application can't be started unless it is re-created. Graphically speaking, the application name does not appear in the created application list (right-hand pane) of the ApplicationManager.

Services Used LifeCycle.releaseObject()

Alternative Scenarios: None

1.1.7.2 UML Sequence Diagram

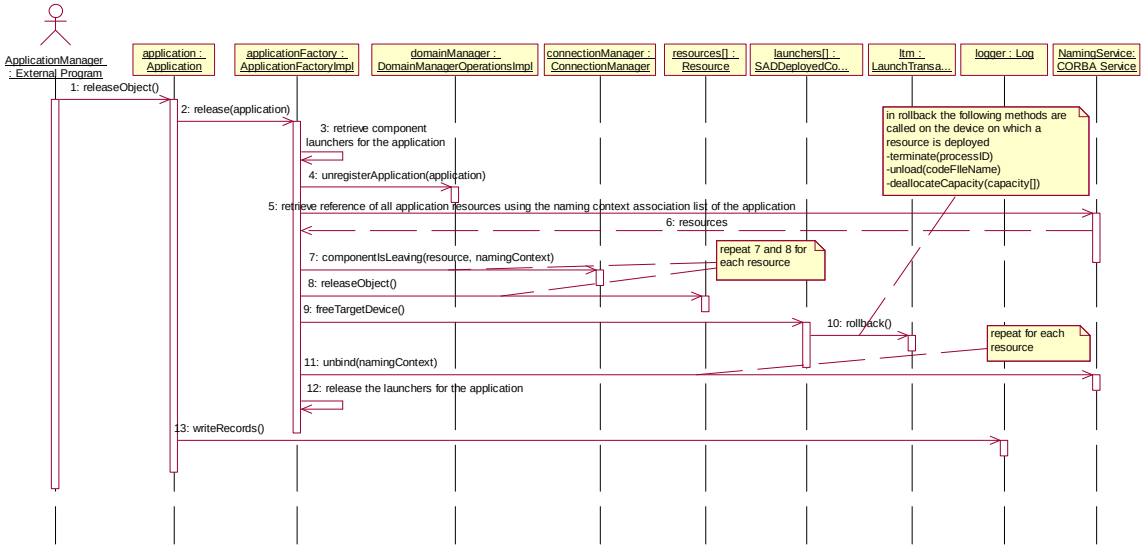

Figure 8 - CF. ApplicationRelease Sequence Diagram

1.1.8 CF.DomainManagerUninstallApplication Scenario

1.1.8.1 Scenario Textual Description

ID: CF. DomanManagerUninstallApplication

Description: Illustrates how a waveform application is removed from a radio such that it will not be available for use anymore.

Actor(s): PackagedApplicationInstaller (GUI)

Precondition: The Naming Server accepts request, a DomainManager is registered with the active Naming Server, and the application AudioEffect0 is installed. The User has started the GUI PackagedApplicationInstaller, selected the application AudioEffect0 from the list, and pressed the Uninstall button.

Responsibilities:

1. The GUI *PackagedApplicationInstaller* ask the *DomainManager* to uninstall the application
2. The *DomainManager* retrieves from its internal list the *applicationFactory* that has created the application AudioEffect0
3. The *DomainManager* remove from its internal list the *applicationFactory*
4. The *DomainManager* retrieves from its internal list the applications created by the *applicationFactory*

For each application created by the *applicationFactory* the step 5 and 6 are performed

5. The *DomainManager* stops an application created by the *applicationFactory*
6. The *DomainManager* releases the application created by the *applicationFactory*

7. The *DomainManager* deletes from its *fileManager* the files and directories related to the application AudioEffect0
8. The *DomainManager* removes AudioEffect0 from its list of applications to install at boot up
9. The *DomainManager* logs that the uninstallation of AudioEffect0 has succeed.

Results The waveform application can't be created anymore. Graphically speaking, the waveform application name (that is a 'type' application) does not appear in the installed application list (left-hand pane) of the ApplicationManager.

Services Used: DomainManager.uninstallApplication(…), Resource.stop(), LifeCycle.releaseObject(), FileManager.list(…), FileManager.remove(…), FileManager.rmDir(…), Log.writeRecords(…).

Alternative Scenarios: None

1.1.8.2 UML Sequence Diagram

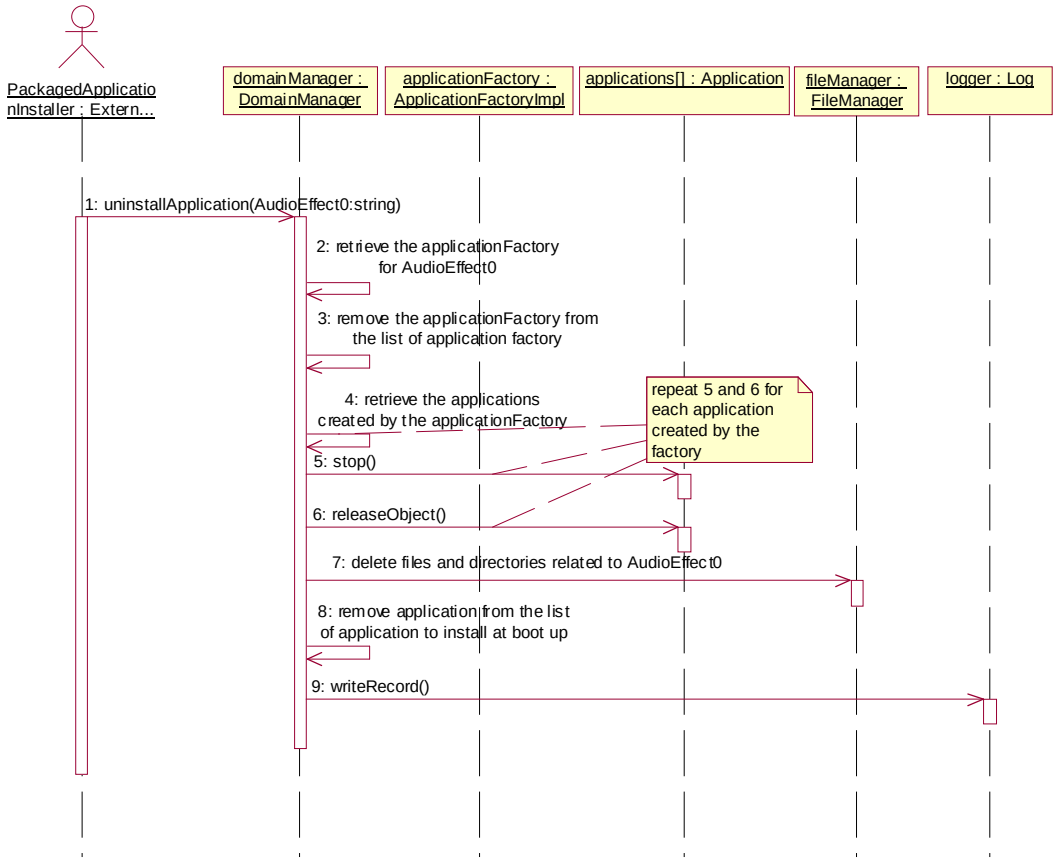

Figure 9 - CF. DomainManagerUninstallApplication Sequence Diagram

1.2 Domain Profile Parsing and Resolving

The following sections contain uml diagrams and class descriptions for all the Core Framework.

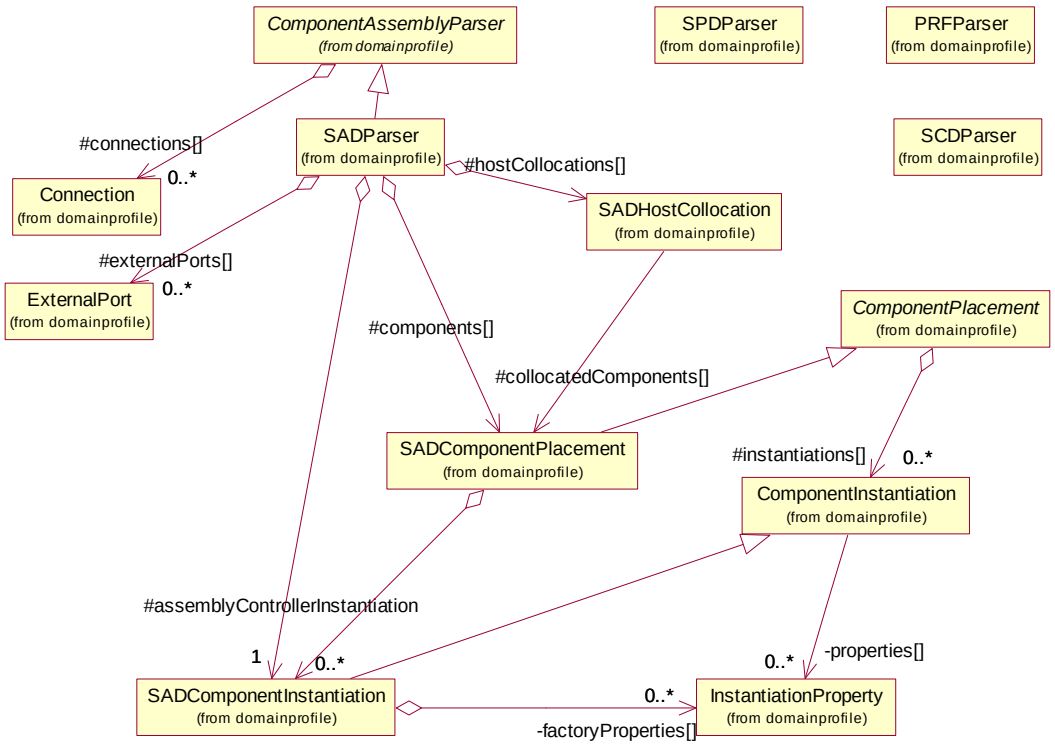

Figure 10 - Class Diagram of XML parsing classes

1.2.1 util.domainprofile.ComponentAssemblyParser

1.2.1.1 Description

This class is used to parse the DCD or SAD XML files.

1.2.1.2 Class Diagram

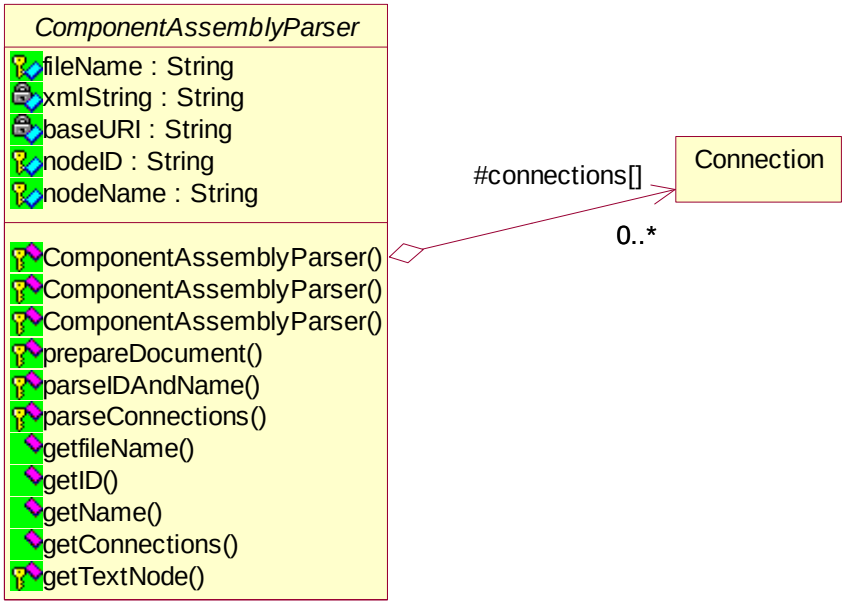

Figure 11 - Class Diagram of ComponentAssemblyParser

1.2.1.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
protected ComponentAssemblyParser()
```

Protected constructor. Required to enable inheritance.

```
protected ComponentAssemblyParser(String fileName)
                                  throws SCA.CF.InvalidProfile
```

Protected complete constructor.
To obtain an instance of this class, use the newInstance() method.
Parameters:
fileName file name of either the DCD or SAD
Throws:
InvalidProfile when the file is malformed

`protected` **`ComponentAssemblyParser`**`(String xmlString, String baseURI)`
`throws SCA.CF.InvalidProfile`

Protected Constructor. To obtain an instance of this class,
use the newInstance() method.
Parameters:
xmlString XML SAD or DCD in a string
baseURI baseURI to help the parser resolve entities

`public Connection[]` **`getConnections`**`()`

Returns: an array of domainprofile.Connection containing
all the parsed connectinterface elements

`public String` **`getfileName`**`()`

Returns: the file name

`public String` **`getID`**`()`

Returns: the component assembly id (either the DCD id or the SAD id)

`public String` **`getName`**`()`

Returns: the component assembly name (either the DCD name or the SAD name)

`protected String` **`getTextNode`**`(Element element)`

Parameters:
element DOM element
Returns: a PCDATA value for an element

`protected void` **`parseConnections`**`(Element rootElement)`
`throws SCA.CF.InvalidProfile`

Parses the connections section which contains all the connectinterface
elements
Parameters:
rootElement the DOM rootElement of the DCD file
Throws:
InvalidProfile when the DCD file is malformed

`protected void` **`parseIDAndName`**`(Element rootElement)`

Parses the deviceconfiguration id and name XML attributs
Parameters:
rootElement the DOM rootElement of the DCD file

```
protected void prepareDocument()
                        throws SCA.CF.InvalidProfile
```

prepareDocument.

This method prepare the XML document to be used by Child class parser

1.2.2 util.domainprofile.Connection

1.2.2.1 Description

This class is used to parse a 'connectinterface' element of a DCD file.

1.2.2.2 Class Diagram

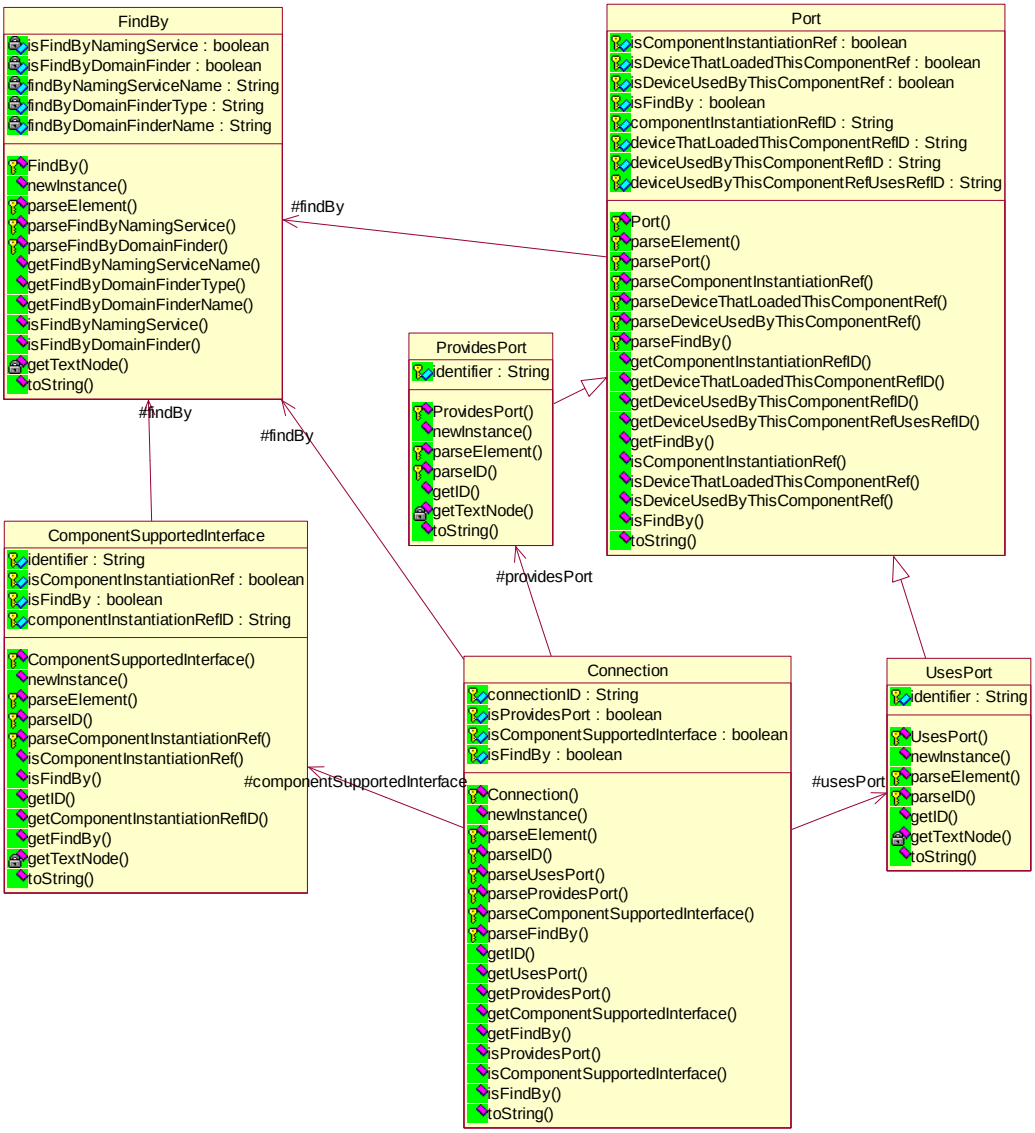

Figure 12 - Class Diagram of Connection

1.2.2.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`protected` **`Connection`**`(Element rootElement)`

Protected Constructor. To obtain an instance of this class, use the newInstance() method.

This class is used to parse a 'connectinterface' element of either the SAD or DCD file of the SCA DomainProfile
Parameters:
rootElement must be a DOM Element representing the 'connectinterface' section

`public ComponentSupportedInterface` **`getComponentSupportedInterface`**`()`

Returns: an object describing the componentsupportedinterface element.
If the componentsupportedinterface element is not used, return null.

`public FindBy` **`getFindBy`**`()`

Returns: an object describing the findby element of the destination end of the connection

`public String` **`getID`**`()`

Returns: the connectinterface id

`public ProvidesPort` **`getProvidesPort`**`()`

Returns: an object describing the providesport element.
If the providesport element is not used, return null.

`public UsesPort` **`getUsesPort`**`()`

Returns: an object describing the usesport element

`public boolean` **`isComponentSupportedInterface`**`()`

Returns: True if the destination end of the connection is specified using a componentsupportedinterface element.
False otherwise.

`public boolean` **`isFindBy`**`()`

Returns: True if the destination end of the connection is specified using a findby element.
False otherwise.

`public boolean` **`isProvidesPort`**`()`

Returns: True if the destination end of the connection is specified using a providesport element.
False otherwise.

```
public static Connection newInstance(Element rootElement)
                                 throws SCA.CF.InvalidProfile
```

This method is used to create a Connection object.
Parameters:
rootElement must be a DOM Element representing the 'connectinterface' section
Throws:
InvalidProfile if the element is not valid

```
protected void parseComponentSupportedInterface(Element rootElement)
```

Parses a componentsupportedinterface element which describes the destination end of the connection.
Parameters:
rootElement must be a DOM Element representing the 'connectinterface' section

```
protected void parseElement()
                      throws SCA.CF.InvalidProfile
```

Parses and extracts its attributes and sub-elements
Throws:
InvalidProfile if the element is not valid

```
protected void parseFindBy(Element rootElement)
                    throws SCA.CF.InvalidProfile
```

Parses a findby element which describes the destination end of the connection.
Parameters:
rootElement must be a DOM Element representing the second child of the 'connectinterface' section
Throws:
InvalidProfile if the element is not valid

`protected void` **`parseID`**`(Element rootElement)`

Parses the id attribute of the connectinterface element which uniquely identifies the connection.

Parameters:

rootElement must be a DOM Element representing the 'connectinterface' section

`protected void` **`parseProvidesPort`**`(Element rootElement)`
`throws SCA.CF.InvalidProfile`

Parses a providesport element which describes the destination end of the connection.

Parameters:

rootElement must be a DOM Element representing the 'connectinterface' section

Throws:

InvalidProfile if the element is not valid

`protected void` **`parseUsesPort`**`(Element rootElement)`
`throws SCA.CF.InvalidProfile`

Parses a providesport element which describes the destination end of the connection.

Parameters:

rootElement must be a DOM Element representing the 'connectinterface' section

Throws:

InvalidProfile if the element is not valid

`public String` **`toString`**`()`

Returns: a string listing every properties

1.2.3 util.domainprofile.ComponentPlacement

1.2.3.1 Description

This class is used to parse a componentplacement element of either a SAD or a DCD file. It is responsible for linking the information between a componentplacement element and the componentfiles element.

1.2.3.2 Class Diagram

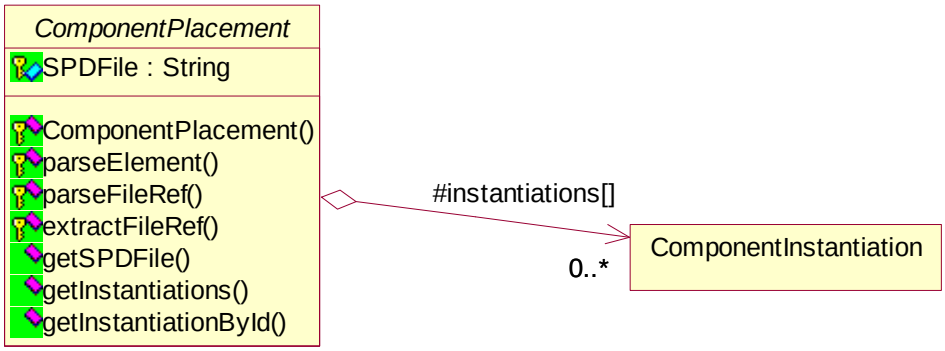

Figure 13 - ComponentPlacement Class Diagram

1.2.3.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
protected ComponentPlacement(Element element, Document document)
                          throws SCA.CF.InvalidProfile
```

Protected Constructor. Use specialized classes to obtain an instantiation of this class.

Parameters:

element must be a DOM Element representing a componentplacement

document must be a DOM Document representing the SAD ot the DCD file

Throws:

InvalidProfile when the SAD or the DCD file is malformed

```
protected abstract void extractFileRef(Element element)
                                  throws SCA.CF.InvalidProfile
```

Parse the componentfileref refid which should be a link to a componentfile declared in the local SAD or DCD.

Parameters:

element must be a DOM Element representing a componentplacement
Throws:
InvalidProfile if the XML file is not a properties file

`public ComponentInstantiation getInstantiationById(String id)`
Parameters:
id UUID of a componentinstantiation element
Returns: the ComponentInstantiation with an id matching the input argument id. Returns Null if not found.

`public ComponentInstantiation[] getInstantiations()`
Returns: an array containing each componentinstantiation

`public String getSPDFile()`
Returns: the componentfile localfile name.

`protected void parseElement()`
`throws SCA.CF.InvalidProfile`
Parses and extracts its attributes and sub-elements
Throws:
InvalidProfile if the element is not valid

`protected void parseFileRef(Element rootElement)`
`throws SCA.CF.InvalidProfile`
Parse the componentfileref refid which should be a link to a componentfile declared in the local SAD or DCD.
Parameters:
rootElement must be a DOM Element representing a componentplacement
Throws:
InvalidProfile if the XML file is not a properties file

1.2.4 util.domainprofile.ComponentInstantiation

1.2.4.1 Description

This class is used to parse the componentinstantiation element of a DCD file.

1.2.4.2 Class Diagram

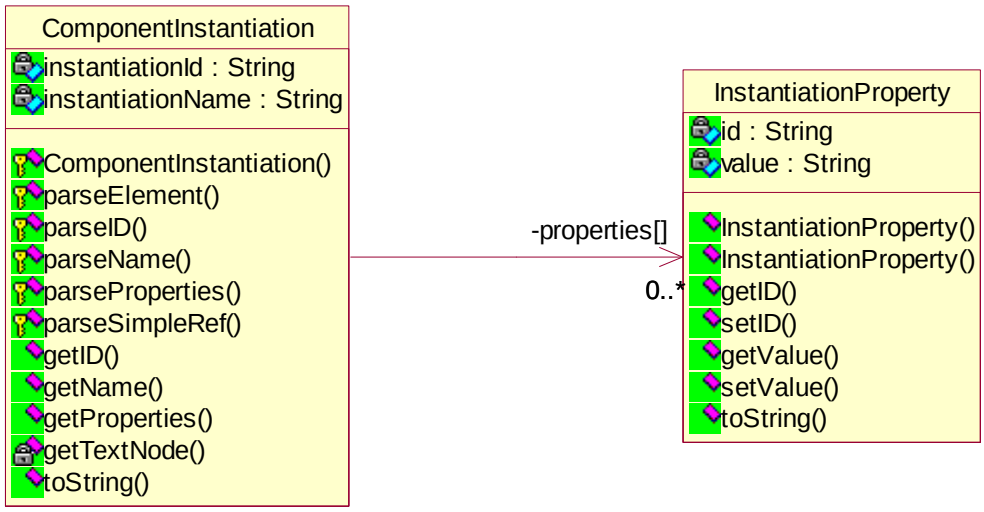

Figure 14 - ComponentInstantiation Class Diagram

1.2.4.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`protected` **`ComponentInstantiation`**`(Element rootElement)`

Protected Constructor. To obtain an instance of this class, use the newInstance() method.

Parameters:

rootElement must be a DOM Element representing a componentinstantiation

`public String` **`getID`**`()`

Returns: componentinstantiation id

`public String` **`getName`**`()`

Returns: the usagename

`public InstantiationProperty[] getProperties()`
Returns: an array containing each componentproperties property.

`protected void parseElement()`
Parses and extracts its attributes and sub-elements
Throws:
InvalidProfile if the element is not valid

`protected void parseID(Element rootElement)`
Parse the componentinstantiation id
Parameters:
rootElement must be a DOM Element representing a componentinstantiation

`protected void parseName(Element rootElement)`
Parse the componentinstantiation name
Parameters:
rootElement must be a DOM Element representing a componentinstantiation

`protected void parseProperties(Element rootElement)`
Parse the componentproperties element
Parameters:
rootElement must be a DOM Element representing a componentinstantiation

`protected InstantiationProperty parseSimpleRef(Element element)`
Parse a componentproperties simpleref element.
Parameters:
element must be a DOM Element representing a simpleref
Returns: an object containing a pair of String for the propertyId and the property value

`public String toString()`
Returns: a string listing every properties

1.2.5 util.domainprofile.InstantiationProperty

1.2.5.1 Description

This class is used as a placeholder for the id/value pair of a simpleref component property.

1.2.5.2 Class Diagram

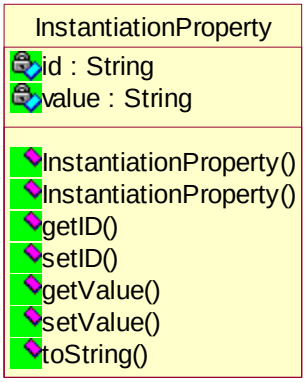

Figure 15 - InstantiationProperty Class Diagram

1.2.5.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`public` **`InstantiationProperty`**`()`

Default constructor.
This CTOR initializes all attributes to null.

`public` **`InstantiationProperty`**`(String id, String value)`

Complete constructor.
Parameters:
id the id of a simpleref property
value the value of a simpleref property

`public String` **`getID`**`()`

Returns: the id of the simpleref property represented by this class

`public String` **`getValue`**`()`

Returns: the value of the simpleref property represented by this class

`public void` **`setID`**`(String id)`

Set the id of the simpleref property represented by this class.

Parameters:

id the id of a simpleref property

`public void` **`setValue`**`(String value)`

Set the id of the simpleref property represented by this class.

Parameters:

value the value of a simpleref property

`public String` **`toString`**`()`

Returns: a string listing every properties

1.2.6 util.domainprofile.SADParser

1.2.6.1 Description

This class is used to parse the SAD XML file, which identify which components need to be launched when an application is loaded.

1.2.6.2 Class Diagram

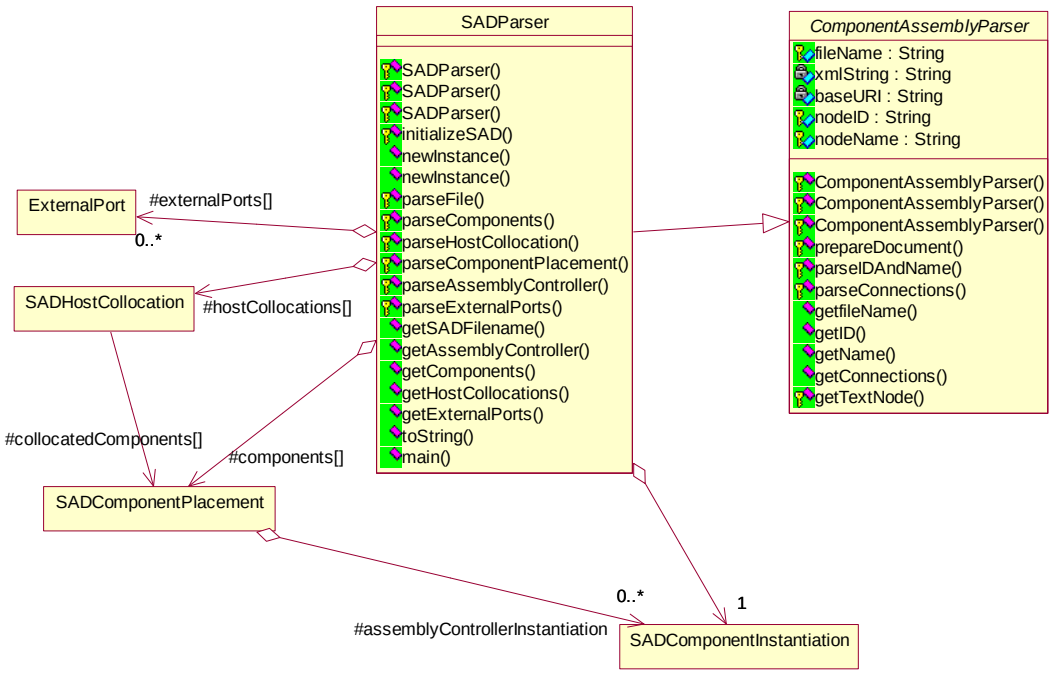

Figure 16 - SADParser Class Diagram

1.2.6.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`protected` **`SADParser`**`()`

Protected constructor. Required to enable inheritance.

`protected` **`SADParser`**`(String SADFile)`
`throws` `SCA.CF.InvalidProfile`

Protected complete constructor.

To obtain an instance of this class, use the newInstance() method.

Parameters:
SADFile SAD file name
Throws:
InvalidProfile when the SAD file is malformed

```
protected SADParser(String sadString, String baseURI)
                throws SCA.CF.InvalidProfile
```

Protected complete constructor.
To obtain an instance of this class, use the newInstance() method.
Parameters:
sadString XML String containing SAD
baseURI base URI used to resolve the relative file references
Throws:
InvalidProfile when the SAD file is malformed

```
public SADComponentInstantiation getAssemblyController()
```

Returns: a SADComponentInstantiation representing the assembly controller

```
public SADComponentPlacement[] getComponents()
```

Returns: an array of component placement containing
all the components that do not need to be collocated

```
public ExternalPort[] getExternalPorts()
```

Returns: an array of external ports

```
public SADHostCollocation[] getHostCollocations()
```

Returns: an array of host collocation containing
all the components that do need to be collocated

```
public String getSADFilename()
```

Returns: the SAD file name

```
protected void initializeSAD()
                       throws SCA.CF.InvalidProfile
```

Used by the CTORs to initialize the attributes to null and get
the DOM root element of the SAD.
Parameters:
SADFile SAD file name
Throws:
InvalidProfile when the SAD file is malformed

`public static void main(String[] args)`

A simple main() used to test this class.

Parameters:

args

`public static SADParser newInstance(String SADFile) throws SCA.CF.InvalidProfile`

This method is used to create an SADParser onto which the parseFile() method has already been invoked.

Parameters:

SADFile SAD file name

Throws:

InvalidProfile when the SAD file is malformed

`public static SADParser newInstance(String sadString, String baseURI) throws SCA.CF.InvalidProfile`

This method is used to create an SADParser onto which the parseFile() method has already been invoked.

Parameters:

sadString XML String containing SAD

baseURI base URI used to resolve the relative file references

Throws:

InvalidProfile when the file is malformed

`public String toString()`

Returns: a string listing every properties

1.2.7 util.domainprofile.SADComponentPlacement

1.2.7.1 Description

This class is used to parse a componentplacement element of a SAD file. It is responsible for linking the information between a componentplacement element the componentfiles element.

1.2.7.2 Class Diagram

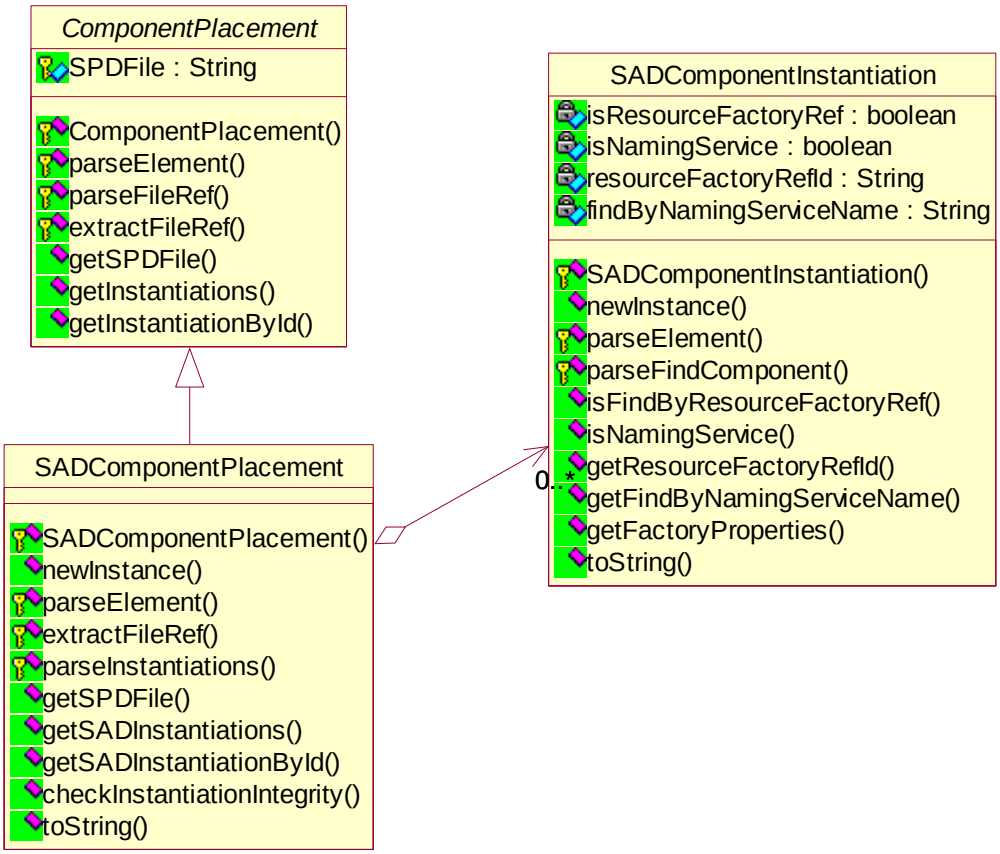

Figure 17 - SADComponentPlacement Class Diagram

1.2.7.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`protected` **`SADComponentPlacement`**`(Element element, Document document)`
`throws SCA.CF.InvalidProfile`

Protected Constructor. To obtain an instance of this class, use the newInstance() method.

Parameters:
element must be a DOM Element representing a componentplacement
document must be a DOM Document representing the SAD file

Throws:
InvalidProfile when the SAD file is malformed

`public void` **`checkInstantiationIntegrity`**`()`
`throws SCA.CF.InvalidProfile`

This methods verified that the instance are all deployed or are all being created by the same ResourceFactory.

The XML does not prevent developper to create a component placement where the instance could be deployed and/or create by different ResourceFactory.

Throws:
InvalidProfile if the instance are not using the same deployment mecanism

`protected void` **`extractFileRef`**`(Element element)`

Parse the componentfileref refid which should be a link to a componentfile declared in the local SAD or DCD.

Parameters:
element must be a DOM Element representing a componentplacement

Throws:
InvalidProfile if the XML file is not a properties file

`public SADComponentInstantiation` **`getSADInstantiationById`**`(String id)`

Parameters:
id UUID of a SAD componentinstantiation element.

Returns: the SADComponentInstantiation with an id matching the input argument id. Returns Null if not found.

`public SADComponentInstantiation[] getSADInstantiations()`

Returns: an array containing each componentinstantiation

`public String getSPDFile()`

Returns: returns the componentfile localfile name. Otherwise, null is returned.

`public static SADComponentPlacement newInstance(Element element, Document document)`
`throws SCA.CF.InvalidProfile`

This method is used to create an SADComponentPlacement onto which the parseElement() method has already been invoked.

Parameters:
element must be a DOM Element representing a componentplacement
document must be a DOM Document representing the SAD file

Throws:
InvalidProfile when the SAD file is malformed

`public String toString()`

Returns: a string listing every properties

1.2.8 util.domainprofile.SADComponentInstantiation

1.2.8.1 Description

This class is used to parse the componentinstantiation element of a SAD file.

1.2.8.2 Class Diagram

Figure 18 - SADComponentInstantiation Class Diagram

1.2.8.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`protected` **`SADComponentInstantiation`**`(Element rootElement)`

Protected Constructor. To obtain an instance of this class, use the newInstance() method.

Parameters:

rootElement must be a DOM Element representing a componentinstantiation

`public` `InstantiationProperty[]` **`getFactoryProperties`**`()`

Returns: an array containing each resourcefactoryproperties of a componentresourcefactoryref.

`public String` **`getFindByNamingServiceName`**`()`

Returns: a string representing the namingservice name attribute.

`public String` **`getResourceFactoryRefId`**`()`

Returns: a string representing the componentresourcefactoryref refid attribute.

`public boolean` **`isFindByResourceFactoryRef`**`()`

Returns: true if the componentinstantiation uses the componentresourcefactoryref element. False otherwise.

`public boolean` **`isNamingService`**`()`

Returns: true if the componentinstantiation uses the namingservice element. False otherwise.

`public static` `SADComponentInstantiation` **`newInstance`**`(Element rootElement)`

This method is used to create an SADComponentInstantiation onto which the parseElement() method has already been invoked.

Parameters:

rootElement must be a DOM Element representing a componentinstantiation

`public String` **`toString`**`()`

Returns: a string listing every properties

1.2.9 util.domainprofile.SADHostCollocation

1.2.9.1 Description

This class is used to parse the hostcollocation element of a SAD XML file.

1.2.9.2 Class Diagram

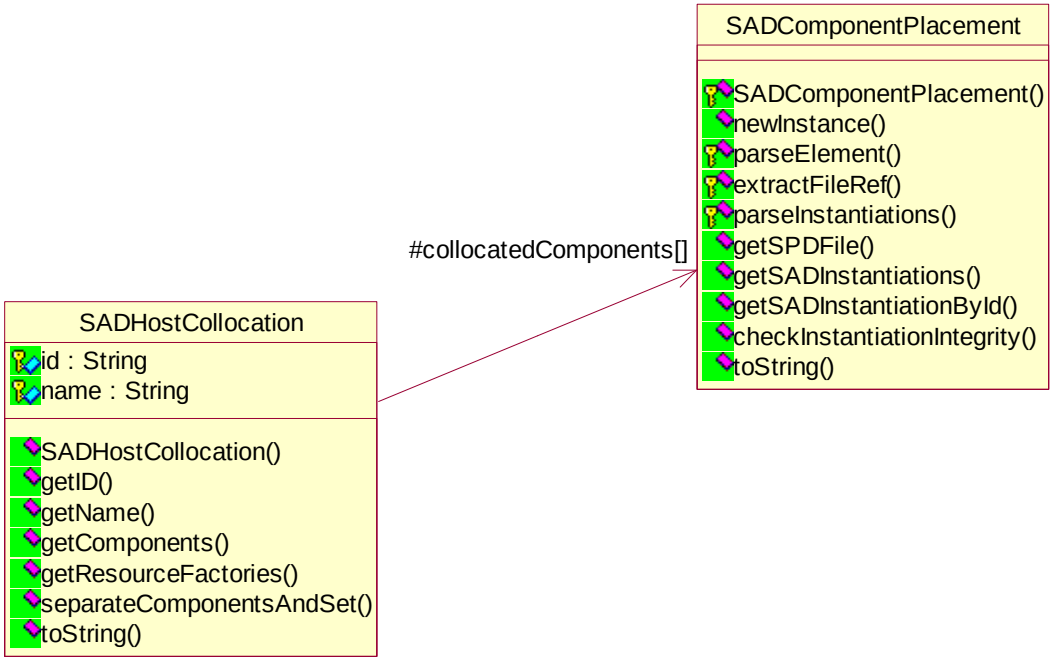

Figure 19 - SADHostCollocation Class Diagram

1.2.9.3 Implementation Specific Service

This section contains a description of all the methods that are implementation specific.

```
public SADHostCollocation(String id, String name, ArrayList
allCollocatedComponents)
                    throws SCA.CF.InvalidProfile
```

Complete constructor.

Parameters:

id is an id that may be used to uniquely identify a set of collocated components within a SAD file.

name is a name that may be used to uniquely identify a set of collocated components within a SAD file.

colocatedComponents is the list of component placements that will be collocated on the same host platform.

`public SADComponentPlacement[] getComponents()`

Returns: an array of components that must be colocated

`public String getID()`

Returns: the id attribute

`public String getName()`

Returns: the name attribute

`public ResourceFactoryPlacement[] getResourceFactories()`

Returns: an array of resourceFactory Placement that holds ResourceFactory componentplacement as well as componentplacement for resources that needs to be created by the resourceFactory

```
public void separateComponentsAndSet(ArrayList allCollocatedComponents)
                              throws SCA.CF.InvalidProfile
```

Utility method to separate the resource created by resourcefactory with a normal deployed collocated component. Once separated class attribut collocatedComponents and resourceFactoryPlacement will be set accordingly.

Throws:

InvalidProfile if a resource use an id for resource factory that does not exists

`public String toString()`

Returns: a string listing every properties

1.2.10 util.domainprofile.ExternalPort

1.2.10.1 Description

This class is used to hold the information about an external port located in a SAD file.

1.2.10.2 Class Diagram

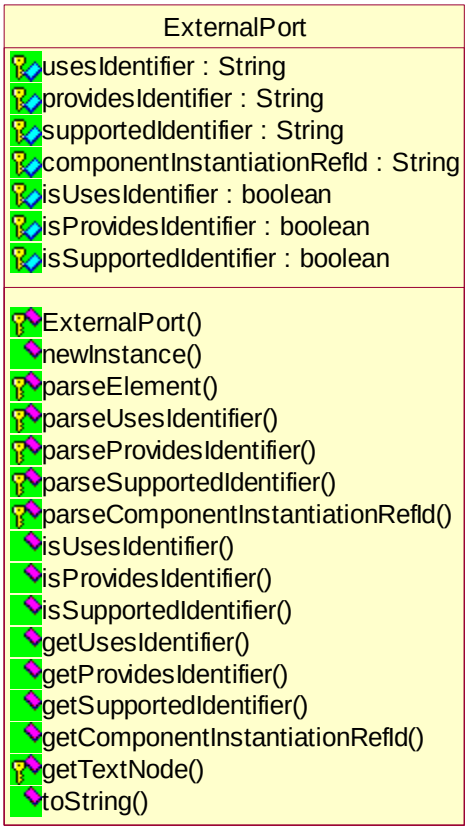

Figure 20 - ExternalPort Class Diagram

1.2.10.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`protected` **`ExternalPort`**`(Element rootElement)`

Complete constructor.

Parameters:

rootElement is a port element describing an external port

`public static` `ExternalPort` **`newInstance`**`(Element rootElement)`
`throws` `SCA.CF.InvalidProfile`

This method is used to create an ExternalPort onto which the parseFile() method has already been invoked.

Parameters:

rootElement must be a DOM Element representing the 'port' section

Throws:

InvalidProfile if the element is not valid

`public String` **`toString`**`()`

Returns: a string listing every properties

1.2.11 util.domainprofile.SCDParser

1.2.11.1 Description

The SCDParser class is used to parse the XML Software Component Descriptor file describing a component, its ports and its interfaces.

1.2.11.2 Class Diagram

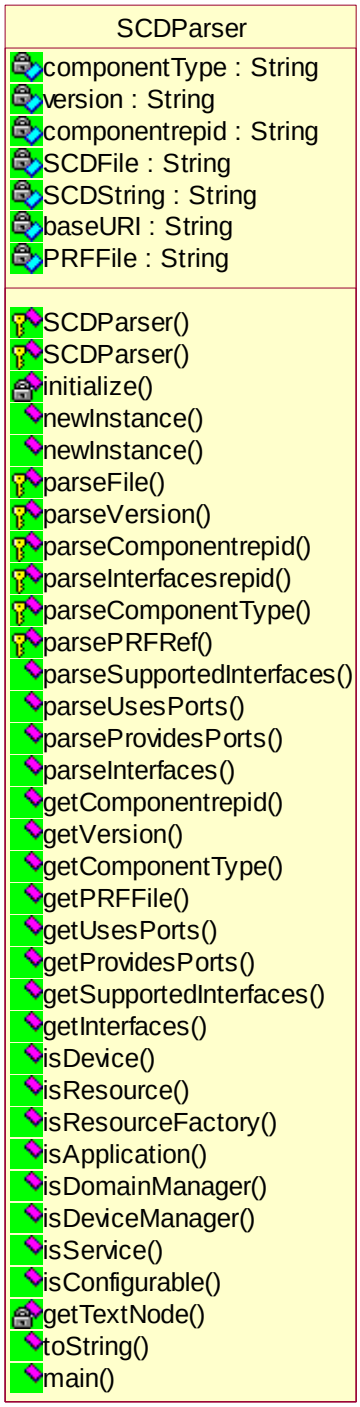

Figure 21 - SCDParser Class Diagram

1.2.11.3 Implementation Specific Services

`protected` **`SCDParser`**`(String SCDFile)`

Protected Constructor. To obtain an instance of this class, use the newInstance() method.

Parameters:

SCDFile XML Software Component Descriptor file name

`protected` **`SCDParser`**`(String SCDString, String baseURI)`

Protected Constructor. To obtain an instance of this class, use the newInstance() method.

Parameters:

SCDFile XML Software Component Descriptor file name

baseURI baseURI to help the parser resolve entities

`public static SCDParser` **`newInstance`**`(String SCDFile)`
`throws SCA.CF.InvalidProfile`

This method is used to create an SCDParser onto which the parseFile() method has already been invoked.

Parameters:

SCDFile XML Software Component Descriptor file name

Throws:

InvalidProfile when the file is malformed

`public static SCDParser` **`newInstance`**`(String SCDString, String baseURI)`
`throws SCA.CF.InvalidProfile`

This method is used to create an SCDParser onto which the parseFile() method has already been invoked.

Parameters:

inputStream inputStream to an XML Software Component Descriptor

baseURI base URI used to resolve the relative file references

Throws:

InvalidProfile when the file is malformed

`public void` **`parseProvidesPorts`**`(Element rootElement)`
`throws SCA.CF.InvalidProfile`

Retrieves the uses port interfaces

`public void` **`parseSupportedInterfaces`**`(Element rootElement)`
`throws SCA.CF.InvalidProfile`

Retrieves the interfaces supported by the component

```
public void parseUsesPorts(Element rootElement)
                    throws SCA.CF.InvalidProfile
```

Retrieves the uses port interfaces

```
public String toString()
```

Returns: String a string giving the type of the component and its PRF file name.

1.2.12 util.domainprofile.SPDParser

1.2.12.1 Description

This class is used to parse the XML Software Package Descriptor (SPD) file describing the various implementations of a component.

1.2.12.2 Class Diagram

Figure 22 - SPDParser Class Diagram

1.2.12.3 Implementation Specific Services

```
protected SPDParser(String SPDFile)
```

Protected Constructor. To obtain an instance of this class, use the newInstance() method.
Parameters:
SPDFile XML Software Package Descriptor file name

`protected SPDParser(String SPDString, String baseURI)`
Protected Constructor. To obtain an instance of this class, use the newInstance() method.
Parameters:
SPDString XML Software Package Descriptor in a string
baseURI baseURI to help the parser resolve entities

`public boolean isScaCompliant()`
Returns: True is the spd type is SCA_COMPLIANT. False otherwise.

`public boolean isScaNonCompliant()`
Returns: True is the spd type is SCA_NON_COMPLIANT. False otherwise.

`public static void main(String[] args)`
A simple main() used to test this class.
Parameters:
args param 0 must be the name of the SPD file

`public static SPDParser newInstance(String SPDFile) throws SCA.CF.InvalidProfile`
This method is used to create an SPDParser onto which the parseFile() method has already been invoked.
Parameters:
SPDFile XML Software Profile Descriptor file name
Throws:
InvalidProfile when the file is malformed

`public static SPDParser newInstance(String SPDString, String baseURI) throws SCA.CF.InvalidProfile`
This method is used to create an SPDParser onto which the parseFile() method has already been invoked.
Parameters:
inputStream inputStream to an XML Software Profile Descriptor
baseURI base URI used to resolve the relative file references
Throws:
InvalidProfile when the file is malformed

`public String` **`toString`**`()`

Returns: a string providing the name of the PRF file and SCD file of the component and the list of every implementations.

1.2.13 util.domainprofile.PRFParser

1.2.13.1 Description

The PRFParser class is used to parse the XML properties file of a component and to get its various properties.

1.2.13.2 Class Diagram

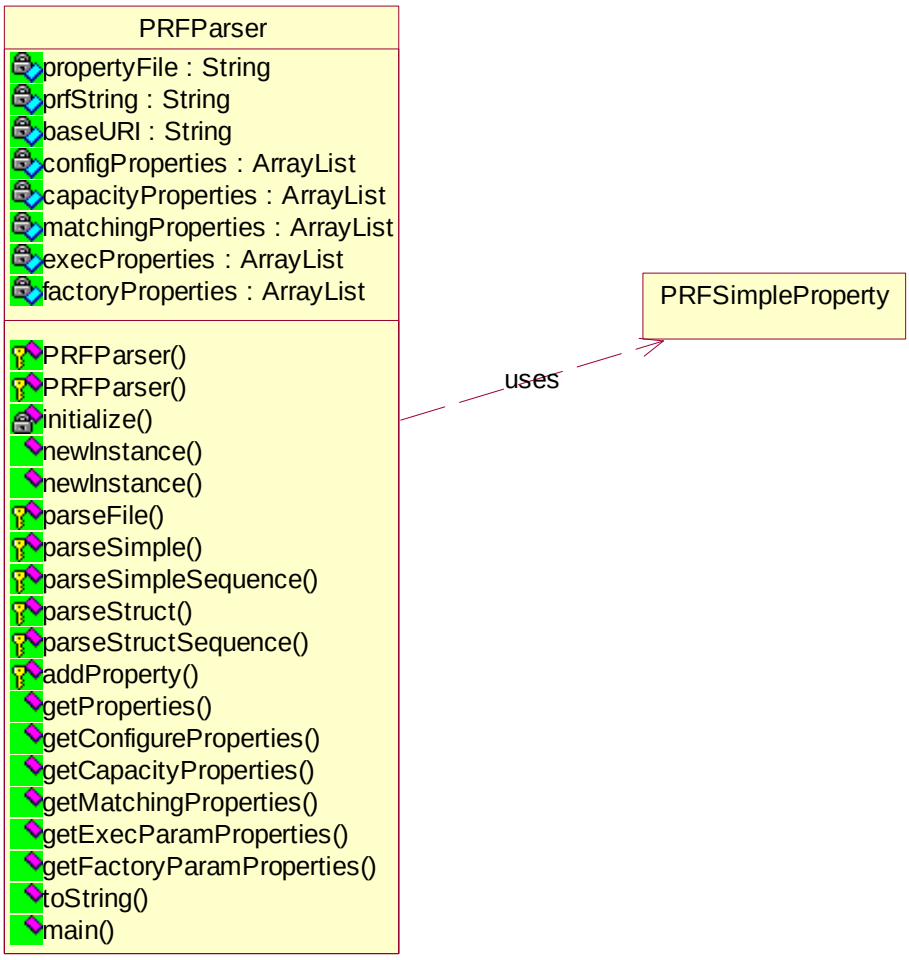

Figure 23 - PRFParser Class Diagram

1.2.13.3 Implementation Specific Services

`protected` **`PRFParser`**`(String propertyFile)`

Constructor.

Parameters:

propertyFile XML Properties file name

`protected` **`PRFParser`**`(String prfString, String baseURI)`

Protected Constructor. To obtain an instance of this class, use the newInstance() method.

Parameters:

prfString XML Property Descriptor in a string

baseURI baseURI to help the parser resolve entities

`public static PRFParser` **`newInstance`**`(String PRFFile)`
`throws SCA.CF.InvalidProfile`

This method is used to create a PRFParser onto which the ParseFile() method has already been invoked.

Parameters:

propertyFile XML Properties file name

Throws:

InvalidProfile when the file is malformed

`public static PRFParser` **`newInstance`**`(String prfString, String baseURI)`
`throws SCA.CF.InvalidProfile`

This method is used to create an PRFParser onto which the parseFile() method has already been invoked.

Parameters:

prfString XML Property Descriptor in a string

baseURI baseURI to help the parser resolve entities

Throws:

InvalidProfile when the file is malformed

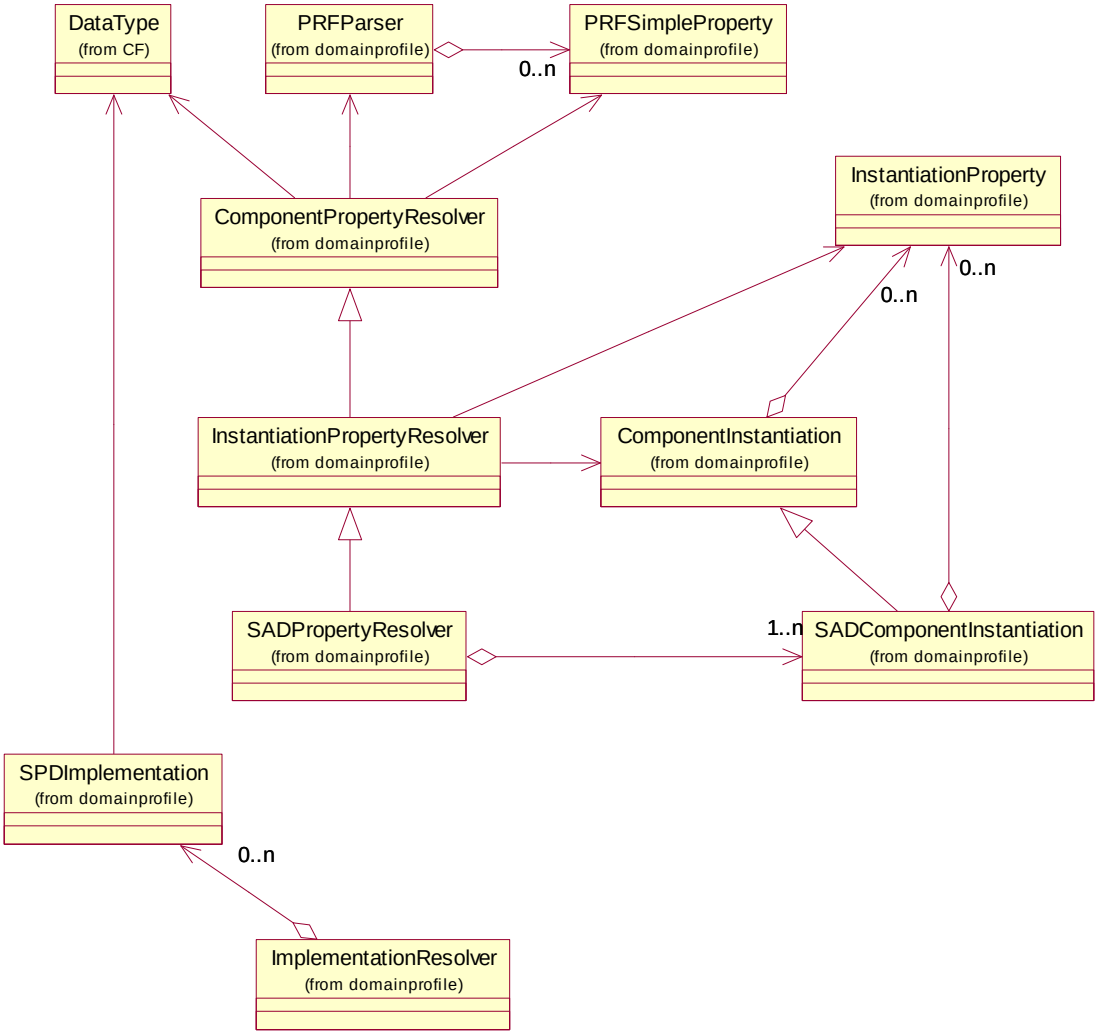

Figure 24 - Class Diagram of property resolving classes

1.2.14 util.domainprofile.ComponentPropertyResolver

1.2.14.1 Description

Given the proper property files, this class will determine which value should be used for a property based on the precedence of value described in sections 2.1 of the appendix D.

1.2.14.2 Class Diagram

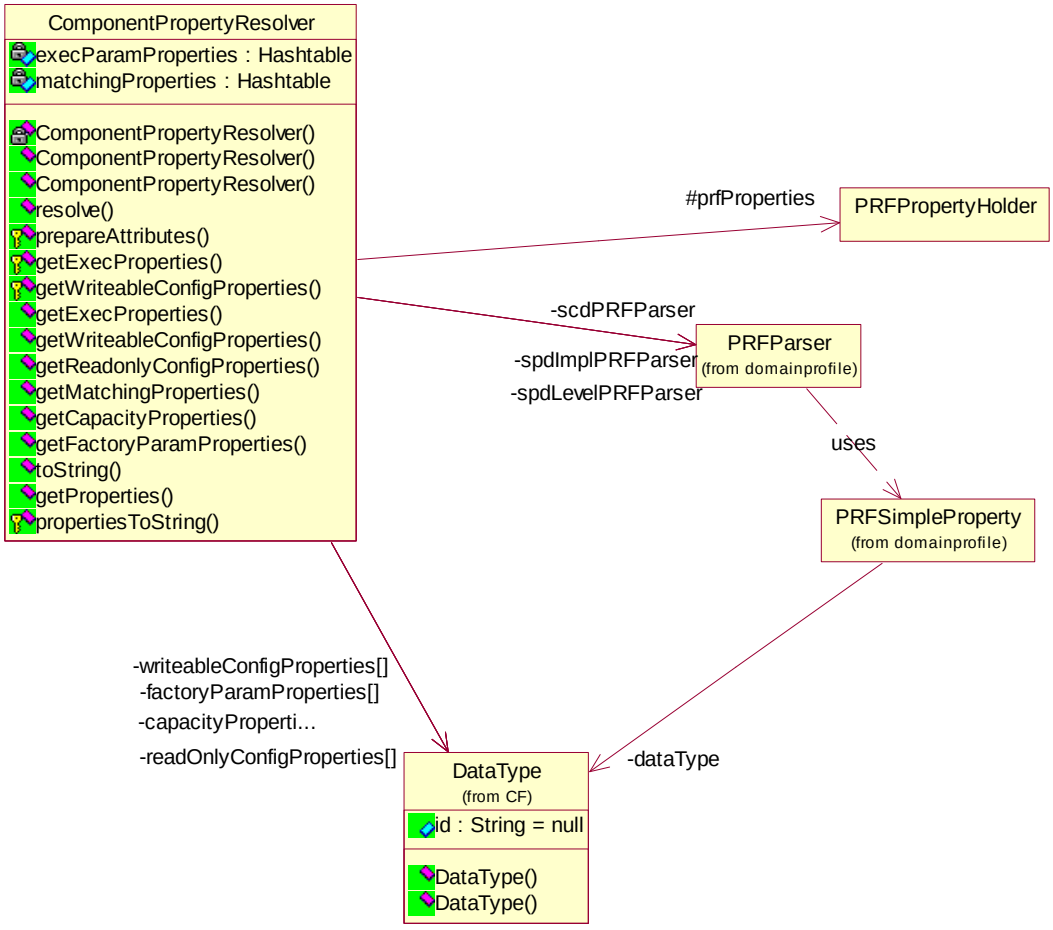

Figure 25 - ComponentPropertyResolver Class Diagram

1.2.14.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public ComponentPropertyResolver(PRFParser scdPRFParser, PRFParser
spdLevelPRFParser, PRFParser spdImplPRFParser)
```

Provided the 3 PRF parser specified as input arguments belong to the same component, the resolving will yield a combined set of properties.

Parameters:

scdPRFParser PRFParser that contains the file property referenced in the component's SCD file

spdLevelPRFParser PRFParser that contains the file property referenced in the component's SPD file at the SPD level

SPDImplPropFile PRFParser that contains the file property referenced in the component's SPD file at the SPD impl

```
public ComponentPropertyResolver(String SCDPropFile, String
SPDLevelPropFile, String SPDImplPropFile)
                              throws SCA.CF.InvalidProfile
```

Provided the 3 PRF file names specified as input arguments belong to the same component, the resolving will yield a combined set of properties.

Parameters:

SCDPropFile file name for the component's PRF file referenced in the component's SCD file

SPDLevelPropFile file name for the component's PRF file referenced in the component's SPD file at the SPD level

SPDImplPropFile file name for the component's PRF file referenced in the component's SPD file at the SPD impl

```
public SCA.CF.DataType[] getCapacityProperties()
```

Returns: the capacity properties for a component

```
protected Hashtable getExecProperties()
```

Returns: the execution parameters. The value for these properties may have been overloaded in which case they are the result of a resolving procedure described in D.2.1

```
public Hashtable getExecProperties(String UUID)
```

Parameters:

UUID UUID of an spd implementation

```
public SCA.CF.DataType[] getFactoryParamProperties()
```

Returns: the factory param properties for a component

```
public Hashtable getMatchingProperties()
```

Returns: the matching properties for a component

`public PRFSimpleProperty[] getProperties()`

`public SCA.CF.DataType[] getReadonlyConfigProperties()`
Returns: the configuration parameters defined as readonly.

`protected SCA.CF.DataType[] getWriteableConfigProperties()`
Returns: the configuration parameters defined as writable (i.e. read/write or writeonly). The value for these properties may have been overloaded in which case they are the result of a resolving procedure described in D.2.1 .

`public SCA.CF.DataType[] getWriteableConfigProperties(String UUID)`
Parameters:
UUID UUID of an spd implementation

`protected void prepareAttributes()`
This method creates all the property holders ready to be used by the get methods. It should only be invoked after the resolving is done.

`protected String propertiesToString(java.util.Enumeration enumeration)`
Utility method to transform an Enumeration of PRFSimpleProperty into a string
Parameters:
enumeration an Enumeration of SimplePropertyHolder or String

`public void resolve()`
`throws SCA.CF.InvalidProfile`
This method implements the resolving of properties as described in sections 2.1 of the appendix D.

As specified, the properties for one component are resolved as the union of all properties in the following precedence order (from most important to least important):

1. SPD Implementation
2. SPD Level
3. SCD

Note that the the SCARI project segragates the allocation properties into 2 types of properties: capacity and matching properties.

1. A capacity property is an allocation property used to describe the requirements in capacity for an implementation. For example, an implementation can require 10 megs of ram and X MIPS from a Device before being deployed.

2. A matching property is an allocation property used to describe the requirements in terms of execution environment for an implementation. For example, an implementation can be compiled for a specific processor or use a particular Runtime version. Therefore, the matching properties have an impact on which Device a implementation will be loaded.

As a consequence, this resolver supports the following types of properties:

1. Matching Properties
2. Capacity Properties
3. Execution Properties
4. Writable Configuration Properties
5. Factory Param Properties

Throws:
InvalidProfile if any of the PRF files is malformed

`public String` **`toString`**`()`

Returns: a string listing every properties

1.2.15 util.domainprofile.InstantiationPropertyResolver

1.2.15.1 Description

This class takes care of resolving the value of properties for each instantiation of a component. Given the proper property files, this class will determine which value should be used for a property based on the precedence of value described in sections 2.1 and 6.3.1.2 of the appendix D.

1.2.15.2 Class Diagram

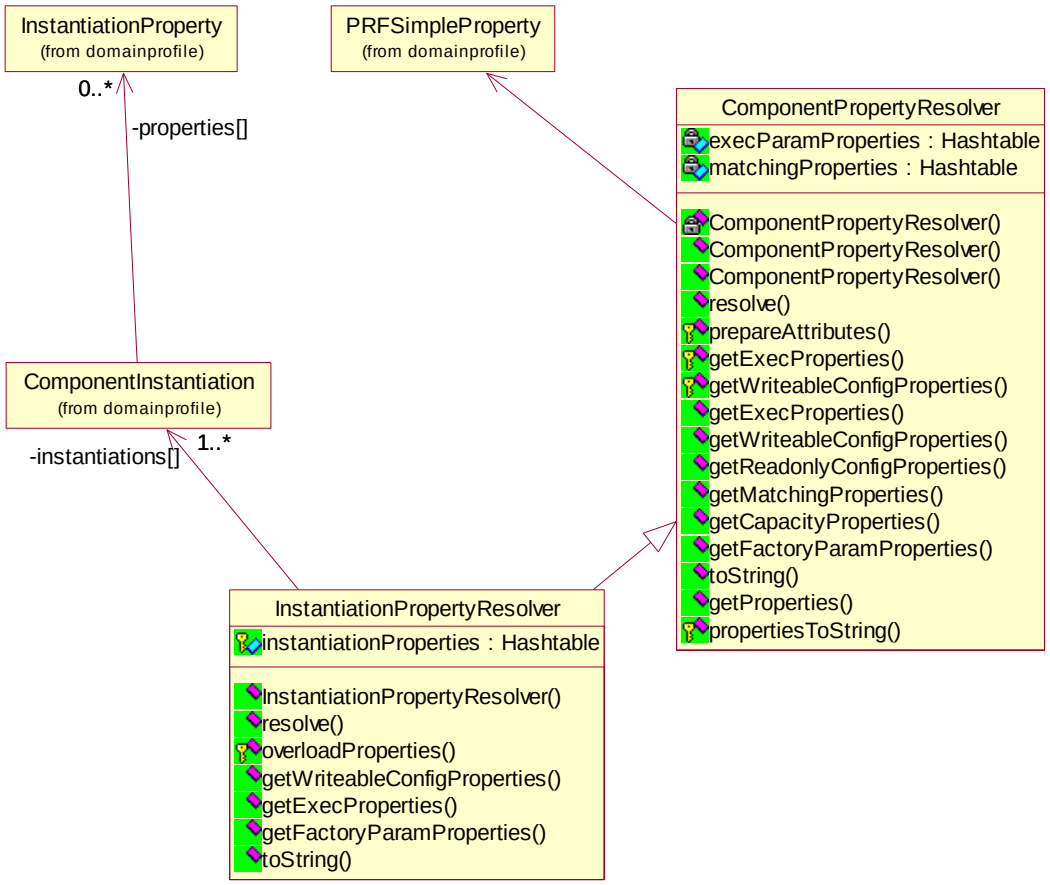

Figure 26 - InstantiationPropertyResolver Class Diagram

1.2.15.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`public` **`InstantiationPropertyResolver`**`(ComponentInstantiation[] instantiations, String SCDPropFile, String SPDLevelPropFile, String SPDImplPropFile)`
`throws SCA.CF.InvalidProfile`

Provided the 3 PRF files specified as input arguments belong to the same component, the resolving will yield one set of properties. These properties will be associated with the uniqueIdentifier of each instantiation of the input argument for future retrieval. This CTOR is typically used when properties need to be resolved for a number of instantiations of a same component.

Parameters:
instantiations an array of Component instantiations for a same component
SCDPropFile name of the property file (PRF) referenced in the component's SCD file
SPDLevelPropFile name of the property file (PRF) referenced in the component's SPD file at the SPD level
SPDImplPropFile name of the property file (PRF) referenced in the component's SPD file at the SPD impl

Throws:
InvalidProfile if the instantiations array is null or empty

`public Hashtable` **`getExecProperties`**`(String inst_UUID)`

Get the execution parameters for a particular instantiation

Parameters:
inst_UUID unique identifier of an instantiation

Returns: the execution parameters. The value for these properties may have been overloaded in which case they are the result of a resolving procedure described in D.6.3.1.2 .

`public SCA.CF.DataType[]` **`getFactoryParamProperties`**`(String inst_UUID)`

Parameters:
inst_UUID unique identifier of a component instantiation

Returns: an array of DataType

`public SCA.CF.DataType[]` **`getWriteableConfigProperties`**`(String inst_UUID)`

Get the execution parameters for a particular instantiation

Parameters:
inst_UUID unique identifier of an instantiation
Returns: the configuration parameters defined as writable (i.e. read/write or writeonly). The value for these properties may have been overloaded in which case they are the result of a resolving procedure described in D.6.3.1.2 .

```
protected ResolvedProperties overloadProperties(InstantiationProperty[]
overloadingProps)
                                                    throws SCA.CF.InvalidProfile
```

Overload the current set of properties with the values specified in the DCD is or SAD for one single instance.
Parameters:
overloadingProps an array of InstantiationProperty containing the properties listed in the DCD or the SAD componentinstantiation.componentproperties Element.
Throws:
InvalidProfile if an invalid overload is performed

```
public void resolve()
            throws SCA.CF.InvalidProfile
```

This method implements the resolving of properties as described in section 6.3.1.2 of the appendix D.

As specified, the execution properties and the writable configuration properties for one instantiation of a component are resolved as the union of all properties in the following precedence order (from most important to least important):

1. DCD component instantiations
2. SPD Implementation
3. SPD Level
4. SCD

This method resolves every type of properties for all specified instantiations such that when it is done, all the properties can be obtained for each instantiations.

Note that the the SCARI project further segragates the allocation properties into 2 types of properties: capacity and matching properties. For more info see the ComponentPropertyResolver.

NB as specified in D.6.3.1.2, this resolver only allows the overloading of the following type of properties:

1. Execution Properties
2. Writable Configuration Properties

Throws:
InvalidProfile if any of the XML file is malformed.

`public String toString()`

Returns: a string listing every properties

1.2.16 util.domainprofile.SADPropertyResolver

1.2.16.1 Description

This class takes care of resolving the value of properties for each instantiation of a component of an application. The component properties described in the SAD and the component's resource factory properties are also used in the resolving. Given the proper property files, this class will determine which value should be used for a property based on the precedence of value described in sections 2.1 and 6.3.1.2 of the appendix D.

1.2.16.2 Class Diagram

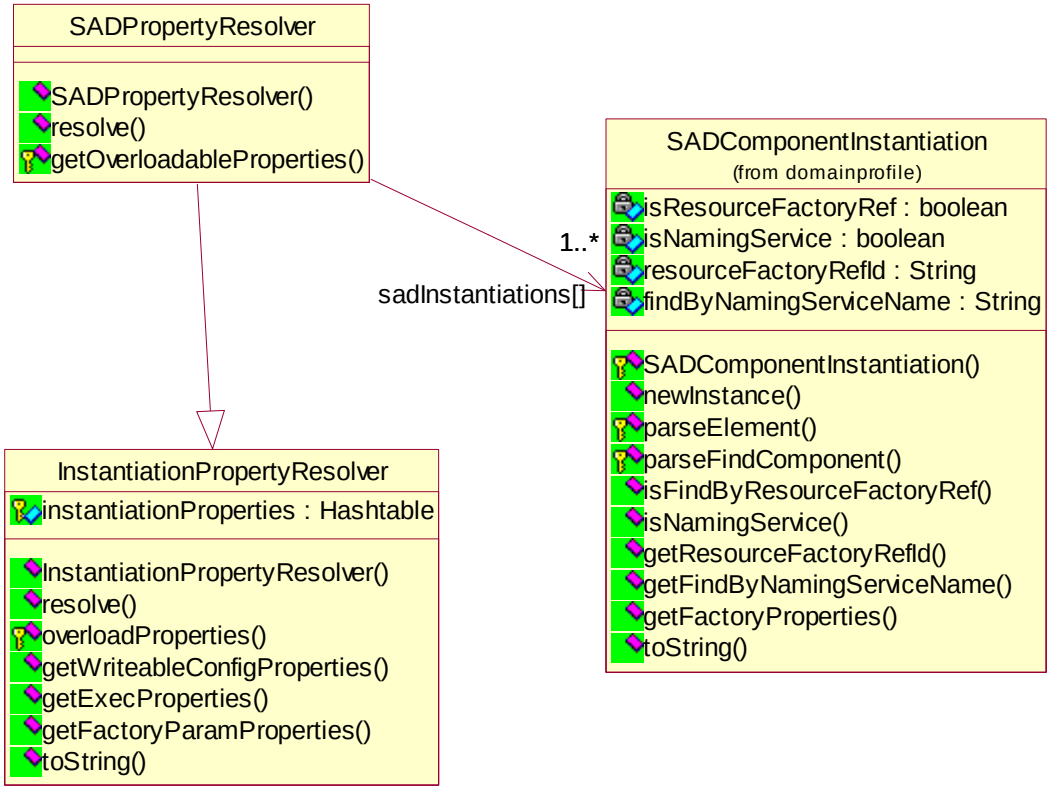

Figure 27 - SADPropertyResolver Class Diagram

1.2.16.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public SADPropertyResolver(SADComponentInstantiation[]
sadInstantiations, String SCDPropFile, String SPDLevelPropFile, String
```

SPDImplPropFile)
throws SCA.CF.InvalidProfile

Provided the 3 PRF files specified as input arguments belong to the same component, the resolving will yield one set of properties. These properties will be associated with the uniqueIdentifier of each instantiation of the input argument for future retrieval. This CTOR is typically used when properties need to be resolved for a number of SAD instantiations of a same component.

Parameters:

sadInstantiations an array of SAD Component instantiations for a same component

SCDPropFile name of the property file (PRF) referenced in the component's SCD file

SPDLevelPropFile name of the property file (PRF) referenced in the component's SPD file at the SPD level

SPDImplPropFile name of the property file (PRF) referenced in the component's SPD file at the SPD impl

Throws:

InvalidProfile if the instantiations array is null or empty

protected ResolvedProperties **getOverloadableProperties**()

get a clone copy of the base properties that can be used for factoryparam overloading.

Returns: ResolvedProperties that contains a clone of the base properties which can be overloaded

public void **resolve**()
throws SCA.CF.InvalidProfile

This method implements the resolving of properties as described in section 6.3.1.2 of the appendix D.

As specified, the execution properties and the writable configuration properties for one instantiation of a component are resolved as the union of all properties in the following precedence order (from most important to least important):

1. SAD component resource factory properties

2. SAD component properties

3. SPD Implementation

4. SPD Level

5. SCD

This method resolves every type of properties for all specified instantiations such that when it is done, all the properties can be obtained for each instantiations.

Note that the the SCARI project further segragates the allocation properties into 2 types of properties: capacity and matching properties. For more info see the ComponentPropertyResolver.

NB as specified in D.6.3.1.2, this resolver only allows the overloading of the following type of properties:

1. Execution Properties

2. Writable Configuration Properties

3. Factory param Properties

Throws:
InvalidProfile if the the properties to overload are not of type factoryparam.

1.3 Core Framework Design

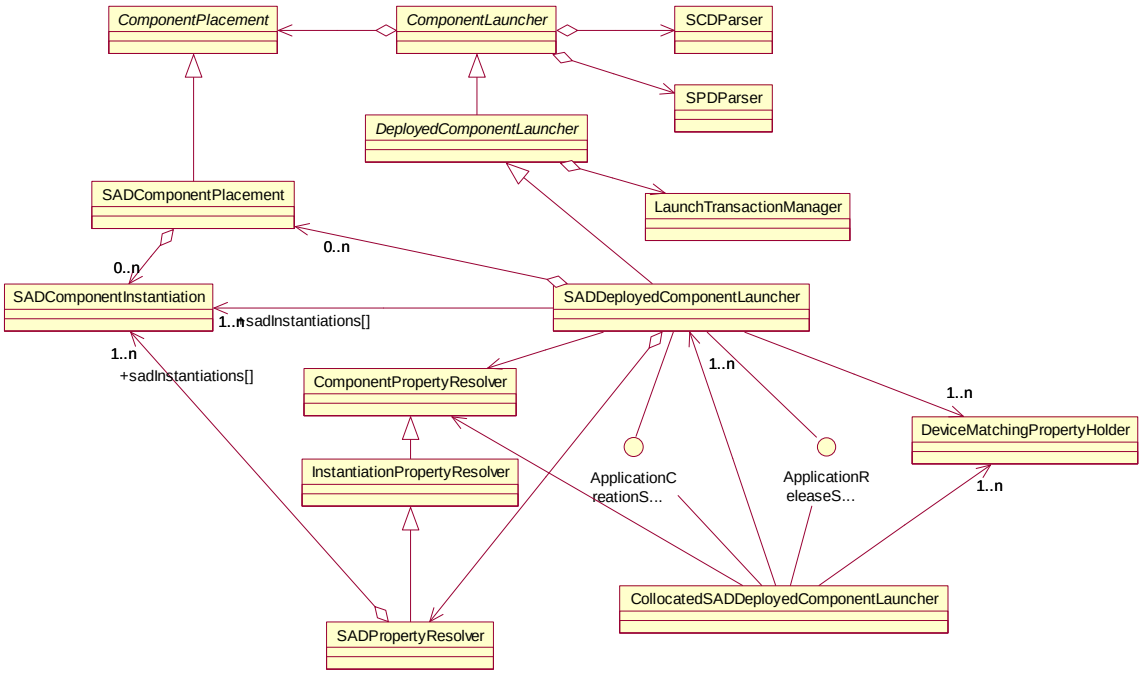

Figure 28 - Class Diagram of launching classes

1.3.1 util.domainProfile.ComponentLauncher

1.3.1.1 Description

This is a base class for all component launchers. It contains all the features for performing a basic component launch. It chooses the proper implementation for the component to be launched and all the components it depends on. Based on the selected implementations, this class determines which property value should be used for exec and config params. Once all that is done, this class uses the abstract methods load() and execute() to launch the component and load the component it depends on. Therefore, to create a component launcher, this base class must be specialized in order to provide an implementation for both load() and execute().

1.3.1.2 Class Diagram

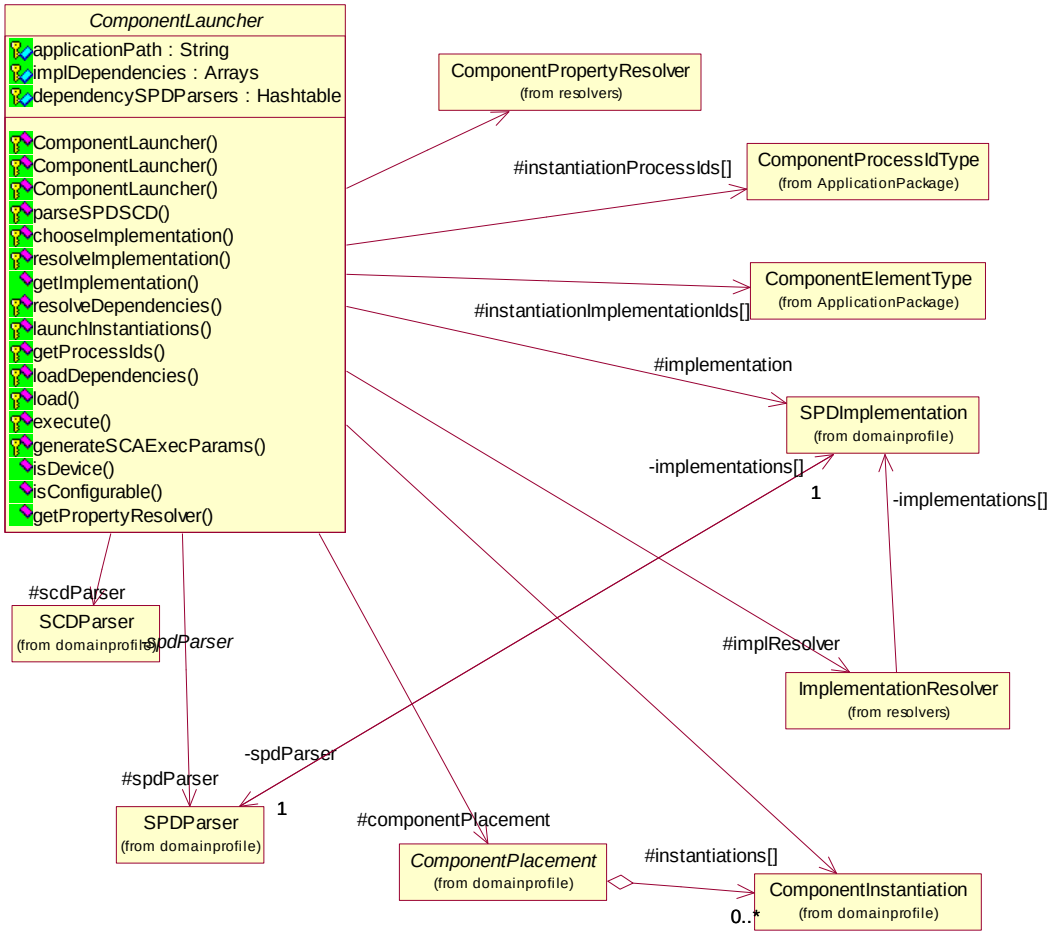

Figure 29 - ComponentLauncher Class Diagram

1.3.1.3 Implementation Specific Services

This class has been modified in order to support specialization by subclasses implementing specific services for the component placements of a DCD or a SAD file. The class implements the services generic to all subclasses. The class DCDComponentLauncher now provides the other services provided by the old version of this class.

`protected` **`ComponentLauncher`**`()`

Default Constructor. Initializes all attributes to 'null'.

`protected` **`ComponentLauncher`**`(String applicationPath, ComponentPlacement componentPlacement)`
`throws SCA.CF.InvalidProfile, SCA.CF.InvalidObjectReference`

Partial constructor.

Parameters:

applicationPath is the native OS file system to be concatenated to the XML file name in order to be used by the parsers. This is necessary since the parsers don't support the SCA file system and files.
componentPlacement DCD component placement describing a component to be launched. A component placement may contain more than one instantiation which will all be launched.

Throws:

InvalidProfile When either the SPD or its SCD is malformed
InvalidObjectReference When a input parameters have invalid reference

`protected` **`ComponentLauncher`**`(String applicationPath, ComponentPlacement componentPlacement, Hashtable matchingProperties)`
`throws SCA.CF.InvalidProfile, util.domainprofile.resolvers.BadImplementationMatch, SCA.CF.InvalidObjectReference`

Complete constructor.

Parameters:

applicationPath is the native OS file system to be concatenated to the XML file name in order to be used by the parsers. This is necessary since the parsers don't support the SCA file system and files.
componentPlacement DCD component placement describing a component to be launched. A component placement may contain more than one instantiation which will all be launched.
matchingProperties a list of properties describing the requirements for the selection of an implementation of the

component to be launched. Typically, these properties contain an OS name, a Processor name and maybe a Runtime name.

Throws:

InvalidProfile When either the SPD or its SCD is malformed

InvalidObjectReference When a input parameters have invalid reference

```
protected void chooseImplementation(Hashtable matchingProperties)
                                throws
util.domainprofile.resolvers.BadImplementationMatch,
                                       SCA.CF.InvalidProfile
```

This method selects the implemention of the component to launch depending on the matching properties.

Parameters:

matchingProperties a list of properties describing the requirements for the selection of an implementation of the component to be launched. Typically, these properties contain an OS name, a Processor name and maybe a Runtime name.

Throws:

BadImplementationMatch when no implementation matches the matching properties.

InvalidProfile when the SPD profile file is malformed.

```
protected abstract int execute(String instantiationID,
SPDImplementation implementation, Hashtable execParams, ArrayList
implDependencies)
                            throws LaunchError
```

This abstract method must be overloaded by specialized classes in order to implement the execution of the code file. In other words, this method must implement the OS execution of a file.

Throws:

LaunchError when the execution process fails

```
protected Hashtable
generateSCAExecParams(util.domainprofile.ComponentInstantiation
instantiation)
```

Returns: the SCA parameters (i.e. command line params) as mandated in section 3.1.3.2.8.5 of the SCA specification.

```
public SPDImplementation getImplementation()
```

Returns the SPDImplementation for the selected implementation of the component to be launched.
Returns: the Implementation that was selected to launch this component

`protected ComponentProcessIdType[] getProcessIds()`

`public abstract ComponentPropertyResolver getPropertyResolver()`
Returns: the ComponentPropertyResolver used to launch all the instantiations of the component to be launched by this ComponentLauncher. This PropertyResolver contains all the properties (config, exec, capacity, matching) for each specific instance launched.

`public boolean isConfigurable()`
Returns: true if the component to be launched implements SCA PropertySet. Returns false otherwise.

`public boolean isDevice()`
Returns: true if the component to be launched is an SCA Device. Returns false otherwise.

1.3.2 util.domainprofile.DeployedComponentLauncher

1.3.2.1 Description

This class is a specialized component launcher used to launch any deployed component using an SCA Device. All the deployed components of a DCD are launched using this component launcher, which inherits from the basic ComponentLauncher. Therefore this class provides an implementation for the abstract methods load() and execute(). Both these methods are implemented such that they delegate their operation to a target SCA Device.

1.3.2.2 Class Diagram

Figure 30 - DeployedcomponentLauncher Class Diagram

1.3.2.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public DCDDeployedComponentLauncher(DCDComponentPlacement
dcdComponentPlacement, DeviceManager deviceManager,
DeviceMatchingPropertyHolder deviceHolder)
                                throws SCA.CF.InvalidProfile,
```

util.domainprofile.resolvers.BadImplementationMatch,
SCA.CF.InvalidObjectReference

Constructor.

Invokes the complete constructor with aggregateDeviceIOR=null. This constructor is used when the deployed component to be launched is not a child Device of an AggregateDevice.

Parameters:
dcdComponentPlacement component placement describing a component to be launched. A component placement may contain more than one instantiation which will all be launched.
deviceManager The DeviceManager to which the deployed component (either a Device or a Service) should report to.
deviceHolder must contain a pair composed of a loadable device to be used for deployment and its associated matching properties.
Throws:
InvalidProfile when component XML is malformed
BadImplementationMatch when no implementation of the componentPlacement matches the matchingProperties
InvalidObjectReference when the fileSystem's reference is invalid or the device is not a loadable.

```
public DCDDeployedComponentLauncher(DCDComponentPlacement
dcdComponentPlacement, DeviceManager deviceManager, String
aggregateDeviceIOR, DeviceMatchingPropertyHolder deviceHolder)
                                throws SCA.CF.InvalidProfile,

util.domainprofile.resolvers.BadImplementationMatch,
                                SCA.CF.InvalidObjectReference
```

Constructor.

This constructor is used when the deployed component to be launched is a child Device of an AggregateDevice.

Parameters:
dcdComponentPlacement component placement describing a component to be launched. A component placement may contain more than one instantiation which will all be launched.
deviceManager The DeviceManager to which the deployed component (either a Device or a Service) should report to.
aggregateDeviceIOR IOR of the parent device to which this component should report to.
deviceHolder must contain a pair composed of a loadable device to be used for deployment and its associated matching properties.

Throws:
InvalidProfile when component XML is malformed
BadImplementationMatch when no implementation of the componentPlacement matches the matchingProperties
InvalidObjectReference when the fileSystem's reference is invalid or the device is not a loadable.

```
protected Hashtable
generateSCAExecParams(util.domainprofile.ComponentInstantiation
instantiation)
```

This method returns the SCA parameters (i.e. command line arguments) mandated in section 3.1.3.2.8.5 of the SCA specification. In addition, this method returns special parameters when it is launching a DomainManager. The parameters are the following 3:
1. "PROFILE_NAME" which must indicate the file name for the DMD xml file

2. "DOMAIN_MGR_ID" which must indicate the DomainManager's UUID (typically its the tag named partitioning.componentplacement.componentinstantiation.id of the DCD)

3. "DOMAIN_MGR_NAME" which must indicate the DomainManager's name (typically its the tag named partitioning.componentplacement.componentinstantiation.usagename of the DCD)

This additional functionnality is not specified in the SCA spec but is necessecary for proper functionning of the CF. It will be submited to the JTRS JPO as a change proposal.
Parameters:
instantiation a ComponentInstantiation for which the execution (command line) arguments need to be retrieved
Returns: the SCA parameters

```
public ComponentPropertyResolver getPropertyResolver()
```

Returns: the PropertyResolver used to launch all the instantiations of the component to be launched by this ComponentLauncher. This PropertyResolver contains all the properties (config, exec, capacity, matching) for each specific instance launched.

```
public String[][] getServiceIdentifiers()
```

```
public boolean isService()
```

```
public void launchComponent()
                    throws SCA.CF.DevicePackage.InvalidState,
                           SCA.CF.DevicePackage.InvalidCapacity,
                           InsufficientCapacityException,
                           LaunchError
```

Used to launch the component managed by this launcher. It performs capacity reservation and launches all the instantiations.

Throws:

InvalidState if the device is not in a state in which it can allocate capacities

InvalidCapacity When a component makes a reference (requires) to an invalid capacity property

InsufficientCapacityException when the target device does not have enough capacity for the component

LaunchError When a problem occurs during the selection of an implementation for the component to be launched

```
public void shutdown()
              throws LaunchError
```

This method will do a gracefull release of the component that has been launched. It will terminate the execution of the component, unload its code file, and deallocate any reserved capacity. This method is typically invoked by the DeviceManager when the node is shutdown.

Throws:

LaunchError when one of the releasing stage fails.

1.3.3 util.domainprofile.DeviceManagerNativeLauncher

1.3.3.1 Description

This class is a native specialized component launcher used to launch a DeviceManager at the OS level. Therefore this class provides an implementation for the abstract methods load() and execute(). Both these methods are implemented using OS primitives.

1.3.3.2 Class Diagram

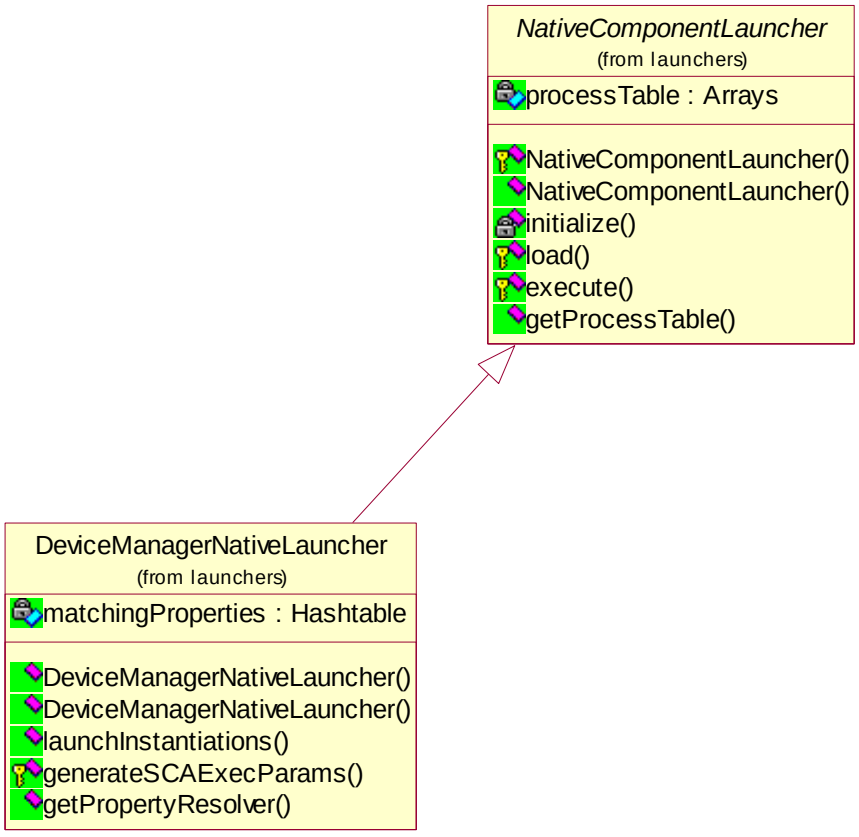

Figure 31 - DeployedcomponentLauncher Class Diagram

1.3.3.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`public` **`DeviceManagerNativeLauncher`**`(DCDParser dcdParser, Hashtable matchingProperties)`
`throws SCA.CF.InvalidProfile, util.domainprofile.resolvers.BadImplementationMatch, SCA.CF.InvalidObjectReference`

Complete constructor.
Parameters:
dcdParser DCDParser from which the DeviceManager to be launched must be found
matchingProperties matching properties of the node where the DeviceManager will be launched.
Throws:
InvalidProfile When either the SPD or its SCD is malformed
BadImplementationMatch when no implementation of the componentPlacement matches the matchingProperties
InvalidObjectReference When the DCDParser is a null reference

`protected Hashtable` **`generateSCAExecParams`**`(SPDImplementation implementation)`

This method overloads the parent in order to generate the proper SCA parameters for a the DeviceManager.
Parameters:
implementation SPDImplementation for which the SCA execution parameters need to generated
Returns: Hashtable returns the SCA parameters (i.e. command line params) See section 3.1.3.2.8.5 of the specification for the mandated SCA params.

`public ComponentPropertyResolver` **`getPropertyResolver`**`()`

Returns: the PropertyResolver used to launch all the instantiations of the component to be launched by this ComponentLauncher. This PropertyResolver contains all the properties (config, exec, capacity, matching) for each specific instance launched.

`public void` **`launchInstantiations`**`()`
`throws LaunchError`

Launch one instantiation of a DeviceManager by selecting a proper implementation based on the matching properties provided as an input argument.

Parameters:
matchingProperties a list of properties describing the requirements for the selection of an implementation of the DeviceManager to be launched. Typically, these properties contain an OS name, a Processor name and maybe a Runtime name.
Throws:
LaunchError When a problem occurs during the selection of an implementation for the DeviceManager to be launched.

1.3.4 util.domainprofile.LaunchTransactionManager

1.3.4.1 Description

This class is used to perform the operations on a device required to launch an instance of a component. The class reminds the operations for all instances launched and can undo all operations if required.

1.3.4.2 Class Diagram

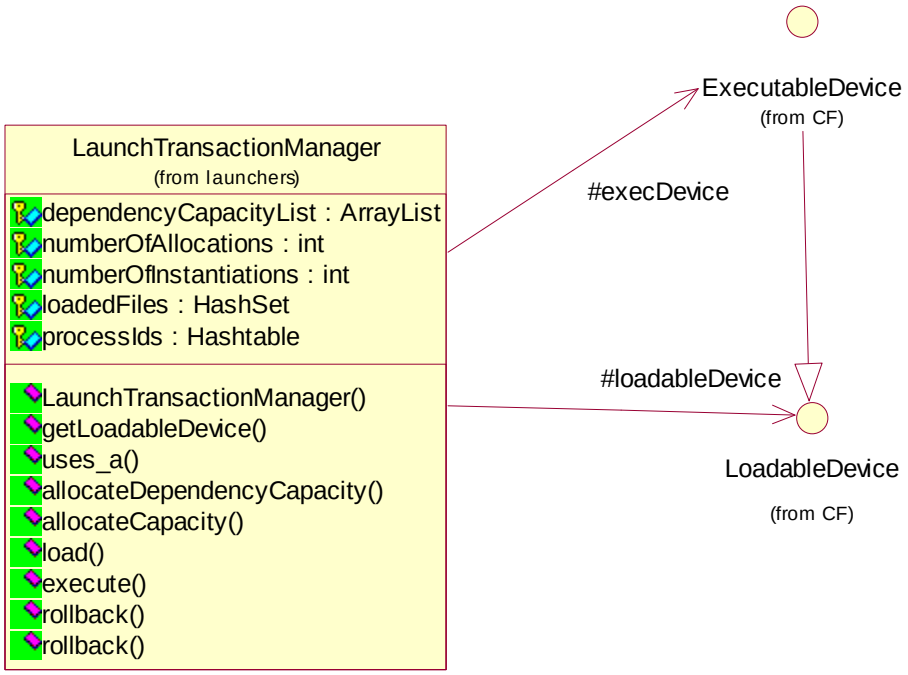

Figure 32 - LaunchTransactionManager Class Diagram

1.3.4.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public LaunchTransactionManager(SCA.CF.LoadableDevice device)
```

Parameters:
device Device on which the operations will be executed.

```
public boolean allocateCapacity(SCA.CF.DataType[] capacities, int
nbInstantiations)
                          throws SCA.CF.DevicePackage.InvalidCapacity,
                                 SCA.CF.DevicePackage.InvalidState
```

Parameters:
capacities to allocate on the device
nbInstantiations number of component instantiations for which the capacities must be allocated on the device
Returns: true if the capacities have been allocated, otherwise false.
Throws:
InvalidCapacity if the device doesn't know one or more capacities in the list
InvalidState if the device is not in a state in which it can allocate capacities

```
public boolean allocateDependencyCapacity(SCA.CF.DataType[] capacities)
                                    throws
SCA.CF.DevicePackage.InvalidCapacity,

SCA.CF.DevicePackage.InvalidState
```

Parameters:
capacities capacities to allocate on the device
Returns: true if the capacities have been allocated, otherwise false.
Throws:
InvalidCapacity if the device doesn't know one or more capacities in the list
InvalidState if the device is not in a state in which it can allocate capacities

```
public int execute(String instantiationID, String name,
SCA.CF.DataType[] options, SCA.CF.DataType[] parameters)
             throws SCA.CF.DevicePackage.InvalidState,
                    SCA.CF.ExecutableDevicePackage.ExecuteFail,
                    SCA.CF.ExecutableDevicePackage.InvalidFunction,
                    SCA.CF.ExecutableDevicePackage.InvalidParameters,
                    SCA.CF.ExecutableDevicePackage.InvalidOptions,
```

SCA.CF.InvalidFileName,
java.lang.UnsupportedOperationException

Execute a file on the device.

Parameters:

instantiationID UUID of the component instantiation to be executed
name name of the function or file to execute
options options to set on the OS for the given name
parameters parameters to the executable function or file

Throws:

InvalidState if the device is not in a state in which it can execute a process.
InvalidFunction if the name doesn't identify an existing function.
InvalidParameters if the parameters name or value are not of string types.
InvalidOptions when input options are not valid
InvalidFileName if the file name indicated by the input name parameter doesn't exist for the device.
UnsupportedOperationException if the device is not an ExecutableDevice thus not supporting the execute operation.

```
public SCA.CF.LoadableDevice getLoadableDevice()
```

Returns: the loadable Device that was provided a object creation

```
public void load(SCA.CF.FileSystem fs, String fileName,
SCA.CF.LoadableDevicePackage.LoadType loadKind)
            throws SCA.CF.DevicePackage.InvalidState,
                   SCA.CF.LoadableDevicePackage.InvalidLoadKind,
                   SCA.CF.LoadableDevicePackage.LoadFail,
                   SCA.CF.InvalidFileName
```

Load a file on the device.

Parameters:

fs file system containing the file
fileName name of the file to load
loadKind type of the file to load

Throws:

InvalidState if the device is not in a state in which it can load a file.
InvalidLoadKind if the device cannot load a file of type loadKind.
InvalidFileName if the file system doesn't contains the specified file.

`public void rollback(String instantiationID)`
`throws LaunchError`

Undo the launch of a component previous launched. This means freeing up (in terms of processes, code files, and capacity) the Device used to do the launch. This method can be invoked to cleanup the mess left by an unsuccessful launch that failed at any stage.

Parameters:
instantiationID UUID of the component instantiation to rollback

Throws:
LaunchError when either the termination of the component processes fails or

when the unload of the code file fails or

when the deallocation of capacity for the instantiations fails.

`public void rollback()`
`throws LaunchError`

Undo the launch of a component previous launched. This means freeing up (in terms of processes, code files, and capacity) the Device used to do the launch. This method can be invoked to cleanup the mess left by an unsuccessful launch that failed at any stage.

Throws:
LaunchError when either the termination of the component processes fails or

when the unload of the code file fails or

when the deallocation of capacity for the instantiations fails.

`public boolean uses_a(String repositoryIdentifier)`

Parameters:
repositoryIdentifier string representing a CORBA interface identifier

Returns: true if the device used by this LaunchTransactionManager is of type identified by reporitoryIdentifier.

1.3.5 util.domainprofile.SADDeployedComponentLauncher

1.3.5.1 Description

This class is a specialized component launcher used to launch any deployed component using an SCA Device. All the deployed components of a SAD are launched using this component launcher that inherits from the basic DeployedComponentLauncher.

1.3.5.2 Class Diagram

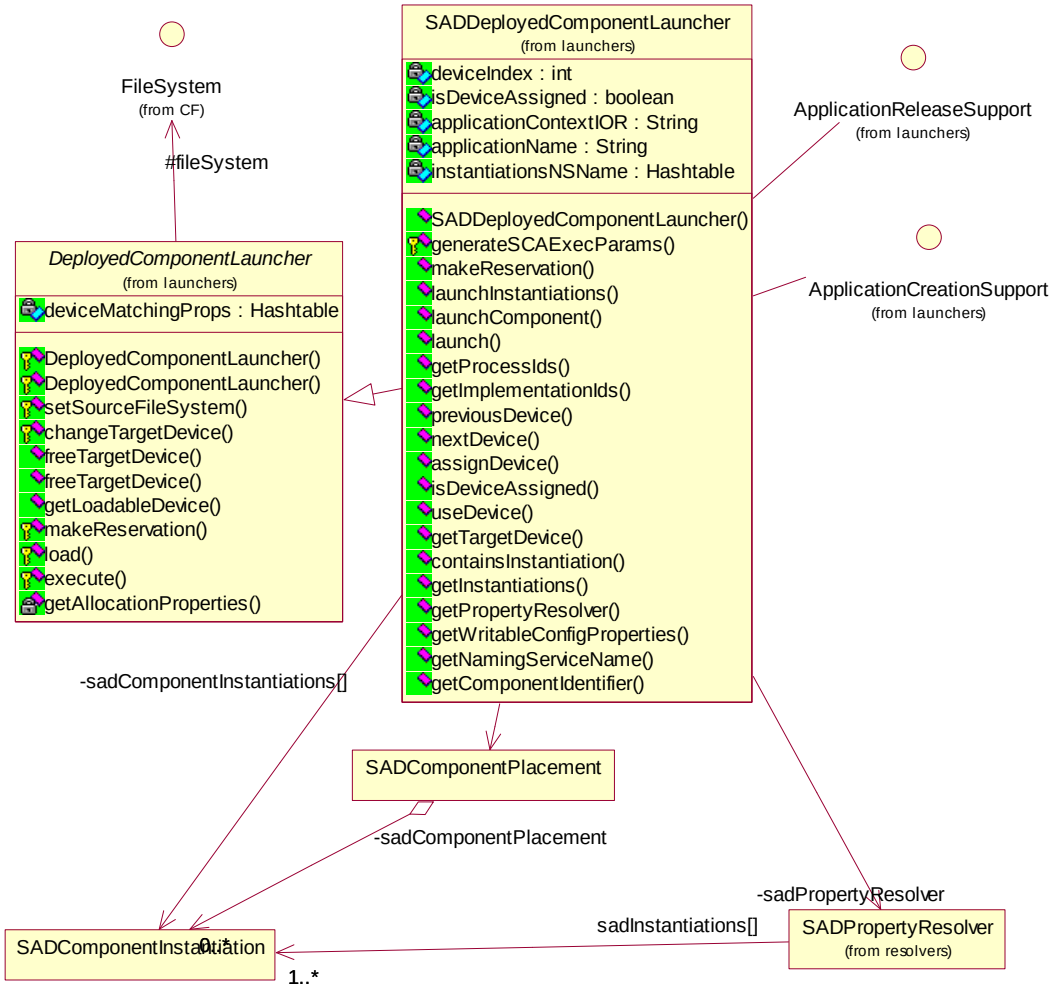

Figure 33 – SADDeployedComponentLauncher Class Diagram

1.3.5.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public SADDeployedComponentLauncher(String applicationPath,
SADComponentPlacement sadComponentPlacement,
DeviceMatchingPropertyHolder[] deviceHolders, SCA.CF.FileSystem
fileSystem, String applicationContextIOR, String applicationName)
                                  throws SCA.CF.InvalidProfile,
                                         LaunchError,
                                         SCA.CF.InvalidObjectReference
```

Partial constructor.

Invokes the complete constructor with aggregateDeviceIOR=null. This constructor is used when the deployed component to be launched is not a child Device of an AggregateDevice.

Parameters:
applicationPath is the native OS file system which will serve as the root directory to fetch the XML files specified. In otherwords, this rootPath will be prepended to the file paths of each file referenced in a SAD profile. This is necessary since the parsers don't support the SCA file system and files.
sadComponentPlacement SAD component placement describing a component to be launched. A component placement may contain more than one instantiation which will all be launched.
deviceHolders an array of holders that must contain pairs of loadable devices to be used and their associated matching properties.
fileSystem the file system where the application has been installed (more specifically, the SCA file system must be mapped to the directory where the SAD has been installed)
applicationContextIOR the IOR of the naming context of the application to which the components will bind
applicationName the name of the application the component is in
Throws:
InvalidProfile when either the SPD or its SCD is malformed
LaunchError when a problem occurs during the selection of an implementation for the component to be launched
InvalidObjectReference never raised (due to overload)

```
public void assignDevice(DeviceMatchingPropertyHolder deviceHolder)
                  throws LaunchError,

util.domainprofile.resolvers.BadImplementationMatch,
                         SCA.CF.InvalidProfile
```

This method is used to assign a loadable device to this launcher. The Launcher will try to launch the component using only one device: the assigned Device.

Parameters:
deviceHolder must contain the loadable device to be used and its asssociated matching properties.
Throws:
LaunchError when the assignDevice() method is invoked after a device has already been assigned to the launcher
InvalidProfile when an XML file is malformed
BadImplementationMatch when a problem occurs during the selection of an implementation for the component to be launched

`public boolean` **`containsInstantiation`**`(String instantiationID)`

Parameters:
instantiationID id of a component instantiation
Returns: true if this component launcher is responsible for launching the specified component instantiation

`protected Hashtable`
`generateSCAExecParams``(util.domainprofile.ComponentInstantiation instantiation)`

Parameters:
instantiation a ComponentInstantiation for which the execution (command line) arguments need to be retrieved
Returns: the SCA parameters (i.e. command line arguments) as mandated in section 3.1.3.2.8.5 of the SCA specification.

`public String` **`getComponentIdentifier`**`(String nsName)`

Parameters:
nsName name used to register a component instance to the naming service.
Returns: the component identifier

`public SCA.CF.ApplicationPackage.ComponentElementType[]`
`getImplementationIds``()`

Returns: an array containing only one implementation id; the id of the selected implementation for the component.

`public SADComponentInstantiation[]` **`getInstantiations`**`()`

Returns: an array of component instantiations for which this launcher is responsible.

`public String` **`getNamingServiceName`**`(String instantiationID)`

Parameters:
instantiationID id of a component instantiation
Returns: the name used to register the component instance to the naming service.

`public SCA.CF.ApplicationPackage.ComponentProcessIdType[] getProcessIds()`
Returns: an array containing the process id of every instantiation

`public ComponentPropertyResolver getPropertyResolver()`
Returns: the PropertyResolver used to launch all the instantiations of the component to be launched by this ComponentLauncher. This PropertyResolver contains all the properties (config, exec, capacity, matching) for each specific instance launched.

`public DeviceMatchingPropertyHolder getTargetDevice()`
Returns: the device that will be (or has been) used to launch the component

`public SCA.CF.DataType[] getWritableConfigProperties(String instantiationID)`
Parameters:
instantiationID id of a component instantiation
Returns: an array of DataType containing only the writable configuration parameters of the specified instantiation.

`public boolean isDeviceAssigned()`
Returns: true if this launcher was assigned a device, false otherwise.

`public void launch()`
`throws LaunchError`
Simply invokes launchComponent(). This method is necessary for the ApplicationCreateSupport interface.
Throws:
LaunchError when a problem occurs during the selection of an implementation for the component to be launched

`public void launchComponent()`
`throws LaunchError`
Used to launch the component managed by this launcher. It performs capacity reservation and launches all the instantiations.

Throws:
LaunchError when a problem occurs during the selection of an implementation for the component to be launched

```
public void launchInstantiations()
                            throws LaunchError
```

Launch all the instantiations of the component to be launched using the implementation passed as an input argument (which has been selected using ComponentLauncher.resolveImplementation()). This method loads ans executes all the instances. It also takes care of resolving the properties for the selected implementation so the execute method can be invoked with the proper exec params. The PropertyResolver created can be obtained through the method getPropertyResolver().

Throws:
LaunchError when a problem occurs during the selection of an implementation for the component to be launched

```
public boolean makeReservation()
                          throws SCA.CF.DevicePackage.InvalidState,
                                 SCA.CF.DevicePackage.InvalidCapacity,
                                 LaunchError,
                                 InsufficientCapacityException
```

Allocate capacity for every instantiation to be launched and for all of their dependencies (that is, for each instantiation of the components they depend on)

Throws:
InvalidState if the device is not in a state in which it can allocate capacities
InvalidCapacity when a component makes a reference (requires) to an invalid capacity property
LaunchError when a problem occurs during the selection of an implementation for the component to be launched
InsufficientCapacityException when the selected Device does not have enough capacity

```
public boolean nextDevice()
                     throws LaunchError,
                            SCA.CF.InvalidProfile
```

Among the available Devices (device holder), choose the matching device that follows the one currently selected.

Throws:

LaunchError when a problem occurs during the selection of an implementation for the component to be launched
InvalidProfile when either the SPD of the component to be launched or the SPD of a dependency is malformed

```
public boolean previousDevice()
                        throws LaunchError,
                               SCA.CF.InvalidProfile
```

Among the available Devices (device holder), choose the matching device that precedes the one currently selected.

Throws:

LaunchError when a problem occurs during the selection of an implementation for the component to be launched
InvalidProfile when either the SPD of the component to be launched or the SPD of a dependency is malformed

```
public boolean useDevice(DeviceMatchingPropertyHolder deviceHolder)
                    throws LaunchError,
                           SCA.CF.InvalidProfile
```

Verifies that the component to be launched can be launched using the input device (in terms of implementation and capacity).
If the component can be launched, the input device will be used when the launchComponent() method is invoked.

Parameters:

deviceHolder must contain the loadable device to be used and its associated matching properties.

Returns: true if the device can be used to launch the component, false otherwise.

Throws:

LaunchError when the assignDevice() method is invoked after a device has already been assigned to the launcher
InvalidProfile when an XML file is malformed

1.3.6 util.domainprofile.launchers.CollocatedSADDeployedComponent Launcher

1.3.6.1 Description

This class is a specialized component launcher used to launch any collocated deployed component using an SCA Device. All the collocated deployed components of a SAD are launched using this component launcher that aggregates one or more SADDeployedComponentLauncher.

1.3.6.2 Class Diagram

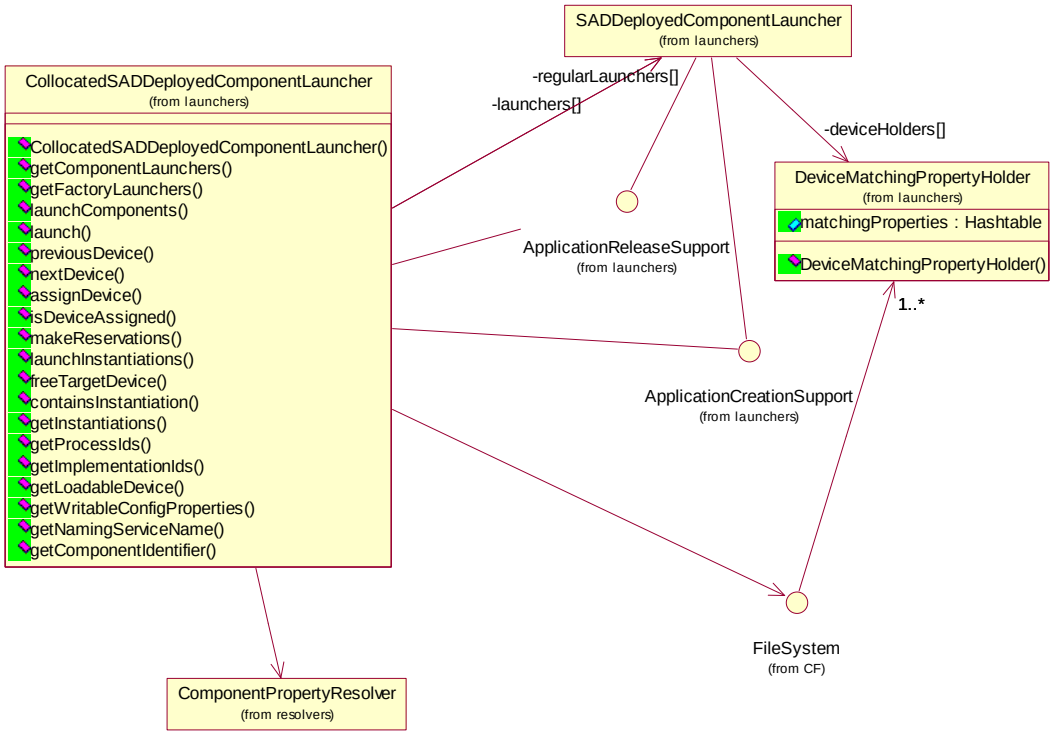

Figure 34 - CollocatedSADComponentLauncher Class Diagram

1.3.6.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public CollocatedSADDeployedComponentLauncher(String applicationPath,
SADHostCollocation hostCollocation, DeviceMatchingPropertyHolder[]
deviceHolders, SCA.CF.FileSystem fileSystem, String
applicationContextIOR, String applicationName)
                                        throws SCA.CF.InvalidProfile,
                                               LaunchError,
```

SCA.CF.InvalidObjectReference

Creates a launcher to launch the collocated components of the SAD.

Parameters:

applicationPath is the native OS file system which will serve as the root directory to fetch the XML files specified. In otherwords, this rootPath will be prepended to the file paths of each file referenced in a SAD profile. This is necessary since the parsers don't support the SCA file system and files.

sadComponentPlacement SAD component placement describing a component to be launched. A component placement may contain more than one instantiation which will all be launched.

deviceHolders an array of holders that must contain pairs of loadable devices to be used and their associated matching properties.

fileSystem the file system where the application has been installed (more specifically, the SCA file system must be mapped to the directory where the SAD has been installed)

applicationContextIOR the IOR of the naming context of the application to which the components will bind

applicationName the name of the application the component is in

Throws:

InvalidProfile when either the SPD or its SCD is malformed

LaunchError when a problem occurs during the selection of an implementation for the component to be launched

InvalidObjectReference never raised (due to overload)

```
public void assignDevice(DeviceMatchingPropertyHolder deviceHolder)
                  throws LaunchError,

util.domainprofile.resolvers.BadImplementationMatch,
                         SCA.CF.InvalidProfile
```

This method is used to assign a loadable device to this launcher. The Launcher will try to launch the component using only one device: the assigned Device.

Parameters:

deviceHolder must contain the loadable device to be used and its asssociated matching properties.

Throws:

LaunchError when the assignDevice() method is invoked after a device has already been assigned to the launcher

InvalidProfile when an XML file is malformed

BadImplementationMatch when a problem occurs during the selection of an implementation for the component to be launched

`public boolean` **`containsInstantiation`**`(String instantiationID)`

Parameters:
instantiationID id of a component instantiation
Returns: true if this component launcher is responsible for launching the specified component instantiation

`public void` **`freeTargetDevice`**`()`
`throws LaunchError`

This method will do a gracefull release of a component that has been launched. It will terminate the execution of the component, unload its code file, and deallocate any reserved capacity. This method also supports the release of a component partially launched (i.e. when the launch fails at any stage)
Throws:
LaunchError when one of the releasing stage fails.

1.3.7 util.domainprofile.ApplicationCreationSupport

1.3.7.1 Description

This interface defines the services that the launchers launching the components of an application must implement.

1.3.7.2 Class Diagram

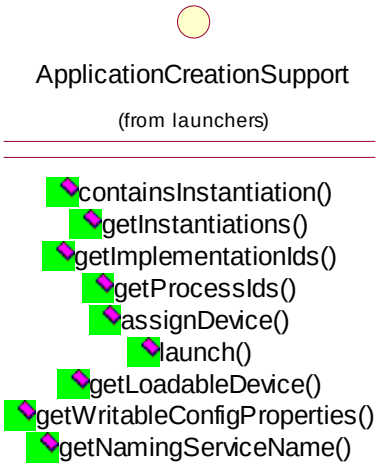

Figure 35 - ApplicationCreationSupport Class Diagram

1.3.7.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public void assignDevice(DeviceMatchingPropertyHolder deviceHolder)
                  throws LaunchError,

util.domainprofile.resolvers.BadImplementationMatch,
                         SCA.CF.InvalidProfile
```

This method is used to assign a loadable device to this launcher. The Launcher will try to launch the component using only one device: the assigned Device.

Parameters:

deviceHolder must contain the loadable device to be used and its asssociated matching properties.

Throws:

LaunchError when the assignDevice() method is invoked after a device has already been assigned to the launcher

InvalidProfile when an XML file is malformed

BadImplementationMatch when a problem occurs during the selection of an implementation for the component to be launched

```
public void launch()
            throws LaunchError
```

Used to launch the component managed by this launcher. It performs capacity reservation and launches all the instantiations.

Throws:

LaunchError when a problem occurs during the selection of an implementation for the component to be launched

1.3.8 util.domainprofile.ApplicationReleaseSupport

1.3.8.1 Description

This interface defines the services that the launchers launching the components of an application must implement.

1.3.8.2 Class Diagram

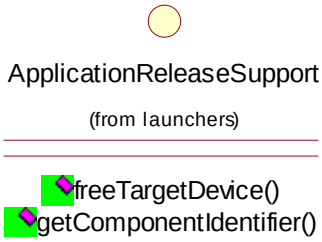

Figure 36 - ApplicationReleaseSupport Class Diagram

1.3.8.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public void freeTargetDevice()
                        throws LaunchError
```

This method will do a gracefull release of a component that has been launched. It will terminate the execution of the component, unload its code file, and deallocate any reserved capacity. This method also supports the release of a component partially launched (i.e. when the launch fails at any stage)

Throws:

LaunchError when one of the releasing stage fails.

```
public String getComponentIdentifier(String nsName)
```

Parameters:

nsName name used to register a component instance to the naming service.

Returns: the component identifier

1.3.9 SCA.CFImpl.ApplicationFactoryImpl

1.3.9.1 Description

This class implements the functionalities to create a specific type of Application in the domain. The software profile determines the type of Application that is created by the ApplicationFactory.

1.3.9.2 Class Diagram

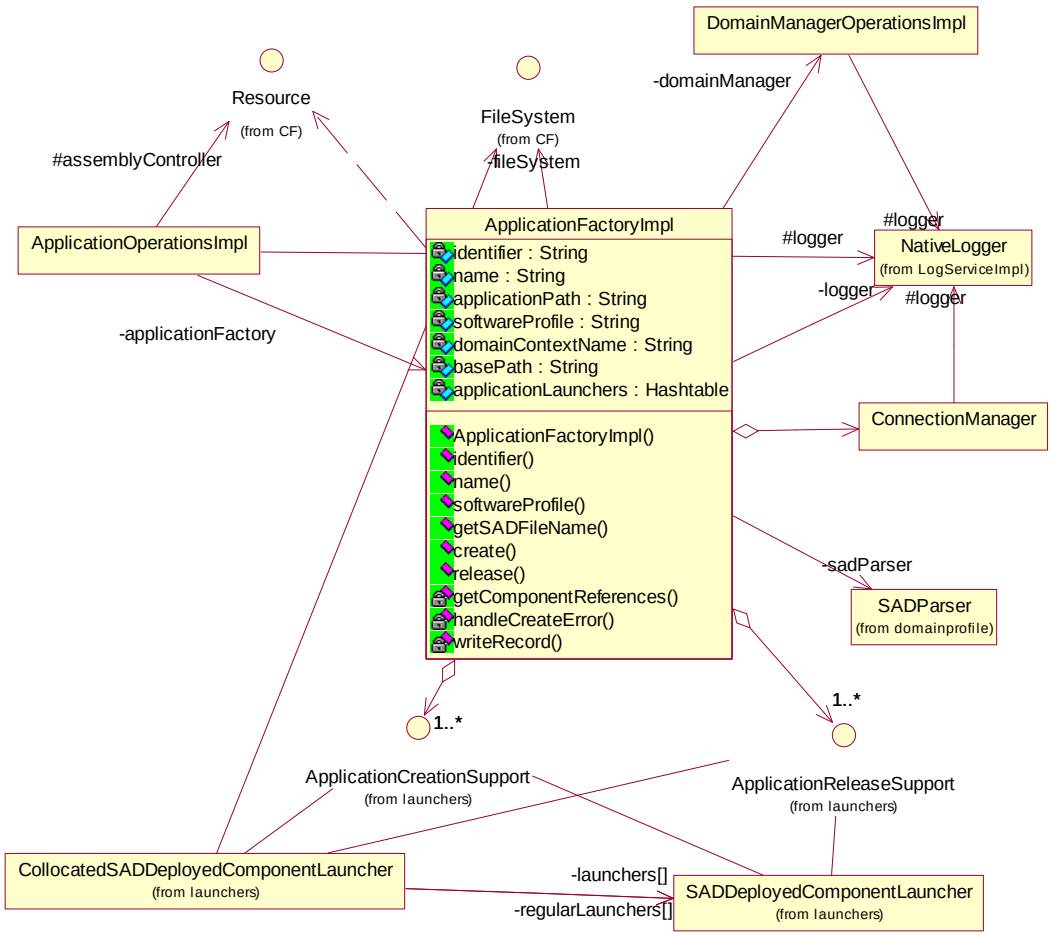

Figure 37 - ApplicationFactoryImpl Class Diagram

1.3.9.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public ApplicationFactoryImpl(String applicationPath, String
softwareProfile, String domainContextName, DomainManagerOperationsImpl
domainManager, SCA.CF.FileSystem fileSystem, org.omg.PortableServer.POA
poa, ProxyPushConsumer odmPPConsumer)
                          throws SCA.CF.InvalidProfile,
                                 SCA.CF.InvalidFileName,
                                 org.omg.CORBA.ORBPackage.InvalidName
```

Create an application factory that creates the type of application described in the software profile.

Parameters:

applicationPath the native OS file system to be concatenated to the XML file name in order to be used by the parsers. This is necessary since the parsers don't support the SCA file system and files.
softwareProfile the name of the SAD file describing an application
domainContextName the domain context name of the radio
domainManager the domain manager that creates this factory
fileSystem the filesystem to use by the component launchers that load and execute the application components
poa the poa used to CORBA enable the applications created by the factory
odmPPConsumer the consumer to pushing the events to the ODM_Channel

Throws:

InvalidProfile if the SAD file is not valid
InvalidFileName if the name of the SAD file is not valid
InvalidName if the CORBANameServiceRegistrar cannot be created

```
public synchronized Application create(String name, SCA.CF.DataType[]
initConfiguration, DeviceAssignmentType[] deviceAssignments)
                                 throws
SCA.CF.ApplicationFactoryPackage.CreateApplicationError,

SCA.CF.ApplicationFactoryPackage.CreateApplicationRequestError,

SCA.CF.ApplicationFactoryPackage.InvalidInitConfiguration
```

This method creates an instance of an application and registers it to the domain manager.

Parameters:

name Name of the application to create.
initConfiguration parameters used to initialize the application
deviceAssignments list of components to assign to specific devices

Returns: an application
Throws:
CreateApplicationError if the application cannot be created because of a processing error
CreateApplicationRequestError if the device assignments cannot be satisfied
InvalidInitConfiguration if the configuration parameters are not valid

`public String` **`getSADFileName`**`()`

Returns: the file name of the software profile describing the type of application.

`public String` **`identifier`**`()`

Returns: the application factory identifier

`public String` **`name`**`()`

Returns: the type of application that can be created using the ApplicationFactory.

`public synchronized void` **`release`**`(ApplicationOperationsImpl applicationImpl)`
`throws SCA.CF.LifeCyclePackage.ReleaseError`

Release an appplication. Perform the reverse work done in the create method. This method is used by an application when its releaseObject method is invoked.
Parameters:
applicationImpl the aplication to release.
Throws:
ReleaseError if an error occurs while releasing the application

`public String` **`softwareProfile`**`()`

Returns: the software profile describing the type of application.

1.3.10 SCA.CFImpl.ApplicationOperationsImpl

1.3.10.1 Description

This class is a generic application created by an ApplicationFactory. The Application class is a front-end to the assembly controller resource of a waveform application. The application forwards the invocation of its methods to the assembly controller, which is set by the ApplicationFactory.

1.3.10.2 Class Diagram

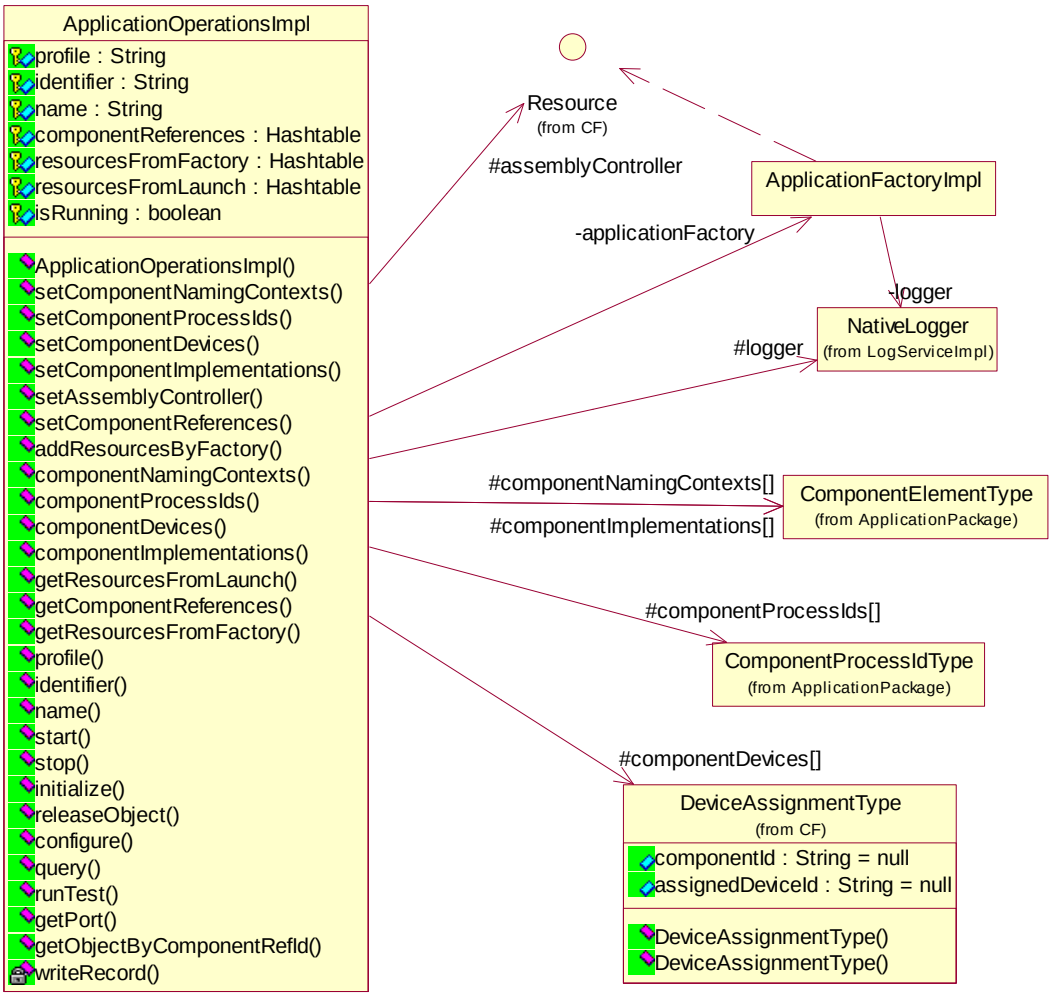

Figure 38 - ApplicationOperationsImpl Class Diagram

1.3.10.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`public` **`ApplicationOperationsImpl`**`(String identifier, String name, String profile, ExternalPort[] ports, ApplicationFactoryImpl applicationFactory, SCA.LogServiceImpl.NativeLogger logger)`

Complete constructor.

Parameters:

identifier the identifier of the application
name the name of the application
profile the name of the SAD file describing the application
ports the external ports described in the SAD file
applicationFactory the factory used to created this application

`public void` **`addResourcesByFactory`**`(Hashtable compRefsById)`

This method receives all the resource reference that were created by ResourceFactory

Parameters:

componentRefs the reference to the resources created by resource factory

`public SCA.CF.DeviceAssignmentType[]` **`componentDevices`**`()`

The componentDevices attribute shall contains the list of components' device assignments within the application. Each component (componentinstantiation element in the Application's SAD) is associated with a device.

`public ComponentElementType[]` **`componentImplementations`**`()`

The componentImplementations attribute contains the list of components' SPD implementation IDs within the Application for those components created.

`public ComponentElementType[]` **`componentNamingContexts`**`()`

The componentNamingContexts attribute contains the list of components' Naming Service Context within the Application for those components using CORBA Naming Service.

`public ComponentProcessIdType[]` **`componentProcessIds`**`()`

The componentProcessIds attribute contains the list of components' process IDs within the Application for components that are executing on a device.

```
public void configure(SCA.CF.DataType[] configProperties)
                throws SCA.CF.PropertySetPackage.InvalidConfiguration,
                       SCA.CF.PropertySetPackage.PartialConfiguration
```

Forwards the configure operation call to the application's assembly controller.

Parameters:
configProperties list of properties name/value to configure.

Throws:
InvalidConfiguration when a configuration error occurs that prevents any property configuration on the application.
PartialConfiguration when some configuration properties were successfully set and some configuration properties were not successfully set.

```
public Hashtable getComponentReferences()
```

Returns: the list of references of the resources composing the application

```
public org.omg.CORBA.Object getObjectByComponentRefId(String compRefId)
```

this method provides the CORBA object reference which is associated with the compenent reference id

Parameters:
compRefId the component reference id

Returns: the object reference associated with the compRefId, otherwise it returns null

```
public org.omg.CORBA.Object getPort(String name)
                                throws
SCA.CF.PortSupplierPackage.UnknownPort
```

Forwards the getPort operation call to the application's assembly controller.

Parameters:
name indicates the name of the port requested

Throws:
UnknownPort if the port name is invalid.

see 3.1.3.1.4.5.1

```
public Hashtable getResourcesFromFactory()
```

Returns: the list of references of the resources composing the application

```
public Hashtable getResourcesFromLaunch()
```

Returns: the list of references of the resources composing the application

`public String` **`identifier`**`()`

The identifier attribute contains the identifier of the created Application. The ApplicationFactory interface's create operation provides the identifier content.

`public void` **`initialize`**`()`
`throws SCA.CF.LifeCyclePackage.InitializeError`

No action is performed by an application for this method.

Throws:

InitializeError.

see 3.1.3.2.1.5

`public String` **`name`**`()`

The name attribute contains the name of the created Application. The ApplicationFactory interface's create operation name parameter provides the name content.

`public String` **`profile`**`()`

This attribute is the XML profile information for the application. The string value contains either a profile element with a file reference to the SAD profile file or the actual xml for the SAD profile. Files referenced within a profile will have to be obtained from a FileManager. The Application will have to be queried for profile information for Component files that are referenced by an ID instead of a file name.

`public void` **`query`**`(SCA.CF.PropertiesHolder configProperties)`
`throws SCA.CF.UnknownProperties`

`public void` **`releaseObject`**`()`
`throws SCA.CF.LifeCyclePackage.ReleaseError`

Release the application.

Throws:

ReleaseError if the application cannot be released

`public void` **`runTest`**`(int testid, SCA.CF.PropertiesHolder testValues)`
`throws SCA.CF.TestableObjectPackage.UnknownTest,`
`SCA.CF.UnknownProperties`

Forwards the runTest operation call to the application's assembly controller.

Parameters:

testid test number to be executed

testValues additional information to the implementation-specific test to be run

Throws:

UnknownTest when there is no underlying test implementation that is associated with the input testid given.

UnknownProperties when the input parameter testValues contains any DataTypes that are not known by the component's test implementation or any values that are out of range for the requested test. The exception parameter invalidProperties contains the invalid inputValues properties id(s) that are not known by the component or the value(s) are out of range.

see 3.1.3.1.3.5.1

`public void` **`setAssemblyController`**`(SCA.CF.Resource assemblyController)`

Parameters:

assemblyController the reference to the assemblyController resource

`public void` **`setComponentDevices`**`(SCA.CF.DeviceAssignmentType[] componentDevices)`

Parameters:

componentDeviceslist list of device assignments for the coomponents within the application. Each component (componentinstantiation element in the Application's SAD) is associated with a device.

`public void` **`setComponentImplementations`**`(ComponentElementType[] componentImplementations)`

Parameters:

componentImplementations list of implementation IDs of the components within the Application for those components created.

`public void` **`setComponentNamingContexts`**`(ComponentElementType[] componentNamingContexts)`

Parameters:

componentNamingContexts list of Naming Service Context of the components within the Application for those components using CORBA Naming Service.

`public void` **`setComponentProcessIds`**`(ComponentProcessIdType[] componentProcessIds)`

Parameters:

componentProcessIds list of process IDs of the components within the Application for the components that are executing on a device.

`public void` **`setComponentReferences`**`(Hashtable compRefsById)`

This method receives all the component reference for this application

Parameters:

componentRefs the reference to the resources composing the application

`public void` **`start`**`()`
`throws SCA.CF.ResourcePackage.StartError`

Start the application processing by forwarding the call to the application's assembly controller.

Throws:

StartError if the application cannot be started

`public void` **`stop`**`()`
`throws SCA.CF.ResourcePackage.StopError`

Stop the application processing by forwarding the call to the application's assembly controller.

Throws:

StartError if the application cannot be stopped

1.3.11 util.install.ComponentInstaller

1.3.11.1 Description

This class is used to install a component file from a native file system into a SCA file system.

1.3.11.2 Class Diagram

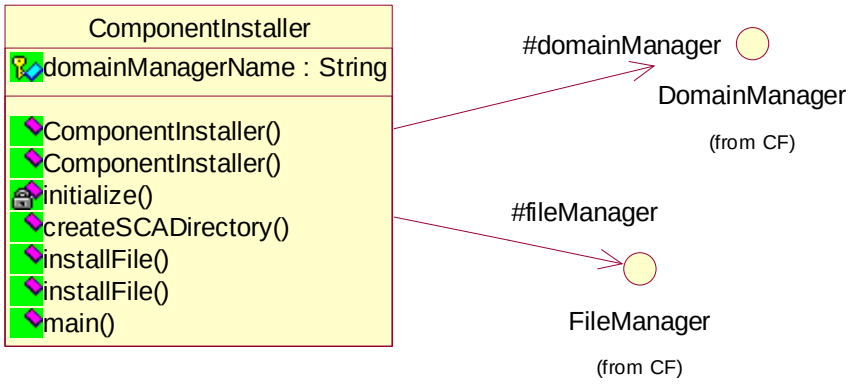

Figure 39 - ComponentInstaller Class Diagram

1.3.11.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public ComponentInstaller(String[] args, String domainManagerName)
                    throws org.omg.CORBA.ORBPackage.InvalidName
```

Parameters:
args command line arguments to use to initialize the ORB.
domainManagerName the name to use to retrieve the domain manager reference from the naming service.
Throws:
InvalidName when the naming service or the domain manager reference cannot be found.

```
public ComponentInstaller(String initialHost, String initialPort,
String domainManagerName)
                    throws org.omg.CORBA.ORBPackage.InvalidName
```

Parameters:
initialHost address to use to initialize the ORB.
initialHost port to use to initialize the ORB.

domainManagerName the name to use to retrieve the domain manager reference from the naming service.

Throws:

InvalidName when the naming service or the domain manager reference cannot be found.

```
public void createSCADirectory(String directoryName)
                        throws SCA.CF.InvalidFileName,
                               SCA.CF.FileException
```

This method create a directory in the file system of the domain manager if it doesn't exist.

Parameters:

directoryName the name of the directory to create

Throws:

InvalidFileName if the directory name is not valid

FileException if the diretory already exist

```
public void installFile(String nativeFileName, String scaFileName)
                    throws InstallException
```

Copy a file from the native file system into the file system of the domain manager.

Parameters:

nativeFileName path name of the native file

scaFileName absolute path name of the SCA file to create

Throws:

InstallException if an error occurs during the installation of the file.

```
public void installFile(java.io.InputStream nativeFileIn, String
scaFileName)
                    throws InstallException
```

Copy a file from the native file system into the file system of the domain manager.

Parameters:

nativeFileIn input stream to read the native file

scaFileName absolute path name of the SCA file to create

Throws:

InstallException if an error occurs during the installation of the file.

1.3.12 util.install.ApplicationInstaller

1.3.12.1 Description

This class is used to install an SCA application. Methods are provided to copy the files composing the application from the native file system into the SCA file system of a domain manager.

1.3.12.2 Class Diagram

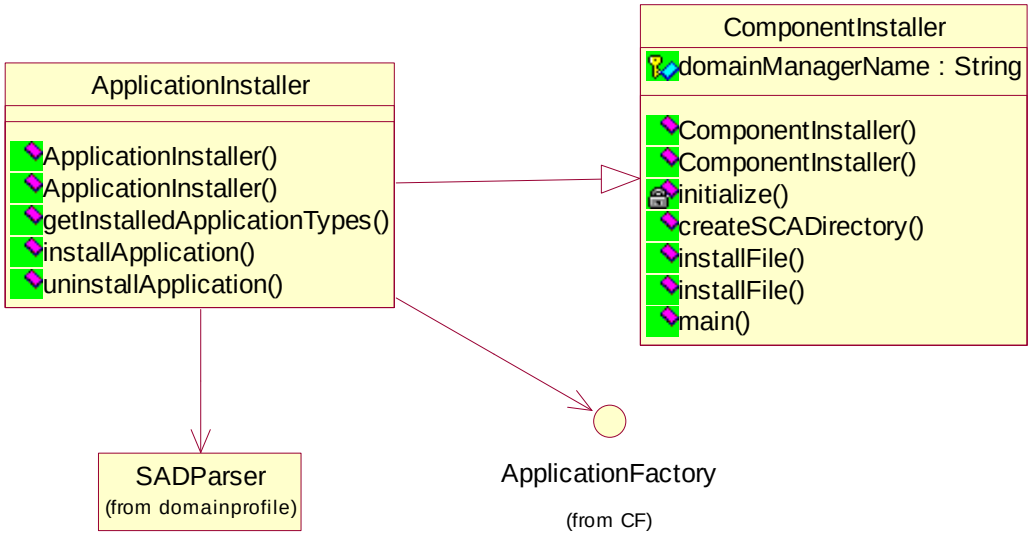

Figure 40 - ApplicationInstaller Class Diagram

1.3.12.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

```
public ApplicationInstaller(String[] args, String domainManagerName)
                    throws org.omg.CORBA.ORBPackage.InvalidName
```

Parameters:
args command line arguments to use to initialize the ORB.
domainManagerName the name to use to retrieve the domain manager reference from the naming service.
Throws:
InvalidName when the naming service or the domain manager reference cannot be found.

```
public ApplicationInstaller(String initialHost, String initialPort,
String domainManagerName)
                        throws org.omg.CORBA.ORBPackage.InvalidName
```

Parameters:
initialHost address to use to initialize the ORB.
initialHost port to use to initialize the ORB.
domainManagerName the name to use to retrieve the domain manager reference from the naming service.
Throws:
InvalidName when the naming service or the domain manager reference cannot be found.

```
public ApplicationType[] getInstalledApplicationTypes()
```

Returns: a list of names of application already installed in the domain manager.

```
public void installApplication(String applicationJarFile)
                          throws InstallException
```

Copy the application files packaged in a JAR file in the native file system into the file system of the domain manager. The path of each file in the JAR file is used as the path relative to the application root directory when the file is created in the SCA file system.
Parameters:
applicationJarFile name of the application JAR file
Throws:
InstallException if an error occurs during the installation of the files.

```
public void uninstallApplication(String applicationType)
                            throws java.lang.Exception
```

Uninstalls an application from the domain manager domain profile.
Parameters:
applicationType the type of the application to uninstall (correspond to the name element value in the SAD file of the application).
Throws:
Exception if an error occurs during the uninstallation

1.3.13 SCA.CFImpl.DomainManagerOperationsImpl

1.3.13.1 Description

Upon their registration, the DomainManager probes *DeviceManager* to get a list of their Devices and Service. The *DomainManager* also reads the *DeviceManager's* DCD to find out if the DCD components need to be connected each other. If it's the case, the *DomainManager* delegates this work to a class called the *ConnectionManager*. This new class is responsible for establishing the connections between registered components. It runs on a separate Thread and is controlled by the *DomainManager*.

1.3.13.2 Class Diagram

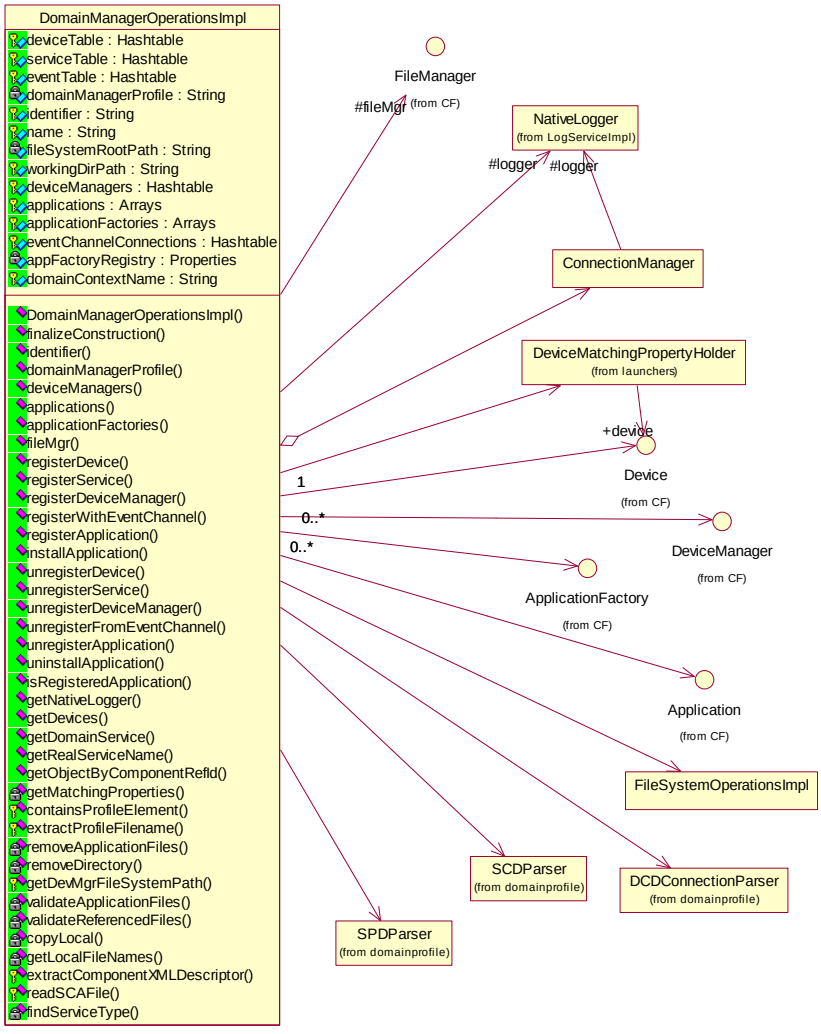

Figure 41 - DomainManagerOperationsImpl Class Diagram

1.3.13.3 Implementation Specific Services

`public` SCA.CF.ApplicationFactory[] **`applicationFactories`**`()`

The readonly applicationFactories attribute contains a list with one ApplicationFactory per application(SAD file and associated files) successfully installed(i.e. no exception raised). The DomainManager writes an ADMINISTRATIVE_EVENT log record to a DomainManager's Log, when
the applicationFactories attribute is obtained by a client.
Returns: returns an empty list of application factories

`public` SCA.CF.Application[] **`applications`**`()`

The applications attribute is read-only containing a sequence of instantiated Applications in the domain. The DomainManager contains a list of Applications that have been instantiated. The DomainManager writes an ADMINISTRATIVE_EVENT log record to a DomainManager's Log, when the
applications attribute is obtained by a client.
Returns: returns an empty list of applications

`public` SCA.CF.DeviceManager[] **`deviceManagers`**`()`

The deviceManagers attribute is read-only containing a sequence of registered DeviceManagers in the domain. The DomainManager contains a list of registered DeviceManagers that have registered with the DomainManager. The DomainManager writes an ADMINISTRATIVE_EVENT log to a
DomainManager's Log, when the deviceManagers attribute is obtained by a client.
Returns: a list of registered DeviceManagers

`public String` **`domainManagerProfile`**`()`

The domainManagerProfile attribute contains the DomainManager's profile. The readonly domainManagerProfile attribute contains either a profile element with a file reference to the DomainManager's (DMD) profile or the XML for the DomainManager's(DMD) profile. Files referenced within the profile will have to be obtained from the DomainManager's FileManager.
Returns: the name of the domain manager profile

`public` SCA.CF.FileManager **`fileMgr`**`()`

The fileMgr attribute is read only containing the mounted FileSystems in the domain. The DomainManager writes an ADMINISTRATIVE_EVENT log record

to a DomainManager's Log, when the fileMgr attribute is obtained by a client.
Returns: the DomainManager's FileManager

```
public void finalizeConstruction(org.omg.CORBA.ORB orb,
org.omg.PortableServer.POA poa)
                          throws java.lang.InstantiationException
```

This is how the DomainManager gets hold of the CORBA POA servant that represents it. It is called by a DomainManagerServer that is constructing this DomainManager Object.
Parameters:
poa reference to the poa used to activate the CORBA object representing this implementation instance of a DomainManager
Throws:
InstantiationException when the poa is invalid or an invalid fileSystemRootPath name was used at construction time or when a CORBA Exception occurs during instantiation of the Root FileSystem

```
public DeviceMatchingPropertyHolder[] getDevices()
```

Returns: an Array of DeviceMatchingPropertyHolder which contains the mathing properties for each Device

```
public org.omg.CORBA.Object getDomainService(String type, String name)
```

This method returns a registered service (i.e an object) based on the input type an name arguments. As specified in the "domainfinder" section (e.g. D.6.5.1.1.3.2), this method will:

- return the DomainManager's registered service corresponding to the input name argument
- return the DomainManager's FileManager when the type is "filemanager" and no name is provided
- return a null reference when the type is "log" and no name is provided
- return the DomainManager's event channel IDM_Channel when the type is "eventchannel" and no name is provided
- return the DomainManager's Naming Service when the type is "namingservice" and no name is provided

Parameters:
type can be "filemanager", "log", "eventchannel", or "namingservice"
name the name of the service (optional except when type is "log")
Returns: the corresponding service object. Null otherwise.

`public SCA.LogServiceImpl.NativeLogger getNativeLogger()`
Returns: the native logger used by the DomainManager. Upon connection of the DomainManager with its SCA Log, the native logger will transfers all its messages to the SCA Log.

`public org.omg.CORBA.Object getObjectByComponentRefId(String compRefId)`
this method provides the CORBA object reference
which is associated with the compenent reference id
Parameters:
compRefId the component reference id
Returns: the object reference associated with the compRefId,
otherwise it returns null

`public String getRealServiceName(org.omg.CORBA.Object serviceObject)`
This method returns the name that has been used to register a service to the DomainManager.
Parameters:
serviceObject the service component for which the name is requested
Returns: the name used to register the serviceObject component

`public String identifier()`
Returns: the domain manager identifier

```
public void installApplication(String profileFileName)
                        throws SCA.CF.InvalidProfile,
                               SCA.CF.InvalidFileName,
SCA.CF.DomainManagerPackage.ApplicationInstallationError
```
This operation is used to register new application software in the DomainManager's Domain Profile. An installer application typically invokes this operation when it has completed the installation of a new Application into the domain.

The profileFileName is the absolute path of the profile filename.

The installApplication operation verifies the application's SAD file exists in the DomainManager's FileManager and all the files the application is dependent on are also resident.

The installApplication operation writes an ADMINISTRATIVE_EVENT log Record to a DomainManager's Log, upon successful Application installation.

Throws:
ApplicationInstallationError when the installation of the Application file(s) was not successfully completed.

InvalidFileName when the input SAD file or any referenced file name does not exist in the file system as defined in the absolute path of the input profileFileName. The installApplication operation logs a FAILURE_ALARM log record to a DomainManger's Log when the

InvalidFileName exception occurs and the logged message shall be "installApplication:: invalid file is xxx", where "xxx" is the input or referenced file name is bad.

InvalidProfile when the input SAD file or any referenced file is not compliant with XML DTDs defined in Appendix D or referenced property definitions are missing. The installApplication operation s logs a FAILURE_ALRAM log record ot a DomainManager's Log when the CF InvalidProfile exception occurs and the logged message shall be "installApplication:: invalid Profile is yyy," where "yyy" is and the input or referenced file name that is bad along with the element or position within the profile that is bad.

```
public synchronized boolean isRegisteredApplication(String name)
```

Used to determine if an application name has already been used (e.g. registered)
Parameters:
name name of the application to look for
Returns: true if the name of the application is already registered. Otherwise, returns false.

```
protected String readSCAFile(SCA.CF.FileSystem fileSystem, String
componentFileName)
                        throws SCA.CF.InvalidFileName,
                               SCA.CF.FileException,
                               SCA.CF.FilePackage.IOException
```

This method is used to return a String that has the content of an SCA File mainly use for parsing purposes.
Parameters:

mountPoint mountPoint in the file manager where the SCA file is located
scaFileName name of the SCA file to read content
Returns: String containing the content of the file
Throws:
InvalidFileName when the SCA file name is invalid
FileException when an error occurs in copying the file
IOException when an error occurs in copying the file

```
public synchronized boolean registerApplication(Application
application)
```

This method is used by the ApplicationFactory to add an Application to the DomnainManager's list when the factory creates a new Application
Parameters:
application the application to register to the DM's list

```
public void registerDevice(SCA.CF.Device registeringDevice,
SCA.CF.DeviceManager registeredDeviceMgr)
                    throws SCA.CF.InvalidObjectReference,
                                   SCA.CF.InvalidProfile,
   SCA.CF.DomainManagerPackage.DeviceManagerNotRegistered,
                SCA.CF.DomainManagerPackage.RegisterError
```

The registerDevice operation verifies that the input parameters, registeringDevice and registeredDeviceMgr, are not nil CORBA component references.

The registerDevice operation adds the registeringDevice and the registeringDevice's attributes (e.g., identifier, softwareProfile's allocation properties, etc.) to the DomainManager, if it does not already exist.

The registerDevice operation associates the input registeringDevice with the input registeredDeviceMgr in the DomainManager when the input registeredDeviceMgr is a valid registered DeviceManager in the DomainManager.

The registerDevice operation, upon successful device registration, writes an ADMINISTRATIVE_EVENT log record to a DomainManager's Log, to indicate that the device has successfully registered with the DomainManager.

Upon unsuccessful device registration, the registerDevice operation writes a FAILURE_ALARM log record to a DomainManager's

Log, when the InvalidProfile exception is raised to indicate that the registeringDevice has an invalid profile.

Upon unsuccessful device registration, the registerDevice operation logs a Failure_Alarm event with DomainManager's Logger for the DeviceManagerNotRegistered exception to indicate that the device that cannot be registered to the Device due to the DeviceManager is not registered with the DomainManager.

Upon unsuccessful device registration, the registerDevice operation writes a FAILURE_ALARM log record to a DomainManager's Log, indicating that the device could not register because the DeviceManager is not registered with the DomainManager.

Upon unsuccessful device registration, the registerDevice operation writes a FAILURE_ALARM log record to a DomainManager's Log, because of an invalid reference input parameter.

Throws:
InvalidProfile when the Device's SPD file and the SPD's referenced files do not exist or cannot be processed due to the file not being compliant with XML syntax

DeviceManagerNotRegistered when the input registeredDeviceMgr(not nil reference) is not registered with the DomainManager.

InvalidObjectReference when input parameters registeringDevice or registeredDeviceMgr contains an invalid reference.

```
public void registerDeviceManager(SCA.CF.DeviceManager deviceMgr)
                            throws SCA.CF.InvalidObjectReference,
                                   SCA.CF.InvalidProfile,

SCA.CF.DomainManagerPackage.RegisterError
```

The registerDeviceManager operation verifies that the input parameter, deviceMgr, is a not a nil CORBA component reference.

The registerDeviceManager operation adds the input deviceMgr's registeredServices and each registeredService's names to the DomainManager. The registerDeviceManager operation associates the input deviceMgr's with the input deviceMgr's registeredServices in the DomainManager in order to support the unregisterDeviceManager operation.

The registerDeviceManager operation performs the documented connections as specified in the connections element of the deviceMgr's DCD file. For connections established for a Log, the registerDeviceManager operation creates a unique producer log ID for each log producer. The registerDeviceManager operation invoke the PropertySet configure operation on each log producer in order to set its unique PRODUCER_LOG_ID. If the DeviceManager's DCD describes a connection for a service that has not been registered with the DomainManager, the registerDeviceManager operation establishes any pending connection when the service registers with the DomainManager by the registerDeviceManager operation.

The registerDeviceManager operation adds the input deviceMgr to the DomainManager's deviceManagers attribute, if it does not already exist.

The registerDeviceManager operation adds the input deviceMgr's registeredDevices and each registeredDevice's attributes (e.g., identifier, softwareProfile's allocation properties, etc.) to the DomainManager.

The registerDeviceManager operation associates the input deviceMgr with the input deviceMgr's registeredDevices in the DomainManager in order to support the unregisterDeviceManager operation.

The registerDeviceManager operation obtains all the Software profiles from the registering DeviceManager's FileSystems.

The registerDeviceManager operation mounts the DeviceManager's FileSystem to the DomainManager's FileManager. The mounted FileSystem name shall have the format, "/DomainName/HostName", where DomainName is the name of the domain and HostName is the input deviceMgr's label attribute

The registerDeviceManager operation, upon unsuccessful DeviceManager registration, writes a FAILURE_ALARM log record to a DomainManager's Log.

Throws:
InvalidObjectReference when the input parameter deviceMgr contains an invalid reference to a DeviceManager interface.
InvalidProfile when

1. The DeviceManager's DCD file and the referenced files do not exist or cannot be processed due to the file not being compliant with XML syntax
2. The Device's SPD file and the SPDs referenced files do not exist or cannot be processed due to the file not being compliant with XML syntax.

The description of this exception is missing from section 3.1.3.2.3.6.1.5

```
public void registerService(org.omg.CORBA.Object registeringService,
SCA.CF.DeviceManager registeredDeviceMgr, String serviceName)
                        throws SCA.CF.InvalidObjectReference,
                                    SCA.CF.InvalidProfile,
    SCA.CF.DomainManagerPackage.DeviceManagerNotRegistered,
                    SCA.CF.DomainManagerPackage.RegisterError
```

This operation is used to register a service for a specific DeviceManager with the DomainManager.

The registerService operation verifies the input registeringService and registeredDeviceMgr are valid object references .The registerService operation verifies the input registeredDeviceMgr has been previously registered with the DomainManager. The registerService operation adds the registeringService and the registeringService's name to the DomainManager. However, if the name of the registering service is not a unique name for that type of service(i.e. it is a duplicate name of an already registered service of the same type of service), then the new service is not be registered by the DomainManager.The registerService operation associates the input registeringService with the input registeredDeviceMgr in the DomainManager when the input registeredDeviceMgr is a valid registered DeviceManager in the DomainManager. The registerService operation, upon successful service registration, establish any pending connection requests for the registeringService. For connections established for a Log, the registerService operation creates a unique producer log ID for each log producer. The registerService operation invokes the PropertySet configure operation onceon each log producer in order to set its unique PRODUCER_LOG_ID(see section 3.1.3.3.5.5.1.2 for details).

The registerService operation, upon successful service registration, writes a log record to a DomainManager's Log, with the log level set to ADMINISTRATIVE_EVENT.

The registerService operation, upon unsuccessful service registration, writes a log record to a DomainManager's Log, with the log level set to FAILURE_ALARM.

Throws:
InvalidObjectReference when input parameters registeringService or registeredDeviceMgr contains an invalid reference.
InvalidProfile This exception is NEVER thrown since a service does not implement a specific interface which would enable the DomainManager to extract its profile.
DeviceManagerNotRegistered when the input registeredDeviceMgr(not nil reference) is not registered with the DomainManager.

```
public void registerWithEventChannel(org.omg.CORBA.Object
registeringConsumerObject, String registeringConsumerId, String
eventChannelName)
                                throws SCA.CF.InvalidObjectReference,

SCA.CF.DomainManagerPackage.InvalidEventChannelName,

SCA.CF.DomainManagerPackage.AlreadyConnected
```

The registerWithEventChannel operation is used to connect a consumer to a domain's event channel.

The registerWithEventChannel operation connects the input registeringConsumer to event channel as specified by the input eventChannelName.

Parameters:
registeringConsumerObject Corba Object that is a CosEventComm PushConsumer
registeringId Id that identifies this registration.
eventChannelName name of the event channel that the registering consumer wants to be registered to.

Throws:
InvalidObjectReference when the registeringConsumerObject contains an invalid reference to a CosEventComm PushConsumer interface.
InvalidEventChannelName when the input parameter contains an invalid event channel name

AlreadyConnected when the input parameter contains a connection to the event channel for the input registeringConsumerId parameter.

```
public void uninstallApplication(String applicationID)
                          throws
SCA.CF.DomainManagerPackage.InvalidIdentifier,
SCA.CF.DomainManagerPackage.ApplicationUninstallationError
```

This operation is used to uninstall an application in the DomainManager's Domain Profile. The CF Installer typically invokes this operation when removing an application from the radio domain.

The uninstallApplication operation removes all files associated with the Application.

The uninstallApplication operation makes the ApplicationFactory unavailable from the DomainManager(i.e. its services no longer provided for the Application).

The uninstallApplication operation, upon successful uninstall of an Application, writes an ADMINISTRATIVE_EVENT log record to a DomainManager's Log.

The uninstallApplication operation, upon unsuccessful uninstall of an Application, writes a FAILURE_ALARM log record to a DomainManager's Log.

Throws:
InvalidIdentifier when the ApplicationID is invalid.

```
public synchronized boolean unregisterApplication(Application
application)
```

This method is used by the ApplicationFactory to add an Application to the DomnainManager's list when the factory creates a new Application

Parameters:
application the application to unregister to the DM's list

Returns: true if the application was unregistered from the DM's list

```
public void unregisterDevice(SCA.CF.Device unregisteringDevice)
                      throws SCA.CF.InvalidObjectReference,
```

SCA.CF.DomainManagerPackage.UnregisterError

The unregisterDevice operation removes a device entry from the DomainManager's Domain Profile.

The unregisterDevice operation releases(client-side CORBA release) the unregistering Device from the DomainManager.

The unregisterDevice operation, upon the successful unregistration of a Device, writes an ADMINISTRATIVE_EVENT log record to a DomainManager's Log.

The unregisterDevice operation, upon unsuccessful unregistration of a Device, writes a FAILURE_ALARM log record to a DomainManager's Log.

Throws:
InvalidObjectReference exception when the input parameter contains an invalid reference to a Device interface.

```
public void unregisterDeviceManager(SCA.CF.DeviceManager
unregisteringDeviceMgr)
                    throws SCA.CF.InvalidObjectReference,
             SCA.CF.DomainManagerPackage.UnregisterError
```

This operation is used to unregister a DeviceManager object from the DomainManager's Domain Profile.

The unregisterDeviceManager operation unregisters a DeviceManager component from the DomainManager.

The unregisterDeviceManager operation releases(client-side CORBA release) all device(s) and service(s) associated with the DeviceManager that is being unregistered.

The unregisterDeviceManager operation shall unmount all DeviceManager's FileSystems from its File Manager.

The unregisterDeviceManager operation, upon the successful unregistration of a DeviceManager, writes an ADMINISTRATIVE_EVENT log record to a DomainManager's Log.

The unregisterDeviceManager operation, upon unsuccessful unregistration of a DeviceManager, writes a FAILURE_ALARM log record to a DomainManager's Log.

Throws:
InvalidObjectReference when the input parameter DeviceManager contains an invalid reference to a DeviceManager interface.

```
public void unregisterFromEventChannel(String registeredConsumerId,
String eventChannelName)
                                    throws
SCA.CF.DomainManagerPackage.InvalidEventChannelName,
SCA.CF.DomainManagerPackage.NotConnected
```

The unregisterFromEventChannel operation is used to disconnect a consumer from a domain's event channel.

The unregisterFromEventChannel operation shall disconnect a registered consumer from the event channel as identified by the input parameters.

Parameters:
registeredConsumerId Id that identifies the consumer that was registerd.
eventChannelName name of the event channel that the registering consumer wants to be unregistered from.
Throws:
InvalidEventChannelName when the input parameter contains an invalid name of an event channel
NotConnected when the input parameter registeredConsumerId parameter is not connected to specified input event channel.

```
public void unregisterService(org.omg.CORBA.Object
unregisteringService, String serviceName)
                        throws SCA.CF.InvalidObjectReference,

SCA.CF.DomainManagerPackage.UnregisterError
```

This operation is used to remove a service entry from the DomainManager for a specific DeviceManager.

The unregisterService operation removes a service entry from the DomainManager. The unregisterService operation releases(client-side CORBA release) the unregisteringService from the DomainManager. The unregisterService operation, upon the successful unregistration of a Service, writes a log record to a DomainManager's Log, with the log level set to ADMINISTRATIVE_EVENT. The unregisterService operation, upon

unsuccessful unregistration of a Service, write a log record to a DomainManager's Log, with the log level set to FAILURE_ALARM.

Throws:

The unregisterService operation raises the CF InvalidObjectReference exception when the input parameter contains an invalid reference to a Service interface.

1.3.14 SCA.CFImpl.DeviceManagerOperationsImpl

1.3.14.1 Description

The DeviceManager interface is used to manage a set of logical Devices and services. The interface for a DeviceManager is based upon its attributes, which are:

1. Device Configuration Profile - a mapping of physical device locations to meaningful labels (e.g., audio1, serial1, etc.), along with the Devices and services to be deployed.

2. File System - the FileSystem associated with this DeviceManager.

3. Device Manager Identifier - the instance-unique identifier for this DeviceManager.

4. Device Manager Label - the meaningful name given to this DeviceManager.

5. Registered Devices - a list of Devices that have registered with this DeviceManager.

6. Registered Services - a list of Services that have registered with this DeviceManager.

The DeviceManager upon start up registers itself with a DomainManager. This requirement allows the system to be developed where at a minimum only the DomainManager's component reference needs to be known. A DeviceManager uses the DeviceManager's deviceConfigurationProfile attribute for determining:

1. How to obtain the DomainManager component reference, whether Naming Service is being used or a DomainManager stringified IOR is being used,

2. Services to be deployed for this DeviceManager (for example, log(s))

3. Devices to be created for this DeviceManager (when the DCD deployondevice element is not specified then the DCD componentinstantiation element is deployed on the same hardware device as the DeviceManager),

4. Devices to be deployed on (executing on) another Device,

5. Devices to be aggregated to another Device,

6. Mount point names for FileSystems,

7. The DCD's id attribute for the DeviceManager's identifier attribute value, and

8. The DCD's name attribute for the DeviceManager's label attribute value.

The DeviceManager creates FileSystem components implementing the FileSystem interface for each OS file system. If multiple FileSystems are to be created, the DeviceManager mounts created FileSystems to a FileManager component (widened to a FileSystem through the FileSys attribute). Each mounted FileSystem name must be unique within the DeviceManager.

The DeviceManager supplies execute operation parameters (IDs and format values) for a Device consisting of:

a. DeviceManager IOR - The ID is "DEVICE_MGR_IOR" and the value is a string that is the DeviceManager stringified IOR.

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

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

d. 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 usageelement is specified.

e. 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 is a composite part of another componentinstantiation element.

f. The execute ("execparam") properties as specified in the DCD for a componentinstantiation element. The DeviceManager shall pass the componentinstantiation element "execparam" properties that have values as parameters.

The DeviceManager passes "execparam" parameters' IDs and values as string values.The DeviceManager uses the componentinstantiation element's SPD implementation code's stacksize and priority elements, when specified, for the execute options parameters.

The DeviceManager initializes and configures logical Devices that are started by the DeviceManager after they have registered with the DeviceManager.

The DeviceManager configures a DCD's componentinstantiation element provided the componentinstantiation element has "configure" readwrite or writeonly properties with values.

If a Service is deployed by the DeviceManager, the DeviceManager supplies execute operation parameters (IDs and format values) consisting of:

a. DeviceManager IOR - The ID is "DEVICE_MGR_IOR" and the value is a string that is the DeviceManager stringified IOR.

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

1.3.14.2 Class Diagram

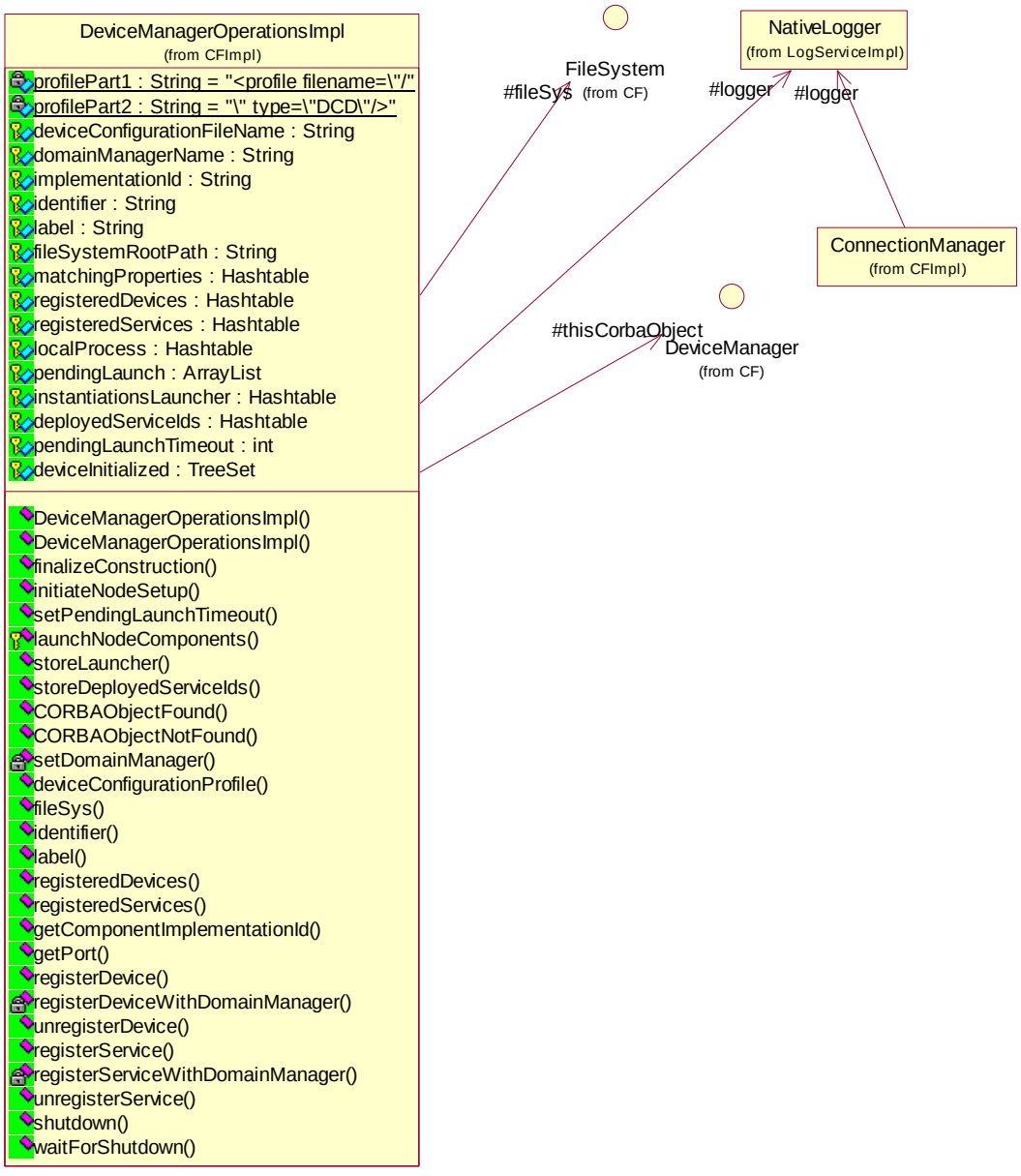

Figure 42 - DomainManagerOperationsImpl Class Diagram

1.3.14.3 Implementation Specific Services

`public` **`DeviceManagerOperationsImpl`**`()`

Default Constructor. This invokes the complete constructor as follows:

1. deviceConfigurationFileName = "Not Specified!"
2. domainManagerName = "Not Specified!"
3. implementationId = "Not Specified!"
4. identifier = "Not Specified!"
5. usageName = "Not Specified!"
6. fileSystemRootPath = "/"
7. matchingProperties = null

`public` **`DeviceManagerOperationsImpl`**`(String deviceConfigurationFileName, String domainManagerName, String implementationId, String identifier, String usageName, String fileSystemRootPath, java.util.Hashtable matchingProperties,` `SCA.LogServiceImpl.NativeLogger` `logger)`

Complete constructor.

Parameters:

deviceConfigurationFileName XML DCD file name (SCA filename)
domainManagerName XML DCD
implementationId XML UUID
identifier XML deviceconfiguration.id
usageName XML deviceconfiguration.name
fileSystemRootPath Native Directory path used for the creation of the DeviceManager's FileSystem (the DCD file must be located in this directory).
matchingProperties Properties of the native device used by the DeviceManager to launch all the local components

`public void` **`CORBAObjectFound`**`(org.omg.CORBA.Object obj)`

This method is invoked by a CORBANameServiceRegistrar search Thread when a reference to the DomainManager is found. Its input arguments supplies the CORBA object reference found. This method invokes the setDomainManager method to initiate registration to the found DomainManager.

Parameters:

obj reference to the CORBA object found

`public void` **`CORBAObjectNotFound`**`(String reason)`

This method is invoked by a CORBANameServiceRegistrar search Thread when a reference to the DomainManager cannot be found after a certain time. Its input argument supplies a reason that might indicate a CORBA problem.

Parameters:
reason a message indicating why the object was not found

`public String` **`deviceConfigurationProfile`**`()`

The readonly deviceConfigurationProfile attribute contains the DeviceManager's profile. The readonly deviceConfigurationProfile attribute contains either a profile element with a file reference to the DeviceManager's device configuration (DCD) profile or the XML for the DeviceManager's device configuration (DCD) profile. Files referenced within the profile are obtained from a FileSystem.

Returns: device configuration profile (DCD) file name

`public` `SCA.CF.FileSystem` **`fileSys`**`()`

The readonly fileSys attribute contains the FileSystem associated with this DeviceManager or a nil CORBA object reference if no FileSystem is associated with this DeviceManager.

Returns: Filesystem used by this DeviceManager

```
public void finalizeConstruction(SCA.CF.DeviceManager deviceManager,
org.omg.PortableServer.POA poa)
                          throws java.lang.InstantiationException,
                                 org.omg.CORBA.ORBPackage.InvalidName
```

This method is used to give an DeviceManager a reference to the CORBA POA servant that represents it so that the DeviceManager can register to its DomainManager. It also gives the Device a reference to the poa that was used to activate its CORBA object such that it may deactivate itself. This method should be invoked immediately after instantiation of the DeviceManager.

This method makes use the utility class CORBANameServiceRegistar to get a reference to the DomainManager (this utility class is described in the following sections). For the moment, this implementation of the DeviceManager looks for a DomainManager named "domainManager" and waits for 60 seconds before giving up. However, this waiting is done on a separate Thread of execution and the DeviceManager is (asynchronously) given a reference to the DomainManager if its found.

This method raises an InstantiationException when the poa is invalid. In

addition, it raises InvalidName when a reference to the CORBA name service cannot be obtained (generally, this happens when the naming service isn't started).

Parameters:

deviceManager a reference to the CORBA object representing this implementation instance of a DeviceManager. This is du to the use of POA tie classes.

poa reference to the poa used to activate the CORBA object representing this implementation instance of a DeviceManager

Throws:

InstantiationException when the poa is invalid or an invalid fileSystemRootPath name was used at construction time or when a CORBA Exception occurs during instantiation of the Root FileSystem

InvalidName When no naming service is running

`public String` **`getComponentImplementationId`**`(String componentInstantiationId)`

This operation returns the SPD implementation ID that the DeviceManager interface used to create a component.

Parameters:

componentInstantiationId instantiation id of the component for which the SPD implementation ID is being requested

Returns: The getComponentImplementationId operation returns the SPD implementation ID attribute that matches the SPD implementation which was used to create the component as identified by the input componentInstantiation ID parameter. The getComponentImplementationId operation returns an empty string when the componentInstantiation is invalid.

`public org.omg.CORBA.Object` **`getPort`**`(String name) throws SCA.CF.PortSupplierPackage.UnknownPort`

The getPort operation provides a mechanism to obtain a specific consumer or producer Port. A PortSupplier may contain zero-to-many consumer and producer port components. The exact number is specified in the component's Software Profile SCD (section Error! Reference source not found.). These Ports can be either push or pull types. Multiple input and/or output ports provide flexibility for PortSuppliers that must manage varying priority levels and categories of incoming and outgoing messages,

provide multi-threaded message handling, or other special message processing.
Parameters:
name for the port object to be returned
Returns: returns the object reference to the named port as stated in the component's SCD. The getPort operation returns the CORBA object reference that is associated with the input port name.
Throws:
UnknownPort if the port name is invalid

`public String identifier()`

The readonly identifier attribute contains the instance-unique identifier for a DeviceManager.
Returns: identifier used by this DeviceManager (e.g. deviceconfiguration.id)

`public void initiateNodeSetup() throws util.domainprofile.launchers.LaunchError, SCA.CF.InvalidProfile`

This method is invoke by the DeviceManager Server to initiate node setup. It parses the Device Configuration Descriptor (DCD) xml file in order to find all the components that need to be launch then call method launchNodeComponents().
Throws:
LaunchError when the DeviceManager is not capable of launching any components because on major DeviceManager problem.

`public String label()`

The readonly label attribute contains the DeviceManager's label. The label attribute is the meaningful name given to a DeviceManager.
Returns: identifier used by this DeviceManager (e.g. deviceconfiguration.name)

`protected void launchNodeComponents() throws util.domainprofile.launchers.LaunchError, SCA.CF.InvalidProfile`

This method is used to launch all the components that are describe in the DCD. It launches components that have no device dependency (aggregate device and/or deploy on device) first. The components that have device

dependencies are added to a pending launch list. Periodically the method loop through the pending launch list to look for the required device that might have registered since the last check. If all device dependencies are fulfilled the component is launched and remove from the the pending list. The loop breaks out when the pending list is empty or the search for missing device has gone for too long (pendingLaunchDuration).

A failure log message is generate when some pending components could not be launched.

Throws:
LaunchError when the DeviceManager is not capable of launching any components because on major DeviceManager problem.

```
public void registerDevice(SCA.CF.Device registeringDevice)
                    throws SCA.CF.InvalidObjectReference
```

This operation provides the mechanism to register a Device with a DeviceManager.

The registerDevice operation adds the input registeringDevice to the DeviceManager's registeredDevices attribute when the input registeringDevice does not already exist in the registeredDevices attribute. The registeringDevice is ignored when duplicated.

The registerDevice operation registers the registeringDevice with the DomainManager when the DeviceManager has already registered to the DomainManager and the registeringDevice has been successfully added to the DeviceManager's registeredDevices attribute.

The registerDevice operation writes a log record with the log level set to FAILURE_ALARM, upon unsuccessful registration of a Device to the DeviceManager's registeredDevices.

Parameters:
registeringDevice device to be registered with this DeviceManager and the DomainManager

Throws:
InvalidObjectReference when the input registeredDevice is a nil CORBA object reference

```
public SCA.CF.Device[] registeredDevices()
```

The readonly registeredDevices attribute contains a list of Devices that have registered with this DeviceManager or a sequence length of zero if no Devices have registered with the DeviceManager.
Returns: list of Devices registered with this DeviceManager

```
public SCA.CF.DeviceManagerPackage.ServiceType[] registeredServices()
```

The readonly registeredServices attribute shall contain a list of Services that have registered with this DeviceManager or a sequence length of zero if no Services have registered with the DeviceManager.
Returns: list of Services registered with this DeviceManager

```
public void registerService(org.omg.CORBA.Object registeringService,
String name)
                    throws SCA.CF.InvalidObjectReference
```

This operation provides the mechanism to register a Service with a DeviceManager.

The registerService operation adds the input registeringService to the DeviceManager's registeredServices attribute when the input registeringService does not already exist in the registeredServices attribute. The registeringService is ignored when duplicated.

The registerService operation registers the registeringService with the DomainManager when the DeviceManager has already registered to the DomainManager and the registeringService has been successfully added to the DeviceManager's registeredServices attribute.

The registerService operation writes a log record with the log level set to FAILURE_ALARM, upon unsuccessful registration of a Service to the DeviceManager's registeredServices.

Parameters:
registeringService Service to be registred
name Name of the service
Throws:
InvalidObjectReference when the input registeredService is a nil CORBA object reference

`public void` **`setPendingLaunchTimeout`**`(int timeout)`

This method is used by the DeviceManager Server to specified the period of time that the launchNodeComponent() has to wait for device(s) that are needed by DCD components. Some DCD components that are aggregated device or need to be deployed on other Loadable/Executable Device cannot be launch until the device that they depend on registers with the DeviceManager. The timeout should bet set before initiateNodeSetup() is called.

Parameters:

timeout the period of time the LaunchNodeComponent has to actively look for required device(s) by any components in the Pending launch list.

`public void` **`shutdown`**`()`

This operation provides the mechanism to terminate a DeviceManager.

The shutdown operation unregisters the DeviceManager from the DomainManager.

The shutdown operation performs releaseObject on all of the DeviceManager's registered Devices (DeviceManager's registeredDevices attribute).

The shutdown operation causes the DeviceManager to be unavailable terminated on the OS), when all of the DeviceManager's registered Devices are unregistered from the DeviceManager.

`public void` **`storeDeployedServiceIds`**`(DCDDeployedComponentLauncher launcher)`

Invoked by the a launchNodeComponents method for each successful launch of a deployed Service. The deployedServiceIds table allows to cross reference a service name and its instantiation UUID.

Parameters:

launcher reference to the DCDDeployedComponentLauncher that was used to launch all the instiatiations for the componentPlacement of the service

`public void` **`storeLauncher`**`(DCDComponentPlacement component, ComponentLauncher launcher)`

Invoked by the a launchNodeComponents method for each successful launch of a componentplacement. The instantiationsLauncher table allows to cross reference an instantiation with its launcher.

Parameters:

component reference to the componentplacement which holds 1..n instantiations

launcher reference to the ComponentLauncher that was use launch all the instiatiations for the componentPlacement

```
public void unregisterDevice(SCA.CF.Device registeredDevice)
                      throws SCA.CF.InvalidObjectReference
```

This operation unregisters a Device from a DeviceManager.

The unregisterDevice operation removes the input registeredDevice from the DeviceManager's registeredDevices attribute. The unregisterDevice operation unregisters the input registeredDevice from the DomainManager when the input registeredDevice is registered with the DeviceManager.

The unregisterDevice operation writes a log record with the log level set to FAILURE_ALARM, when it cannot successfully remove a registeredDevice from the DeviceManager's registeredDevices.

Parameters:

registeredDevice Device to be unregistered

Throws:

InvalidObjectReference when the input registeredDevice is a nil CORBA object referenceo or does not exist in the DeviceManager's registeredDevices attribute

```
public void unregisterService(org.omg.CORBA.Object registeredService,
String name)
                       throws SCA.CF.InvalidObjectReference
```

This operation unregisters a Service from a DeviceManager.

The unregisterService operation removes the input registeredService from the DeviceManager's registeredServices attribute. The unregisterService operation unregisters the input registeredService from the DomainManager when the input registeredService is registered with the DeviceManager.

The unregisterService operation writes a log record with the log

level set to FAILURE_ALARM, when it cannot successfully remove a registeredService from the DeviceManager's registeredServices.

Parameters:
registeredService Service to be unregistered
name Name of the Service to be unregistered
Throws:
InvalidObjectReference The unregisterService operation raises the CF InvalidObjectReference when the input registeredService is a nil CORBA object reference or does not exist in the DeviceManager's registeredServices attribute.

`public void` **`waitForShutdown`**`()`

This method is used to wait for the shutdown of this DeviceManager. Any object invoking this method will be blocked until the releaseObject() method is invoked and completes its execution. Typically, this method is used by a DeviceManagerServer to determine when to unregister its DeviceManager from the naming service.

1.3.15 SCA.LogServiceImpl.NativeLogger

1.3.15.1 Description

This class is used to add any message in a local log. A local log can be used to log any message in the meantime a remote SCA logger become available to any component needing to log something. The messages are stored in a list in memory for ease of use and also written in a file as a backup.

1.3.15.2 Class Diagram

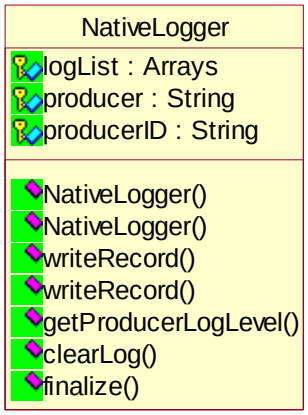

Figure 43 - NativeLogger Class Diagram

1.3.15.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`public` **`NativeLogger`**`(String producerID, String producerName)`

Creates a native logger with an initial log message list of size one.

Parameters:

producerID The unique producer ID dedicated to the producer of the messages of this log.

producerName The name of the producer of the message to put in the log.

`public` **`NativeLogger`**`(int logSize, String producerID, String producerName)`

Creates a native logger.

Parameters:

logSize initial size of the list that will store the messages to be logged.

producerID The unique producer ID dedicated to the producer of the messages of this log.

producerName The name of the producer of the message to put in the log.

`public void` **`clearLog`**`()`

Remove all messages from the logging list and delete the log file.

1.3.16 SCA.LogServiceImpl.LogPort

1.3.16.1 Description

This class is a local logger and is also a port. If an instance of this class is connected to the port of a remote logger the messages that are logged by the producer are written in the remote log. Otherwise, the messages are written in a local log. The messages stored in the local log are transferred automatically when a remote logger is connected to this port.

1.3.16.2 Class Diagram

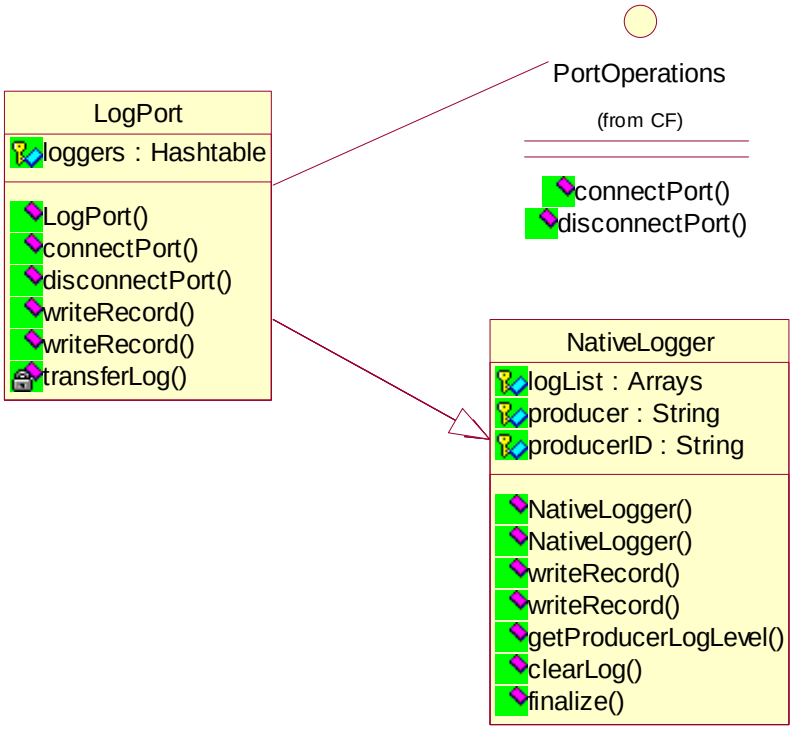

Figure 44 - LogPort Class Diagram

1.3.16.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`public` **`LogPort`**`(String producerID, String producerName)`

Creates a LogPort.

Parameters:

producerID The unique producer ID dedicated to the producer of the messages of this log.

producerName The name of the producer of the message to put in the log.

```
public void connectPort(org.omg.CORBA.Object connection, String
connectionID)
                 throws SCA.CF.PortPackage.InvalidPort,
                        SCA.CF.PortPackage.OccupiedPort
```

Connects this logger to a remote logger. Messages already stored in the log are transmitted to the remote logger.

Parameters:

connection Remote object to connect to this logger (must be a Log).
connectionID Identifier for the connection to create.

Throws:

InvalidPort if the object to connect is not of type Log.
OccupiedPort N/A

```
public void disconnectPort(String connectionID)
                    throws SCA.CF.PortPackage.InvalidPort
```

Unestablish a connection to a remote logger.

Parameters:

connectionID Identifier of the connection to unestablish.

Throws:

InvalidPort if the connection does not exist.

```
public void writeRecord(String message, int level)
```

Writes the message using the default producerID and producer Name

Parameters:

message Message to log.
level Log level identifying the type of message.

```
public void writeRecord(String producerID, String producer, String
message, int level)
```

Writes the message to the local log if no connection exist to a remote logger else write the message to the remote log.

Parameters:

producerID ID of the producer
producer Name of the producer
message Message to log.
level Log level identifying the type of message.

1.3.17 SCA.CFImpl.ConfigurationDoubleProperty

1.3.17.1 Description

This is a specialization of SCA.CF.ConfigurationProperty. It represents a double Configuration property. This Configuration property can be used by any Resource.

1.3.17.2 Class Diagram

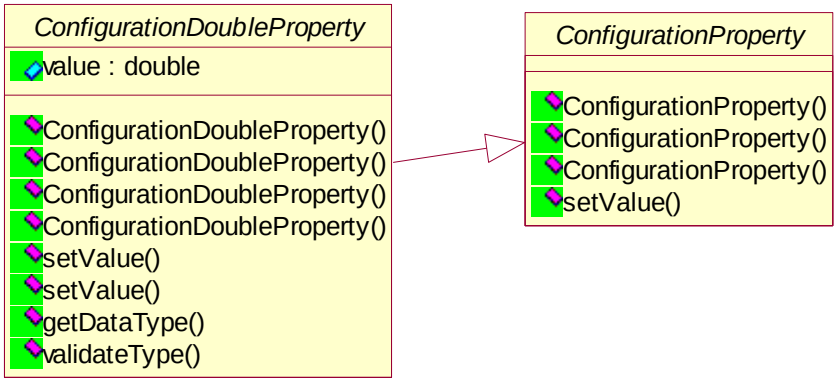

Figure 45 - ConfigurationDoubleProperty Class Diagram

1.3.17.3 Implementation Specific Services

This section contains a description of all the methods that are implementation specific.

`public` **`ConfigurationDoubleProperty`**`()`

Default constructor.This invokes a partial constructor as follows:

1. name = String "Not Specified!"

`public` **`ConfigurationDoubleProperty`**`(String name)`

Partial constructor. This invokes a partial constructor as follows:

1. name = name input argument
2. readOnly = READ_WRITE

Parameters:

name Configuration property name

`public` **`ConfigurationDoubleProperty`**`(String name, boolean readOnly)`

Partial constructor. This invokes a partial constructor as follows:

1. name = name input argument
2. readOnly = readOnly input argument
3. visible = VISIBLE

Parameters:
name Configuration property name
readOnly Type of access. Can be READ_ONLY or READ_WRITE

`public` **`ConfigurationDoubleProperty`**`(String name, boolean readOnly, boolean visible)`

Complete constructor. This invokes a super class constructor as follows:

1. name = name input argument
2. readOnly = readOnly input argument
3. visible = visible input argument

Parameters:
name Configuration property name
readOnly Type of access. Can be READ_ONLY or READ_WRITE
visible Determines if query() supports this property. Can be VISIBLE or HIDDEN

`public` `DataType` **`getDataType`**`()`

Returns this property packaged as an SCA.CF.DataType containing a double CORBA ANY.
Returns: this property packaged as a SCA.CF.DataType

`public boolean` **`setValue`**`(DataType dataType)`

Extracts a double value from the input DataType and invokes setValue(double)

Returns: False when extraction fails. Otherwise, it returns the value returned by setValue(double)

`public abstract boolean` **`setValue`**`(double value)`

Must assign property to 'value' input argument.
Returns: true if assignment succeeded. False otherwise.

`public boolean` **`validateType`**`(DataType dataType)`

Returns: true if the input dataType argument contains a double value