SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION

APPENDIX D-1: PSM - DOCUMENT TYPE DEFINITION (DTD)

01 October 2012
Version: 4.0.1

Prepared by:

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

REVISION SUMMARY

Version	Revision	Date
Next <Draft>	Initial Draft Release	30 November 2010
Candidate Release	Initial Release	27 December 2011
4.0	ICWG Approved Release	28 February 2012
4.0.1	Incorporated transition to JTNC and applied SCA 4.0 Errata Sheet v1.0	01 October 2012

TABLE OF CONTENTS

D-1.1 Scope **10**
D-1.2 Conformance **10**
D-1.3 Conventions **10**
D-1.4 Normative References **10**
D-1.5 Informative References **10**
D-1.6 Software Package Descriptor **10**
D-1.6.1 Software Package 11
D-1.6.1.1 title 12
D-1.6.1.2 author 12
D-1.6.1.3 description 13
D-1.6.1.4 propertyfile 13
D-1.6.1.4.1 localfile 14
D-1.6.1.5 descriptor 14
D-1.6.1.6 implementation 14
D-1.6.1.6.1 description 16
D-1.6.1.6.2 propertyfile 16
D-1.6.1.6.3 code 16
D-1.6.1.6.4 compiler 17
D-1.6.1.6.5 programminglanguage 17
D-1.6.1.6.6 humanlanguage 18
D-1.6.1.6.7 runtime 18
D-1.6.1.6.8 os 18
D-1.6.1.6.9 processor 18
D-1.6.1.6.10 dependency 19
D-1.6.1.6.10.1 softpkgref 19
D-1.6.1.6.10.2 propertyref 20
D-1.6.1.6.11 usesdevice 20
D-1.6.1.6.11.1 propertyref 21
D-1.6.1.7 usesdevice 21
D-1.7 Device Package Descriptor **22**
D-1.7.1 Device Package 22

D-1.7.1.1 title 23
D-1.7.1.2 author 23
D-1.7.1.3 description 23
D-1.7.1.4 hwdeviceregistration 23
D-1.7.1.4.1 propertyfile 24
D-1.7.1.4.2 description 25
D-1.7.1.4.3 manufacturer 25
D-1.7.1.4.4 modelnumber 25
D-1.7.1.4.5 deviceclass 25
D-1.7.1.4.6 childhwdevice 25
D-1.7.1.4.6.1 hwdeviceregistration 26
D-1.7.1.4.6.2 devicepkgref 26
D-1.8 Properties Descriptor 27
D-1.8.1 properties 27
D-1.8.1.1 simple 28
D-1.8.1.1.1 description 29
D-1.8.1.1.2 value 29
D-1.8.1.1.3 units 29
D-1.8.1.1.4 range 29
D-1.8.1.1.5 enumerations 29
D-1.8.1.1.6 kind 30
D-1.8.1.1.7 action 31
D-1.8.1.2 simplesequence 31
D-1.8.1.3 test 32
D-1.8.1.3.1 inputvalue 33
D-1.8.1.3.2 resultvalue 33
D-1.8.1.4 struct 33
D-1.8.1.4.1 configurationkind 34
D-1.8.1.5 structsequence 35
D-1.9 Software Component Descriptor 37
D-1.9.1 softwarecomponent 37
D-1.9.1.1 componentrepid 38
D-1.9.1.2 componenttype 38

D-1.9.1.3 componentfeatures 38
D-1.9.1.3.1 supportsinterface 39
D-1.9.1.3.2 ports 39
D-1.9.1.4 interfaces 40
D-1.9.1.5 propertyfile 41
D-1.10 Software Assembly Descriptor **42**
D-1.10.1 softwareassembly 42
D-1.10.1.1description 43
D-1.10.1.2componentfiles 44
D-1.10.1.2.1 componentfile 44
D-1.10.1.3partitioning 44
D-1.10.1.3.1 componentplacement 45
D-1.10.1.3.1.1 componentfileref 45
D-1.10.1.3.1.2 componentinstantiation 45
D-1.10.1.3.1.2.1 componentproperties 46
D-1.10.1.3.1.2.2 deploymentdependencies 47
D-1.10.1.3.1.2.3 findcomponent 47
D-1.10.1.3.1.2.3.1 componentfactoryref 48
D-1.10.1.3.1.2.3.1.1 componentfactoryproperties 48
D-1.10.1.3.2 hostcollocation 50
D-1.10.1.3.2.1 componentplacement 50
D-1.10.1.3.3 assemblyplacement 51
D-1.10.1.3.3.1 assemblyinstantiation 51
D-1.10.1.3.3.1.1 componentproperties 52
D-1.10.1.3.3.1.2 deviceassignments 53
D-1.10.1.3.3.1.3 deploymentdependencies 53
D-1.10.1.4deploymentdependencies 53
D-1.10.1.5assemblycontroller 55
D-1.10.1.6connections 55
D-1.10.1.6.1 connectinterface 55
D-1.10.1.6.1.1 *usesport* 56
D-1.10.1.6.1.1.1 identifier 57
D-1.10.1.6.1.1.4 devicethatloadedthiscomponentref 58
D-1.10.1.6.1.1.5 deviceusedbythiscomponentref 58
D-1.10.1.6.1.1.6 domainfinder 58

D-1.10.1.6.1.2 providesport 59
D-1.10.1.6.1.2.1 identifier 60
D-1.10.1.6.1.2.2 componentinstantiationref 60
D-1.10.1.6.1.2.3 assemblyinstantiationref 61
D-1.10.1.6.1.2.4 devicethatloadedthiscomponentref 61
D-1.10.1.6.1.2.5 deviceusedbythiscomponentref 61
D-1.10.1.6.1.2.6 domainfinder 61
D-1.10.1.6.1.3 componentsupportedinterface 61
D-1.10.1.6.1.3.3 domainfinder 62
D-1.10.1.7externalports 62
D-1.10.1.8deploymentprefs 64
D-1.11 Device Configuration Descriptor 64
D-1.11.1 deviceconfiguration 64
D-1.11.1.1description 65
D-1.11.1.2devicemanagersoftpkg 65
D-1.11.1.3componentfiles 66
D-1.11.1.4partitioning 66
D-1.11.1.4.1 componentplacement 66
D-1.11.1.4.1.1 *componentfileref* 67
D-1.11.1.4.1.2 *deployondevice* 67
D-1.11.1.4.1.3 *compositepartofdevice* 67
D-1.11.1.4.1.4 *devicepkgfile* 67
D-1.11.1.4.1.4.1 *localfile* 67
D-1.11.1.4.1.5 componentinstantiation 67
D-1.11.1.4.1.5.1 *componentproperties* 68
D-1.11.1.4.1.5.2 componentfactoryref 69
D-1.11.1.4.1.5.2.1 componentfactoryproperties 70
D-1.11.1.5connections 72
D-1.11.1.6domainmanager 72
D-1.11.1.7filesystemnames 72
D-1.12 Domain Manager Configuration Descriptor 73
D-1.12.1 domainmanagerconfiguration 73
D-1.12.1.1description 73
D-1.12.1.2domainmanagersoftpkg 74
D-1.12.1.3deploymentlayout 74

D-1.12.1.4services 74
D-1.12.1.4.1 service 75
D-1.13 Platform Deployment Descriptor 76
D-1.13.1 deploymentplatform 76
D-1.13.1.1description 76
D-1.13.1.2channel 77
D-1.13.1.2.1 devicelist 77
D-1.13.1.2.1.1 deviceref 77
D-1.13.1.2.2 servicelist 78
D-1.13.1.2.2.1 serviceref 78
D-1.14 Application Deployment Descriptor 79
D-1.14.1 deploymentprecedence 79
D-1.14.1.1description 79
D-1.14.1.2deploymentoption 79
D-1.14.1.2.1 description 80
D-1.14.1.2.2 channelref 80
D-1.15 Attachments 81

LIST OF FIGURES

Figure 1: *softpkg* Element Relationships 11
Figure 2: *author* Element Relationships 13
Figure 3: *implementation* Element Relationships 15
Figure 4: *code* Element Relationships 17
Figure 5: *dependency* Element Relationships 19
Figure 6: *softpkgref* Element Relationships 20
Figure 7: *devicepkg* Element Relationships 22
Figure 8: *hwdeviceregistration* Element Relationships 24
Figure 9: *childhwdevice* Element Relationships 26
Figure 10: *properties* Element Relationships 27
Figure 11: *simple* Element Relationships 28
Figure 12: *simplesequence* Element Relationships 32
Figure 13: *test* Element Relationships 33
Figure 14: *struct* Element Relationships 34
Figure 15: *structsequence* Element Relationships 35
Figure 16: *softwarecomponent* Element Relationships 37
Figure 17: *componentfeatures* Element Relationships 38
Figure 18: *ports* Element Relationships 40
Figure 19: *softwareassembly* Element Relationships 43
Figure 20: *partitioning* Element Relationships 44
Figure 21: *componentplacement* Element Relationships 45
Figure 22: *componentinstantiation* Element Relationships 46
Figure 23: *findcomponent* Element Relationships 47
Figure 24: *componentfactoryref* Element Relationships 48
Figure 25: *componentfactoryproperties* Element Relationships 49
Figure 26: assemblyplacement Element Relationships 51
Figure 27: *assemblyinstantiation* Element Relationships 52
Figure 28: *componentproperties* Element Relationships 53
Figure 29: *deploymentdependencies* Element Relationships 54
Figure 30: *assemblycontroller* Element Relationships 55
Figure 31: *connectinterface* Element Relationships 56

Figure 32: *usesport* Element Relationships 57
Figure 33: *providesport* Element Relationships 60
Figure 34: *componentsupportedinterface* Element Relationships 61
Figure 35: *port* Element Relationships 63
Figure 36: *deviceconfiguration* Element Relationships 65
Figure 37: *componentplacement* Element Relationships 66
Figure 38: *componentinstantiation* Element Relationships 68
Figure 39: *componentproperties* Element Relationships 69
Figure 40: *componentfactoryref* Element Relationships 70
Figure 41: *componentfactoryproperties* Element Relationships 71
Figure 42: *domainmanagerconfiguration* Element Relationships 73
Figure 43: *services* Element Relationships 74
Figure 44: *service* Element Relationships 75
Figure 45: *deploymentplatform* Element Relationships 76
Figure 46: *channel Element* Relationships 77
Figure 47: *deploymentprecedence* Element Relationships 79
Figure 48: *deploymentoption* Element Relationships 80

APPENDIX D-1 PSM - DOCUMENT TYPE DEFINITION

D-1.1 SCOPE

This appendix defines the SCA Domain Profiles using XML Document Type Definition (DTD) files.

SCA501 DTD files are installed in the domain and shall have ".dtd" as their filename extension. SCA502 All XML files shall have as the first two lines as an XML declaration (?xml) and a document type declaration (!DOCTYPE). The XML declaration specifies the XML version and whether the document is standalone. The document type declaration specifies the DTD for the document. Example declarations are as follows:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE softwareassembly SYSTEM "softwareassembly.dtd">
```

D-1.2 CONFORMANCE

See SCA Appendix D.

D-1.3 CONVENTIONS

N/A

D-1.4 NORMATIVE REFERENCES

The following documents contain provisions or requirements which by reference constitute requirements of this appendix.

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

D-1.5 INFORMATIVE REFERENCES

N/A

D-1.6 SOFTWARE PACKAGE DESCRIPTOR

The Software Package Descriptor (SPD) is used at deployment time to load a component and its various implementations. The information contained in the SPD will provide the basis for the domain management function to manage the component within the SCA architecture.

The SPD may contain various implementations of any given component. Within the specification of an SPD several other files are referenced including a component level *propertyfile* and a Software Component Descriptor (SCD) file. Within any given implementation there may be additional *propertyfiles*.

SCA503 A Software Package Descriptor file shall have a ".spd.xml" extension.

D-1.6.1 Software Package

The *softpkg* element (**Figure 1**) indicates an SPD definition. The *softpkg* id uniquely identifies the package. The *softpkg id* attribute definition guarantees uniqueness within an XML document, however an implementation specific approach must be utilized to maintain uniqueness within a Domain Profile. The *name* attribute is a user-friendly label for the *softpkg* element. The *type* attribute indicates whether or not the component implementation is SCA compliant. All files referenced by a Software Package are located in the same directory as the SPD file or a directory that is relative to the directory where the SPD file is located.

A software package requires at least one implementation. An implementation is a monolithic loadable/executable artifact. A single monolithic loadable/executable artifact is for a General Purpose Processor, Digital Signal Processor or Field Gate Array Processor.

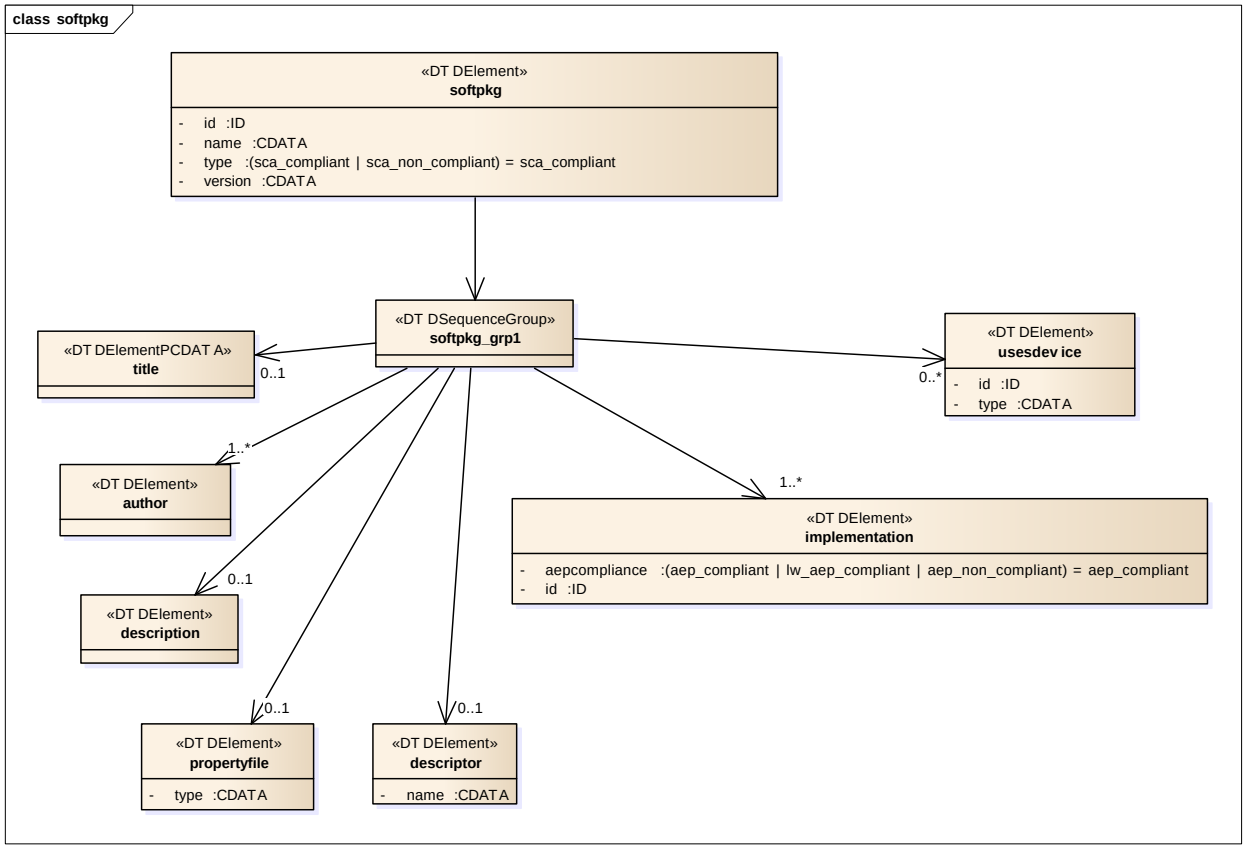


Figure 1: *softpkg* Element Relationships

The set of properties to be used for a Software Package come from the union of these properties sources using the following precedence order:

1. SPD Implementation Properties
2. SPD level properties
3. SCD properties

Any duplicate properties having the same *id* are ignored. Duplicated properties must be the same property type, only the value can be over-ridden. The implementation properties are only used for the initial configuration and creation of a component by an ApplicationFactoryComponent and cannot be referenced by a Software Assembly Descriptor (SAD) *componentinstantiation*, *componentproperties* or *componentfactoryproperties* element.

```
<!ELEMENT softpkg
     ( title?
     , author+
     , description?
     , propertyfile?
     , descriptor?
     , implementation+
     , usesdevice*
)>
<!ATTLIST softpkg
id        ID        #REQUIRED
name      CDATA     #REQUIRED
type (sca_compliant | sca_non_compliant) "sca_compliant"
version   CDATA     #IMPLIED >
```

D-1.6.1.1 title

The *title* element is used for indicating a title for the software component being installed in accordance with the *softpkg* element.

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

D-1.6.1.2 author

The *author* element (see **Figure 2**) is used to indicate the name of the person, the company, and the web page of the developer producing the component being installed into the system.

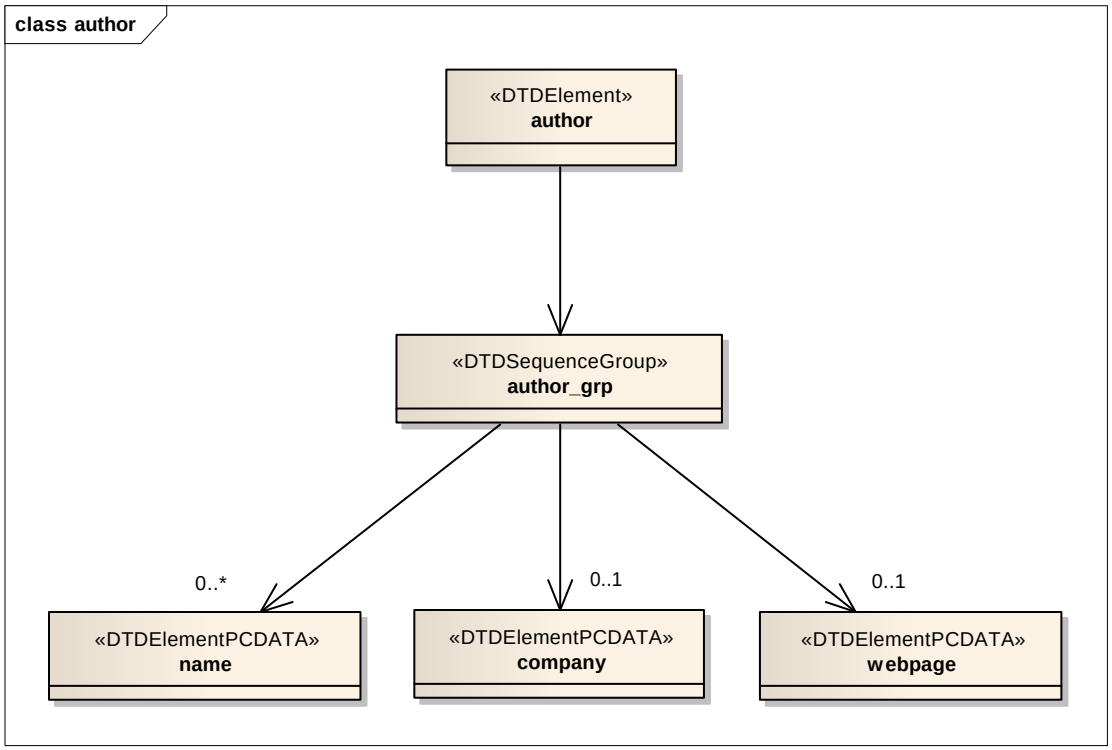


Figure 2: *author* Element Relationships

```
<!ELEMENT author
     ( name*
     , company?
     , webpage?
)>
<!ELEMENT name      (#PCDATA)>
<!ELEMENT company   (#PCDATA)>
<!ELEMENT webpage   (#PCDATA)>
```

D-1.6.1.3 description

The *description* element is used to describe any pertinent information about the software component being delivered to the system.

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

D-1.6.1.4 propertyfile

The *propertyfile* element is used to indicate the local filename of the Property Descriptor (PRF) file associated with the Software Package. The intent of the *propertyfile* will be to provide the definition of *properties* elements common to all component implementations being deployed in accordance with the Software Package (*softpkg*). PRF files may also contain *properties* elements that are used in definition of command and control id value pairs used by the SCA *Resource* configure() and query() operations. The format of the *properties* element is described in the PRF (Section D-1.8.1).

```
<!ELEMENT propertyfile
     (  localfile
)>
<!ATTLIST propertyfile
type      CDATA     #IMPLIED>
```

D-1.6.1.4.1 localfile

The *localfile* element is used to reference a file in the same directory as the SPD file or a directory that is relative to the directory where the SPD file is located. When the *name* attribute is a simple name, the file exists in the same directory as the SPD file. A relative directory indication begins either with "../" meaning parent directory and "./" means current directory in the *name* attribute. Multiple "../" and directory names can follow the initial "../" in the *name* attribute. All *name* attributes must have a simple name at the end of the file name.

```
<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
name CDATA     #REQUIRED>
```

D-1.6.1.5 descriptor

The *descriptor* element points to the local filename of the SCD file used to document the interface information for the component being delivered to the system. In the case of an SCA component, the SCD will contain information about three aspects of the component (the component type, message ports, and interface definitions). The SCD file is optional (see section D-1.9 on SCD file).

```
<!ELEMENT descriptor
     (localfile
)>
<!ATTLIST descriptor
     name      CDATA     #IMPLIED>
```

D-1.6.1.6 implementation

The *implementation* element (see **Figure 3**) contains descriptive information about the particular implementation template for a software component contained in the *softpkg* element. The *implementation* element is intended to allow multiple component templates to be delivered to the system in one Software Package. Each *implementation* element is intended to allow the same component to support different types of processors, operating systems, etc. The *implementation* element will also allow definition of implementation-dependent properties for use in *Device*, *Application*, or *Resource* creation. The *implementation* element's *id* attribute uniquely identifies a specific implementation of the component and an implementation specific approach is required to maintain uniqueness within a Domain Profile. The *compiler*, *programminglanguage*, *humanlanguage*, *os*, *processor*, and *runtime* elements are optional dependency elements.

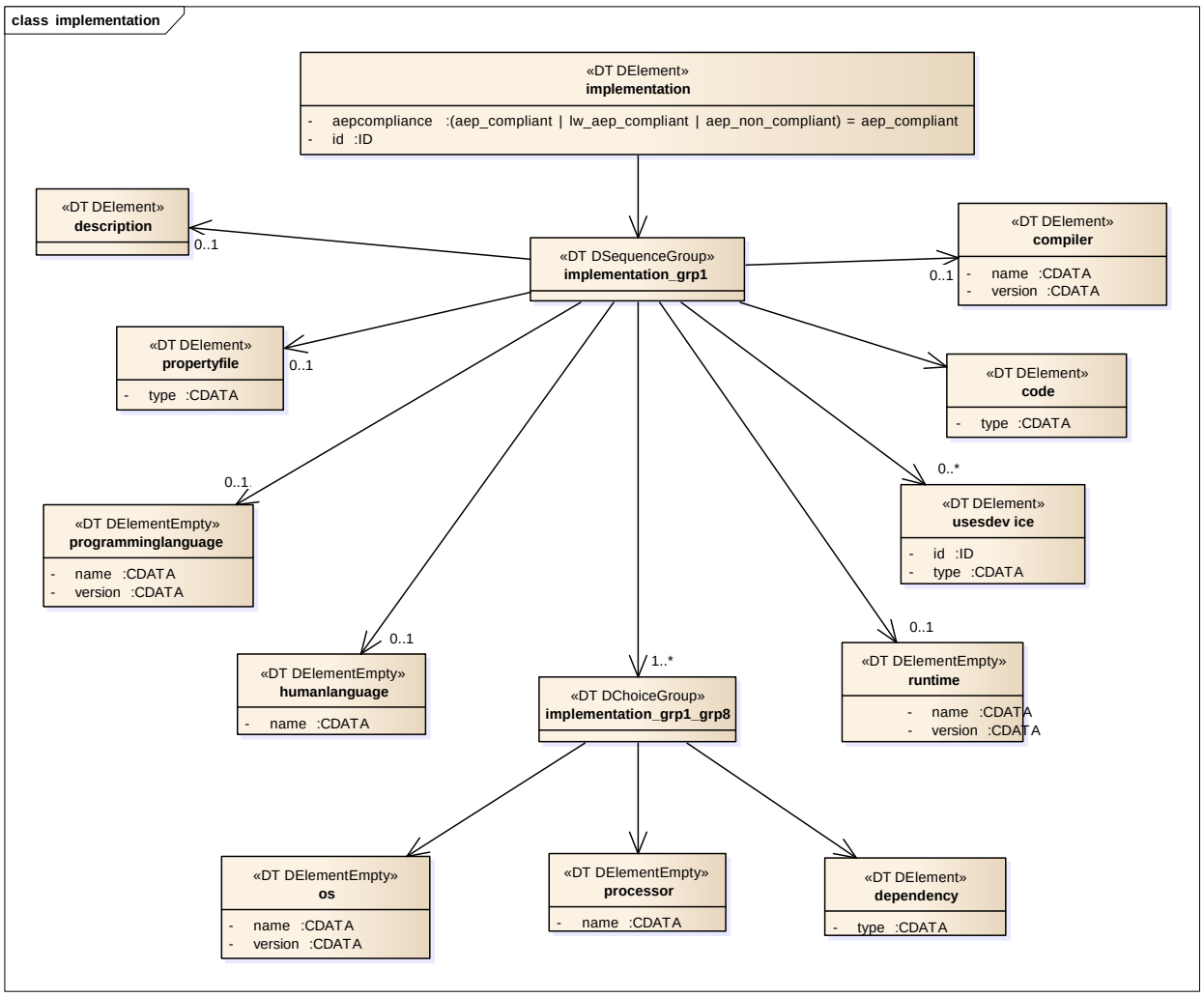


Figure 3: *implementation* Element Relationships

```
<!ELEMENT  implementation
     ( description?
     , propertyfile?
     , code
     , compiler?
     , programminglanguage?
     , humanlanguage?
     , runtime?
     , ( os | processor | dependency )+
     , usesdevice*
     )>
<!ATTLIST implementation
     id   ID   #REQUIRED
     aepcompliance (aep_compliant | lw_aep_compliant |
     aep_non_compliant) “aep_compliant”>
```

D-1.6.1.6.1 description

The *description* element is used to describe any pertinent information about the software component implementation that the software developer wishes to document within the software package profile.

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

D-1.6.1.6.2 propertyfile

The *propertyfile* element is used to indicate the local filename of the PRF file associated with this component package described by the *implementation* element. Although the SCA does not restrict the specific use of the PRF file based on context, it is intended within the *implementation* element to provide component implementation specific *properties* elements for use in command and control id value pair settings to the *Resource* configure() and query() interfaces. See section D-1.8.1 on the description of the *properties* element format in the PRF.

```
<!ELEMENT propertyfile
     (localfile
)>
<!ATTLIST propertyfile
     type      CDATA     #IMPLIED>
```

D-1.6.1.6.3 code

The code element (see **Figure 4**) is used to indicate the local filename of the code that is described by the softpkg element, for a specific implementation of the software component. The *stacksize* and *priority* are options parameters used by the *ExecutableDevice execute* operation. Data types for the values of these options are unsigned long. The *type* attribute for the *code* element will also indicate the type of file being delivered to the system. The *entrypoint* element provides the means for providing the name of the entry point of the component being delivered. The valid values for the *type* attribute are: “Executable”, “KernelModule”, “SharedLibrary”, and “Driver.”

The meaning of the code *type* attribute:

1. Executable means to use *LoadableDeviceObject*::*load* and *ExecutableDevice*::*execute* operations. This is a “main” process.
2. Driver and Kernel Module means load only.
3. SharedLibrary means dynamic linking.
4. Without a code entrypoint element means load only.
5. With a code entrypoint element means load and *ExecutableDevice*::*execute*.

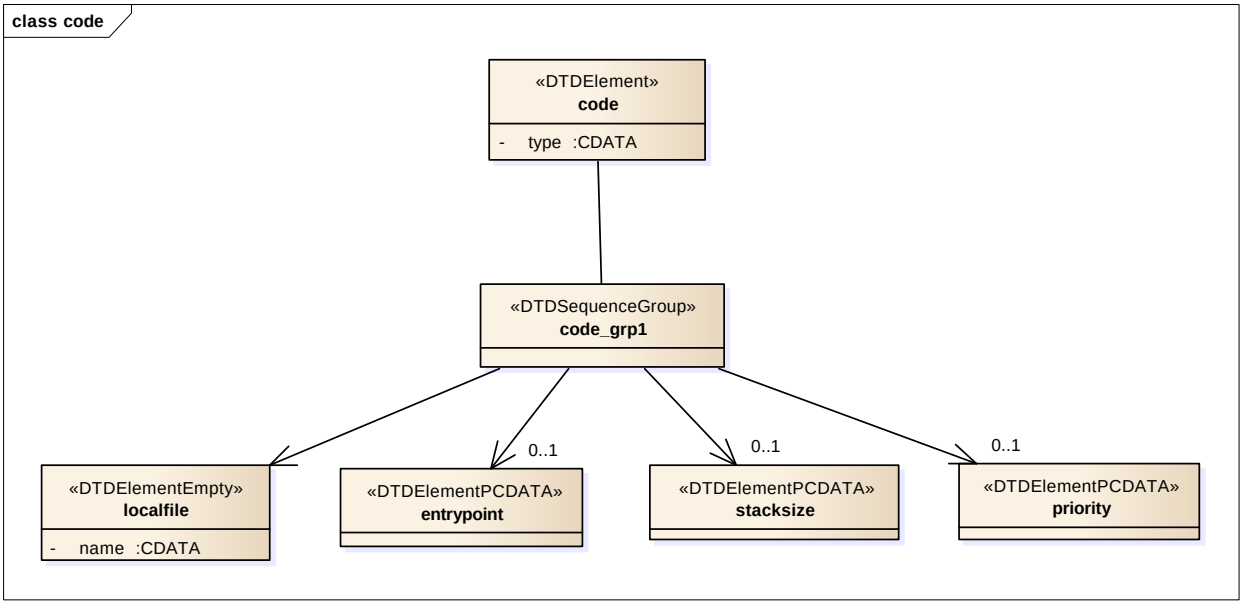


Figure 4: *code* Element Relationships

```
<!ELEMENT code
     ( localfile
     , entrypoint?
     , stacksize?
     , priority?
     )>
<!ATTLIST code
     type CDATA     #IMPLIED>

<!ELEMENT entrypoint (#PCDATA)>

<!ELEMENT stacksize (#PCDATA)>

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

D-1.6.1.6.4 compiler

The *compiler* element is used to indicate the compiler used to build the software component being described by the *softpkg* element. The required *name* attribute will specify the name of the compiler used, and the *version* attribute will contain the compiler version.

```
<!ELEMENT compiler EMPTY>
<!ATTLIST compiler
     name       CDATA     #REQUIRED
     version    CDATA     #IMPLIED>
```

D-1.6.1.6.5 programminglanguage

The *programminglanguage* element is used to indicate the type of programming language used to build the component implementation. The required *name* attribute will specify a language such as “c”, “c++”, or “java”.

```
<!ELEMENT programminglanguage EMPTY>
<!ATTLIST programminglanguage
     name      CDATA     #REQUIRED
     version   CDATA     #IMPLIED>
```

D-1.6.1.6.6 humanlanguage

The *humanlanguage* element is used to indicate the human language for which the software component was developed.

```
<!ELEMENT humanlanguage EMPTY>
<!ATTLIST humanlanguage
     name CDATA     #REQUIRED>
```

D-1.6.1.6.7 runtime

The *runtime* element specifies a runtime required by a component implementation. An example of the runtime is a Java VM.

```
<!ELEMENT runtime EMPTY>
<!ATTLIST runtime
     name      CDATA     #REQUIRED
     version   CDATA     #IMPLIED>
```

D-1.6.1.6.8 os

The *os* element is used to indicate the operating system on which the software component is capable of operating. The required *name* attribute will indicate the name of the operating system and the *version* attribute will contain the operating system. The *os* attributes will be defined in a property file as an allocation property of string type and with names os_name and os_version and with an *action* element value other than "external". The *os* element is automatically interpreted as a dependency and compared against allocation properties with names of *os*_name and *os*_version. The *os_name* attribute allocation property is defined in Attachment 1 to this appendix.

```
<!ELEMENT os EMPTY>
<!ATTLIST os
     name      CDATA     #REQUIRED
     version   CDATA     #IMPLIED>
```

D-1.6.1.6.9 processor

The *processor* element is used to indicate the processor and/or processor family on which this software component will operate. The processor *name* attribute will be defined in a property file as an allocation property of string type and with a name of processor_name and with an *action* element value other than "external". The *processor* element is automatically interpreted as a dependency and compared against an allocation property with a name of processor_name. The *processor_name* attribute allocation property is defined in Attachment 1 to this appendix.

```
<!ELEMENT processor EMPTY>
<!ATTLIST processor
     name CDATA     #REQUIRED>
```

D-1.6.1.6.10dependency

The *dependency* element (see **Figure 5**) is used to indicate the dependent relationships between the components being delivered and other components and devices, in an SCA compliant system. The *softpkgref* element is used to specify a Software Package file that must be resident within the system for the component, described by this *softpkg* element, to load without errors. The *propertyref* will reference a specific allocation property, using a unique identifier, and provide the value that will be used by a ComponentBaseDevice capacity model.

A DomainManagerComponent and DeviceManagerComponent will use these dependency definitions to assure that components and devices that are necessary for proper operation of the implementation are present and available. The *type* attribute is descriptive information indicating the type of dependency.

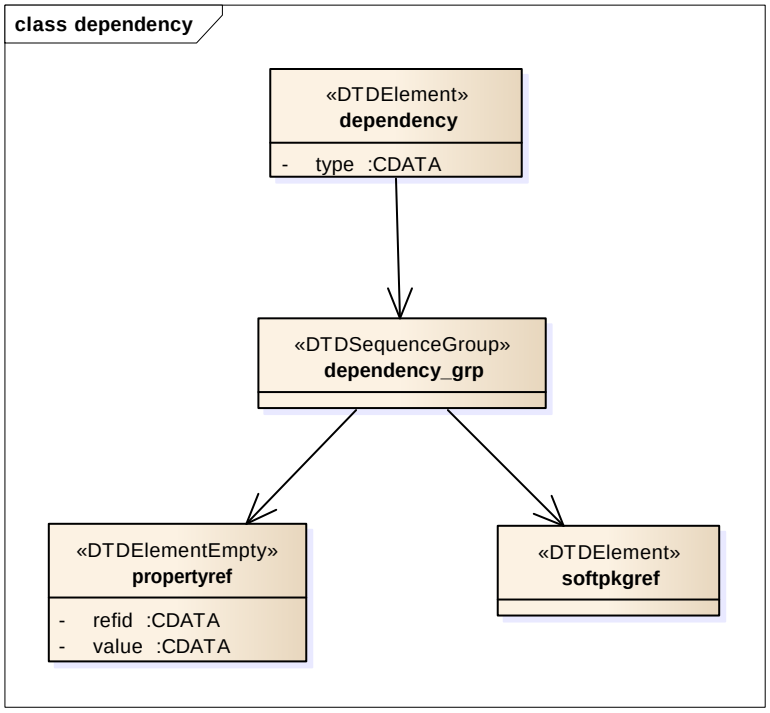


Figure 5: *dependency* Element Relationships

```
<!ELEMENT dependency
     ( softpkgref | propertyref )>
<!ATTLIST dependency
     type CDATA     #REQUIRED>
```

D-1.6.1.6.10.1 softpkgref

The *softpkgref* element (see **Figure 6**) refers to a *softpkg* element contained in another SPD file and indicates a file-load dependency on that file. The other file is referenced by the *localfile* element. An optional *implref* element refers to a particular implementation-unique identifier, within the SPD of the other file.

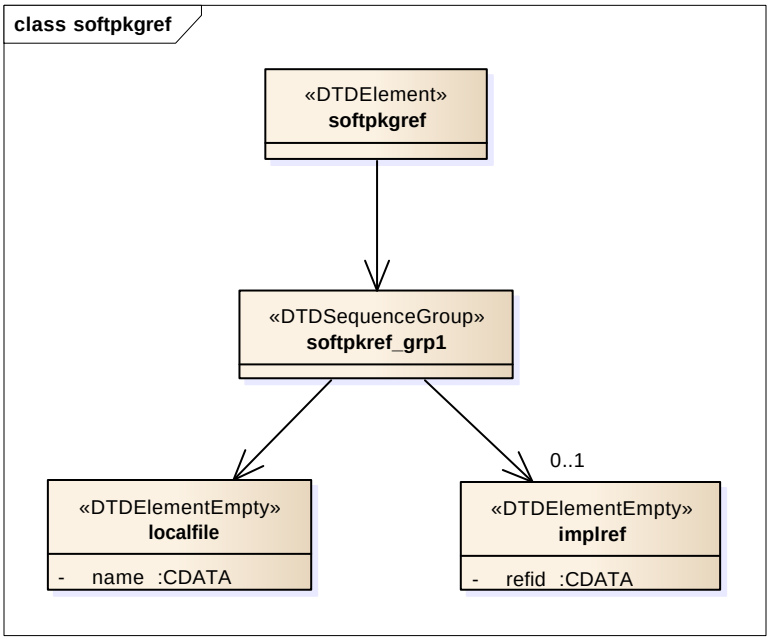


Figure 6: *softpkgref* Element Relationships

```
<!ELEMENT softpkgref
     ( localfile
     , implref?
     )>

<!ELEMENT implref EMPTY>
<!ATTLIST implref
     refid     CDATA     #REQUIRED>
```

D-1.6.1.6.10.2 propertyref

The *propertyref* element is used to indicate a unique *refid* attribute that references a *simple* allocation property, defined in the package, and a property *value* attribute used by the domain Management function to perform the dependency check.

```
<!ELEMENT propertyref EMPTY>
<!ATTLIST propertyref
     refid     CDATA     #REQUIRED
     value     CDATA     #REQUIRED>
```

D-1.6.1.6.11usesdevice

The *usesdevice* element describes any “uses” relationships this component has with a device in the system. The *propertyref* element references allocation properties, which indicate the ComponentBaseDevice to be used, and/or the capacity needed from the ComponentBaseDevice to be used.

```
<!ELEMENT  usesdevice
     ( propertyref+ )>
<!ATTLIST usesdevice
     id   ID        #REQUIRED
     type CDATA          #REQUIRED>
```

D-1.6.1.6.11.1 propertyref

See D-1.6.1.6.10.2 for a definition of the *propertyref* element.

D-1.6.1.7 usesdevice

See D-1.6.1.6.11 for a definition of the *usesdevice* element.

D-1.7 DEVICE PACKAGE DESCRIPTOR

The SCA Device Package Descriptor (DPD) is the part of a Device Profile that contains hardware device Registration attributes, which are typically used by a Human Computer Interface application to display information about the device(s) resident in an SCA-compliant radio system. DPD information is intended to provide hardware configuration and revision information to a radio operator or to radio maintenance personnel. A DPD may be used to describe a single hardware element residing in a radio or it may be used to describe the complete hardware structure of a radio.

SCA504 A Device Package Descriptor File shall have a ".dpd.xml" extension.

D-1.7.1 Device Package

The *devicepkg* element (see **Figure 7**) is the root element of the DPD. The *devicepkg id* attribute uniquely identifies the package and an implementation specific approach is required to maintain uniqueness within a Domain Profile. The *version* attribute specifies the version of the *devicepkg*. The format of the version string is numerical major and minor version numbers separated by commas (e.g., "1,0,0,0"). The *name* attribute is a user-friendly label for the *devicepkg*.

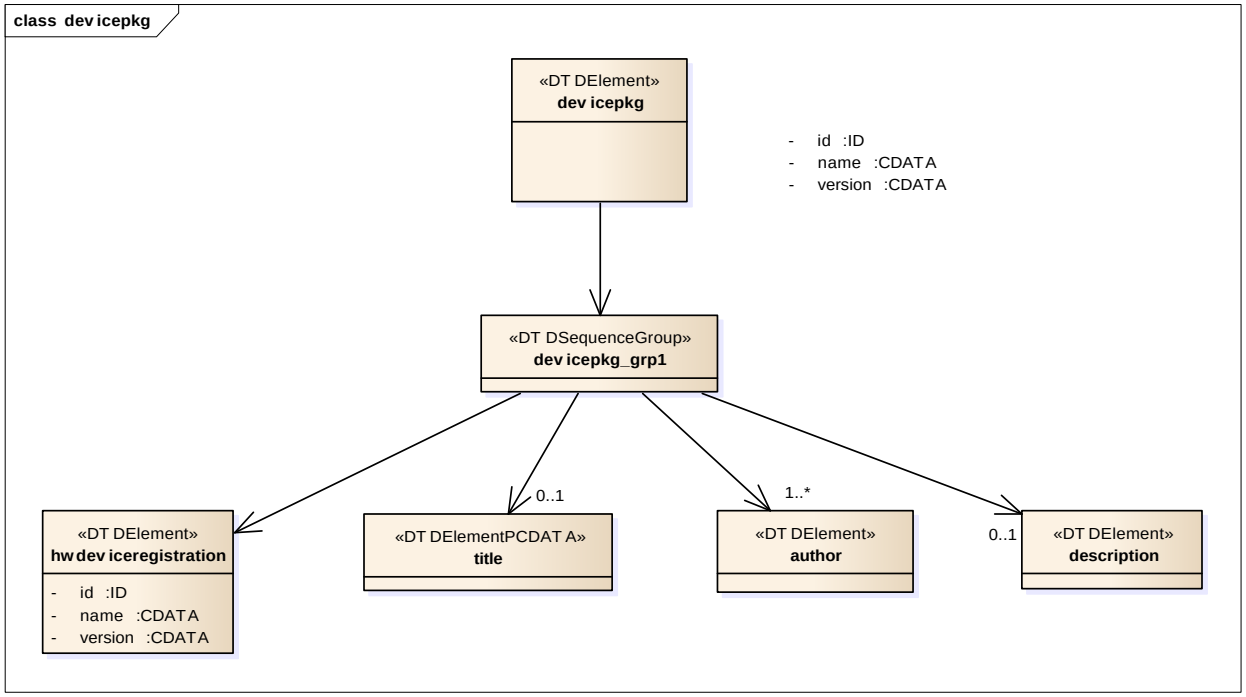


Figure 7: *devicepkg* Element Relationships

```
<!ELEMENT devicepkg
     ( title?
     , author+
     , description?
     , hwdeviceregistration
)>
<!ATTLIST devicepkg
     id         ID   #REQUIRED
     name       CDATA     #REQUIRED
     version    CDATA     #IMPLIED>
```

D-1.7.1.1 title

The *title* element is used for indicating a title for the hardware device being described by *devicepkg*.

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

D-1.7.1.2 author

See D-1.6.1.2 for a definition of the author element.

D-1.7.1.3 description

The *description* element is used to describe any pertinent information about the device implementation that the hardware developer wishes to document within the DPD.

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

D-1.7.1.4 hwdeviceregistration

The *hwdeviceregistration* element (see **Figure 8**) provides device-specific information for a hardware device. The *hwdeviceregistration id* attribute uniquely identifies the device and an implementation specific approach is required to maintain uniqueness within a Domain Profile. The *version* attribute specifies the version of the *hwdeviceregistration* element. The format of the version string is numerical major and minor version numbers separated by commas (e.g., "1,0,0,0"). The *name* attribute is a user-friendly label for the hardware device being registered. At a minimum, the *hwdeviceregistration* element must include a description, the manufacturer, the model number and the device's hardware class(es).

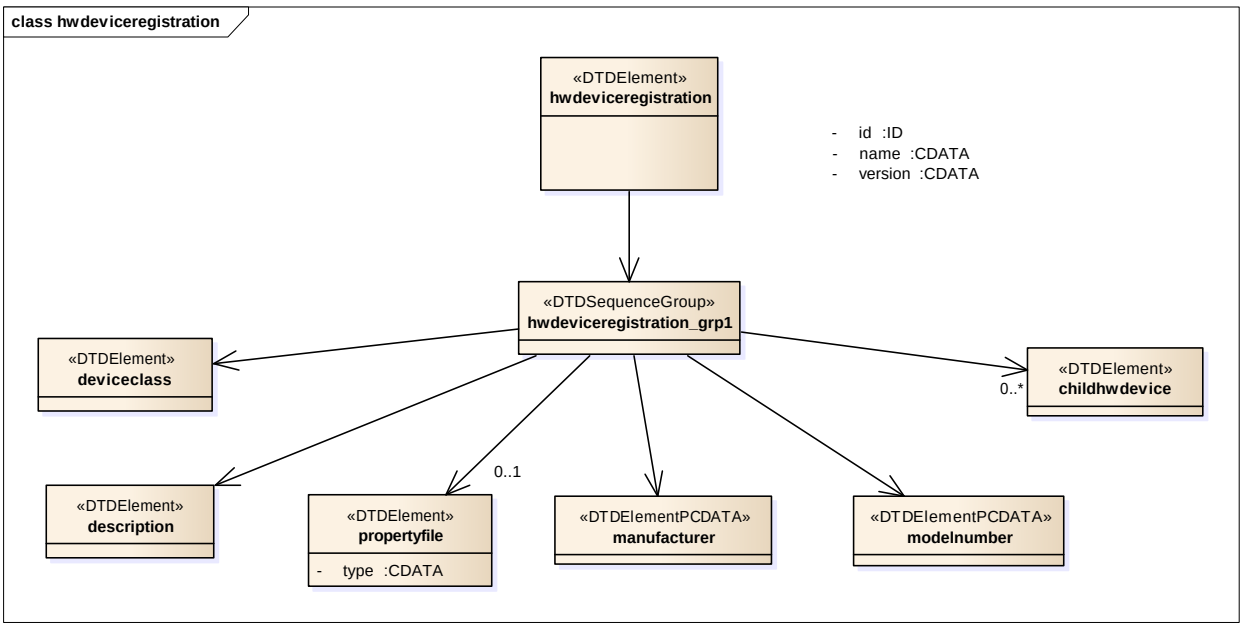


Figure 8: *hwdeviceregistration* Element Relationships

```
<!ELEMENT hwdeviceregistration
     ( propertyfile?
     , description
     , manufacturer
     , modelnumber
     , deviceclass
     , childhwdevice*
)>
<!ATTLIST hwdeviceregistration
     id        ID              #REQUIRED
     name      CDATA           #REQUIRED
     version   CDATA           #IMPLIED>
```

D-1.7.1.4.1 propertyfile

The *propertyfile* element is used to indicate the local filename of the property file associated with the *hwdeviceregistration* element. The format of a property file is described in the Properties Descriptor (PRF) (Section D-1.8).

The intent of the property file is to provide the definition of properties elements for the hardware device being deployed and described in the Device Package (*devicepkg*) or *hwdeviceregistration* element.

```
<!ELEMENT propertyfile
     ( localfile
     )>
<!ATTLIST propertyfile
     type CDATA     #IMPLIED>

<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
     name CDATA     #REQUIRED>
```

D-1.7.1.4.2 description

See D-1.6.1.3 for definition of the *description* element.

D-1.7.1.4.3 manufacturer

The *manufacturer* element is used to convey the name of manufacturer of the device being installed.

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

D-1.7.1.4.4 modelnumber

The *modelnumber* element is used to indicate the manufacture's model number, for the device being installed.

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

D-1.7.1.4.5 deviceclass

The *deviceclass* element is used to identify one or more hardware classes that make up the device being installed.

```
<!ELEMENT deviceclass
( class+
)>
<!ELEMENT class (#PCDATA)>
```

D-1.7.1.4.6 childhwdevice

The *childhwdevice* element (see **Figure 9**) indicates additional device-specific information for hardware devices that make up the root or parent hardware device registration. An example of *childhwdevice* would be a radio's RF module that has receiver and exciter functions within it. In this case, a ComponentBaseDevice representing the RF module itself would be a parent ComponentBaseDevice with its DPD, and the receiver and exciter are child devices to the module. The parent / child relationship indicates that when the RF module is removed from the system, the receiver and exciter devices are also removed.

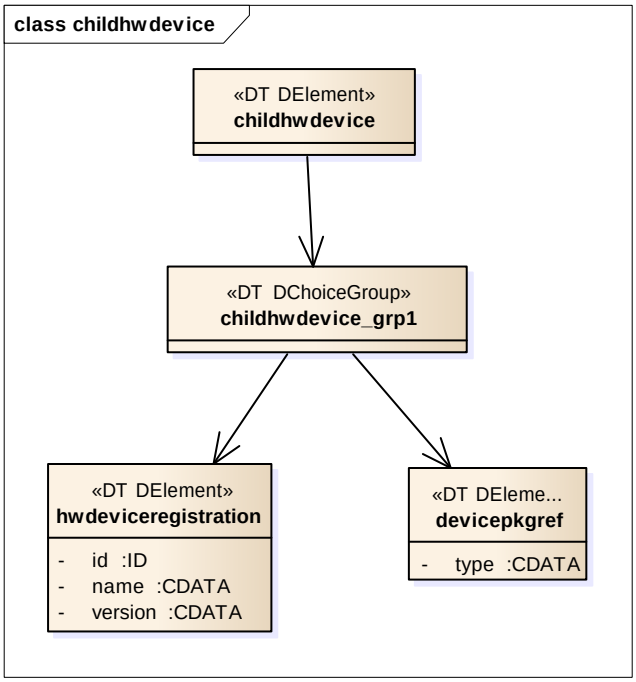


Figure 9: *childhwdevice* Element Relationships

```
<!ELEMENT childhwdevice
     ( hwdeviceregistration | devicepkgref )>
```

D-1.7.1.4.6.1 hwdeviceregistration

The *hwdeviceregistration* element provides device-specific information for the child hardware device. See D-1.7.1.4 for definition of the *hwdeviceregistration* element.

D-1.7.1.4.6.2 devicepkgref

The *devicepkgref* element is used to indicate the local filename of a DPD file pointed to by a DPD (e.g., a *devicepkg* within a *devicepkg*).

```
<!ELEMENT devicepkgref
     ( localfile )>
<!ATTLIST devicepkgref
     type CDATA     #IMPLIED>
```

D-1.8 PROPERTIES DESCRIPTOR

The Properties Descriptor (PRF) file details component and device attribute settings. For purposes of the SCA, Property Descriptor files will contain *simple*, *simplesequence*, *test*, *struct* or *structsequence* elements. These elements will be used to describe attributes of a component that will be used for dependency checking.

These elements will also be used to provide values used by the *PropertySet::configure*, *PropertySet::query*, and *PropertySet::runTest* operations.

SCA494 A Properties Descriptor shall have a ".prf.xml" extension.

D-1.8.1 properties

The *properties* element (see **Figure 10**) is used to describe property attributes that will be used in the *configure* and *query* operations for SCA components that realize the *PropertySet* interface and for definition of attributes used for dependency checking. The *properties* element can also be used in the *TestableObject*::*runTest* operation to configure tests and provide test results.

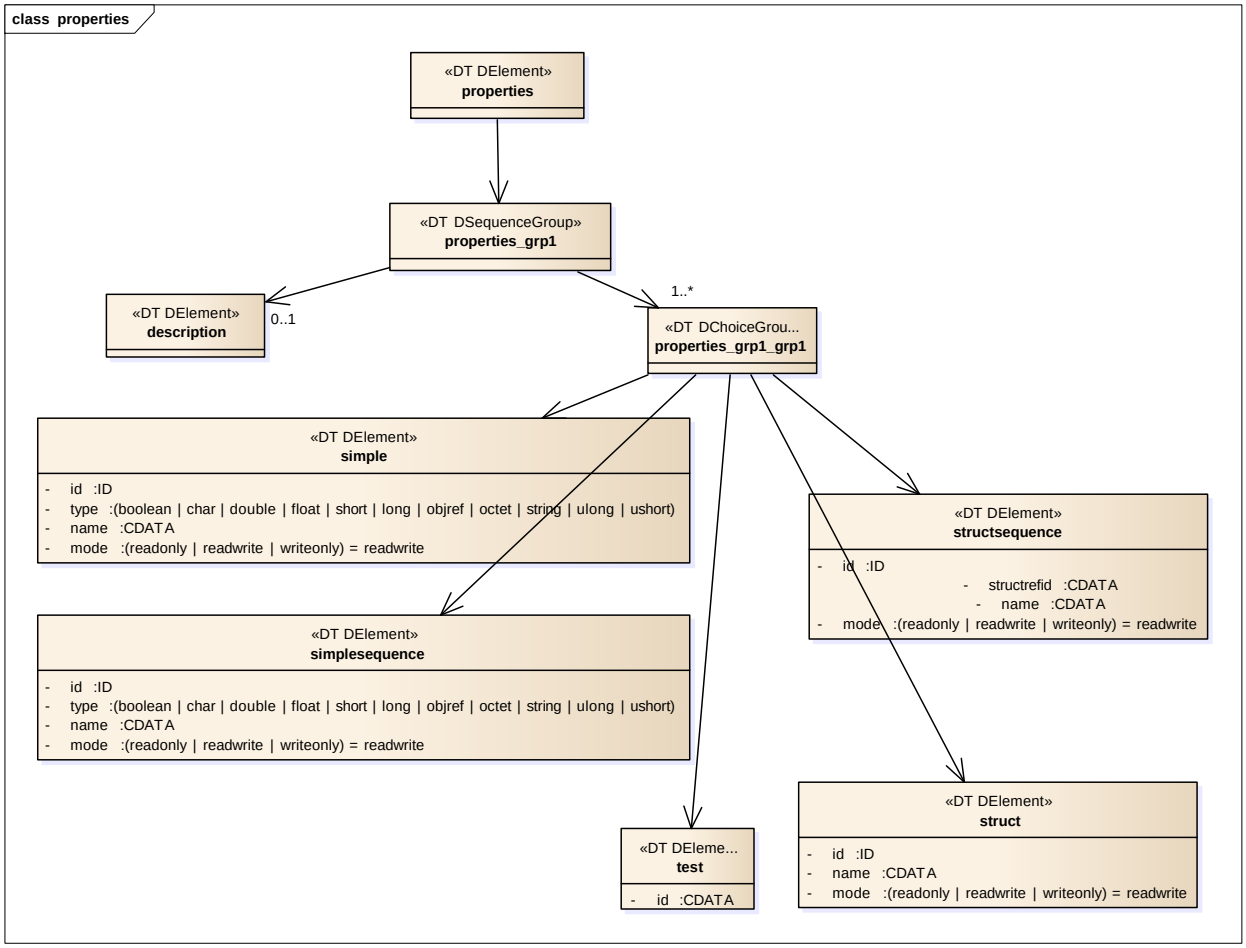


Figure 10: *properties* Element Relationships

```
<!ELEMENT properties
     ( description?
     ,(simple | simplesequence | test | struct | structsequence
     )+
)>
```

D-1.8.1.1 simple

The *simple* element (see **Figure 11**) provides for the definition of a property which includes a unique *id*, *type*, *name* and *mode* attributes of the property that will be used in the *PropertySet configure*() and *query*() operations, for indication of component capabilities, or in the *TestableObject runTest* operation. The *simple* element is specifically designed to support id-value pair definitions. A *simple* property *id* attribute corresponds to the id of the id-value pair. The *value* and *range* of a simple property correspond to the value of the id-value pair. The optional *enumerations* element allows for the definition of a label-to-value for a particular property. The *mode* attribute defines whether the *properties* element is "readonly", "writeonly" or "readwrite". The *id* attribute is an identifier for the *simple* property element. The *id* attribute for a *simple* property that is an allocation type requires an implementation specific approach to maintain uniqueness within a Domain Profile. The *id* attribute for all other *simple* property elements can be any valid XML ID type. The *mode* attribute is only meaningful when the type of the *kind* element is "configure".

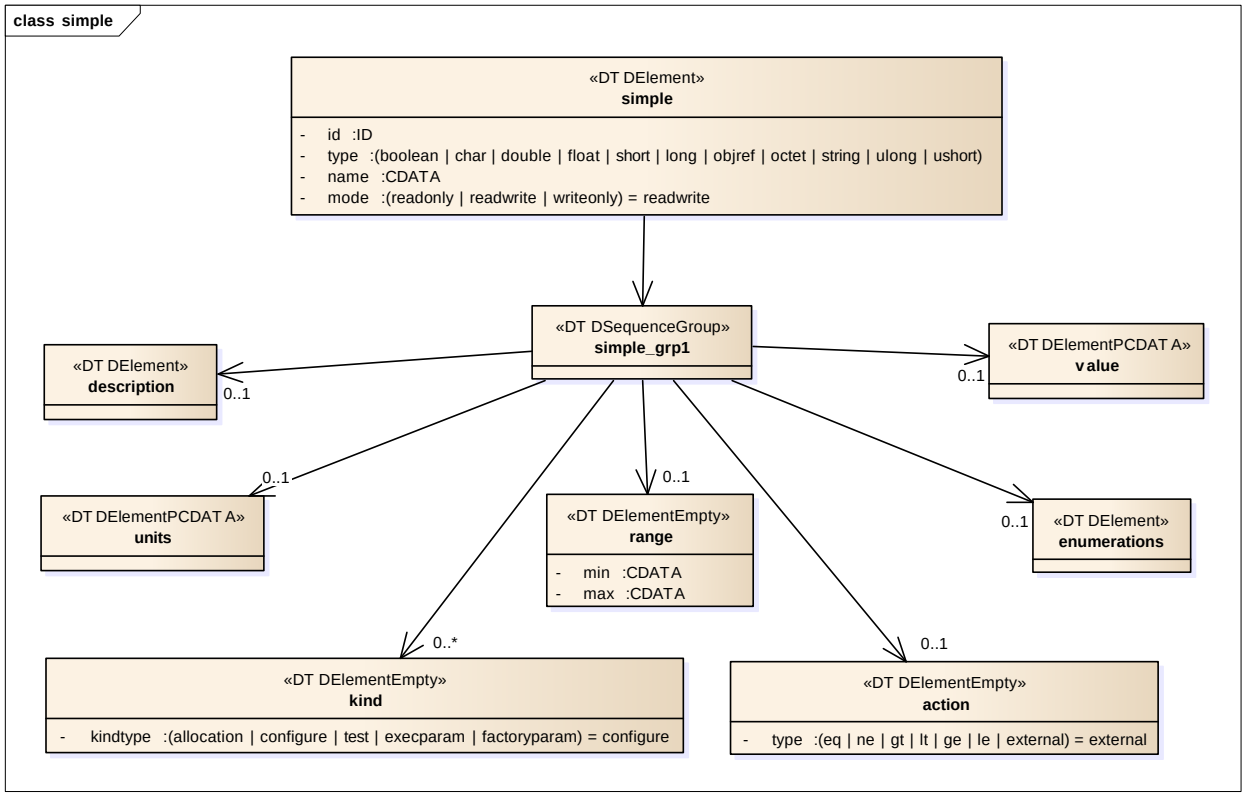


Figure 11: *simple* Element Relationships

```
<!ELEMENT simple
     ( description?
     , value?
     , units?
     , range?
     , enumerations?
     , kind*
     , action?
)>
<!ATTLIST simple
     id   ID        #REQUIRED
     type ( boolean | char   | double | float | short   | long  |
     objref | octet | string | ulong  | ushort ) #REQUIRED
     name CDATA     #IMPLIED
mode ( readonly | readwrite | writeonly)  “readwrite”>
```

D-1.8.1.1.1 description

The *description* element is used to provide a description of the *properties* element that is being defined.

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

D-1.8.1.1.2 value

The *value* element is used to provide a value setting to the *properties* element.

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

D-1.8.1.1.3 units

The *units* element describes the intended practical data representation to be used for the *properties* element.

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

D-1.8.1.1.4 range

The *range* element describes the specific *min* and *max* values that are legal for the *simple* element. The intent of the *range* element is to provide a means to perform range validation. This element is not used by ApplicationFactoryComponents or ApplicationManagerComponents.

```
<!ELEMENT range EMPTY>
<!ATTLIST range
     min  CDATA     #REQUIRED
     max  CDATA     #REQUIRED>
```

D-1.8.1.1.5 enumerations

The *enumerations* element is used to specify one or more *enumeration* elements.

```
<!ELEMENT enumerations
     ( enumeration+ )>
```

The *enumeration* element is used to associate a *value* attribute with a *label* attribute. Enumerations are legal for various integer type *properties* elements. Enumeration values are

implied; if not specified by a developer, the initial implied value is 0 and subsequent values are incremented by 1.

Note: The advantage of the *enumeration* element over the *sequence* element is that the *enumeration* element provides a mechanism to associate a value of a property to a label. The *sequence* element does not allow association of values (only lists of sequences).

```
<!ELEMENT enumeration EMPTY>
<!ATTLIST enumeration
     label     CDATA     #REQUIRED
     value     CDATA     #IMPLIED>
```

D-1.8.1.1.6 kind

The *kind* element's *kindtype* attribute is used to specify the kind of property. The types of *kindtype* attributes are:

1. configure, which is used in the *configure* and *query* operations of the *PropertySet* interface. The ApplicationFactoryComponent will use the *configure* kind of properties to build the CF *Properties* input parameter to the *configure* operation that is invoked on the AssemblyControllerComponent during application creation. The DeviceManagerComponent will use the *configure* kind of properties to build the CF *Properties* input parameter to the *configure* operation that is invoked on components implementing the *PropertySet* interface, during device and service creation. The ApplicationFactoryComponent and DeviceManagerComponent will also use the configure kind of properties for *ComponentFactory* create options parameters. When the mode is readonly, only the *query* behavior is supported. When the mode is writeonly, only the *configure* behavior is supported. When the mode is readwrite, both *configure* and *query* are supported.
2. test, which is used in the *runTest* operation of the *TestableObject* interface. The *test* kind of properties will be used as the *testValues* parameter to the *runTest* operation that is invoked on components that realize the *TestableObject* interface.
3. allocation, which is used in the *allocateCapacity* and *deallocateCapacity* operations of the *Device* interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the simple properties of *kindtype* allocation to build the input capacities parameter to the *allocateCapacity* operation that is invoked on device components during application creation, when the *action* element of those properties is *external*. The application factory and device manager manage *simple* properties of *kindtype* allocation when the *action* is not external. Allocation properties that are external can also be queried using the *PropertySet query* operation.
4. execparam,. which is used in the *execute* operations of the *ExecutableDevice* interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the execparam kind of properties to build the CF *Properties* input parameter to the execute operation that is invoked on ExecutableDeviceComponents during component creation. Only simple elements can be used as execparam types.
5. factoryparam, which is used in the *createComponent* operation of the *ComponentFactory* interface. The ApplicationFactoryComponent and DeviceManagerComponent will use

the factoryparam type of properties to build the CF *Properties* input parameter to the *createComponent* operation.

A property can have multiple *kind* elements and the default *kindtype* is configure.

```
<!ELEMENT kind EMPTY>
<!ATTLIST kind
     kindtype  ( allocation | configure | test | execparam |
     factoryparam) “configure”>
```

D-1.8.1.1.7 action

The *action* element is used to define the type of comparison used to compare an SPD property value to a device property value, during the process of checking SPD dependencies. The *kindtype* attribute of the *action* element, will determine the type of comparison to be made (e.g., equal, not equal, greater than, etc.). The default value for *kindtype* is external.

In principle, the *action* element defines the operation executed during the comparison of the allocation property value, provided by an SPD *dependency* element, to the associated allocation property value of a ComponentBaseDevice. The allocation property is on the left side of the action and the dependency value is on the right side of the action. This process allows for the allocation of appropriate objects within the system based on their attributes, as defined by their dependent relationships.

For example, if a ComponentBaseDevice’s properties file defines a DeviceKind allocation property whose *action* element is set to "equal", then at the time of dependency checking a valid DeviceKind property is checked for equality. If a software component implementation is dependent on a DeviceKind property with its value set to "NarrowBand", then the component's SPD dependency *propertyref* element will reference the *id* of the DeviceKind allocation property with a value of "NarrowBand". At the time of dependency checking, the ApplicationFactoryComponent and DeviceManagerComponent will check ComponentBaseDevices whose *properties kind* element is set to “allocation” and property *id* is DeviceKind for equality against a "NarrowBand" value.

```
<!ELEMENT action EMPTY>
<!ATTLIST action
     type  ( eq  |  ne  | gt  | lt | ge  |  le  | external
     )"external">
```

D-1.8.1.2 simplesequence

The *simplesequence* element (see **Figure 12**) is used to specify a list of *properties* with the same characteristics (e.g., type, range, units, etc.). The *simplesequence* element definition is similar to the *simple* element definition except that it has a list of values instead of one value. The *simplesequence* element maps to the sequence types for the CF and CORBA modules, defined in SCA Appendix C and OMG CORBA Specification version 3.2 [1] , based upon the *type* attribute.

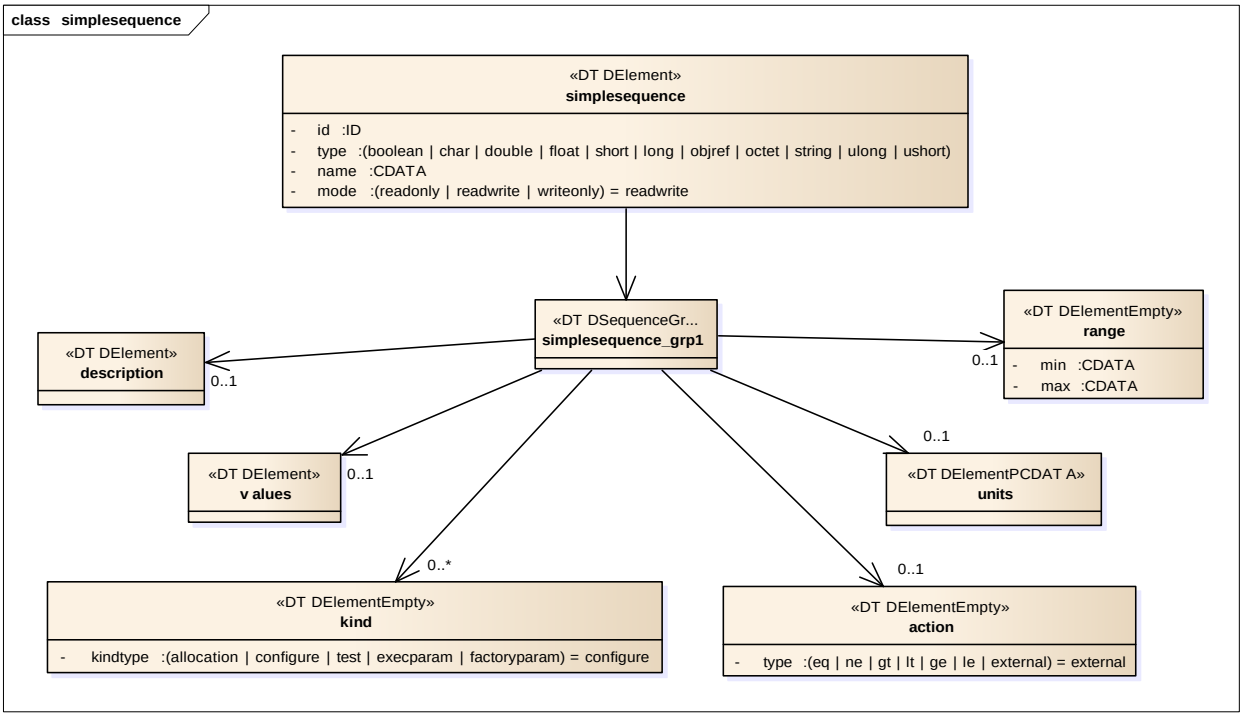


Figure 12: *simplesequence* Element Relationships

```
<!ELEMENT simplesequence
     ( description?
     , values?
     , units?
     , range?
     , kind*
     , action?
     )>
<!ATTLIST simplesequence
     id   ID   #REQUIRED
     type ( boolean | char  | double | float | short | long |
     objref | octet | string | ulong |ushort )    #REQUIRED
     name CDATA     #IMPLIED
     mode (readonly | readwrite | writeonly) “readwrite”>

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

D-1.8.1.3 test

The *test* element (see **Figure 13**) is used to specify a list of test properties for executing the *runTest* operation in order to perform a component specific test. This element contains *inputvalue* and *resultvalue* elements and it has an *id* attribute for grouping test properties to a specific test. The *id* attribute will be represented by a numeric value. *Inputvalues* are used to configure the test to be performed (e.g., frequency and RF power output level). When the test has completed, *resultvalues* contain the results of the testing (e.g., pass or a fault code/message)

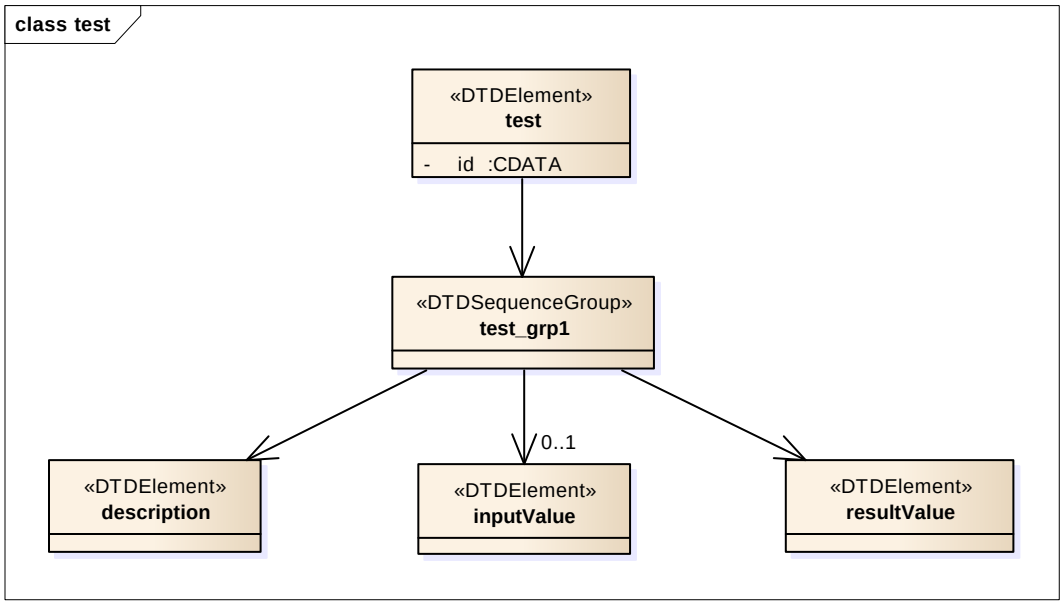


Figure 13: *test* Element Relationships

```
<!ELEMENT test
     ( description
     , inputvalue?
     , resultvalue
     )>
<!ATTLIST test
     id   CDATA     #REQUIRED>
```

D-1.8.1.3.1 inputvalue

The *inputvalue* element is used to provide test configuration properties. The *simple* properties it contains must have a *kindtype* value of test.

```
<!ELEMENT inputvalue
     ( simple+ )>
```

D-1.8.1.3.2 resultvalue

The *resultvalue* element is used to specify the desired results of the *runTest* operation. The *simple* properties it contains must have a *kindtype* value of test.

```
<!ELEMENT resultvalue
     ( simple+ )>
```

D-1.8.1.4 struct

The *struct* element (see **Figure 14**) is used to group properties with different characteristics (i.e., similar to a structure or record entry). Each item in the *struct* element can be a different *simple* type (e.g., short, long, etc.). The *struct* element corresponds to the CF *Properties* type where each *struct* item (ID, value) corresponds to a *properties* element list item. The *properties* element list size is based on the number of *struct* items.

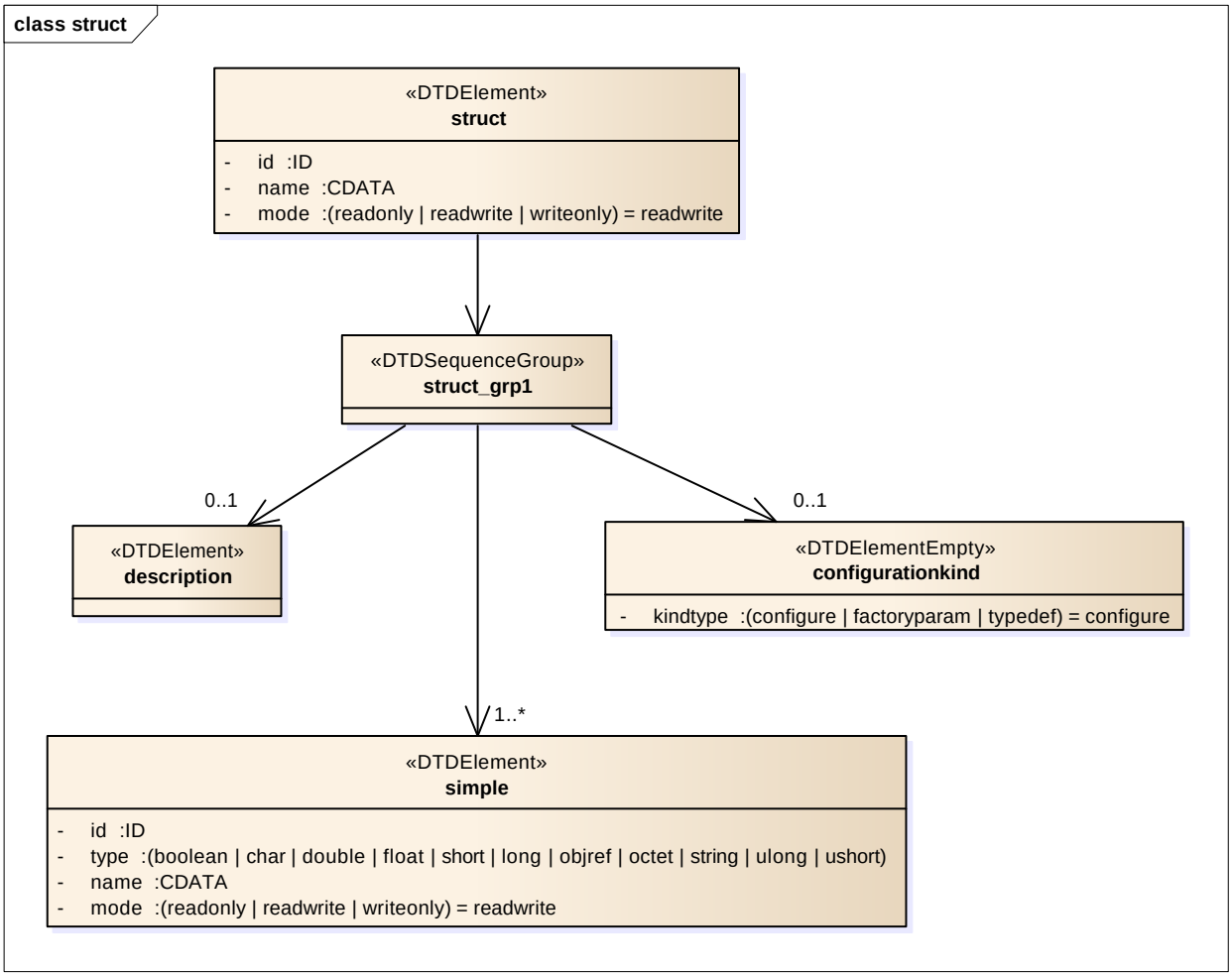


Figure 14: *struct* Element Relationships

```
<!ELEMENT struct
     ( description?
     , simple+
     , configurationkind?
)>
<!ATTLIST struct
     id   ID        #REQUIRED
     name CDATA     #IMPLIED
     mode (readonly | readwrite | writeonly) "readwrite">
```

D-1.8.1.4.1 configurationkind

The *configurationkind* element's *kindtype* attribute is used to specify the kind of property. The kindtypes are:

1. configure, which is used in the *configure* and *query* operations of the *PropertySet* interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the configure kind of properties to build the CF Properties input parameter to the *configure* operation that is invoked on SCA components that realize the *PropertySet* interface during application creation. When the *mode* is readonly, only the query

behavior is supported. When the *mode* is writeonly, only the configure behavior is supported. When the *mode* is readwrite, both configure and query are supported.

2. factoryparam, which is used in the *createComponent* operations of the *ComponentFactory* interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the factoryparam kind of properties to build the CF Properties input parameter to the *createComponent()* operation. A property can have multiple *configurationkind* elements and their default *kindtype* is "configure".
3. typedef, which is used when this definition is to be used a type definition for a *structsequence*. A property of this type will not have an identity independent of the element that references it (i.e. it would not be possible to use the *configure* or *query* operations on this element).

```
<!ELEMENT configurationkind EMPTY>
<!ATTLIST configurationkind
     kindtype  (configure | factoryparam | typedef) "configure">
```

D-1.8.1.5 structsequence

The *structsequence* element (see **Figure 15**) is used to specify a list of properties with the same *struct* characteristics. The *structsequence* element maps to a *properties* element having the CF Properties type. Each item in the CF Properties type will be the same *struct* definition as referenced by the *structrefid* attribute. Any values specified within the *struct* definition are ignored and values for the sequence are provided by the *structvalue* element.

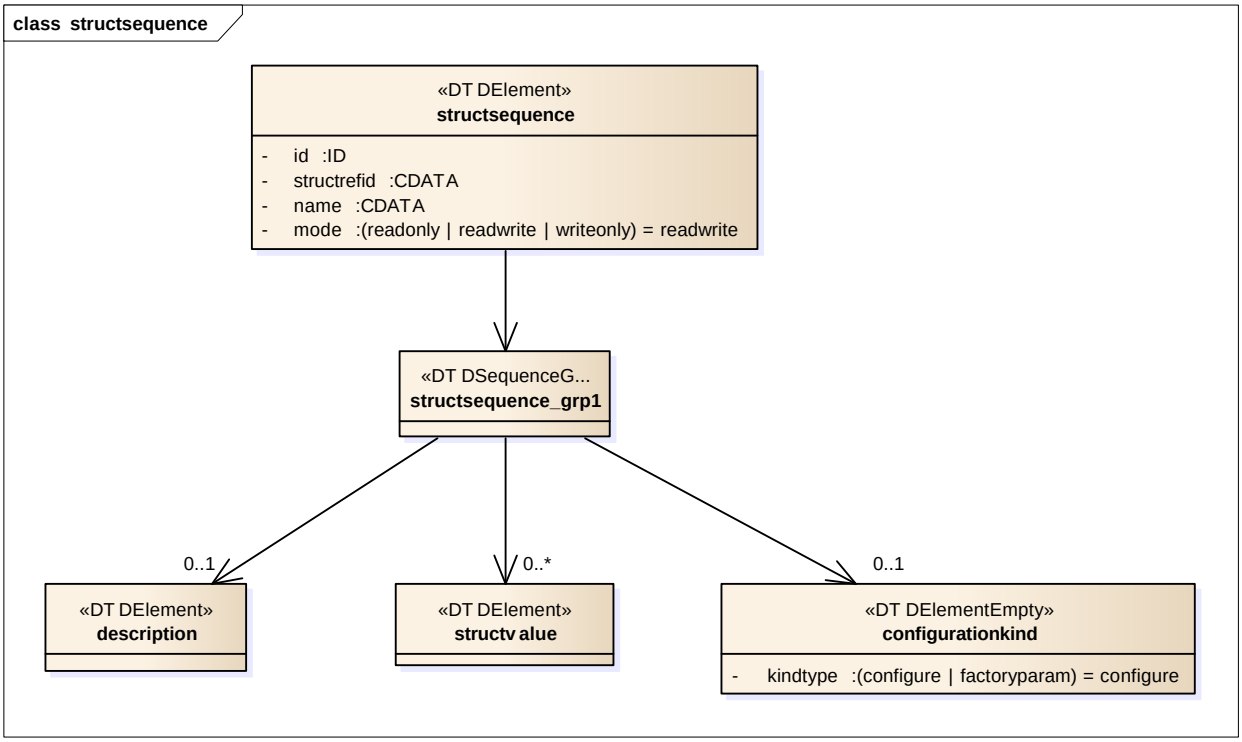


Figure 15: *structsequence* Element Relationships

```
<!ELEMENT structsequence
     ( description?
     , structvalue*
     , configurationkind?
)>
<!ATTLIST structsequence
     id                  ID        #REQUIRED
     structrefid         CDATA     #REQUIRED
     name                CDATA     #IMPLIED
     mode  (readonly | readwrite | writeonly) "readwrite">

<!ELEMENT structvalue
      ( simpleref+ )>

<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
     refid     CDATA     #REQUIRED
     value     CDATA     #REQUIRED>
```

D-1.9 SOFTWARE COMPONENT DESCRIPTOR

The SCA components ResourceComponent, DeviceComponent, LoadableDeviceComponent, ExecutableDeviceComponent, ComponentFactoryComponent, and ServiceComponents that are described by the Software Component Descriptor (SCD) are based on the SCA specification, and the following specification concentrates on definition of the elements necessary for describing the ports and interfaces of these components.

SCA495 A Software Component Descriptor file shall have a ".scd.xml" extension.

D-1.9.1 softwarecomponent

The *softwarecomponent* element (see **Figure 16**) is the root element of the SCD file. For use within the SCA the sub-elements that are supported include:

1. *componentrepid* – is the repository id of the component
2. *componenttype* – identifies the type of software component object
3. *componentfeatures* – provides the supported message ports for the component
4. *interface* – describes the component unique id and name for supported interfaces.

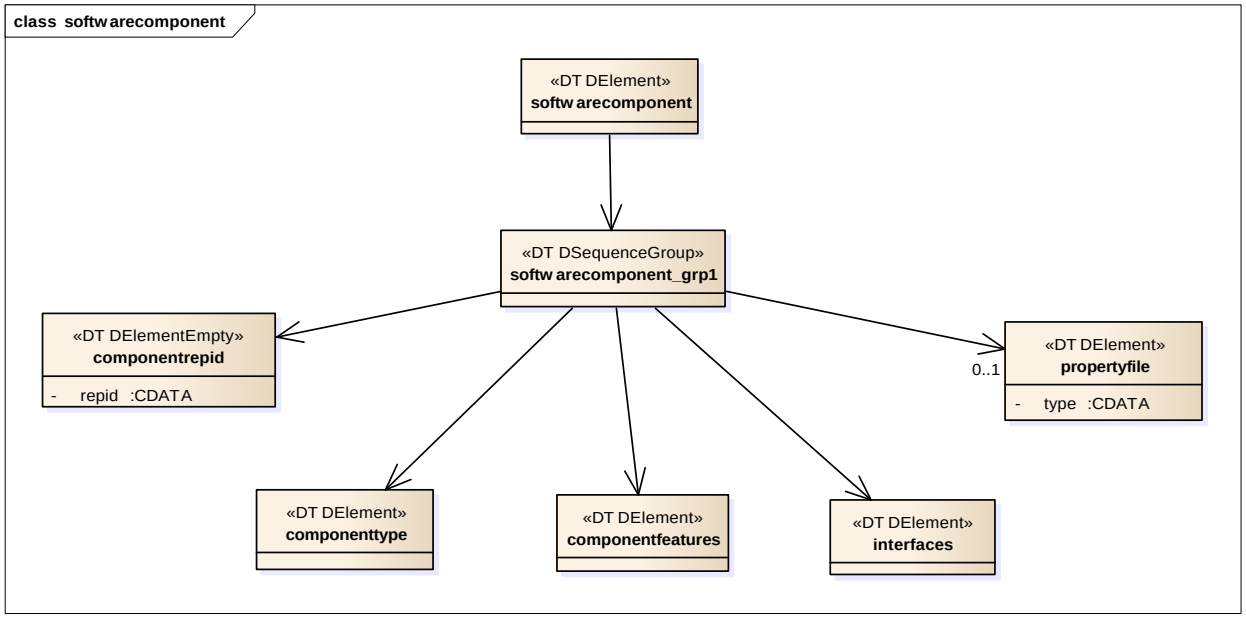


Figure 16: *softwarecomponent* Element Relationships

```
<!ELEMENT softwarecomponent
      ( componentrepid
     , componenttype
     , componentfeatures
     , interfaces
     , propertyfile?
     )>
```

D-1.9.1.1 componentrepid

The *componentrepid* uniquely identifies the interface that the component is implementing. The *componentrepid* may be referred to by the *componentfeatures* element. The *componentrepid* is derived from the *Resource*, *Device*, *LoadableDevice*, *ExecutableDevice*,, *ComponentFactory* interface or represents a ServiceComponent.

```
<!ELEMENT componentrepid EMPTY>
<!ATTLIST componentrepid
     repid     CDATA     #REQUIRED>
```

D-1.9.1.2 componenttype

The *componenttype* describes properties of the component. For SCA components, the component types include APPLICATION_COMPONENT, DEVICE_COMPONENT, CF_SERVICE_COMPONENT, NON_CF_SERVICE_COMPONENT, and FRAMEWORK_COMPONENT.

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

D-1.9.1.3 componentfeatures

The *componentfeatures* element (see **Figure 17**) is used to describe a component with respect to the components that it inherits from, the interfaces the component supports, its provides and uses *ports*. If a component extends any of the following interfaces, *Resource*, *ComponentFactory*, or *Device*, *LoadableDevice*, *ExecutableDevice* ,then all the inherited interfaces (e.g., *Resource*) are depicted as *supportsinterface* elements.

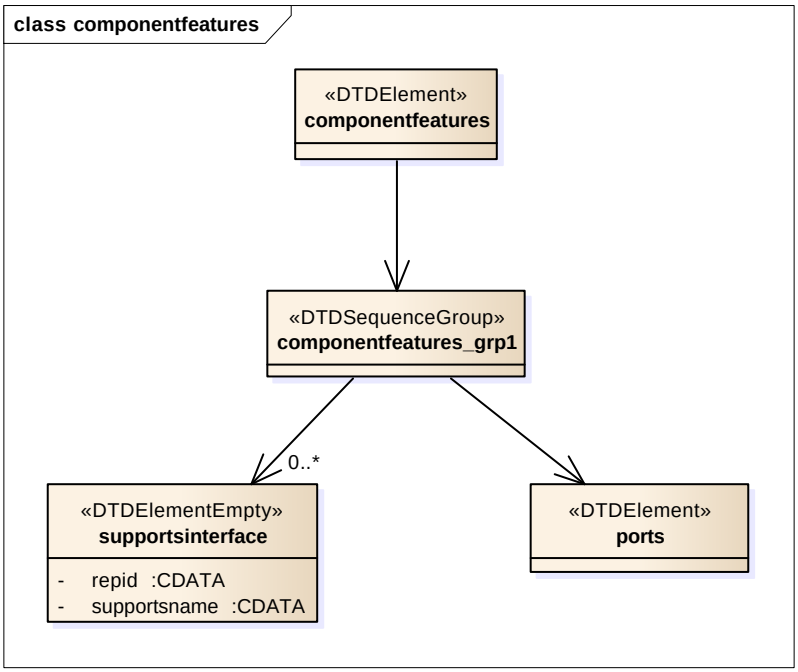


Figure 17: *componentfeatures* Element Relationships

```
<!ELEMENT componentfeatures
     ( supportsinterface*
     , ports
)>
```

D-1.9.1.3.1 supportsinterface

The *supportsinterface* element is used to identify an interface definition that the component supports. These interfaces are distinct interfaces that were inherited by the component's specific interface. One can widen the component's interface to be a *supportsinterface*. The *repid* is used to refer to the *interface* element (see *interfaces* section D-1.9.1.4).

```
<!ELEMENT supportsinterface EMPTY>
<!ATTLIST supportsinterface
     repid          CDATA     #REQUIRED
     supportsname   CDATA     #REQUIRED>
```

D-1.9.1.3.2 ports

The *ports* element (see **Figure 18**) describes what interfaces a component provides and uses.

The *provides* elements are interfaces that are not part of a component's interface but are independent interfaces known as facets (i.e. a provides port at the end of a path, like I/O Device or Modem Device). The *uses* element is uses port at the start of a path (i.e. I/O Device) that is connected to a *provides* or *supportinterfaces* interface. Any number of *uses* and *provides* elements can be given in any order. Each *ports* element has a name and references an interface by *repid* (see interfaces section D-1.9.1.4). The port names are used in the SAD to connect ports together. The *maxconnections* attribute allows the developer to specify how many simultaneous connections are allowed to be made using that port. A *ports* element also has an optional *porttype* element that allows for identification of port classification. Values for *porttype* include "data", "control", "responses", and "test". If a *porttype* is not given then "control" is assumed.

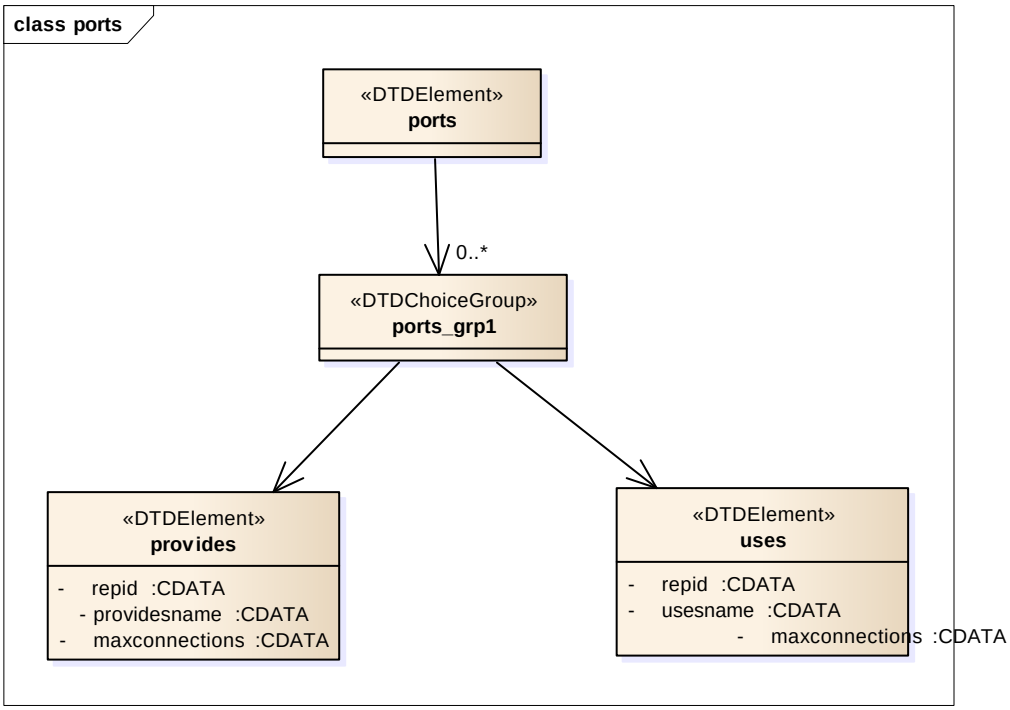


Figure 18: *ports* Element Relationships

```
<!ELEMENT ports
     ( provides | uses )*
>

<!ELEMENT provides
          ( porttype* )>
<!ATTLIST provides
     repid          CDATA     #REQUIRED
     providesname   CDATA     #REQUIRED
     maxconnections CDATA     #REQUIRED>

<!ELEMENT uses
      ( porttype*
)>
<!ATTLIST uses
     repid     CDATA          #REQUIRED
     usesname  CDATA          #REQUIRED
     maxconnections CDATA     #REQUIRED>

<!ELEMENT porttype EMPTY>
<!ATTLIST porttype
     type ( data | control | responses | test ) #REQUIRED>
```

D-1.9.1.4 interfaces

The *interfaces* element is made up of one to many *interface* elements.

```
<!ELEMENT interfaces
     ( interface+ )>
```

The *interface* element describes an interface that the component, either directly or through inheritance, provides, uses, or supports. The *name* attribute is the character-based non-qualified name of the interface. The *repid* attribute is the unique repository id of the interface. The *repid* is also used to reference an *interface* element elsewhere in the SCD, for example from the *inheritsinterface* element.

For ServiceComponents the *inheritsinterface* element is not expected to contain a value.

```
<!ELEMENT interface
     ( inheritsinterface*) >
<!ATTLIST interface
     repid     CDATA     #REQUIRED
     name      CDATA     #REQUIRED>

<!ELEMENT inheritsinterface EMPTY>
<!ATTLIST inheritsinterface
     repid     CDATA     #REQUIRED>
```

D-1.9.1.5 propertyfile

The *propertyfile* element is used to indicate the local filename of the PRF file associated with the software component. The definition of the *propertyfile* element can be found in section D-1.6.1.4. Within the SCD, the *localfile sub*-element of the *propertyfile* element is a relative pathname referencing a file in the same directory as the SCD or in a directory that is relative to the directory where the SCD file is located.

D-1.10 SOFTWARE ASSEMBLY DESCRIPTOR

This section describes the XML elements of the Software Assembly Descriptor (SAD) XML file; the *softwareassembly* element (see Figure 19). The intent of the software assembly is to provide the means of describing the assembled functional application and the interconnection characteristics of the SCA components within that application. Created applications are assembled from a combination of one or more component instantiations and nested assembly instantiations that are interconnected with each other and to platform devices and services.

SCA496 A Software Assembly Descriptor file shall have a ".sad.xml" extension.

D-1.10.1 softwareassembly

The installation of an application into the system includes the installation of a main SAD file and one or more SPD and SAD files. The SAD file references component SPD files to obtain deployment information for these components, as well SAD files for nested applications. The *softwareassembly* element's *name* attribute uniquely identifies the assembly, requiring an implementation specific approach to maintain uniqueness within a Domain Profile. The *softwareassembly* element's *version* attribute is the version of the application.

The SAD *deploymentdependencies* are merged with and overridden by lower-level *deploymentdependencies* defined within the *componentinstantiation* and *assemblyinstantiation* elements.

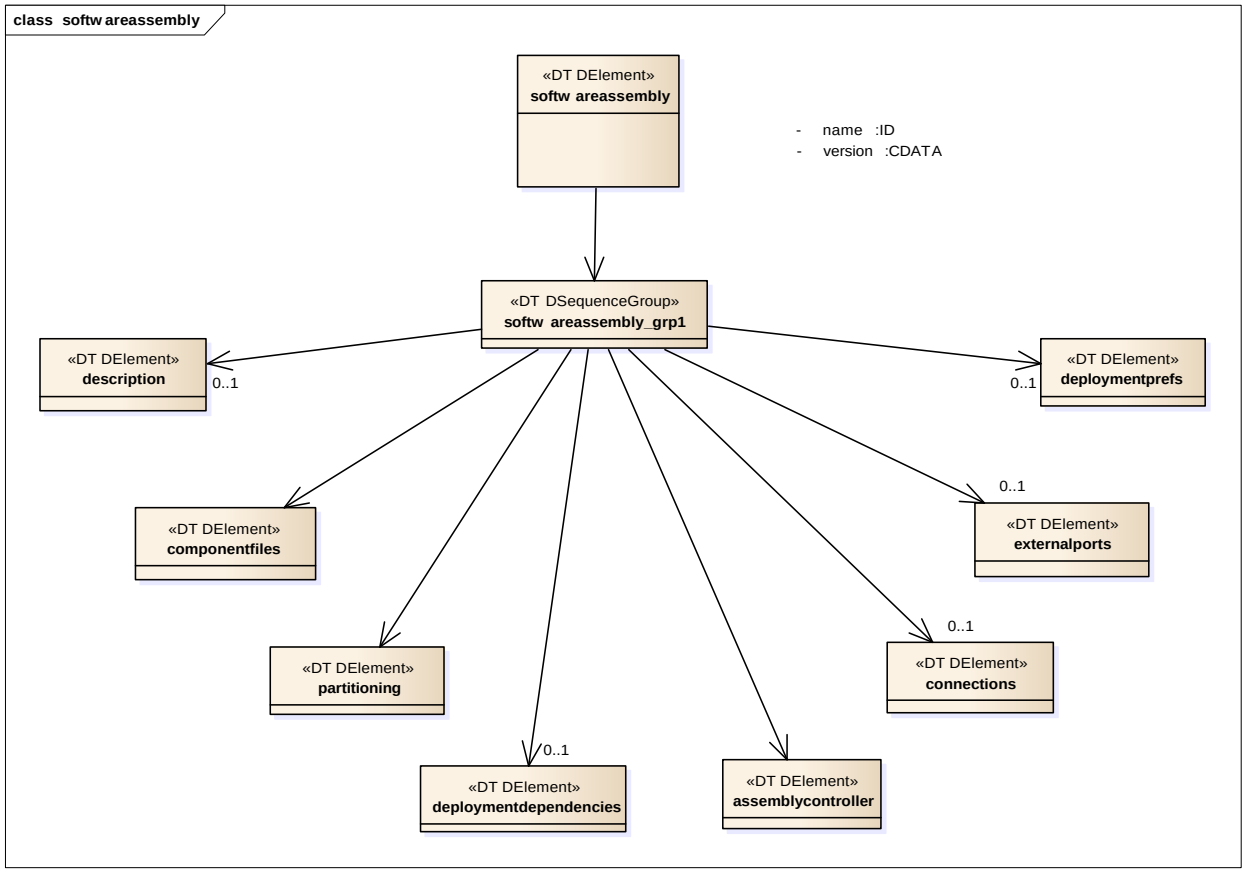


Figure 19: *softwareassembly* Element Relationships

```
<!ELEMENT softwareassembly
     ( description?
     , componentfiles
     , partitioning
     , deploymentdependencies?
     , assemblycontroller
     , connections?
     , externalports?
     , deploymentprefs?
)>
     <!ATTLIST softwareassembly name ID #REQUIRED
     version CDATA #IMPLIED>
```

D-1.10.1.1 description

The *description* element of the component assembly may be used to describe any information the developer would like to indicate about the assembly.

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

D-1.10.1.2 componentfiles

The *componentfiles* element is used to indicate that an assembly is made up of 1..n component files and / or subassemblies. The *componentfile* element contains a reference to a local file, which can be an SPD or SAD file.

```
<!ELEMENT componentfiles
     ( componentfile+ )>
```

D-1.10.1.2.1componentfile

The *componentfile* element is a reference to a local file. See section D-1.6.1.4.1 for the definition of the *localfile* element. The *type* attribute is "Software Package Descriptor" or "Software Assembly Descriptor".

```
<!ELEMENT componentfile
     ( localfile )>
<!ATTLIST componentfile
     id         ID   #REQUIRED
     type       CDATA     #IMPLIED>
```

D-1.10.1.3 partitioning

A component *partitioning* element (see **Figure 20**) specifies a deployment pattern of components and their components-to-hosts relationships as well as nested sub-applications. A component instantiation is captured inside a *componentplacement* element. The *hostcollocation* element allows the components to be placed on a common device. When the *componentplacement* is by itself and not inside a *hostcollocation*, it then has no collocation constraints. An assembly instantiation (nested sub-application) is captured inside an *assemblyplacement* element.

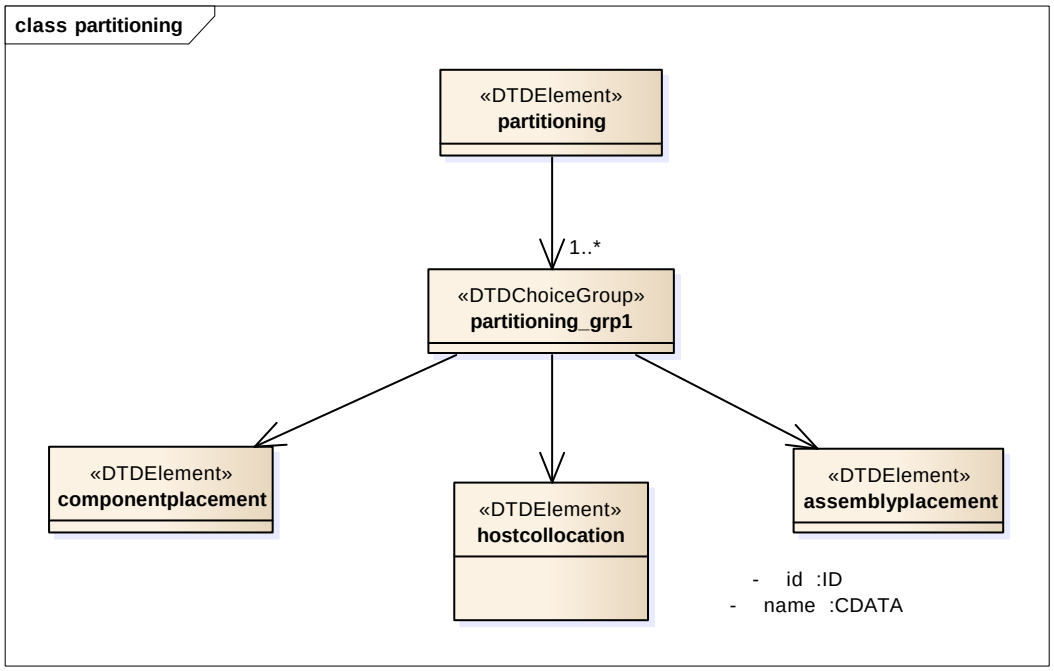


Figure 20: *partitioning* Element Relationships

```
<!ELEMENT partitioning
     ( componentplacement | hostcollocation |
       assemblyplacement )+>
```

D-1.10.1.3.1componentplacement

The *componentplacement* element (see Figure 21) defines a particular deployment of a component. The component can be deployed either directly or by using a *ComponentFactory*.

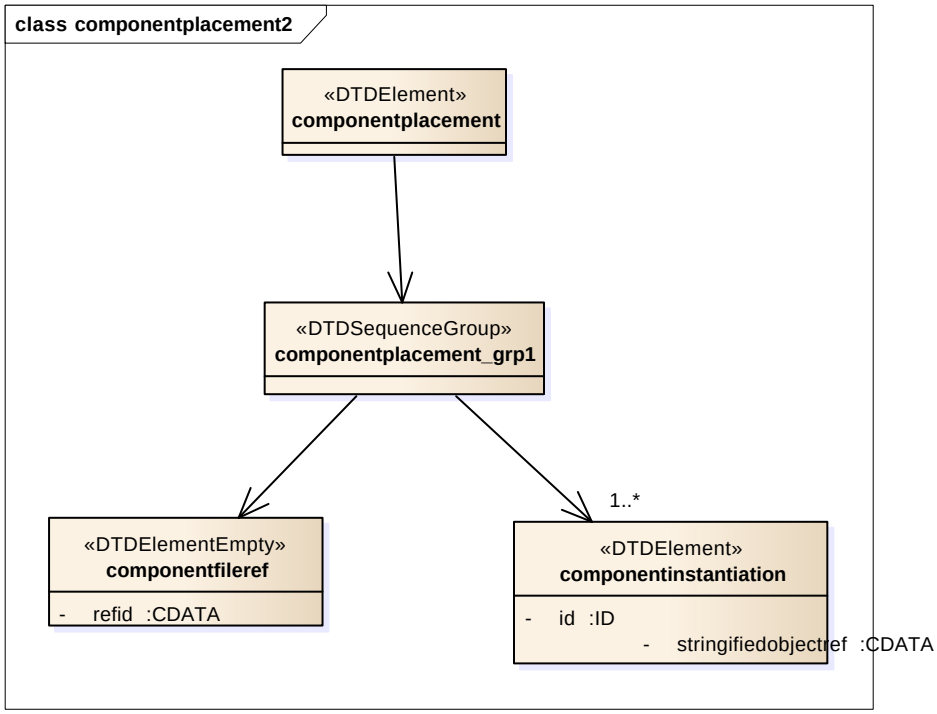


Figure 21: *componentplacement* Element Relationships

```
<!ELEMENT componentplacement
     ( componentfileref
     , componentinstantiation+
)>
```

D-1.10.1.3.1.1 componentfileref

The *componentfileref* element is used to reference a particular SPD or a SAD file. The *componentfileref* element's *refid* attribute corresponds to the *componentfile* element's *id* attribute.

```
<!ELEMENT componentfileref  EMPTY>
<!ATTLIST componentfileref
     refid     CDATA     #REQUIRED>
```

D-1.10.1.3.1.2 componentinstantiation

The *componentinstantiation* element (see Figure 22) is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation's id* attribute is an implementation specific value that uniquely identifies the component within a Domain Profile. The *componentinstantiation* element's id may be

referenced by the *usesport* and *providesport* elements within the SAD file. It is the component name for the instantiation not the application name. The *componentinstantiation* element's stringifiedobjectref attribute, when specified, is the component instantiation object reference that requires dynamic connections.

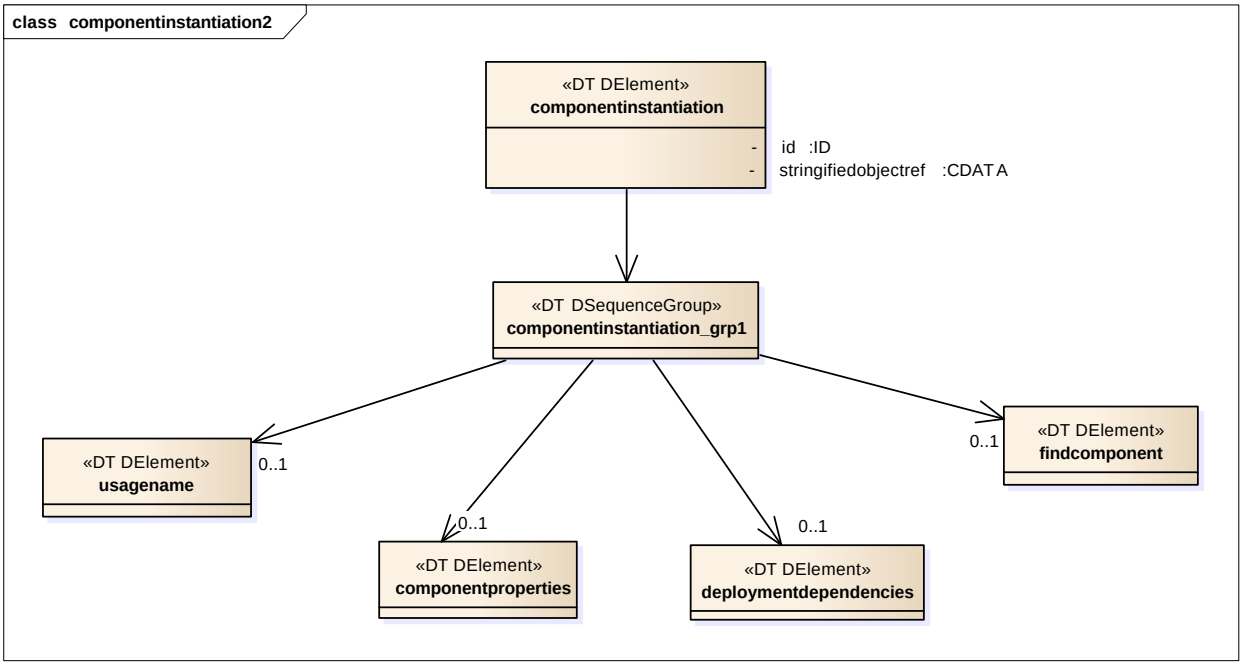


Figure 22: *componentinstantiation* Element Relationships

```
<!ELEMENT componentinstantiation
     ( usagename?
     , componentproperties?
     , deploymentdependencies?
     , findcomponent?
)>
<!ATTLIST componentinstantiation
     id   ID   #REQUIRED
     stringifiedobjectref CDATA #implied>

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

D-1.10.1.3.1.2.1componentproperties

The optional *componentproperties* element (see D-1.10.1.3.3.1.1) is a list of configure, factoryparam, and/or execparam properties values that are used in creating the component or for the initial configuration of the component.

The following sources will be searched in the given precedence order for initial values for simple properties with a *kindtype* of "execparam" or "configure" and a *mode* attribute of "readwrite" or "writeonly":

1. The SAD partitioning : componentplacement : componentinstantiation element,

2. The value or default value, if any, from the SPD using the properties precedence stated in D-1.6.1.

If no values are found in the sources above, the property is discarded.

The following sources will be searched in the given precedence order for initial values for simple properties with a *kindtype* of "factoryparam":

1. The SAD *partitioning* : *componentplacement* : *componentinstantiation* : *findcomponent* : *componentfactoryref* : *componentfactoryproperties* element,
2. The SAD *partitioning* : *componentplacement* : *componentinstantiation* : *componentproperties* element,
3. The value or default value, if any, from the SPD using the properties precedence stated in D-1.6.1.

If no values are found in the sources above, the property is discarded.

D-1.10.1.3.1.2.2 deploymentdependencies

The optional *deploymentdependencies* element (described generically in section D-1.10.1.4) overrides *componentinstantiation's* SPD and SAD dependencies.

D-1.10.1.3.1.2.3 findcomponent

The optional *findcomponent* element (see **Figure 23**) is used to obtain the object reference from a *componentfactoryref.*

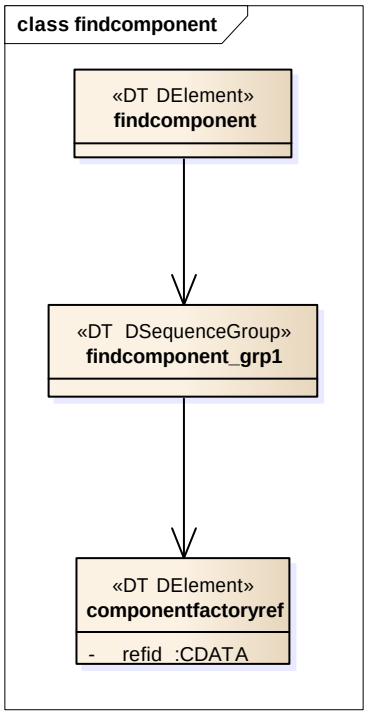


Figure 23: *findcomponent* Element Relationships

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

D-1.10.1.3.1.2.3.1 componentfactoryref

The *componentfactoryref* element (see Figure 24) refers to a particular ApplicationComponentFactory *componentinstantiation* element found in the SAD, which is used to obtain a Resource instance for this *componentinstantiation* element. The *refid* attribute refers to a unique *componentinstantiation id* attribute.

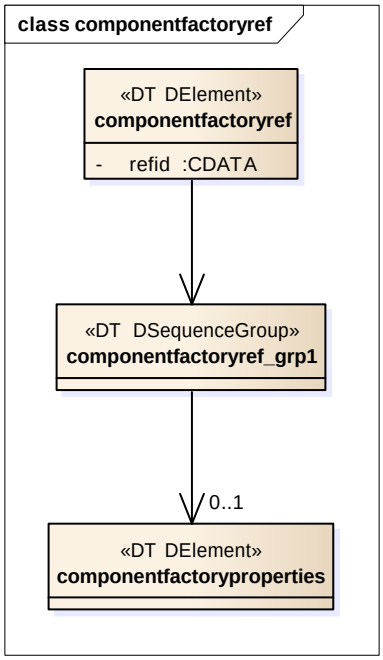


Figure 24: *componentfactoryref* Element Relationships

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

D-1.10.1.3.1.2.3.1.1 componentfactoryproperties

The optional *componentfactoryproperties* element (see Figure 25) specifies the properties "qualifiers", for the *ComponentFactory::create* call.

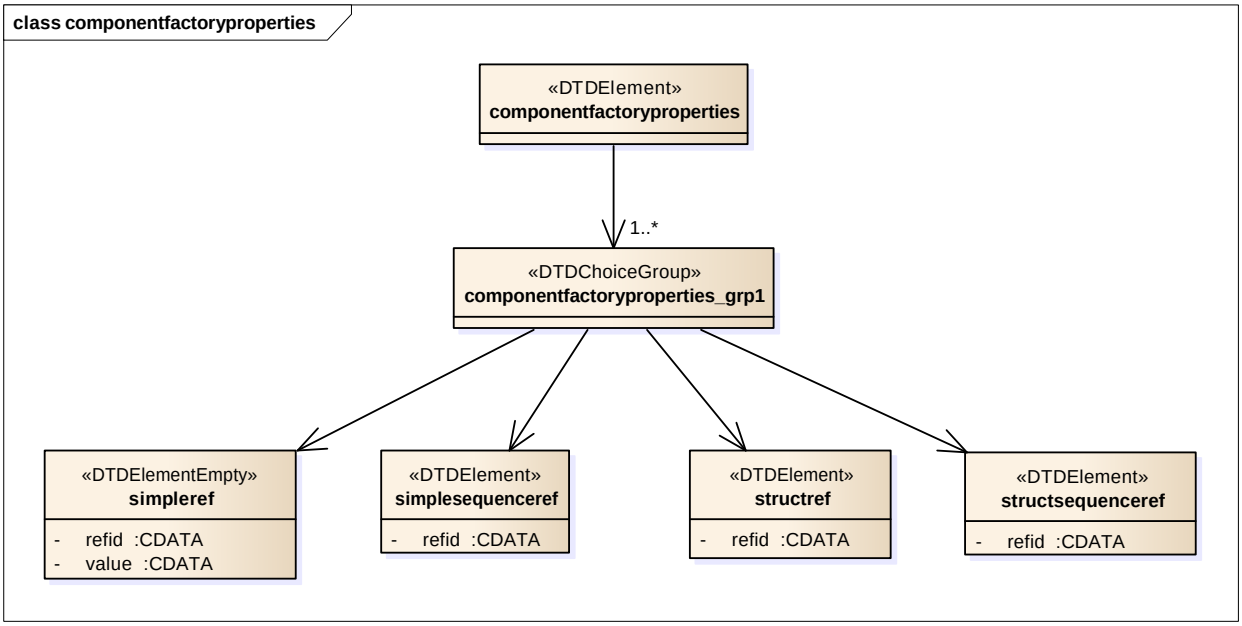


Figure 25: *componentfactoryproperties* Element Relationships

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

<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
     refid     CDATA     #REQUIRED
     value     CDATA     #REQUIRED>

<!ELEMENT simplesequenceref
          ( values )>
<!ATTLIST simplesequenceref
     refid     CDATA     #REQUIRED>

<!ELEMENT structref
          ( simpleref+ )>
<!ATTLIST structref
     refid     CDATA     #REQUIRED>

<!ELEMENT structsequenceref
      ( structvalue+ )>
<!ATTLIST structsequenceref
     refid     CDATA     #REQUIRED>

<!ELEMENT structvalue
      ( simpleref+ )>

<!ELEMENT values
      ( value+ )>

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

D-1.10.1.3.2hostcollocation

The *hostcollocation* element specifies a group of component instances that are to be deployed together on a single host. For purposes of the SCA, the *componentplacement* element will be used to describe the 1...n components that will be collocated on the same host platform. Within the SCA specification, a host platform will be interpreted as a single device. The id and *name* attributes are optional but may be used to uniquely identify a set of collocated components within a SAD file.

```
<!ELEMENT hostcollocation
     ( componentplacement )+>
<!ATTLIST hostcollocation
     id   ID   #IMPLIED
     name CDATA     #IMPLIED>
```

D-1.10.1.3.2.1 componentplacement

See *componentplacement*, section D-1.10.1.3.1.

D-1.10.1.3.3assemblyplacement

The *assemblyplacement* element (see Figure 26) defines a particular deployment of a nested subassembly. It references the SAD file for that nested subassembly and an *assemblyinstantiation* element defining its creation.

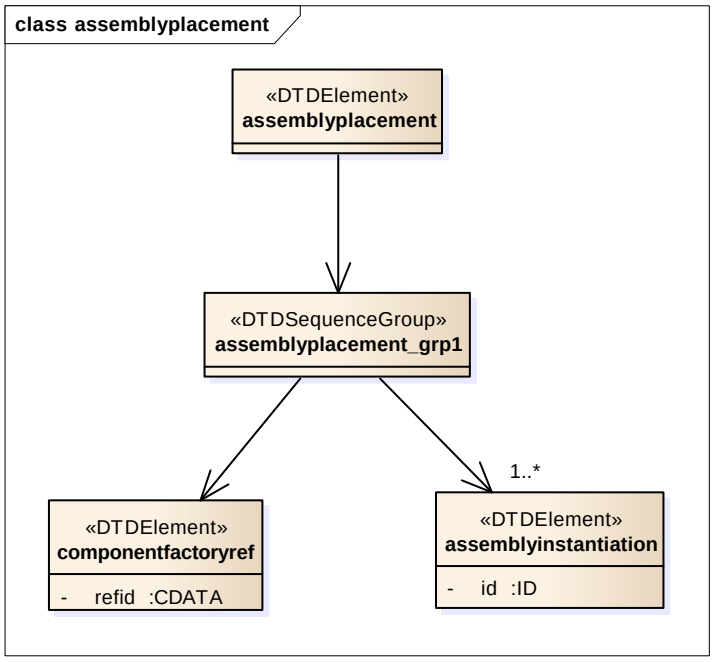


Figure 26: assemblyplacement Element Relationships

```
<!ELEMENT assemblyplacement
     ( componentfileref
     , assemblyinstantiation+
)>
```

D-1.10.1.3.3.1 assemblyinstantiation

The *assemblyinstantiation* element (see **Figure 27**) describes an instantiation of an assembly as a nested sub-application relative to an *assemblyplacement* element. This *assemblyinstantiation* (which itself can potentially contain other nested sub-applications) can then be connected and controlled through the *assemblycontroller* and external ports elements defined in the referenced SAD file. *Assemblyinstantiation* creation can be viewed as essentially similar to a normal application created by an ApplicationFactoryComponent, with the omission of the registration of the sub-application's ApplicationManagerComponent with the DomainManagerComponent.

The *assemblyinstantiation's id* attribute is an implementation specific value that uniquely identifies the assembly within a Domain Profile. The *assemblyinstantiation* element's id may be referenced by the *usesport* and *providesport* elements within the enclosing SAD file when connecting to / from ports listed in the sub-application's SAD *externalports* element.

The *assemblyinstantiation* element contains a number of sub-elements used by the core framework in the creation, deployment and configuration of the sub-application. Most of these elements appear only as sub-elements of the *assemblyinstantiation* element, and are described here.

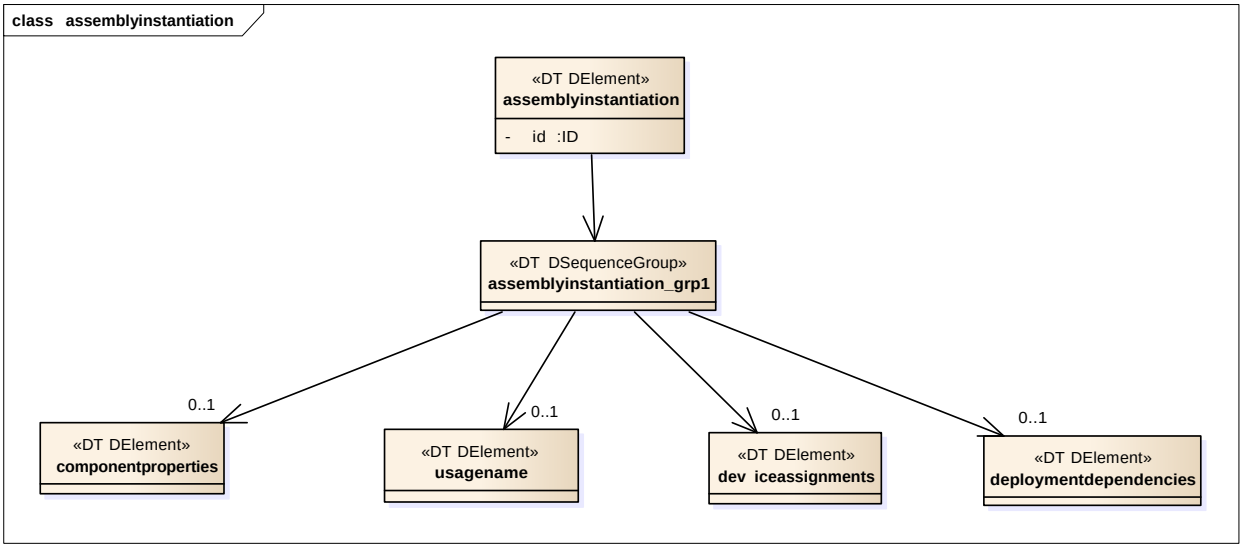


Figure 27: *assemblyinstantiation* Element Relationships

```
<!ELEMENT assemblyinstantiation
     ( usagename?
     , componentproperties?
     , deviceassignments?
     , deploymentdependencies?
)>

<!ATTLIST assemblyinstantiation
     id   ID   #REQUIRED>
```

D-1.10.1.3.3.1.1componentproperties

The optional *componentproperties* element (see **Figure 28**) is a list of configure, factoryparam, and/or execparam property values that are used in creating and / or initially configuring the components of the sub-application. For valid properties with a *kindtype* of “execparam” or “factoryparam”, or “configure” properties with a *mode* attribute of “readwrite” or “writeonly”, values will supplement or (if of the same name), override sub-application values following the given precedence order:

1. The outer SAD partitioning : *assemblyplacement : assemblyinstantiation* element
6. The nested SAD *componentinstantiation* / *assemblyinstantion* element
2. (N/A for factoryparam values) The value or default value, if any, from the component’s SPD using the properties precedence stated in D-1.6.1.

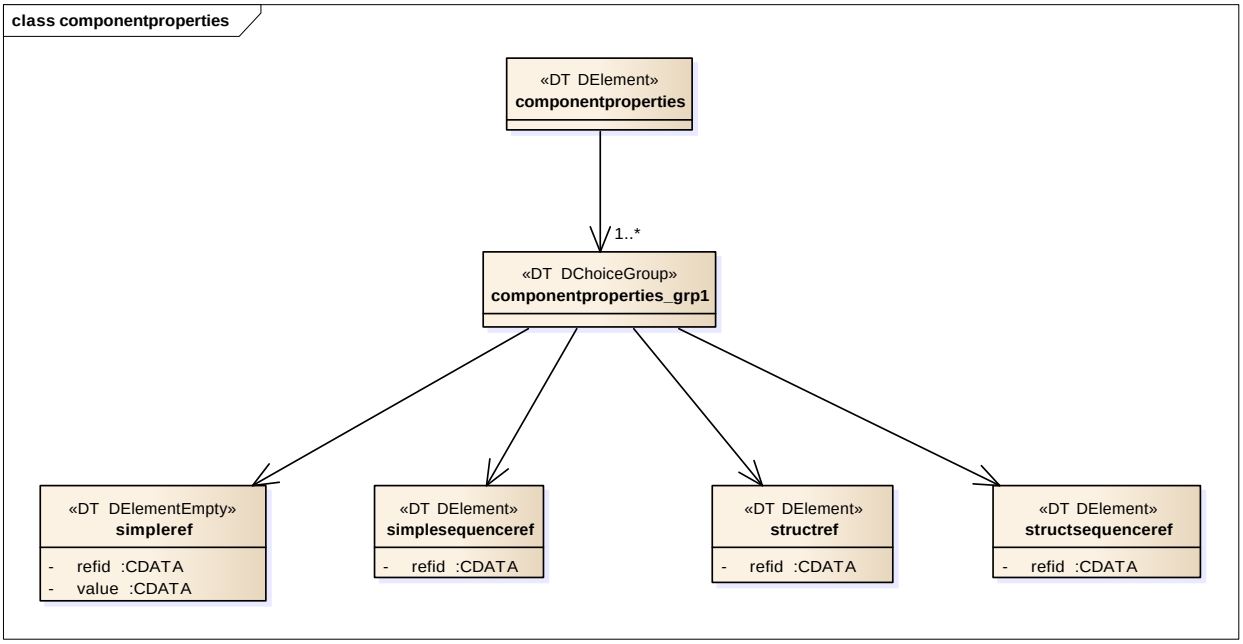


Figure 28: *componentproperties* Element Relationships

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

D-1.10.1.3.3.1.2 deviceassignments

The optional *deviceassignments* element provides a list of *deviceassignment* elements which are used when deploying the sub-application's components.

In a *deviceassignment* element, the *componentid* attribute refers to the *componentinstantiationref* within the scope of the sub-application being created, while the *assignedDeviceid* refers a device identifier (Device Configuration Descriptor (DCD) c*omponentinstantiation id*) in the domain.

```
<!ELEMENT deviceassignments
     ( deviceassignment  )+>

<!ELEMENT deviceassignment EMPTY>
<!ATTLIST deviceassignment
     componentid         CDATA     #REQUIRED
     assignedDeviceid    CDATA     #REQUIRED>
```

D-1.10.1.3.3.1.3 deploymentdependencies

The *deploymentdependencies* element (described generically in section D-1.10.1.4) overrides referenced SAD dependencies.

D-1.10.1.4 deploymentdependencies

The *deploymentdependencies* element (see **Figure 29**) is cited in multiple places within the SAD. It supplies, within its governing scope, overriding values for like-named dependencies defined within the scope. This allows the addition of scope-dependent deployment information, e.g. specification of a particular radio channel or security domain for the deployed application, sub-

application or component. Each *dependencies* element is a *propertyref* referencing a property of *kind* "allocation", and overrides any values from

- narrower-scope deploymentdependencies
- SPD-defined dependency propertyref elements (see section D-1.6.1.6.10.2)
- SPD-defined usesdevice propertyref elements (see section D-1.6.1.6.11.1).

When, for any *componentinstantiation*, there are no matching dependencies (same id) specified in the SPD file, then the *deploymentdependencies*-supplied *propertyref* is not used to constrain deployment. In this way, dependencies can be overridden where they are specified, while not imposing new dependencies where they are not intended.

Deployment dependencies precedence order in order of highest to lowest is:

1. *Application Factory::create* deploymentdependencies parameter
2. *Assemblyinstantiation deploymentdependencies*
3. *Componentinstantiation deploymentdependencies*
4. SAD deployment *deploymentdependencies*
5. *Componentinstantiation* SPD *dependency* and *usesdevice*

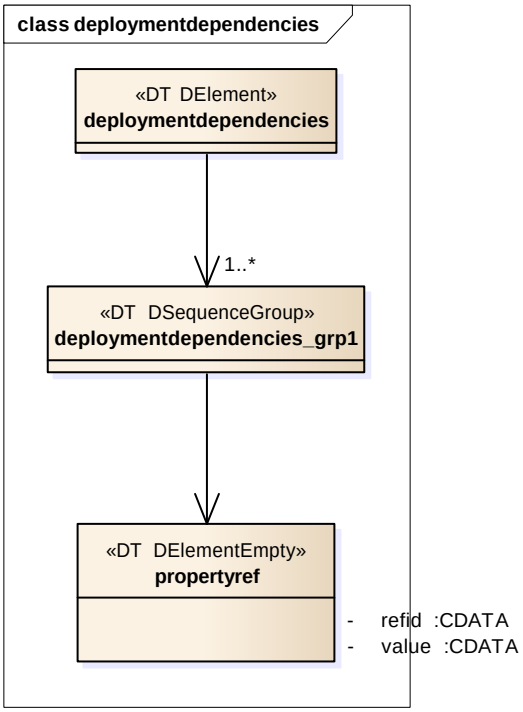


Figure 29: *deploymentdependencies* Element Relationships

```
<!ELEMENT deploymentdependencies
     (propertyref
     )+>
```

D-1.10.1.5 assemblycontroller

The *assemblycontroller* element (see Figure 30) indicates the *componentinstantiation* and / or *assemblyinstantiation*(s) that form the control point(s) for the assembly. The ApplicationManagerComponent delegates its *Resource*::*configure*, *query*, *start*, *stop*, and *runTest* operations to the elements identified by the *assemblycontroller* element.

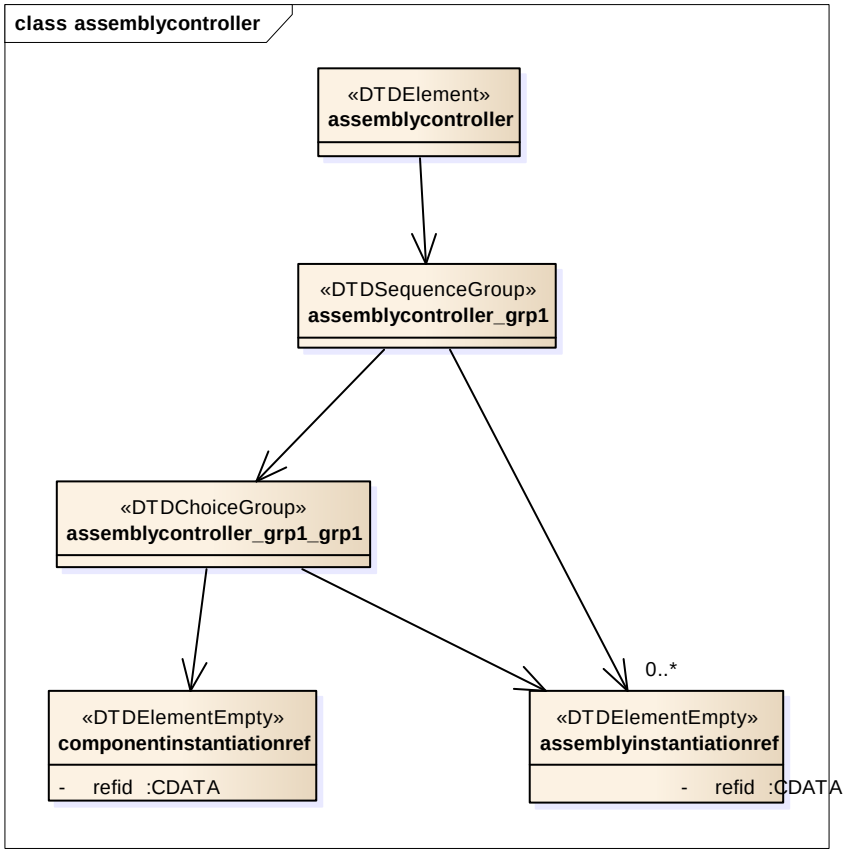


Figure 30: *assemblycontroller* Element Relationships

```
<!ELEMENT assemblycontroller
     (( componentinstantiationref | assemblyinstantiationref),
     assemblyinstantiationref* )>
```

D-1.10.1.6 connections

The *connections* element is a child element of the *softwareassembly* element. The *connections* element is intended to provide the connection map between components in the assembly.

```
<!ELEMENT connections
     ( connectinterface* )>
```

D-1.10.1.6.1connectinterface

The *connectinterface* element (see **Figure 31**) is used when application components are being assembled to describe connections between their port interfaces. The *connectinterface* element consists of a *usesport* element and a *providesport* or *componentsupportedinterface* element. These elements are intended to connect two compatible components.

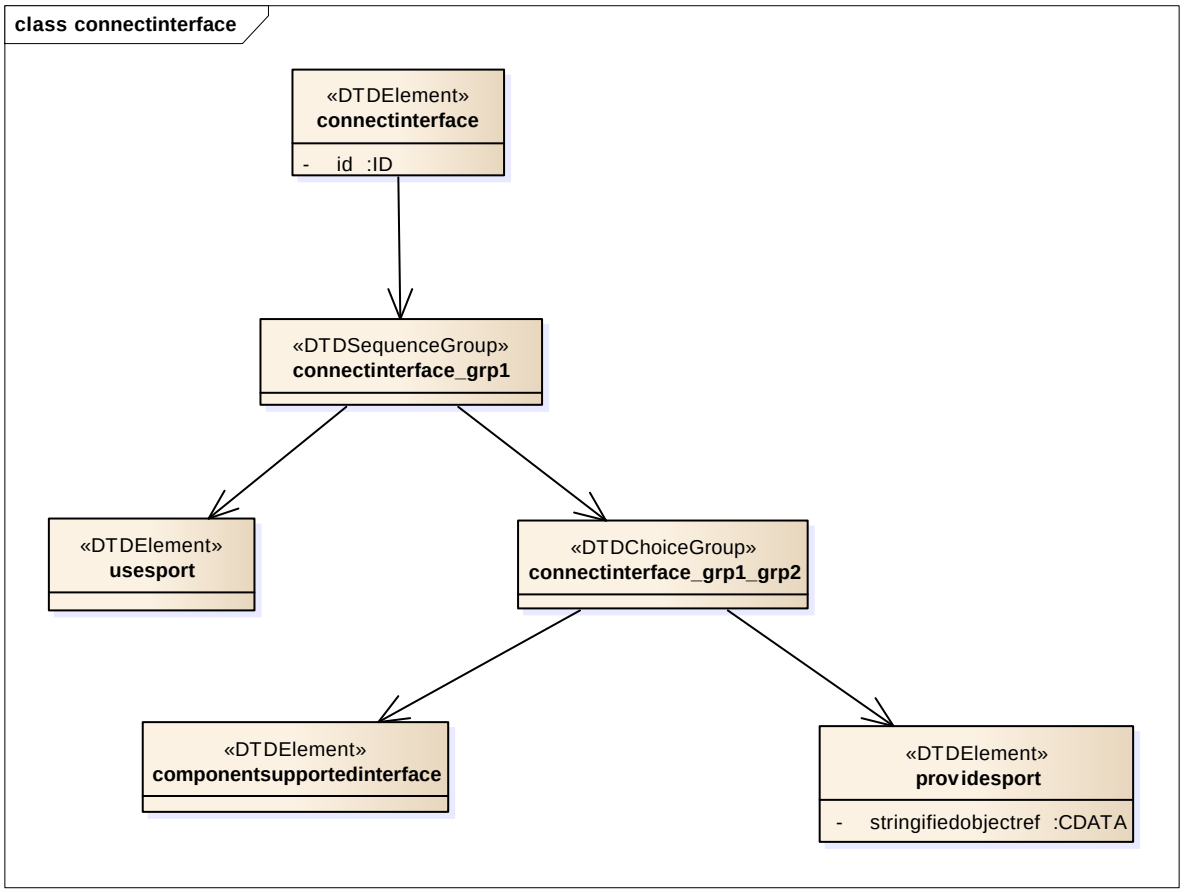


Figure 31: *connectinterface* Element Relationships

```
<!ELEMENT connectinterface
     ( usesport
     , ( providesport | componentsupportedinterface )
     )>
<!ATTLIST connectinterface
     id   ID   #IMPLIED>
```

D-1.10.1.6.1.1 *usesport*

The *usesport* element (see **Figure 32**) identifies, using the *identifier* element, the component port that is using the provided interface from the *providesport* element. A *Resource* type component may be referenced by one of five elements: *componentinstantiationref*, *assemblyinstantiationref*, *domainfinder*, *devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref*.

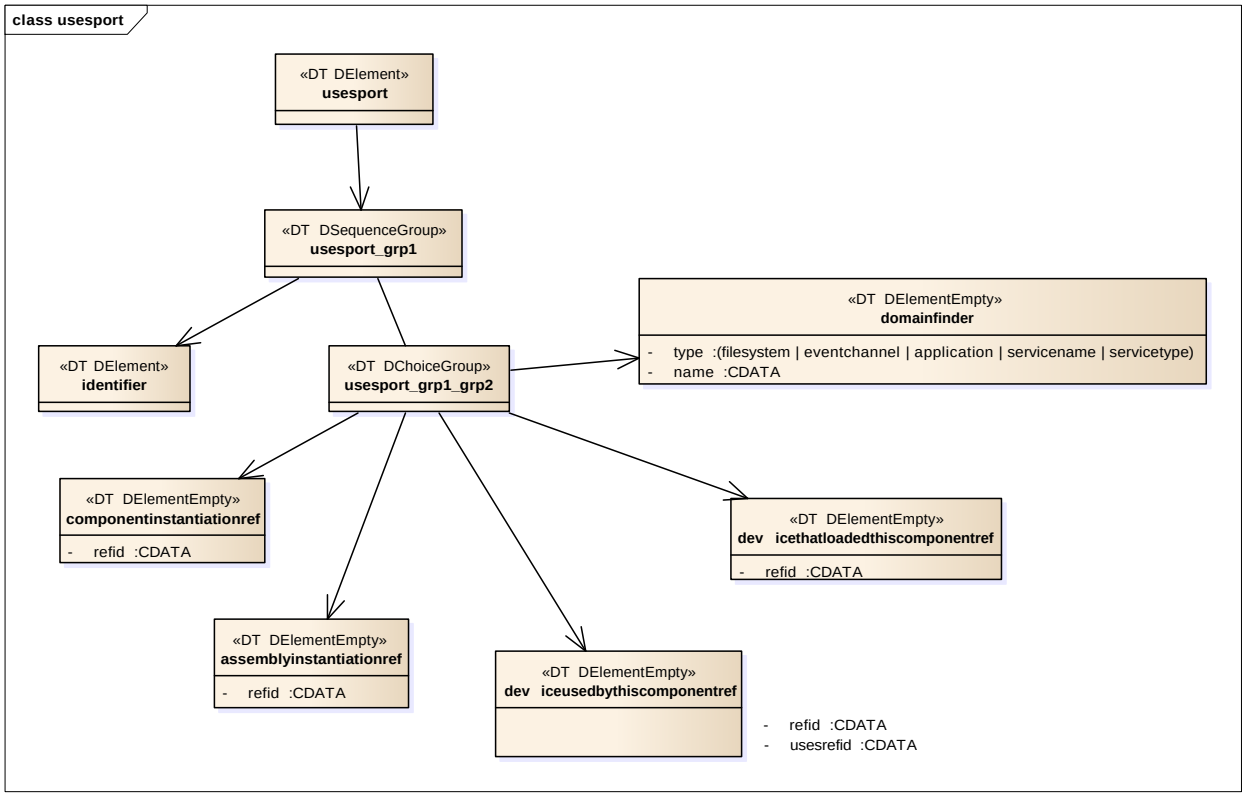


Figure 32: *usesport* Element Relationships

```
<!ELEMENT usesport
     (identifier
     , ( componentinstantiationref |
     assemblyinstantiationref |
     devicethatloadedthiscomponentref |
     deviceusedbythiscomponentref | domainfinder )
)>
```

D-1.10.1.6.1.1.1identifier

The *identifier* element identifies which “uses port” on the component is to participate in the connection relationship. This identifier will correspond with an *id* for one of the component ports specified in the SCD.

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

D-1.10.1.6.1.1.2 componentinstantiationref

The *componentinstantiationref* element refers to the *id* attribute of the *componentinstantiation* element within the SAD file. The *refid* attribute will correspond to the unique *componentinstantiation id* attribute.

```
<!ELEMENT componentinstantiationref EMPTY>
<!ATTLIST componentinstantiationref
     refid     CDATA     #REQUIRED>
```

D-1.10.1.6.1.1.3 assemblyinstantiationref

The *assemblyinstantiationref* element refers to the *id* attribute of the *assemblyinstantiation* element within the SAD file. The *refid* attribute will correspond to the unique *assemblyinstantiation id* attribute.

```
<!ELEMENT assemblyinstantiationref EMPTY>
<!ATTLIST assemblyinstantiationref
     refid          CDATA     #REQUIRED>
```

D-1.10.1.6.1.1.4devicethatloadedthiscomponentref

The *devicethatloadedthiscomponentref* element refers to a specific component found in the assembly, which is used to obtain the ComponentBaseDevice that was used to load the referenced component from the ApplicationFactoryComponent. The ComponentBaseDevice obtained is then associated with this component instance.

```
<!ELEMENT devicethatloadedthiscomponentref EMPTY>
<!ATTLIST devicethatloadedthiscomponentref
     refid     CDATA     #REQUIRED>
```

D-1.10.1.6.1.1.5deviceusedbythiscomponentref

The *deviceusedbythiscomponentref* element refers to a specific component, within the assembly, which is used to obtain the ComponentBaseDevice (e.g., logical device) that is being used by the specific component from the ApplicationFactoryComponent. This relationship is needed when a component is pushing or pulling data and/or commands to another component that exists in the system such as an audio device.

```
<!ELEMENT deviceusedbythiscomponentref EMPTY>
<!ATTLIST deviceusedbythiscomponentref
     refid          CDATA     #REQUIRED
     usesrefid      CDATA     #REQUIRED>
```

D-1.10.1.6.1.1.6domainfinder

The *domainfinder* element is used to indicate to the ApplicationFactoryComponent the necessary information to find an object reference that is of specific type and may also be known by an optional name within the domain. The valid *type* attributes are “filesystem”, “eventchannel”, “application”, “servicename”, and “servicetype”.

For “filesystem” type when *name* attribute is not supplied then the closest FileSystemComponent proximity-wise (e.g. file system residing on the same physical device as the component on the other end of the connection would be used first) is provided.

The *type* attribute value of “eventchannel” is used to specify the event channel to be used in the OE’s Event Service for producing or consuming events. If the *name* attribute is not supplied and the *type* attribute has a value of “eventchannel” then the Incoming domain management event channel is used.

For “application” type the *name* attribute must be specified. For “application” type the *name* attribute format is ApplicationFactoryComponent name followed by forward slash “/” followed by Application name (e.g. “the_applicationfactory_name/the_application_name”). The options for “application” type name and meaning are:

1. When only the application name is specified then any existing ApplicationManagerComponent in the domain with that name can be used.
2. When both the Application Factory name and Application name (e.g. “the_applicationfactory_name/the_application_name”) is specified the ApplicationManagerComponent with that name created by the specified Application FactoryComponent is returned.
3. When only the Application Factory name followed by a forward slash is specified then any ApplicationManagerComponent created by the specified ApplicationFactoryComponent with that name can be used.

If “servicename” or “servicetype” is specified then name must be specified. Both values correspond to a service identified in a DeviceMangerComponent’s DCD. The DCD *usagename* element contains a value in an “identifier\type” format for a component service type. For “servicename” the name corresponds to the “identifier” portion of the *usagename* element. For “servicetype” the name corresponds to the “type” portion of the *usagename* element. The *type* attribute value of “servicename” is used to locate registered ServiceComponents on a per name basis. The *type* attribute value of “servicetype” is used to locate registered ServiceComponents on a per type basis.

```
<!ELEMENT domainfinder EMPTY>
<!ATTLIST domainfinder
     type (filesystem | eventchannel | application | servicename
     | servicetype) #REQUIRED
     name CDATA     #IMPLIED>
```

D-1.10.1.6.1.2 providesport

The *providesport* element (see Figure 33) identifies, using the *identifier* element, the component port that is provided to the usesport interface within the *connectinterface* element. A *Resource* type component may be referenced by one of five elements: *componentinstantiationref*, *assemblyinstantiationref*, *domainfinder*, *devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref*. The *domainfinder* element by itself is used when the object reference is not a Resource type. The *providesport* element’s *stringifiedobjectref* attribute, when specified, is the component instantiation provides port object reference that does not require component registration but still requires dynamic connections

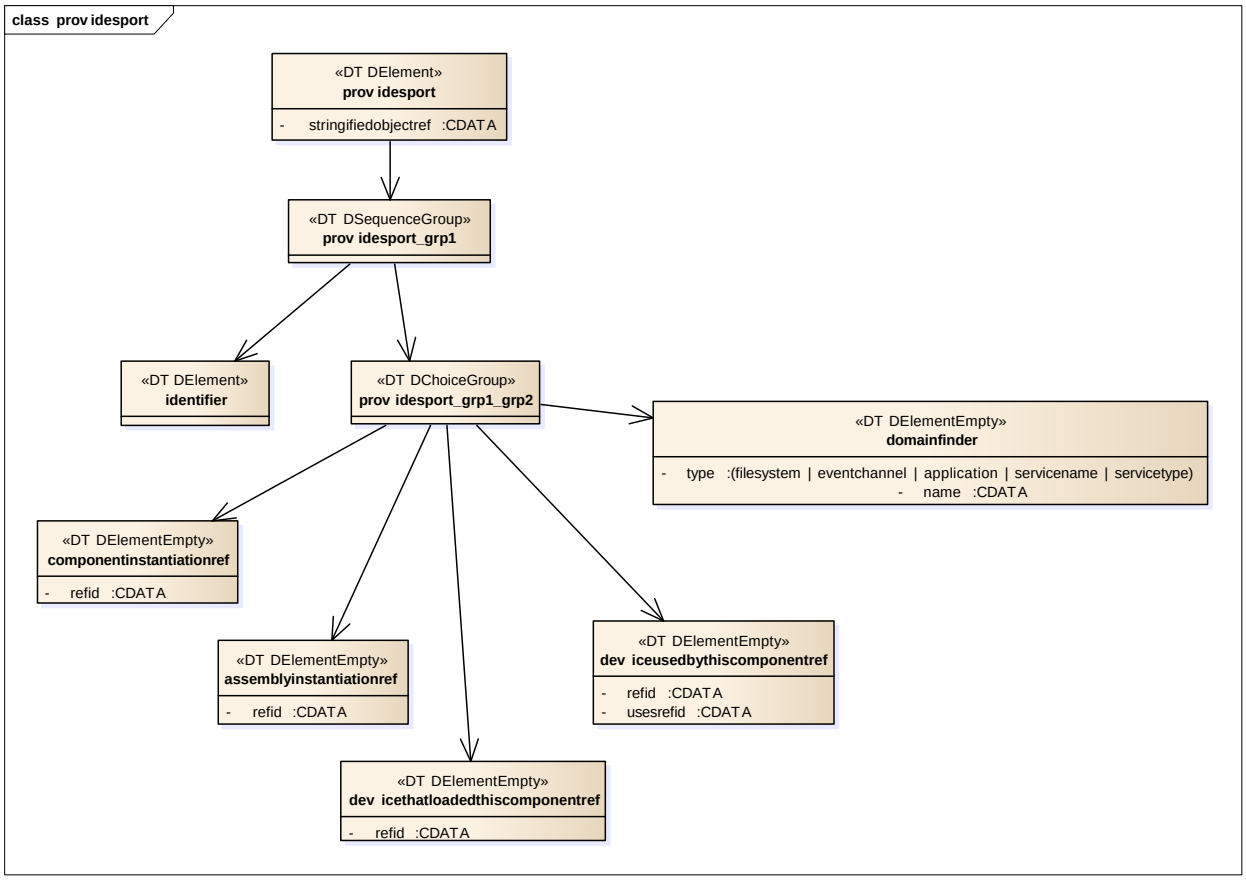


Figure 33: *providesport* Element Relationships

```
<!ELEMENT providesport
(identifier
     , ( componentinstantiationref |
     assemblyinstantiationref |
     devicethatloadedthiscomponentref |
     deviceusedbythiscomponentref | domainfinder )
     )>
<!ATTLIST providesport
     stringifiedobjectref CDATA #implied>
```

D-1.10.1.6.1.2.1identifier

The *identifier* element identifies which “provides port” on the component is to participate in the connection relationship. This identifier will correspond with a *repid* attribute for one of the component ports elements, specified in the SCD.

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

D-1.10.1.6.1.2.2componentinstantiationref

See D-1.10.1.6.1.1.2 for a description of the *componentinstantiationref* element.

D-1.10.1.6.1.2.3assemblyinstantiationref

See D-1.10.1.6.1.1.3 for a description of the *assemblyinstantiationref* element.

D-1.10.1.6.1.2.4devicethatloadedthiscomponentref

See D-1.10.1.6.1.1.4 for a description of the *devicethatloadedthiscomponentref* element.

D-1.10.1.6.1.2.5deviceusedbythiscomponentref

See D-1.10.1.6.1.1.5 for a description of the *deviceusedbythiscomponentref* element.

D-1.10.1.6.1.2.6domainfinder

See section D-1.10.1.6.1.1.6 for a description of the *domainfinder* element.

D-1.10.1.6.1.3 componentsupportedinterface

The *componentsupportedinterface* element (see Figure 34) specifies a component, which has a *supportsinterface* element, that can satisfy an interface connection to a port specified by the *usesport* element, within a *connectinterface* element. A component within the assembly may be referenced by one of four elements: *componentinstantiationref*, *domainfinder*, *devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref*. The *componentinstantiationref* identifies a component within the assembly. The *domainfinder* element points to an existing component that can be found within a DomainManagerComponent.

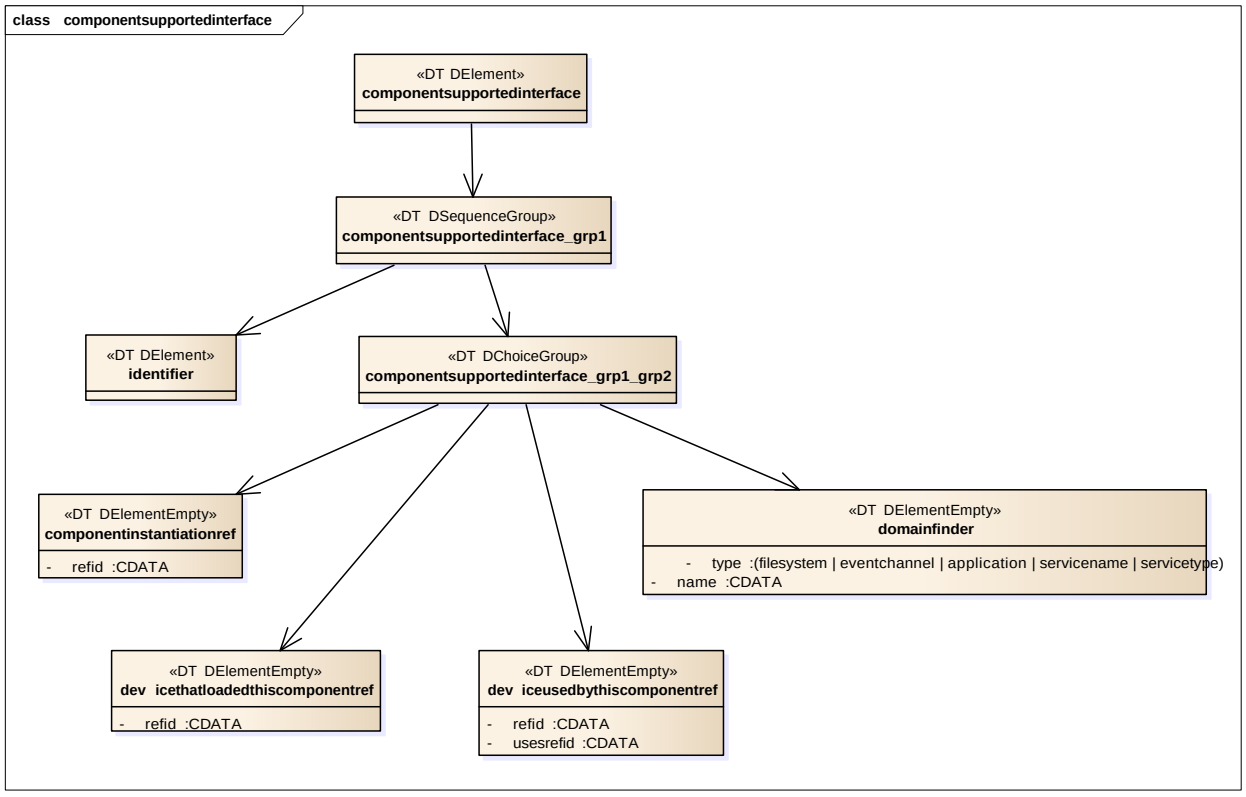


Figure 34: *componentsupportedinterface* Element Relationships

```
<!ELEMENT componentsupportedinterface
(identifier
     , ( componentinstantiationref |
     devicethatloadedthiscomponentref |
     deviceusedbythiscomponentref | domainfinder)
     )>
```

D-1.10.1.6.1.3.1 identifier

The *identifier* element identifies which supported interface on the component is to participate in the connection relationship. This identifier will correspond with the *repid* attribute of one of the component's *supportsinterface* elements, specified in the SCD.

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

D-1.10.1.6.1.3.2 componentinstantiationref

See section D-1.10.1.6.1.1.2 for a description of the *componentinstantiationref* element.

D-1.10.1.6.1.3.3domainfinder

See section D-1.10.1.6.1.1.6 for a description of the *domainfinder* element.

D-1.10.1.7 externalports

The optional *externalports* element (see Figure 35) is a child element of the *softwareassembly* element (see Figure 19). The *externalports* element is used to identify the visible ports for the software assembly. The *Application::getProvidesPorts* operation is used to access the assembly's obtainable provides ports.

The *usesidentifier* element identifies which supported interface of the software assembly is to participate in the connection relationship. This identifier will correspond with an *identifier* element from one of the assembly component's *usesport* definitions.

The *providesidentifier* element identifies which supported interface of the software assembly is to participate in the connection relationship. This identifier will correspond with an *identifier* element from one of the assembly component's *providesport* definitions.

The *supportedidentifier* element identifies which supported interface of the software assembly is to participate in the connection relationship. This identifier will correspond with an *identifier* element from one of the assembly component's *componentsupportedinterface* definitions.

```
<!ELEMENT externalports
     ( port+
     )>
```

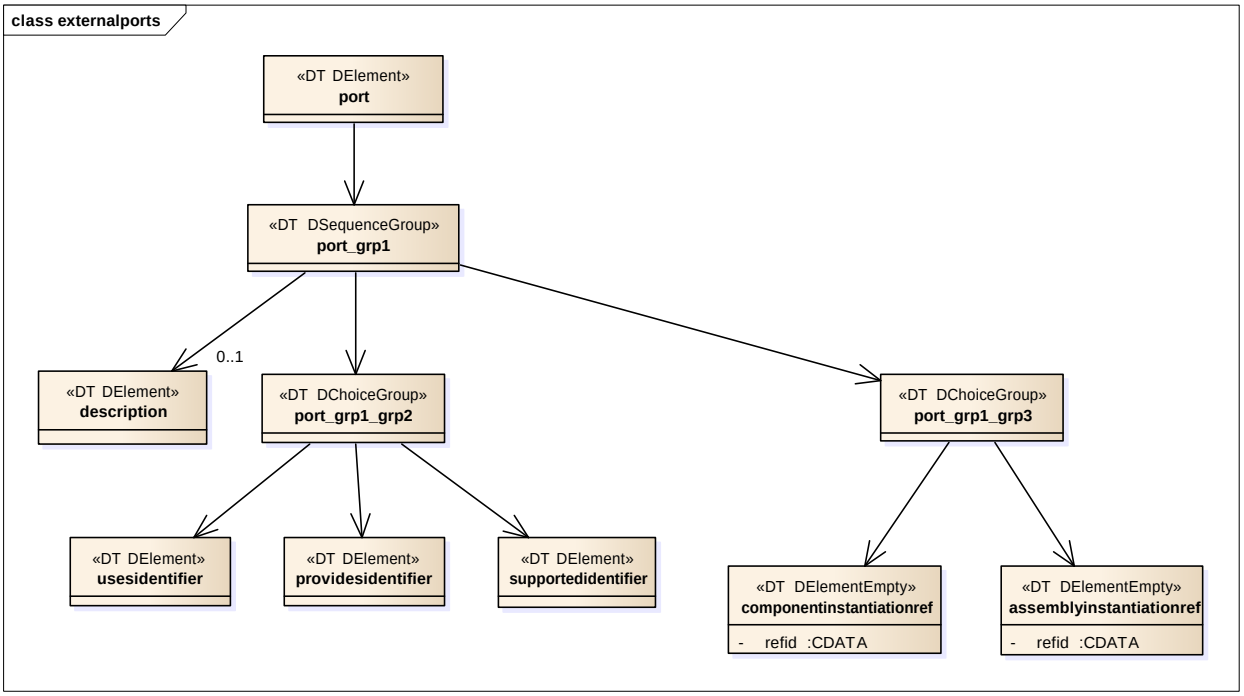


Figure 35: *port* Element Relationships

```
<!ELEMENT port
     ( description?
     , ( usesidentifier | providesidentifier |
     supportedidentifier)
     , (componentinstantiationref | assemblyinstantiationref)
     )>

<!ELEMENT description (#PCDATA)>
<!ELEMENT usesidentifier (#PCDATA)>
<!ELEMENT providesidentifier (#PCDATA)>
<!ELEMENT supportedidentifier (#PCDATA)>
```

D-1.10.1.8 deploymentprefs

The optional *deploymentprefs* element is a reference to a local file. See section D-1.6.1.4.1 for the definition of the *localfile* element. The file refers to an Application Deployment Descriptor (ADD) file.

```
<!ELEMENT deploymentprefs
     ( localfile
     )>
```

D-1.11 DEVICE CONFIGURATION DESCRIPTOR

This section describes the XML elements of the Device Configuration Descriptor (DCD) XML file; the *deviceconfiguration* element (see Figure 36). The DCD is based on the SAD (e.g., componentfiles, partitioning, etc.) DTD. The intent of the DCD is to provide the means of describing the components that are initially started on the DeviceManagerComponent node, how to obtain the DomainManagerComponent reference, connections of services to components (ComponentBaseDevices, DeviceManagerComponent), and the characteristics (file system names, etc.) for a DeviceManagerComponent. The *componentfiles* and *partitioning* elements are optional; if not provided, that means no components are started up on the node, except for a DeviceManagerComponent. If the *partitioning* element is specified then a *componentfiles* element has to be specified also.

SCA497 A Device Configuration Descriptor file shall have a ".dcd.xml" extension.

D-1.11.1 deviceconfiguration

The *deviceconfiguration* element's *id* attribute is a unique identifier within the domain for the device configuration. This *id* attribute is a unique identifier within the Domain Profile for the device configuration. The *name* attribute is the user-friendly name for the DeviceManagerComponent's *label* attribute.

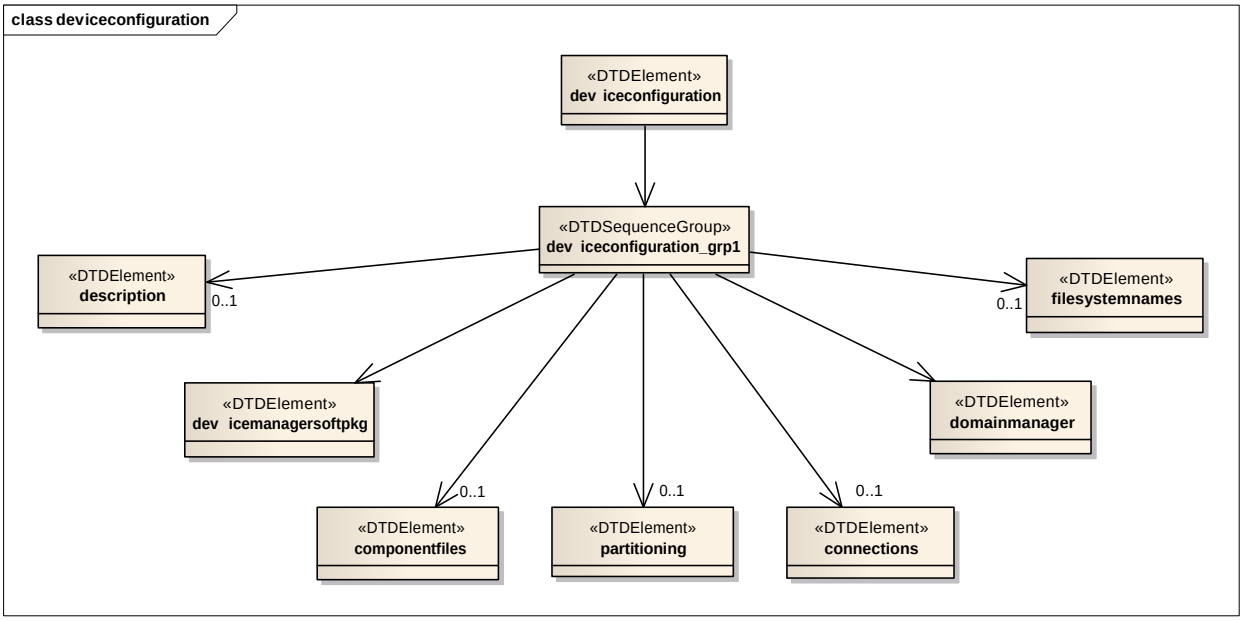


Figure 36: *deviceconfiguration* Element Relationships

```
<!ELEMENT deviceconfiguration
     ( description?
     , devicemanagersoftpkg
     , componentfiles?
     , partitioning?
     , connections?
     , domainmanager
     , filesystemnames?
     )>
<!ATTLIST deviceconfiguration
     id   ID   #REQUIRED
     name CDATA     #IMPLIED>
```

D-1.11.1.1 description

The optional *description* element, of the *deviceconfiguration* element, may be used to provide information about the device configuration.

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

D-1.11.1.2 devicemanagersoftpkg

The *devicemanagersoftpkg* element refers to the SPD for the DeviceManagerComponent that corresponds to this DCD. The SPD file is referenced by a *localfile* element. The referenced file can be used to describe the *DeviceManager* realization and to specify the *usesports* for the services (Log*(s)*, etc.) used by the DeviceManagerComponent. See section D-1.6.1.4.1 for a description of the *localfile* element.

```
<!ELEMENT devicemanagersoftpkg
     ( localfile
     )>
```

D-1.11.1.3 componentfiles

The optional *componentfiles* element is used to reference deployment information for components that are started up on the device. The *componentfile* element references a SPD. The SPD, for example, can be used to describe ComponentBaseDevices, a DeviceManagerComponent, a DomainManagerComponent, and FileSystemComponents. The XML definition of the DCD's *componentfiles* element is the same as the one provided in the SAD's section D-1.10.1.2, see that section for the XML definition of the *componentfiles* element.

D-1.11.1.4 partitioning

The optional *partitioning* element consists of a set of *componentplacement* elements. A component instantiation is captured inside a *componentplacement* element.

```
<!ELEMENT partitioning
     ( componentplacement )*>
```

D-1.11.1.4.1componentplacement

The *componentplacement* element (see Figure 37) is used to define a particular deployment of a component. The *componentfileref* element identifies the component to be deployed. The *componentinstantiation* element identifies the actual component created. Multiple components of the same kind can be created within the same *componentplacement* element.

The optional *deployondevice* element indicates the device on which the *componentinstantiation* element is deployed. The optional *compositepartofdevice* element indicates the parent device of the *componentinstantiation* element. When the component is a logical device, the optional *devicepkgfile* element indicates the hardware device information for the logical device.

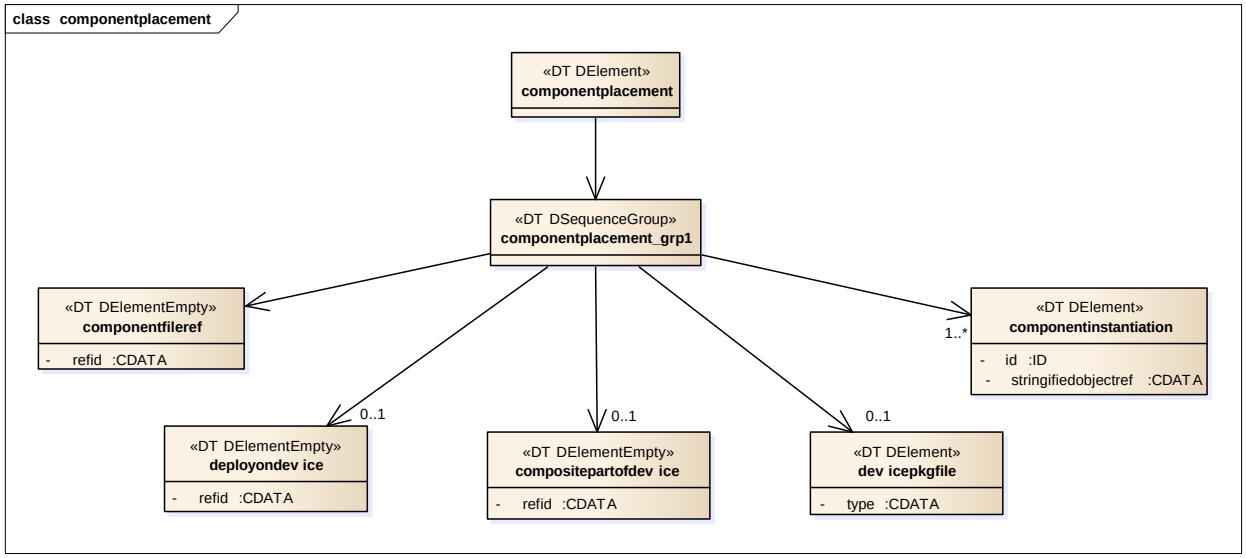


Figure 37: *componentplacement* Element Relationships

```
<!ELEMENT componentplacement
     ( componentfileref
     , deployondevice?
     , compositepartofdevice?
     , devicepkgfile?
     , componentinstantiation+
     )>
```

D-1.11.1.4.1.1 *componentfileref*

The *componentfileref* element is used to reference a *componentfile* element within the *componentfiles* element. The *componentfileref* element's *refid* attribute corresponds to a *componentfile* element's *id* attribute.

```
<!ELEMENT componentfileref  EMPTY>
<!ATTLIST componentfileref
     refid     CDATA     #REQUIRED>
```

D-1.11.1.4.1.2 *deployondevice*

The *deployondevice* element is used to reference a *componentinstantiation* element on which this *componentinstantiation* is deployed.

```
<!ELEMENT deployondevice  EMPTY>
<!ATTLIST deployondevice
     refid     CDATA     #REQUIRED>
```

D-1.11.1.4.1.3 *compositepartofdevice*

The *compositepartofdevice* element is used when a parent-child relationship exists between devices to reference the *componentinstantiation* element that describes the parent device when this device's *componentinstantiation* element describes the child device.

```
<!ELEMENT compositepartofdevice  EMPTY>
<!ATTLIST compositepartofdevice
     refid     CDATA     #REQUIRED>
```

D-1.11.1.4.1.4 *devicepkgfile*

The *devicepkgfile* element is used to refer to a device package file that contains the hardware device definition.

```
<!ELEMENT devicepkgfile
     ( localfile
     )>
<!ATTLIST devicepkgfile
     type CDATA     #IMPLIED>
```

D-1.11.1.4.1.4.1*localfile*

See D-1.6.1.4.1 for a definition of the *localfile* element.

D-1.11.1.4.1.5 componentinstantiation

The *componentinstantiation* element (see Figure 38) is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The

componentinstantiation's *id* attribute is an implementation specific value that uniquely identifies the component. The *componentinstantiation* element's stringifiedobjectref attribute, when specified, is the component instantiation object reference that requires dynamic connections. The *componentinstantiation* contains a *usagename* element that is intended for an applicable name for the component.

For a component service (e.g., Log Service implementation), the *usagename* element is not optional. For ServiceComponents, *usagename* must be provided in an "identifier\type" format. The "identifier" portion of the name must be unique for each service instantiation. The "type" value is common across all instantiations of the same service. The "type" value should be representative of the service that is being provided such as the name or the interfaces. The value "log" represents the type for a Log Service implementation.

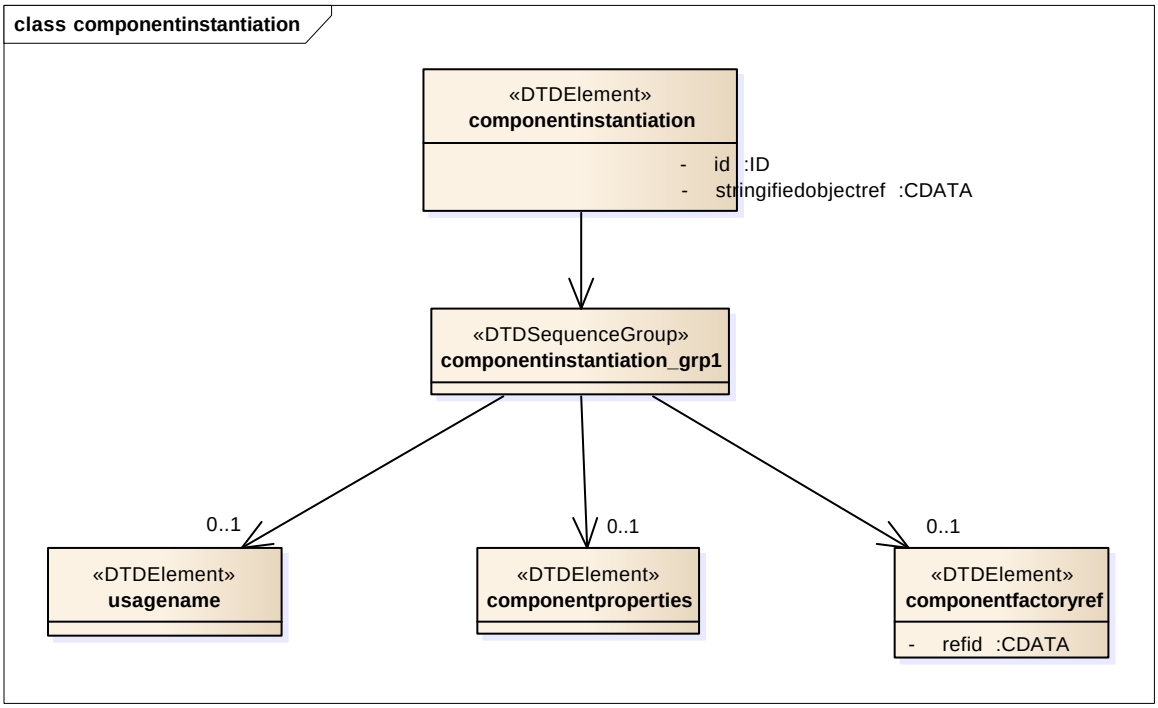


Figure 38: *componentinstantiation* Element Relationships

```
<!ELEMENT componentinstantiation
     ( usagename?
     ,componentproperties?
     ,componentfactoryref?
     )>
<!ATTLIST componentinstantiation
     id   ID   #REQUIRED
     stringifiedobjectref CDATA #implied>

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

D-1.11.1.4.1.5.1*componentproperties*

The optional *componentproperties* element (see Figure 39) is a list of property values with a *kind*type of "configure", "execparam" or "allocation") and a *mode* attribute of "readwrite" or

"writeonly" that are used either during component configuration or execution or for overriding allocation properties. D-1.10.1.3.1.2 defines the property list for the *componentinstantiation* element, which contains initial properties values.

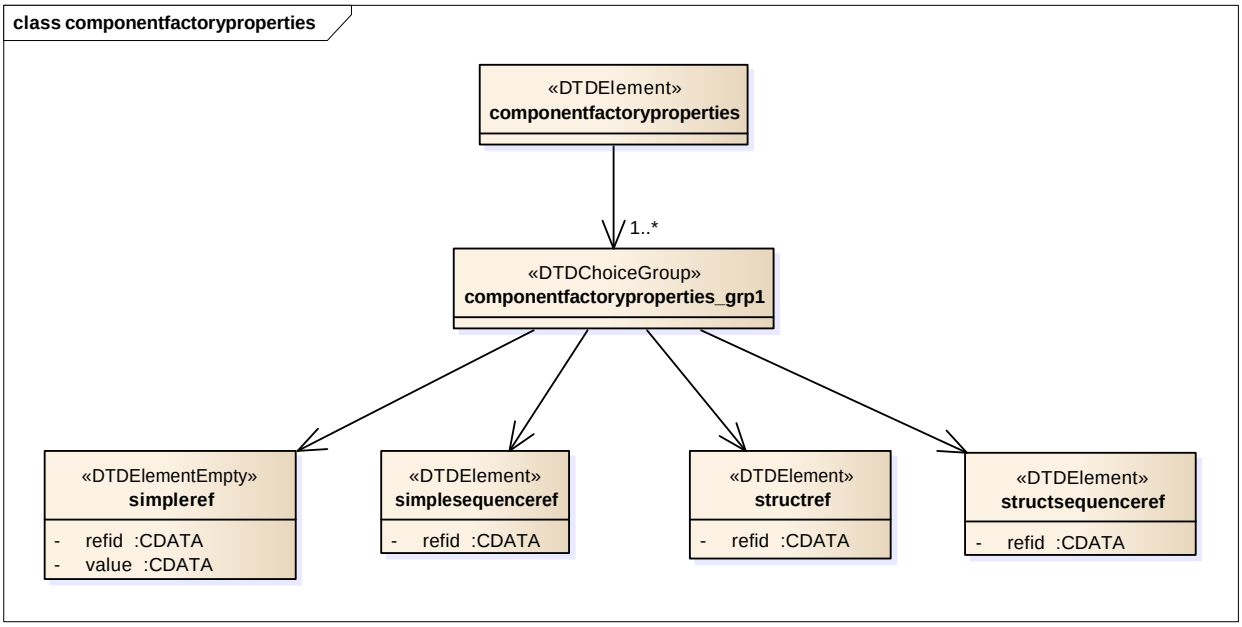


Figure 39: *componentproperties* Element Relationships

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

D-1.11.1.4.1.5.2componentfactoryref

The optional *componentfactoryref* element (see Figure 40) refers to a particular PlatformComponentFactoryComponent *componentinstantiation* element found in the DCD, which is used to obtain a ComponentBaseDevice or ServiceComponent for this *componentinstantiation* element. The *refid* attribute refers to a unique *componentinstantiation id* attribute. The optional *componentfactoryref* element should be specified only when a PlatformComponentFactoryComponent is used to create ComponentBaseDevice or ServiceComponents.

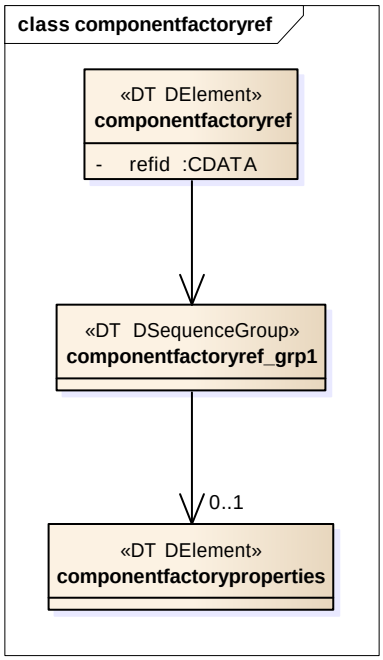


Figure 40: *componentfactoryref* Element Relationships

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

D-1.11.1.4.1.5.2.1 componentfactoryproperties

The optional *componentfactoryproperties* element (see Figure 41) specifies the properties "qualifiers", for the PlatformComponentFactoryComponent *createComponent* call.

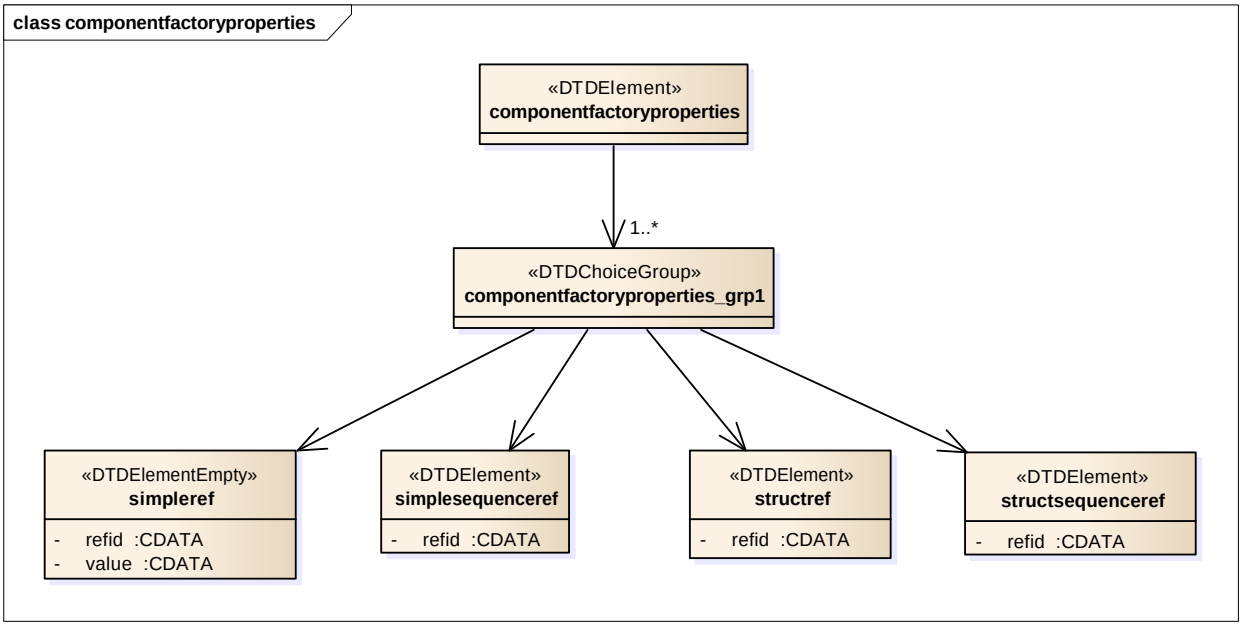


Figure 41: *componentfactoryproperties* Element Relationships

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

<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
     refid     CDATA     #REQUIRED
     value     CDATA     #REQUIRED>

<!ELEMENT simplesequenceref
     ( values )>
<!ATTLIST simplesequenceref
     refid     CDATA     #REQUIRED>

<!ELEMENT structref
     ( simpleref+ )>
<!ATTLIST structref
     refid     CDATA     #REQUIRED>

<!ELEMENT structsequenceref
     ( structvalue+ )>
<!ATTLIST structsequenceref
     refid     CDATA     #REQUIRED>

<!ELEMENT structvalue
```

```
     ( simpleref+ )>

<!ELEMENT values
     ( value+ )>

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

D-1.11.1.5 connections

The *connections* element in the DCD is the same as the *connections* element in the SAD in section D-1.10.1.6. The *connections* element in the DCD is used to indicate the services (Log, etc.) instances that are used by the DeviceManagerComponent and ComponentBase*Devices* in the DCD. To establish connections to a DeviceManagerComponent, the DCD's *deviceconfiguration* element's *id* attribute value is used for the SAD's *usesport* element's *componentinstantiationref* element's *refid* attribute value.

D-1.11.1.6 domainmanager

The *domainmanager* element indicates how to register to the DomainManagerComponent. The value of the *name* attribute is based upon the *type* attribute, which could be the name of a file. The *type* attribute indicates the type of mechanism used for obtaining the DomainManagerComponent registration reference such as the localfile that contains an ior, ior_string, or resolve_initial_reference by ORB.

```
<!ELEMENT domainmanager EMPTY>

<!ATTLIST domainmanager
     name CDATA     #REQUIRED
     type CDATA     #REQUIRED>
```

D-1.11.1.7 filesystemnames

The optional *filesystemnames* element indicates the mounted file system names for DeviceManagerComponent's FileManagerComponent.

The optional *filesystemnames* element indicates the names for file systems mounted within a DeviceManagerComponent's FileManagerComponent. The *mountname* attribute contains a file system name that uniquely identifies a mount point. The *deviceid* attribute is the unique identifier for a specific component, within the DCD, which represents the device hosting this file system. The use of the *deviceid* attribute value is implementation dependent.

```
<!ELEMENT filesystemnames
     ( filesystemname+   )>

<!ELEMENT filesystemname EMPTY>
<!ATTLIST filesystemname
     mountname CDATA     #REQUIRED
     deviceid  CDATA     #REQUIRED>
```

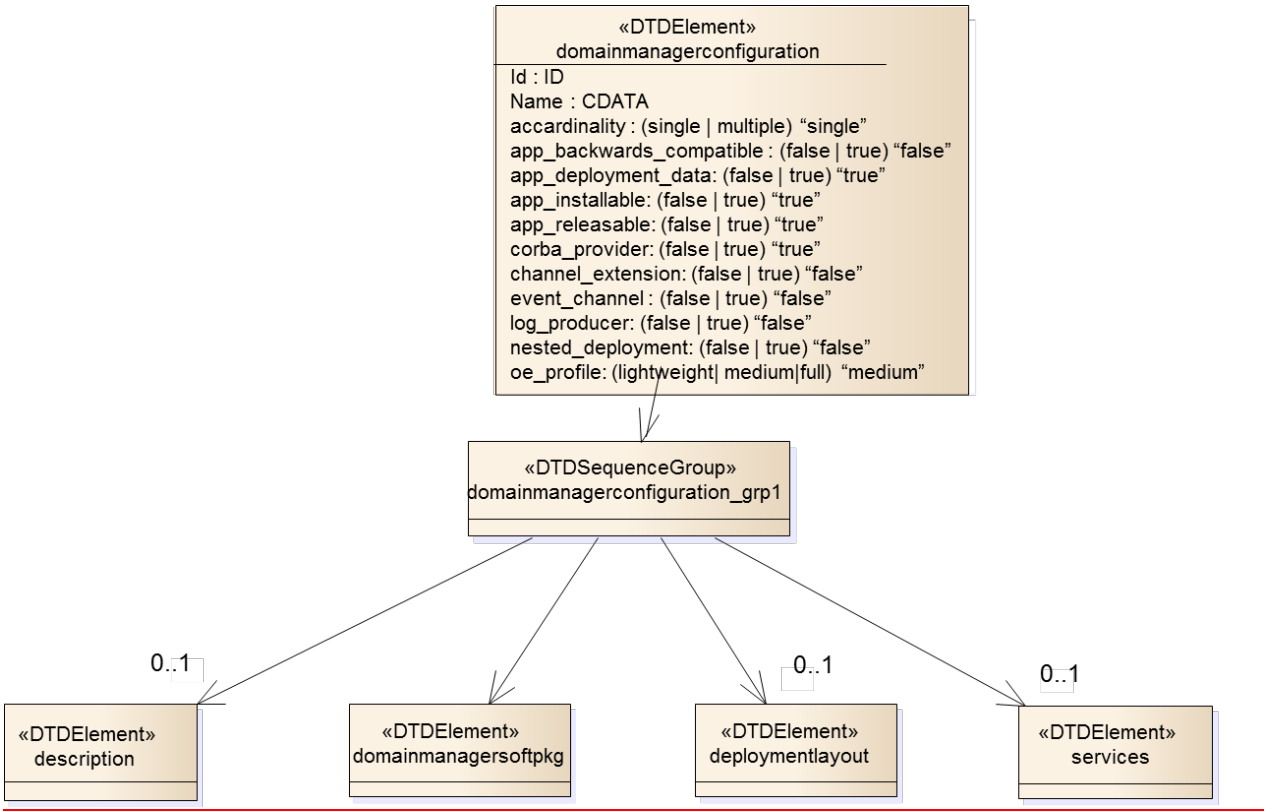
D-1.12 DOMAIN MANAGER CONFIGURATION DESCRIPTOR

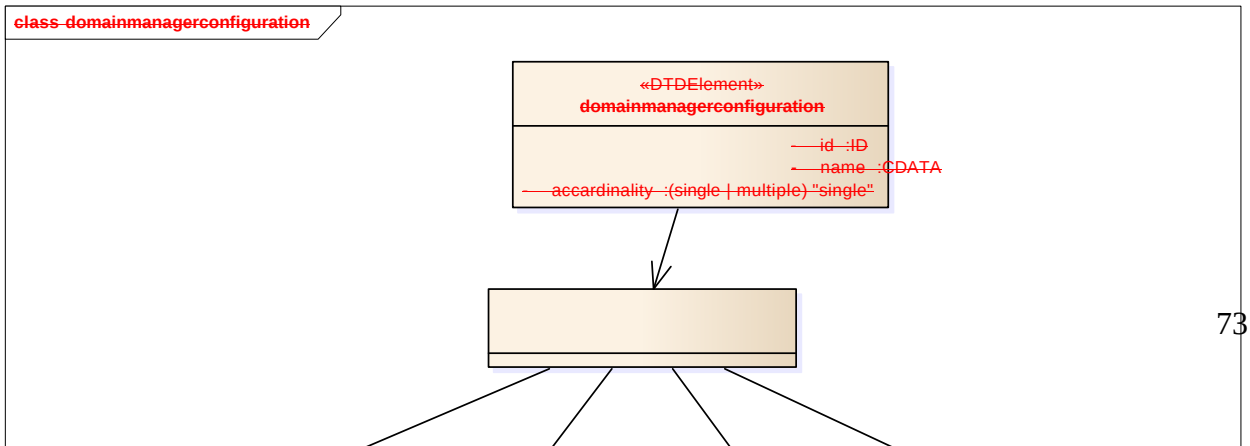
This section describes the XML elements of the Domain Manager Configuration Descriptor (DMD) XML file.

SCA498 A DomainManager Configuration Descriptor file shall have a ".dmd.xml" extension.

D-1.12.1 domainmanagerconfiguration

The *domainmanagerconfiguration* element (see **Figure 42**) *id* attribute is an implementation specific value that uniquely identifies the DomainManagerComponent. The app_backwards_compatible, app_deployment_data, app_installable, app_releasable, corba_provider, channel_extension, event_channel, log_producer, and nested_deployment attributes indicate the optional Units of Functionality (UOF) supported by the DomainManagerComponent. Accardinality indicated whether multiple assembly controllers are supported. The oe_profile attribute indicates the operating environment profile supported by the DomainManagerComponent.





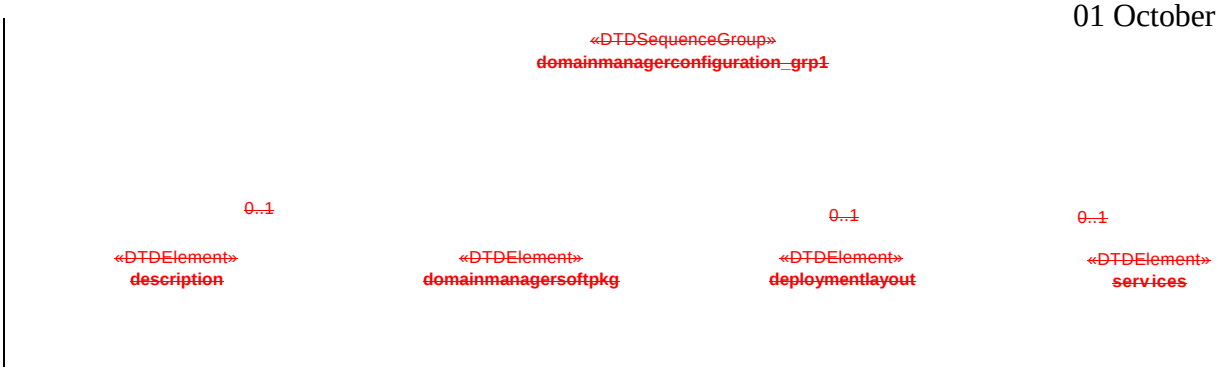


Figure 42: *domainmanagerconfiguration* Element Relationships

```
<!ELEMENT domainmanagerconfiguration
     ( description?
     , domainmanagersoftpkg
     , deploymentlayout?
     , services?
     )>
<!ATTLIST domainmanagerconfiguration
     id   ID   #REQUIRED
     name CDATA     #REQUIRED
     accardinality (single | multiple) "single"
    app_backwards_compatible  (false | true) “false”
    app_deployment_data (false | true) “true”
    app_installable (false | true) “true”
    app_releasable (false | true) “true”
    corba_provider (false | true) “true”
    channel_extension (false | true) “false”
    channel_extension  (false | true) “false”
    log_producer (false | true) “false”
    nested_deployment (false | true) “false”
    oe_profile (lightweight| medium|full) “medium”>
```

D-1.12.1.1 description

The optional *description* element of the DMD may be used to provide information about the configuration.

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

D-1.12.1.2 domainmanagersoftpkg

The *domainmanagersoftpkg* element refers to the SPD for the DomainManagerComponent. The SPD file is referenced by a *localfile* element. This SPD can be used to describe the DomainManagerComponent implementation and to specify the *usesports* for the services (Log*(s)*, etc.) used by the DomainManagerComponent. See section D-1.6.1.4.1 for description of the *localfile* element.

```
<!ELEMENT domainmanagersoftpkg
     ( localfile ) >
```

D-1.12.1.3 deploymentlayout

The optional *deploymentlayout* element is a reference to a local file. See section D-1.6.1.4.1 for the definition of the *localfile* element. The file refers to a Platform Deployment Descriptor (PDD) file.

```
<!ELEMENT deploymentlayout
     ( localfile
     )>
```

D-1.12.1.4 services

The optional *services* element (see Figure 43) in the DMD is used by the DomainManagerComponent to determine which service (Log, etc.) instances to use; it makes use of the *service* element.

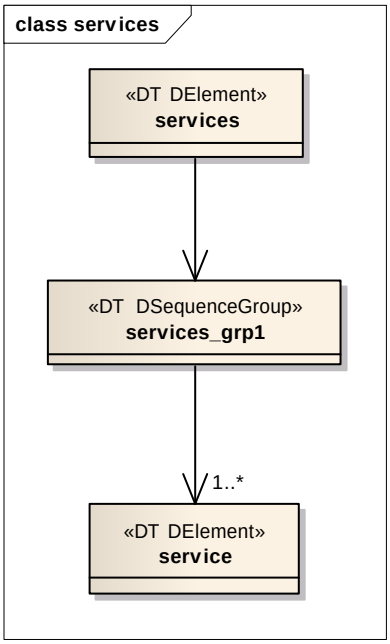


Figure 43: *services* Element Relationships

```
<!ELEMENT services
     ( service+ ) >
```

D-1.12.1.4.1service

The *service* element (see **Figure 44**) defines the service instance. See section D-1.10.1.6.1.1.6 for a description of the *domainfinder* element. See section D-1.10.1.6.1.1.1 for a description of the *identifier* element.

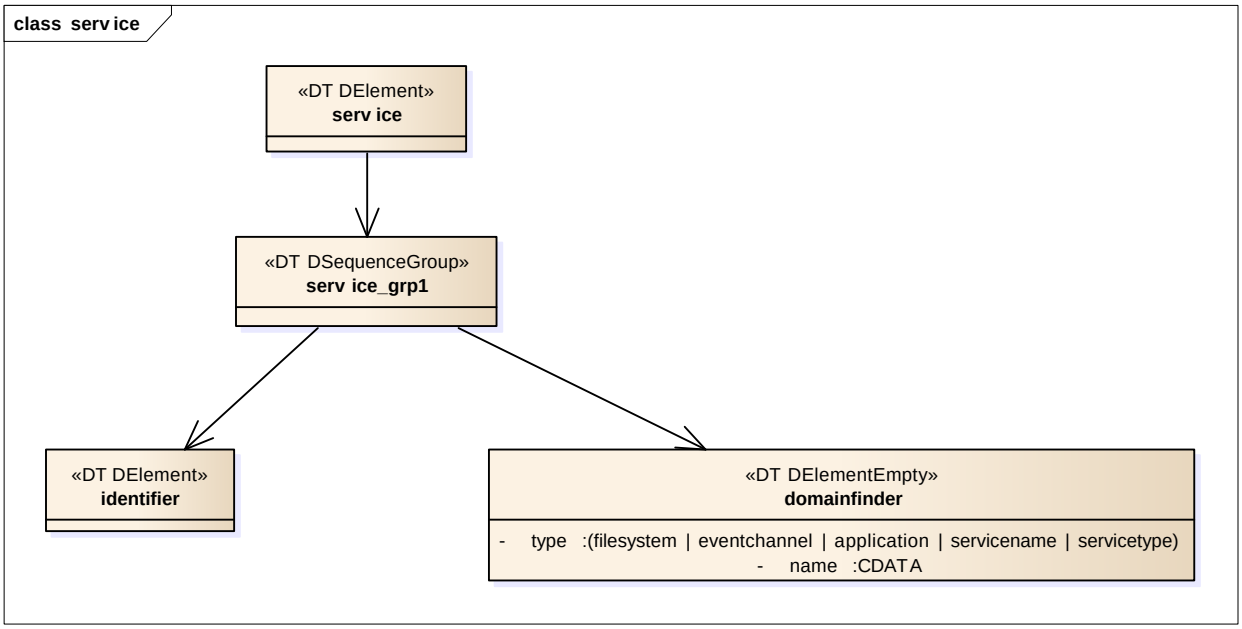


Figure 44: *service* Element Relationships

```
<!ELEMENT service
     ( identifier
     , domainfinder)>
```

D-1.13 PLATFORM DEPLOYMENT DESCRIPTOR

This section describes the XML elements of the Platform Deployment Descriptor (PDD) XML file; the *deploymentplatform* element. The intent of the PDD is to provide a means of describing the collection of services and devices that are associated with a virtual channel. The knowledge of the channel composition can be utilized as part of an overall systems engineering strategy to control the allocation of applications to system resources. Another use of the information could be to improve the efficiency of application deployment as the channel elements would be used to constrain the search space for the allocation of individual application components. The use of the PDD is optional within a system, a system designer is free to use allocation properties or other approaches to manage the allocation of application components to platform resources.

SCA499 A Platform Deployment Descriptor file shall have a ".pdd.xml" extension.

D-1.13.1 deploymentplatform

The *deploymentplatform* element (see **Figure 45**) contains the layout of the virtual channels within a platform domain.

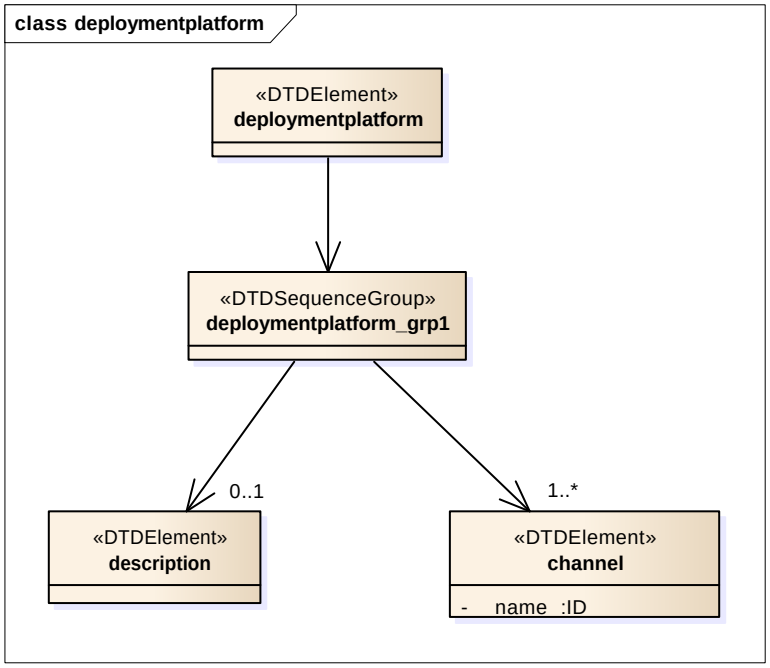


Figure 45: *deploymentplatform* Element Relationships

```
<!ELEMENT deploymentplatform
     ( description?
     , channel+
     ) >
```

D-1.13.1.1 description

The optional *description* element of the PDD may be used to provide information about the platform domain.

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

D-1.13.1.2 channel

The *channel* element (see **Figure 46**) in the PDD defines the collections of devices and services that are used by the *ApplicationFactory* as target resource pools for application deployment. The *channel* element's *name* attribute contains the identifier for the channel that is used by the *ApplicationFactory* and the ADD.

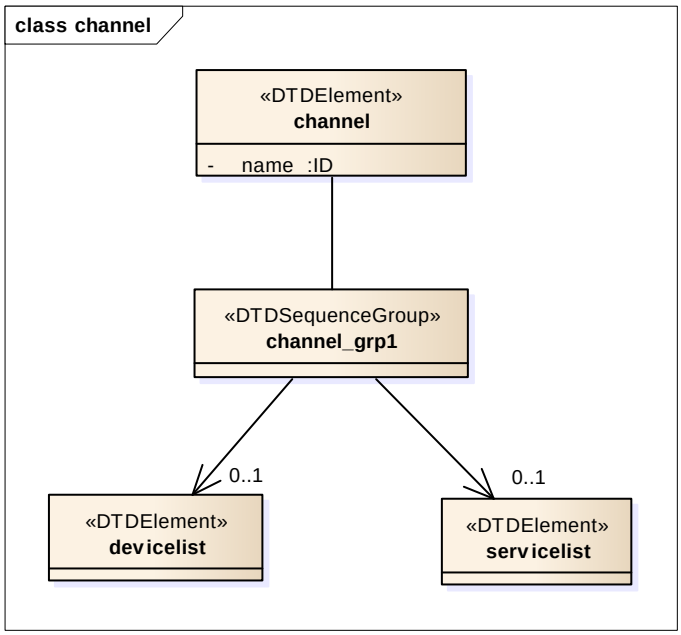


Figure 46: *channel Element* Relationships

```
<!ELEMENT channel
     ( devicelist?
     , servicelist?
     ) >
<!ATTLIST channel
     name ID   #REQUIRED>
```

D-1.13.1.2.1devicelist

The optional *devicelist* element in the PDD defines the collection of devices for a given channel that are used by the *ApplicationFactory* as target resource pools for application deployment.

```
<!ELEMENT devicelist
     ( deviceref+
     )>
```

D-1.13.1.2.1.1 *deviceref*

The *deviceref* element is used to reference a *componentinstantiation* element which is part of the channel. The *refid* attribute points to a *componentinstantiation identifier* for a device that has registered with the platform.

```
<!ELEMENT deviceref EMPTY>
<!ATTLIST deviceref
     refid          CDATA          #REQUIRED>
```

D-1.13.1.2.2servicelist

The optional *servicelist* element in the PDD defines the collection of services for a given channel that are used by the *ApplicationFactory* as target resource pools for application deployment.

```
<!ELEMENT servicelist
     ( serviceref+
     )>
```

D-1.13.1.2.2.1 *serviceref*

The *serviceref* element identifies a service which is part of the channel. The *servicename* attribute is identical to the "identifier" portion *usagename* identifier for a service that has registered with the platform (for a component service type *usagename* is provided in an "identifier\type" format).

```
<!ELEMENT serviceref      EMPTY>
<!ATTLIST serviceref
     servicename          CDATA          #REQUIRED>
```

D-1.14 APPLICATION DEPLOYMENT DESCRIPTOR

This section describes the XML elements of the Application Deployment Descriptor (ADD) XML file; the *deploymentprecedence* element. The intent of the ADD is to provide prioritized lists of deployment alternatives for application instances.

SCA500 An Application Deployment Descriptor file shall have an ".add.xml" extension.

D-1.14.1 deploymentprecedence

The *deploymentprecedence* element (see **Figure 47**) contains the relationship between application instances and their candidate virtual channels.

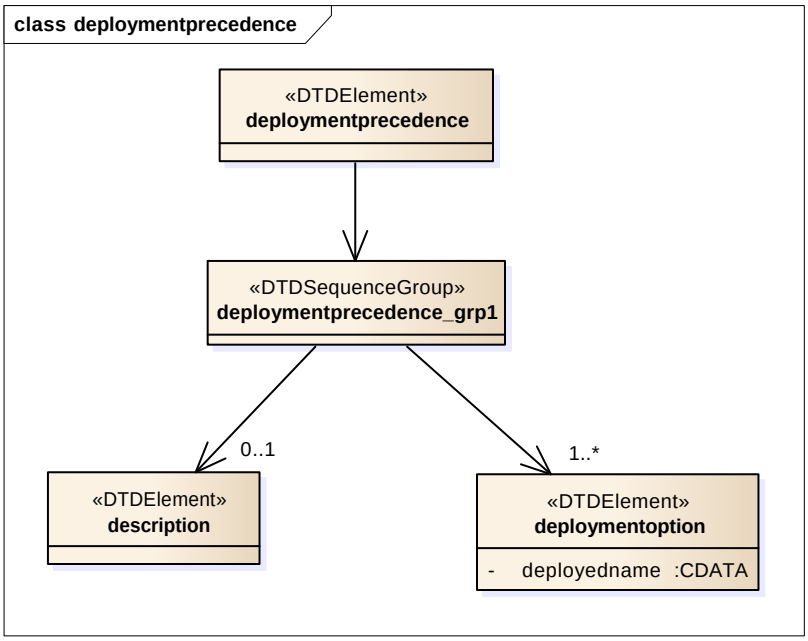


Figure 47: *deploymentprecedence* Element Relationships

```
<!ELEMENT deploymentprecedence
     ( description?
     , deploymentoption+
     ) >
```

D-1.14.1.1 description

The optional *description* element of the ADD may be used to provide information about the application.

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

D-1.14.1.2 deploymentoption

The *deploymentoption* element (see **Figure 48**) in the ADD identifies the ordered list of channels that provide deployment alternatives for a specific application instance. The *deployedname* attribute corresponds to a named application instance (e.g. the name parameter passed to the *ApplicationFactory*::*create* operation).

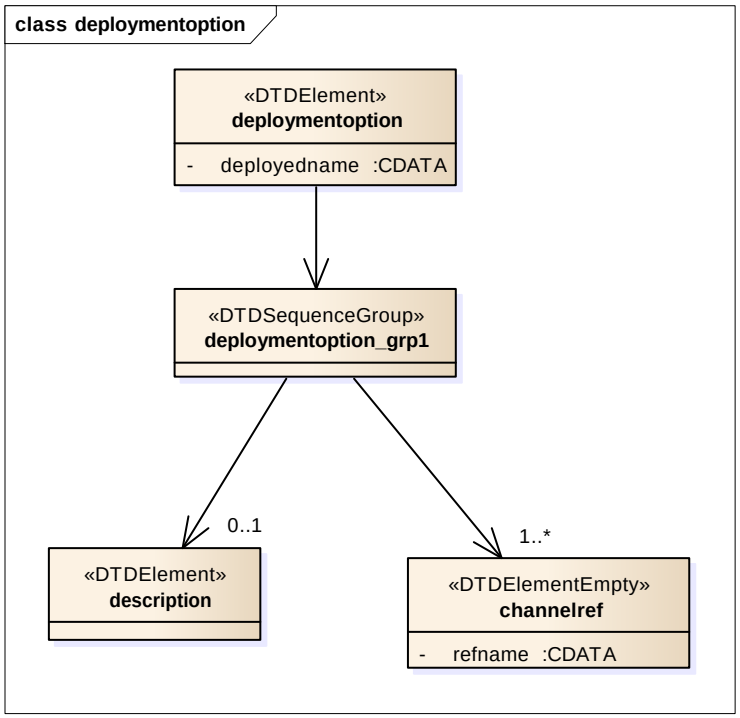


Figure 48: *deploymentoption* Element Relationships

```
<!ELEMENT deploymentoption
     ( description?
     , channelref+
     )>
<!ATTLIST deploymentoption
     deployedname   CDATA      #REQUIRED>
```

D-1.14.1.2.1description

The optional *description* element may be used to provide information about the application instance.

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

D-1.14.1.2.2channelref

The *channelref* element is used to reference a *channel* element from the PDD which provides a deployment alternative. The *refname* attribute points to a *channel* element *name* attribute that identifies a channel.

```
<!ELEMENT channelref EMPTY>
<!ATTLIST channelref
     refname   CDATA     #REQUIRED>
```

D-1.15 ATTACHMENTS

This appendix includes the following:

- Attachment 1: Common Properties Definitions