Date: March 2007

Component Document Type Definitions Specification, v1.0

OMG Available Specification
formal/07-03-03

The PIM and PSM for Software Radio Components Specification (formal/07-03-01) is physically partitioned into 5 volumes:

Communication Channel and Equipment (formal/07-03-02)
Component Document Type Definitions (formal/07-03-03)
Component Framework (formal/07-03-04)
Common and Data Link Layer Facilities (formal/07-03-05)
POSIX Profiles (formal/07-03-06)

OBJECT MANAGEMENT GROUP

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its

attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

DISCLAIMER OF WARRANTY

RESTRICTED RIGHTS LEGEND

TRADEMARKS

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this

specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (*http://www.omg.org/technology/agreement.htm*).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

This specification defines the content of a standard set of Data Type Definition (DTD) files for applications, components, and domain and device management. The complete DTD set is contained in Section 7 Document Type Definitions and in reference 3.2.1.1. XML files that are compliant with these DTD files will contain information about the service components to be started up when a platform is power on and information for deploying installed applications. Examples of the kind of information that could be found in a compliant XML files include things such as:

- A component's properties, ports, interfaces, implementations
- An Application's assembly of components and their interconnections
- A Node's Service Components

2 Conformance

Conformance is at the level of usage. An XML file based upon one of the DTDs in this specification conforms if the XML file is well formed, syntax and semantically correct.

3 References

3.1 Normative References

XML: Extensible Markup Language (XML) 1.0 (Second Edition)
W3C Recommendation, 6 October 2000
[http://www.w3.org/]

3.2 Non-normative References

3.2.1 Domain XML Profile

3.2.1.1 Domain XML Profile Files

Domain DTDs XML Files
Formal OMG document number: dtc/2006-04-13
The Object Management Group, December 2006
[http://www.omg.org]

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

Common Object Request Broker Architecture (CORBA)

An OMG distributed computing platform specification that is independent of implementation languages.

Component

A component can always be considered an autonomous unit within a system or subsystem. It has one or more ports, and its internals are hidden and inaccessible other than as provided by its interfaces. A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component exposes a set of ports that define the component specification in terms of provided and required interfaces. As such, a component serves as a type, whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics).

Interface Definition Language (IDL)

An OMG and ISO standard language for specifying interfaces and associated data structures.

Platform

A set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.

Request for Proposal (RFP)

A document requesting OMG members to submit proposals to the OMG's Technology Committee. Such proposals must be received by a certain deadline and are evaluated by the issuing task force.

Unified Modeling Language (UML)

An OMG standard language for specifying the structure and behavior of systems. The standard defines an abstract syntax and a graphical concrete syntax.

5 Symbols and abbreviated terms

Abbreviation	Definition
CF	Component Framework
CORBA	Common Object Request Broker Architecture
DCD	Device Configuration Descriptor
DMD	DomainManagerComponent Configuration Descriptor
DPD	Device Package Descriptor
DSP	Digital Signal Processor
DTD	Document Type Definition
FPGA	Field Programmable Gate Array
I/O	Input/Output
ID	Identification, Identifier
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
ISO	International Standards Organization
N/A	Not Applicable
OMG	Object Management Group
ORB	Object Request Broker
OS	Operating System
SCD	Software Component Descriptor
UML	Unified Modeling Language
UUID	Universally Unique Identifier
XML	eXtensible Markup Language

6 Additional Information

6.1 Changes to Adopted OMG Specifications

The specifications contained in this document require no changes to adopted OMG specifications.

6.2 Acknowledgements

The following organizations (listed in alphabetical order) contributed to this specification:

- BAE Systems

- The Boeing Company
- Blue Collar Objects
- Carleton University
- Communications Research Center Canada
- David Frankel Consulting
- École de Technologie Supérieure
- General Dynamics Decision Systems
- Harris
- ISR Technologies
- ITT Aerospace/Communications Division
- L-3 Communications Corporation
- Mercury Computer Systems
- The MITRE Corporation
- Mobile Smarts
- Northrup Grumman
- PrismTech
- Raytheon Corporation
- Rockwell Collins
- SCA Technica
- Space Coast Communication Systems
- Spectrum Signal Processing
- THALES
- Virginia Tech University
- Zeligsoft
- 88solutions

7 Document Type Definitions

The specification provides architectural specifications for the deployment of communications software into a device. The intent of a device is to provide a re-configurable platform, which can host software components written by various vendors to support user functional services. The specification requires portable software components to provide common information called a domain profile. The intent of this specification is to clearly define to the component developers the requirements of information and format for the delivery of this information. The domain management functions use the component deployment information expressed in the Domain Profile. The information is used to start, initialize, and maintain the applications that are installed into a compliant system.

This specification defines the XML Document Type Definition (DTD) set for use in deploying components. The complete DTD set is in reference 3.2.1.1 Domain XML Profile Files.

7.1 Deployment Overview

The hardware devices and software components that make up a platform are described by a set of XML descriptor files that are collectively referred to as a Domain Profile. Descriptor files are Software Package, Device Package, Properties, Software Component, Software Assembly, Device Configuration, and DomainManagerComponent Configuration. A Software Profile is either a Software Assembly Descriptor (for applications) or a Software Package Descriptor (for all other software components and hardware devices). These descriptor files describe the identity, capabilities, properties, and inter-dependencies of the hardware devices and software components that make up the system. All of the descriptive data about a system is expressed in the XML vocabulary. This document includes a UML diagram of each complex XML element defined. That is, a UML diagram is provided for each element that makes use of more than one type of XML element as a part of its definition. The UML diagram precedes the XML definition that it represents.

Figure 7.1 depicts the relationships between the descriptor files that are used to describe a system’s hardware and software assets. The XML vocabulary within each of these files describes a distinct aspect of the hardware and software assets.

A Software Assembly Descriptor file describes how multiple components of an assembly, i.e., an application, are deployed and interconnected. A Software Assembly Descriptor file is associated with one or more Software Package Descriptor files. Each component of the Software Assembly Descriptor is described in a Software Package Descriptor file. Information about the interfaces that a component publishes and/or consumes is contained in a Software Component Descriptor file. Each Software Component Descriptor file is associated with a Software Package Descriptor file that describes one or more implementations of the software component. Software properties are described in a Properties Descriptor file that may be applicable to all implementations of the component, i.e., associated at the Software Package Descriptor level or applicable to a single implementation of the component.

Two types of files, a Device Package Descriptor, and a Device Configuration Descriptor, describe hardware devices and are known collectively as a Device Profile. The hardware device is described by the Device Package Descriptor. The logical device is described by the Software Package Descriptor. The Device Configuration Descriptor contains the associations between hardware devices and logical devices. A Device Package Descriptor file identifies the class of the device. Property files, associated with Device Package Descriptors, contain information about the properties of the hardware device being deployed such as serial number, processor type, and allocation capacities.

A Device Configuration Descriptor file describes the components that are initially started up on the device and how to find the DomainManagerComponent. Each component of the Device Configuration Descriptor is described in a Software Package Descriptor file.

The DomainManagerComponent Configuration Descriptor file contains a reference to the DomainManagerComponent Software Package Descriptor file.

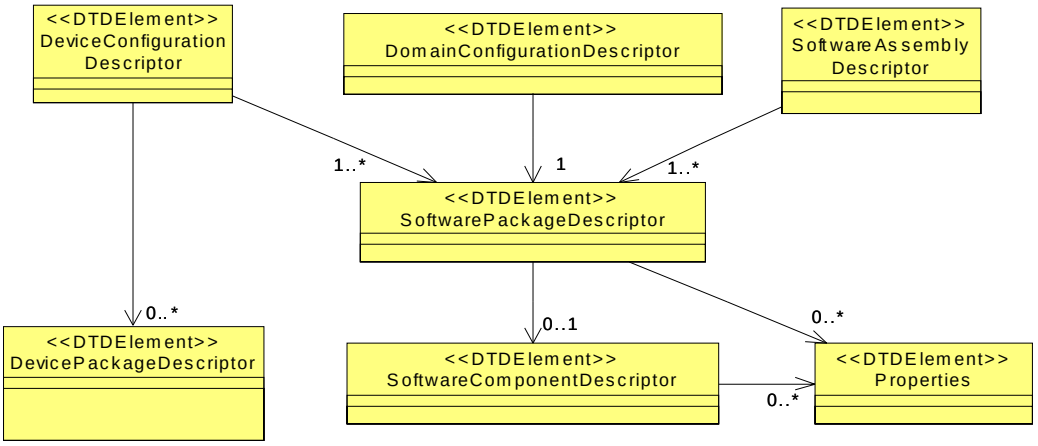

Figure 7.1 - Relationships Between DTD XML File Types

7.2 Software Package Descriptor

The Software Package Descriptor is used at deployment time to load a compliant component and its various implementations. The information contained in the Software Package Descriptor will provide the basis for the Management function to manage the component within a platform or domain.

The software package descriptor may contain various implementations of any given component. Within the specification of a software package descriptor several other files are referenced including a component level *propertyfile* and a software component *descriptor* file. Within any given implementation there may be additional *propertyfiles*.

7.2.1 Software Package

The *softpkg* element (see Figure 7.2) indicates a Software Package Descriptor (SPD) definition. The *softpkg* id uniquely identifies the package. The version attribute specifies the version of the component. The name attribute is a user-friendly label for the *softpkg* element. The name attribute is supplied when the id is not user-friendly such as a DCE UUID. The DCE UUID format starts with the characters “DCE:” and is followed by the printable form of the UUID, a colon, and a decimal minor version number, for example: “DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1.” The decimal minor version number is optional. The type attribute indicates whether or not the component implementation is compliant (e.g., “compliant,” “non_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. The type attribute is used to establish the level of compliance with the spec.

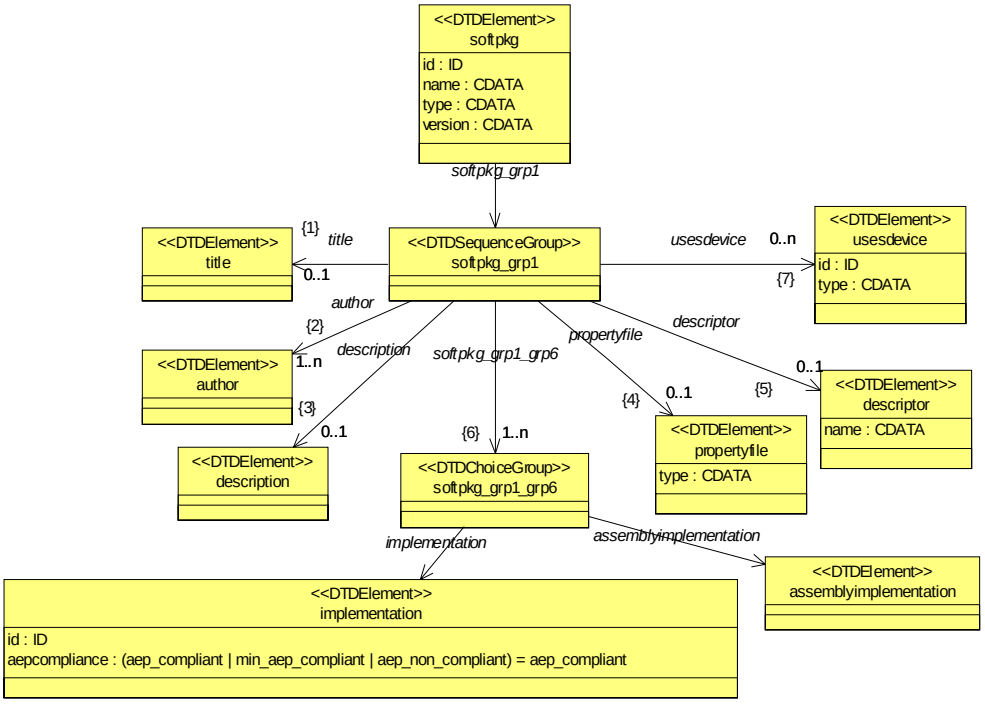

Figure 7.2 - *softpkg* Element Relationships

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

1. SPD Implementation Properties - indicates the implementation values for properties that are specific to one and only one implementation.
2. SPD level properties - indicates the implementation value for a property that is true for all implementations unless over-ridden at the implementation element level.
3. SCD level properties - property definitions for all implementations.

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 SPD-level and implementation-level properties only state what the values are for a component implementation specified in SPD. These property values are not used by ApplicationFactoryComponent for initial configuration of the deployed component and ExecutableProperty(s) for the deployed component main program. ExecutableProperty(s) are used for component construction by the component's main process. These properties are used for the ExecutableDeviceComponent execute operation options parameter.

<!ELEMENT softpkg
(title?
, author+
, description?
, propertyfile?
, descriptor?

```
        , (implementation | assemblyimplementation)+
        , usesdevice*
)>
!ATTLIST softpkg
    id        ID        #REQUIRED
    name    CDATA      #REQUIRED
    type CDATA          #IMPLIED
    version CDATA      #IMPLIED >
```

7.2.1.1 propertyfile

The *propertyfile* element is used to indicate the local filename of the Property Descriptor 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*). Property Descriptor files may also contain *properties* elements that are used in definition of command and control id value pairs used by the CF PropertySet configure() and query() operations. The format of the *properties* element is described in the Properties Descriptor (see Section 7.4, “Properties Descriptor,” on page 21).

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

7.2.1.1.1 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>
```

7.2.1.2 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)>
```

7.2.1.3 author

The *author* element (see Figure 7.3) will be 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.

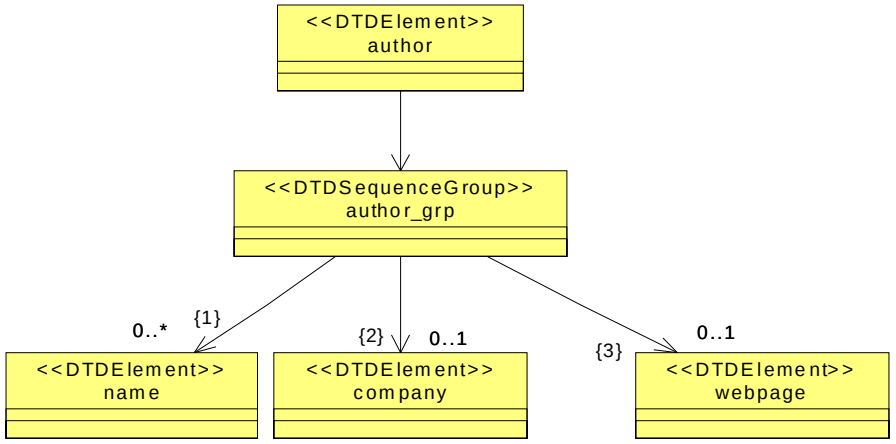

Figure 7.3 - *author* Element Relationships

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

7.2.1.4 description

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

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

7.2.1.5 descriptor

The *descriptor* element points to the local filename of the Software Component Descriptor (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 IDL interfaces). The SCD file is optional, since some SCA components are non-CORBA components, like digital signal processor (DSP) “c” code (see Section 7.5, “Software Component Descriptor,” on page 30).

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

<!ATTLIST descriptor
name CDATA #IMPLIED>

7.2.1.6 implementation

The *implementation* element (see Figure 7.4) 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 CF *Device*, CF *Application*, or CF *Resource* creation. The *implementation* element's id attribute uniquely identifies a specific implementation of the component. The compiler, *programminglanguage*, *humanlanguage*, os, processor, and *runtime* elements are optional dependency elements. The aepcompliance attribute is used to establish the level of compliance with an Application Environment Profiles (AEPs). The aepcompliance indicates if an implementation is compliant with an AEPs and to which one.

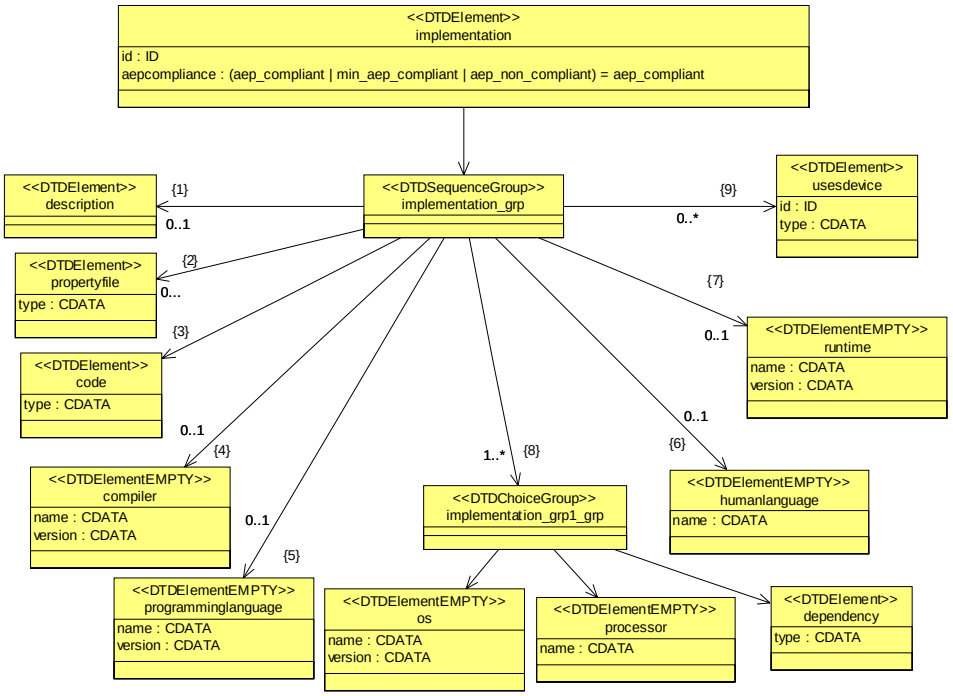

Figure 7.4 - *implementation* Element Relationships

<!ELEMENT implementation
(description?
, propertyfile?
, code
, compiler?
, programminglanguage?
, humanlanguage?
, runtime?
, (os

```
      | processor
      | dependency
      )+
, usesdevice*
    )>
<!ATTLIST implementation
        id   ID   #REQUIRED
aepcompliance (aep_compliant | lwaep_compliant |aep_non_compliant) “aep_compliant”>
```

7.2.1.6.1 propertyfile

The *propertyfile* element is used to indicate the local filename of the Property Descriptor file associated with this *implementation* element. Although the specification does not restrict the specific use of the Property Descriptor 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 ResourceComponent configure() and query() operations. See the description of the *properties* element format in the Properties Descriptor, Section 7.4, “Properties Descriptor,” on page 21.

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

7.2.1.6.2 description

The *description* element will be 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)>
```

7.2.1.6.3 code

The *code* element (see Figure 7.5) will be used to indicate the local filename of the code that is described by the *softpk*g element, for a specific implementation of the software component. The stack size and priority are option parameters used by the ExecutableDeviceComponent *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 LoadableDeviceComponent *load* and ExecutableDeviceComponent *execute*. This is a “main” process.
2. Driver and Kernel Module means load only.
3. SharedLibrary means dynamic linking.
    - Without a code *entrypoint* element means load only.

- With a code *entrypoint* element means load and ExecutableDeviceComponent *execute*.

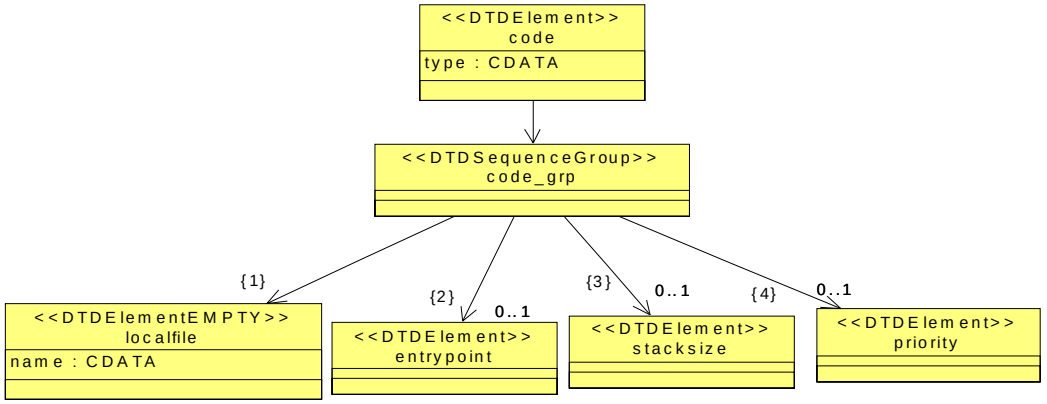

Figure 7.5 - *code* Element Relationships

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

<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
    name   CDATA          #REQUIRED>
<!ELEMENT entrypoint (#PCDATA)>
<!ELEMENT stacksize (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
```

7.2.1.7 compiler

The *compiler* element will be 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>
```

7.2.1.8 programminglanguage

The *programminglanguage* element will be 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>
```

7.2.1.8.1 humanlanguage

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

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

7.2.1.8.2 os

The *os* element will be 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 DeviceComponent's property file as an allocation property (ServiceProperty) 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 DeviceComponent's allocation property (struct sequence Property of struct NameVersionCharacteristic) with the name of os that contains a list of os name and version pairs.

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

7.2.1.8.3 processor

The *processor* element will be 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 DeviceComponent's property file as an allocation property (ServiceProperty) 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 a DeviceComponent's allocation property with a name of processor_name.

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

7.2.1.8.4 dependency

The *dependency* element (see Figure 7.6) is used to indicate the dependent relationships between the components being delivered and other components and devices, in a 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 or propertyvaluesref references a specific ServiceProperty (allocation property), by ServiceProperty identifier, and provide the value that will be used by a ManagedServiceComponent CapabilityModel.

The DomainManagerComponent will use these dependency definitions to assure that ServiceComponent(s) that are necessary for proper operation of the implementation are present and available. The type attribute is descriptive information indicating the type of dependency.

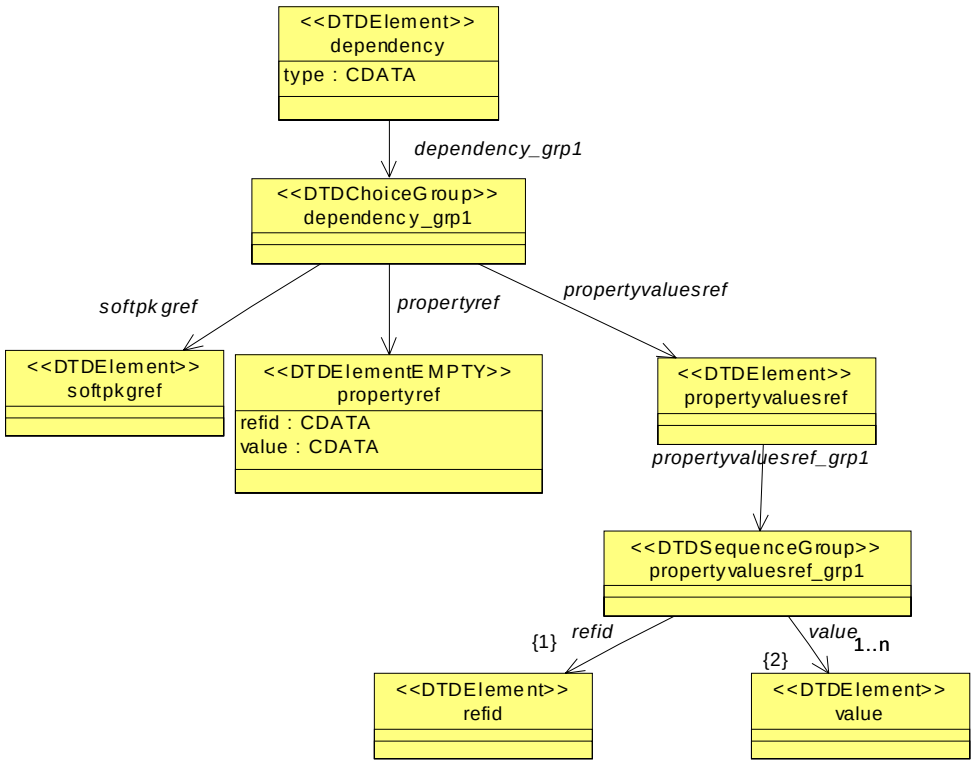

Figure 7.6 - *dependency* Element Relationships

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

7.2.1.8.5 softpkgref

The *softpkgref* element (see Figure 7.7) refers to a *softpkg* element contained in another Software Package Descriptor 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 Software Package Descriptor of the other file.

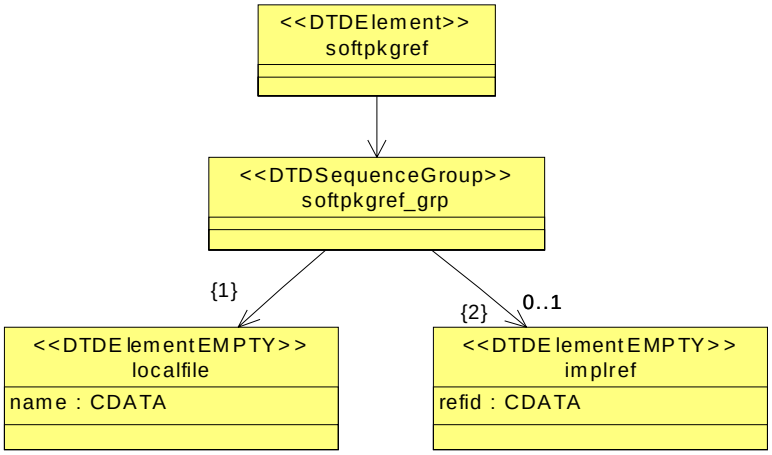

Figure 7.7 - *softpkgref* Element Relationships

```
<!ELEMENT softpkgref
    ( localfile
    , implref?
    )>
<!ELEMENT implref EMPTY>
<!ATTLIST implref
    refid    CDATA      #REQUIRED>
```

7.2.1.8.6 propertyref

The *propertyref* element is used to indicate a reference (refid attribute) to a ServiceProperty (allocation property), defined in some ServiceComponent's property file, and the requested value (value attribute) for the ServiceProperty (allocation property). This deployment requirement is used by the ApplicationFactoryComponent to find the right ServiceComponent or DeviceComponent that can meet the requirements as specified by the value attribute.

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

7.2.1.8.7 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>
```

7.2.1.8.8 propertyvaluesref

The *propertyvaluesref* element is used to indicate a reference (refid element) to a ServiceProperty (allocation property), defined in some ServiceComponent's property file, and the requested values (value element) for the ServiceProperty (allocation property). This deployment requirement is used by the ApplicationFactoryComponent to find the right ServiceComponent or DeviceComponent that can meet the requirements as specified by the value element.

```
<!ELEMENT propertyvaluesref
    (refid
          ,value+)>
<!ELEMENT refid  (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

7.2.1.9 usesdevice

The *usesdevice* element describes any "uses" relationships this component has with a ServiceComponent in the system. The *propertyref* or propertyvaluesref element references ServiceProperty(s) (e.g., allocation properties), which indicate the ServiceComponent to be used (characteristics), and/or the capacity(s) needed from the ServiceComponent.

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

7.2.1.9.1 propertyref

See Section 7.2.1.8.6, "propertyref" for a definition of the *propertyref* element.

7.2.1.9.2 propertyvaluesref

See Section 7.2.1.8.8, "propertyvaluesref" for a definition of the *propertyvaluesref* element.

7.2.1.10 assemblyimplementation

The *assemblyimplementation* element references a Software Assembly Descriptor (SAD) file for implementation of a component.

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

7.3 Device Package Descriptor

The 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 a 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. In either case, the description of the hardware structure should be consistent with hardware partitioning as described in UML Profile for communicaiton equipment and communicaiton channel.

7.3.1 Device Package

The *devicepkg* element (see Figure 7.8) is the root element of the DPD. The *devicepkg* id attribute uniquely identifies the package. 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*.

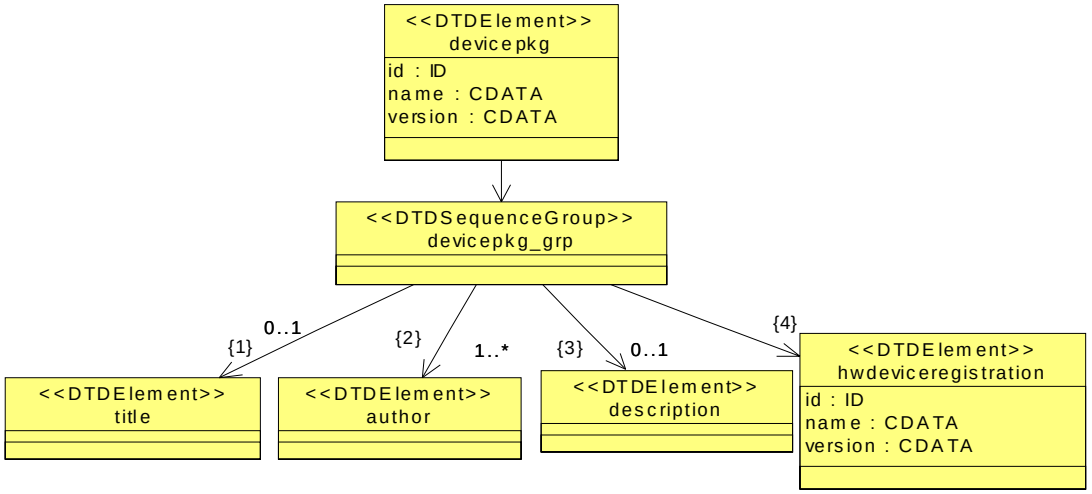

Figure 7.8 - *devicepkg* Element Relationships

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

7.3.1.1 title

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

<!ELEMENT title (#PCDATA)>

7.3.1.2 author

See Section 7.2.1.3, “author” for a definition of the *author* element.

7.3.1.3 description

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

<!ELEMENT description (#PCDATA)>

7.3.1.4 hwdeviceregistration

The *hwdeviceregistration* element (see Figure 7.9) provides device-specific information for a hardware device. The *hwdeviceregistration* id attribute uniquely identifies the device. 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-friendlylabel for the hardware device being registered. The name attribute is supplied when the id is not user-friendly such as a DCE UUID. At a minimum, the *hwdeviceregistration* element must include a description, the manufacturer, the model number, and the device’s hardware class(es).

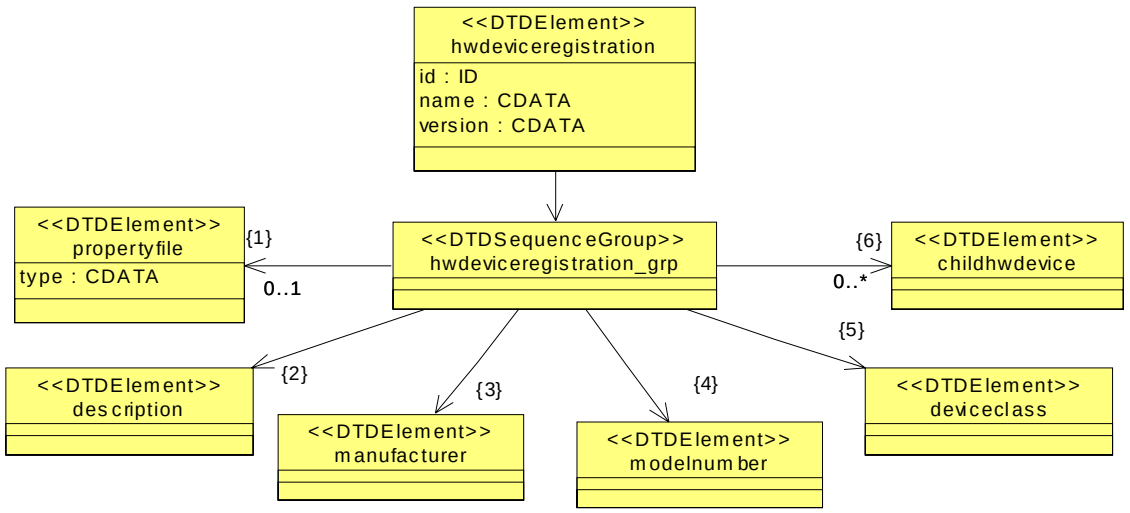

Figure 7.9 - *hwdeviceregistration* Element Relationships

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

```
name     CDATA        #REQUIRED
version  CDATA        #IMPLIED>
```

7.3.1.4.1 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 Section 7.4, “Properties Descriptor,” on page 21.

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>
```

7.3.1.4.2 description

See Section 7.2.1.4, “description” for definition of the *description* element.

7.3.1.4.3 manufacturer

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

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

7.3.1.4.4 modelnumber

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

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

7.3.1.4.5 deviceclass

The *deviceclass* element is used to identify one or more hardware classes that make up the device being installed (as defined in UML Profile for communicaiton equipment).

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

7.3.1.4.6 childhwdevice

The *childhwdevice* element (see Figure 7.10) 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 CF *Device* representing the RF module itself would be a parent *Device* 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.

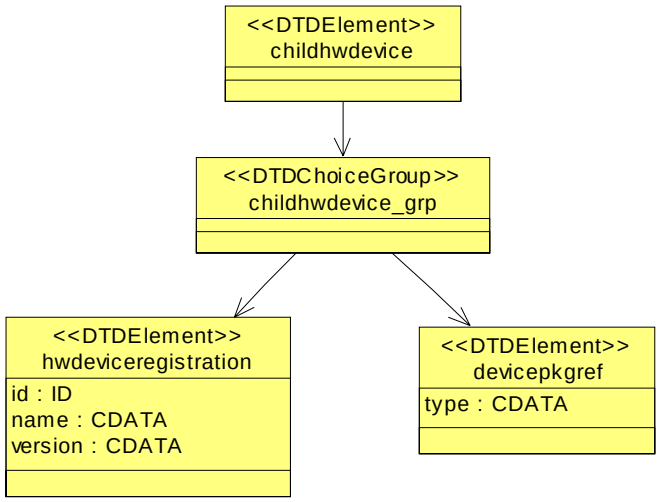

Figure 7.10 - *childhwdevice* Element Relationships

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

7.3.1.4.7 hwdeviceregistration

The *hwdeviceregistration* element provides device-specific information for the child hardware device. See 7.3.1.4 for a definition of the *hwdeviceregistration* element.

7.3.1.4.8 devicepkgref

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

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

7.4 Properties Descriptor

The Properties Descriptor file details component and device attribute settings. For purposes of the 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 for component values used by a ResourceComponent's *configure*(), *query*(), and *runTest*() operations.

7.4.1 properties

The *properties* element (see Figure 7.11) is used to describe property attributes that will be used in the *configure*( ) and *query*( ) operations for CF ResourceComponents 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.

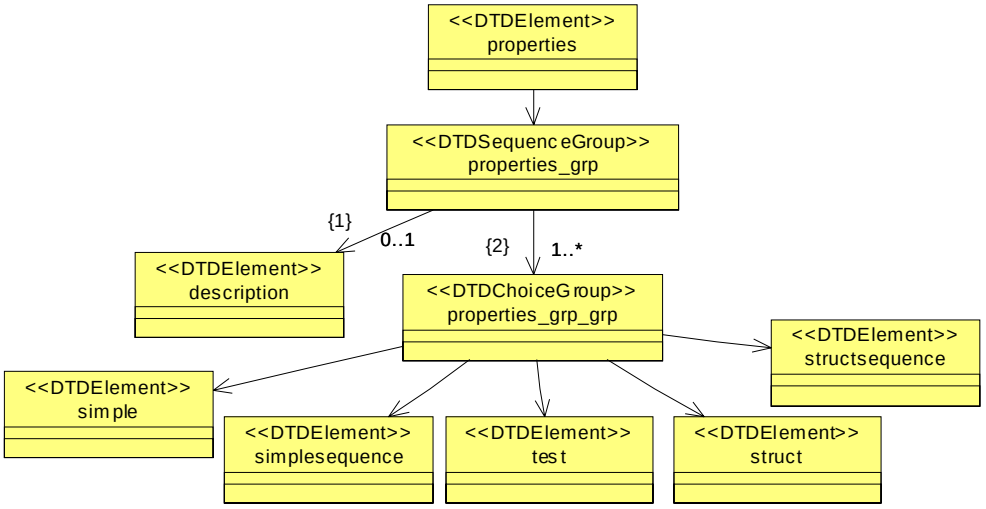

Figure 7.11 - *properties* Element Relationships

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

7.4.1.1 simple

The *simple* element (see Figure 7.12 and Table 7.1) 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, or for indication of component capabilities. 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 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." The integerID attribute is used to specify an integer identifier for a simple property, which when used has precedence over the ID attribute.

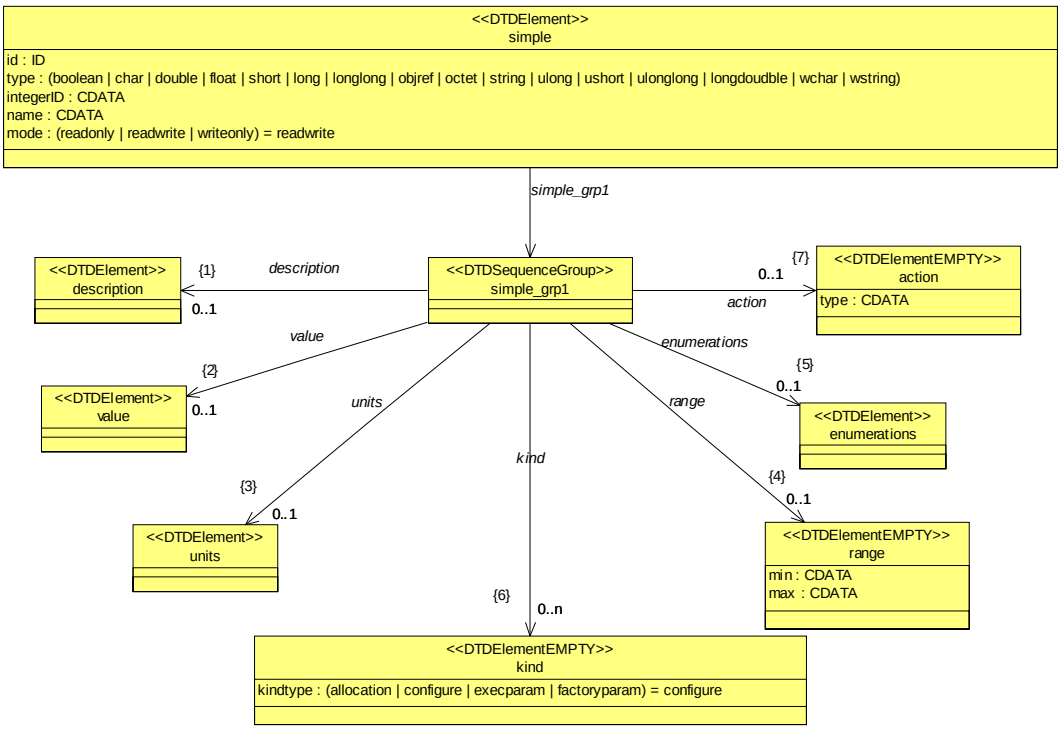

Figure 7.12 - *simple* Element Relationships

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

Table 7.1 - Simple Elements & Attributes Summary

Simple Properties	Id	Type	integer ID	Name	Mode	Value	Units	Range	Enum	Kind	Action
Configure & Query	+	+	int	*	RW (default)	+	*	*	*	configure	N/A
Configure Only	+	+	int	*	WO	+	*	*	*	configure	N/A
Query Only	+	+	int	*	RO	---	*	*	*	configure	N/A
Service Component's ServiceProperty (Locally Managed=false)	+	+	int	*	- - -	+	*	*	*	allocation	Eq, ne gt, lt, ge, le
Service Component's ServiceProperty (Locally Managed=true)	+	+	int	*	- - -	+	*	*	*	allocation	External (default)
Execute Property	+	+	int	*	- - -	+	*	*	*	execparm	N/A
ResourceFactoryComponent's Configure Property	+	+	int	*	- - -	+	*	*	*	factoryparm	N/A
TestProperty's InputValue Property or ResultValue Property	+	+	int	*	- - -	+	*	*	*	test	N/A

Table Legend:

+ Is required and may contain any value, For ID only certain characters can be used. Value must conform to type.

* Is optional and may contain any value

int value is integer characters

N/A Not Applicable, will be ignored if used

7.4.1.1.1 description

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

<!ELEMENT description (#PCDATA)>

7.4.1.1.2 value

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

<!ELEMENT value (#PCDATA)>

7.4.1.1.3 units

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

<!ELEMENT units (#PCDATA)>

7.4.1.1.4 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 the CF *ApplicationFactory* or CF *Application* implementations.

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

7.4.1.1.5 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. An Enumeration value is assigned to a property that implements the CORBA long type. Enumeration values are implied; if not specified by a developer, the initial implied value is 0 and subsequent values are incremented by 1.

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

7.4.1.1.6 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 and DeviceManagerComponent will use the configure kind of properties to build the *Properties* input parameter to the *configure* () operation that is invoked on the Component(s) 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. test, which is used in the runTest() operation in the TestableObject interface. The test kind of properties will be used as the testValues parameter to the runTest() operation.

3. allocation, which is used in the *allocateCapacity*() and *deallocateCapacity*() operations of the Device interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the allocation kind of properties to build the *capacities* inout parameter to the *allocateCapacity*() operation that is invoked on the *Device*Component(s) during application creation when the simple property action element is external. The

ApplicationFactoryComponent and DeviceManagerComponent (not DeviceComponent) manages an Allocation property when the action value 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 Device interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the execparam kind of properties to build the Properties input parameter to the *execute*() operation that is invoked on the ExecutableDevicecomponent(s) during component and/or application creation. Only simple elements can be used as execparam types.

5. factoryparam, are properties that are only for the *createResource*() operation of the ResourceFactory interface. The ApplicationFactoryComponent will use the factoryparam type of properties to build the *Properties* input parameter to the *createResource*() 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”>

7.4.1.1.7 action

The *action* element is used to define the type of comparison used to compare an SPD property value to a DeviceComponent property value, during the process of checking SPD dependencies. The type attribute, of the *action* element, will determine the type of comparison to be made (e.g., equal, not equal, greater than, etc.). When the action’s type is not external then the ApplicationFactoryComponent and DeviceManagerComponent performs the action comparison, not the DeviceComponent. The default value for type is external when not specified.

When the action is “external” then the DeviceComponent is locally managing the allocation propery (e.g., ServiceProperty). The ApplicationFactoryComponent cannot manage these properties, instead it must use the alllocateCapacity operation on a compatible DeviceComponent. For non-external action types, the allocateCapacity operation is not called on a DeviceComponent.

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 DeviceComponent. 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 *DeviceComponent’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 will check DeviceComponent(s) 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 CDATA #REQUIRED>

7.4.1.1.8 simplesequence

The *simplesequence* element (see Figure 7.13) 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 basic primitive sequence types and CF PortTypes CORBA modules, defined in UML Profile for CF, based upon the type attribute.

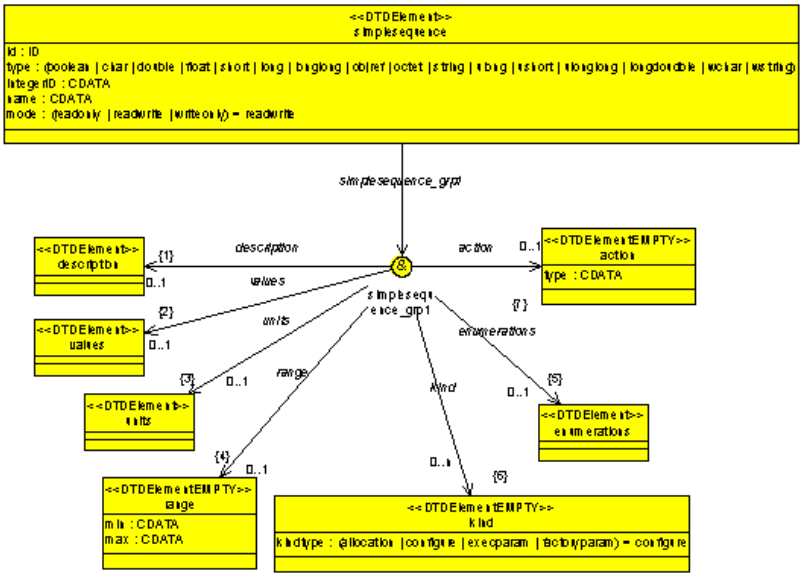

Figure 7.13 - *simplesequence* Element Relationships

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

7.4.1.2 test

The *test* element (see Figure 7.14) is used to specify a list of test properties for executing the TestableObject *runTest*() operation to perform a component specific test. This definition contains *inputvalue* and *resultvalue* elements and it has a testid attribute for grouping test properties to a specific test. *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).

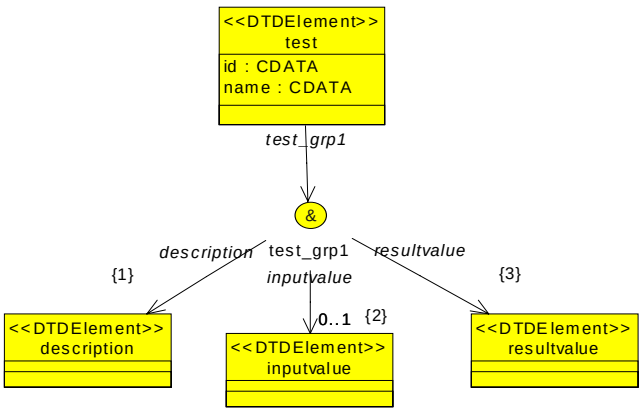

Figure 7.14 - *test* Element Relationships

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

7.4.1.2.1 inputvalue

The *inputvalue* element is used to provide test configuration properties. The Simple properties it contains must be of kind "test".

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

7.4.1.2.2 resultvalue

The *resultvalue* element is used to provide test result properties. The Simple properties it contains must be of kind "test".

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

7.4.1.3 struct

The *struct* element (see Figure 7.15) 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 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.

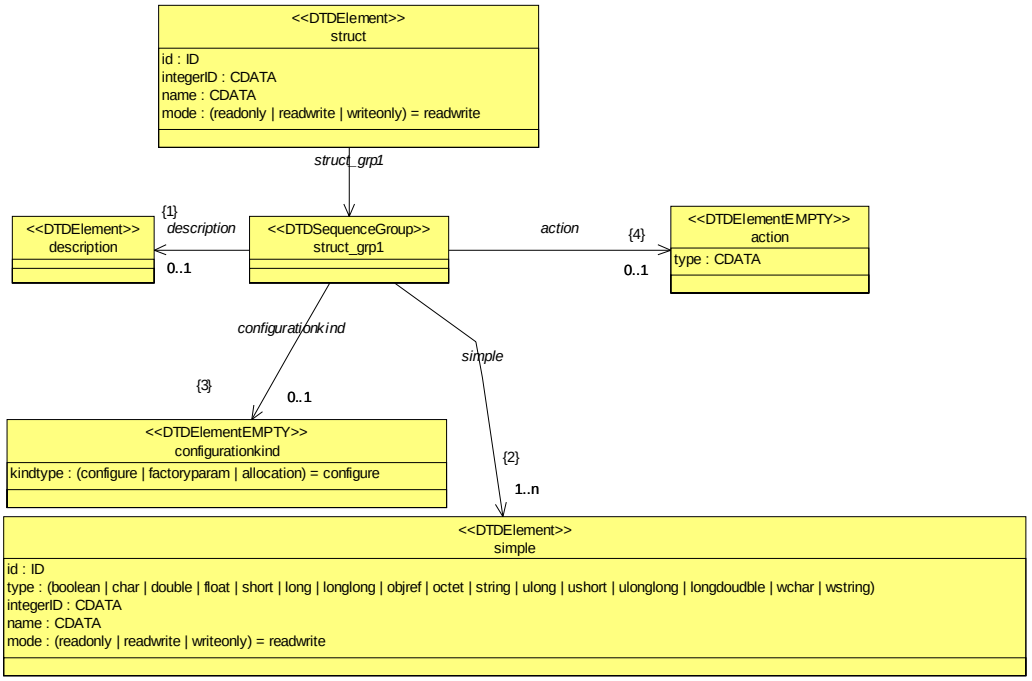

Figure 7.15 - *struct* Element Relationships

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

7.4.1.3.1 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 CF Resource interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the configure kind of properties to build the Properties input parameter to the *configure*() operation that is invoked on the ResourceComponent(s)

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 *createResource* operations of the *ResourceFactory* interface. The ApplicationFactoryComponent will use the factoryparam kind of properties to build the Properties input parameter to the *createResource*() operation. A property can have multiple *configurationkind* elements and their default kindtype is “configure”.

3. allocation, which is used in the *allocateCapacity*() and *deallocateCapacity*() operations of the Device interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the allocation kind of properties to build the *capacities* inout parameter to the *allocateCapacity*() operation that is invoked on the *Device*Component(s) during application creation when the simple property action element is external. The ApplicationFactoryComponent and DeviceManagerComponent (not DeviceComponent) manages an Allocation property when the action value is not external. Allocation properties that are external can also be queried using the PropertySet *query()* operation.

<!ELEMENT configurationkind EMPTY>
<!ATTLIST configurationkind
kindtype(configure | factoryparam | allocation) “configure”>

7.4.1.4 *structsequence*

The *structsequence* element (see Figure 7.16) is used to specify a list of properties with the same *struct* characteristics. The *structsequence* element maps to a *properties* element having the Properties type. Each item in the Properties type will be the same *struct* definition as referenced by the structrefid attribute.

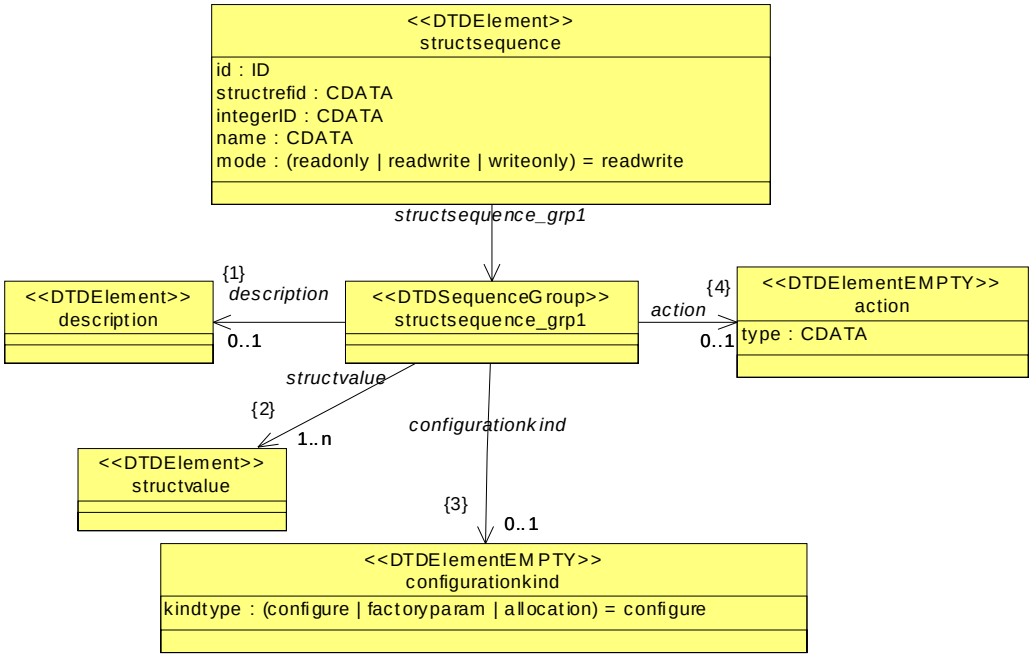

Figure 7.16 - *structsequence* Element Relationships

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

<!ELEMENT structvalue
        (simpleref+
)>
<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
        refid CDATA #REQUIRED
        value CDATA #REQUIRED>
```

7.5 Software Component Descriptor

The Software Component Descriptor (SCD) defines elements necessary for describing the ports, interfaces, and properties for a component definition.

7.5.1 softwarecomponent

The *softwarecomponent* element (see Figure 7.17) is the root element of the software component descriptor file. The sub-elements that are supported include:

- *corbaversion* – indicates which version of CORBA the component is developed for.
- *componentrepid* – is the repository id of the component
- *componenttype* – identifies the type of software component object
- *componentfeatures* – provides the supported message ports for the component
- *interface* – describes the component unique id and name for supported interfaces

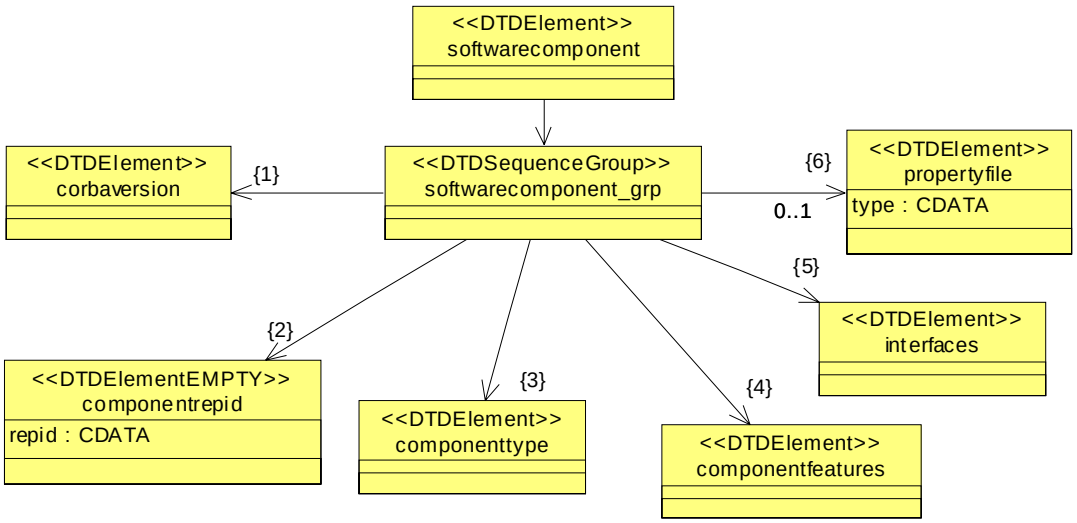

Figure 7.17 - *softwarecomponent* Element Relationships

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

7.5.1.1 corbaversion

The *corbaversion* element is intended to indicate the version of CORBA that the delivered component supports.

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

7.5.1.2 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 interfaces such as the *Resource*, *Device*, or *ResourceFactory.*

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

7.5.1.3 componenttype

The *componenttype* describes properties of the component. For CF components, the component types include elements such as service, resource, device, resourcefactory, domainmanager, log, filesystem, filemanager, devicemanager, namingservice, and eventservice.

<!ELEMENT componenttype (#PCDATA)>

7.5.1.4 componentfeatures

The *componentfeatures* element (see Figure 7.18) is used to describe a component with respect to the components that it inherits from, the interfaces the component supports, and its' provides and uses *ports*. The component interface is usually Resource, ResourceFactory, or service interface such as Device, LoadableDevice, and ExecutableDevice. If a component extends the Resource or Device interfaces, then all the inherited interfaces (e.g., Resource) are depicted as supportsinterface elements.

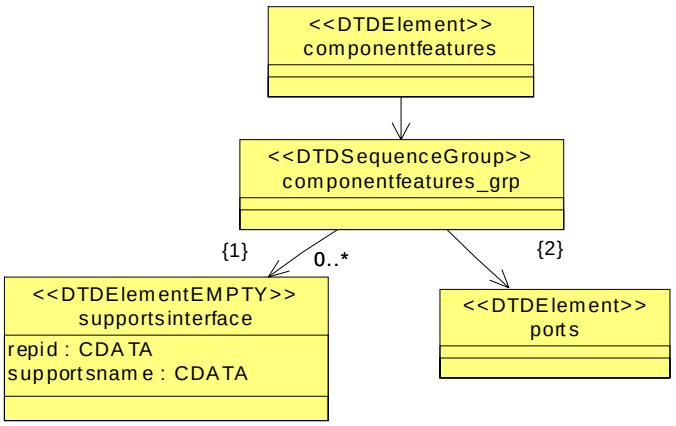

Figure 7.18 - *componentfeatures* Element Relationships

<!ELEMENT componentfeatures
(supportsinterface*
, ports
)>

7.5.1.4.1 supportsinterface

The *supportsinterface* element is used to identify an IDL interface 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 section 7.5.1.5).

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

7.5.1.4.2 ports

The *ports* element (see Figure 7.19) describes what interfaces a component provides and uses (or requires). The *provides* elements are interfaces that are not part of a component's interface but are independent interfaces known as facets (in CORBA Components terminology) (i.e., a *provides* port at the end of a path, like I/O Device or Modem Device). The *uses* element describes the interfaces needed by a component. These uses ports are 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 section 7.5.1.5). The port names are used in the Software Assembly

Descriptor to connect ports together. 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.

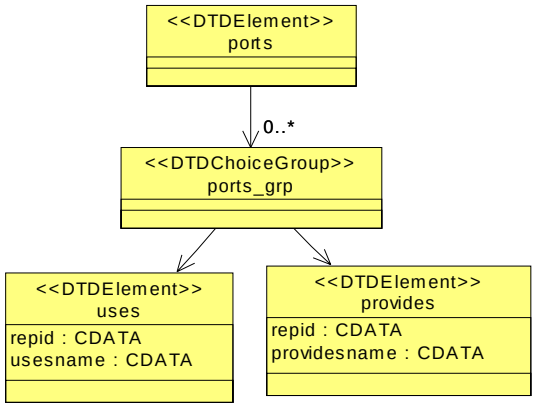

Figure 7.19 - *ports* Element Relationships

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

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

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

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

7.5.1.5 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, which has formats specified in the CORBA specification. The repid is also used to reference an *interface* element elsewhere in the SCD, for example from the *inheritsinterface* element.

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

7.5.1.6 propertyfile

Refer to section 7.2.1.1 *propertyfile* for definition of *propertyfile*. The properties defined at the SCD are the definition of properties supported by all implementations, and be managed by the PropertySet interface as described in the UML Profile for CF.

7.6 Software Assembly Descriptor

This section describes the XML elements of the Software Assembly Descriptor (SAD) XML file; the *softwareassembly* element (see Figure 7.20). The SAD is based on the CORBA Components Specification Component Assembly Descriptor. The intent of the software assembly is to provide the means of describing the assembled functional application and the interconnection characteristics of the components within that application. The component assembly provides four basic types of application information for Radio Management. The first is partitioning information that indicates special requirements for collocation of components, the second is the assembly controller for the software assembly, the third is connection information for the various components that make up the application assembly, and the fourth is the visible ports for the application assembly.

The installation of an application into the system involves the installation of a SAD file. The SAD file references component's SPD files to obtain deployment information for these components. The *softwareassembly* element's id attribute uniquely identifies the assembly. The *softwareassembly* element's name attribute is the user-friendly name for the ApplicationFactoryComponent name attribute. The name attribute is supplied when the id is not user-friendly such as a DCE UUID. The softwareassembly element's version attribute is the version of the application.

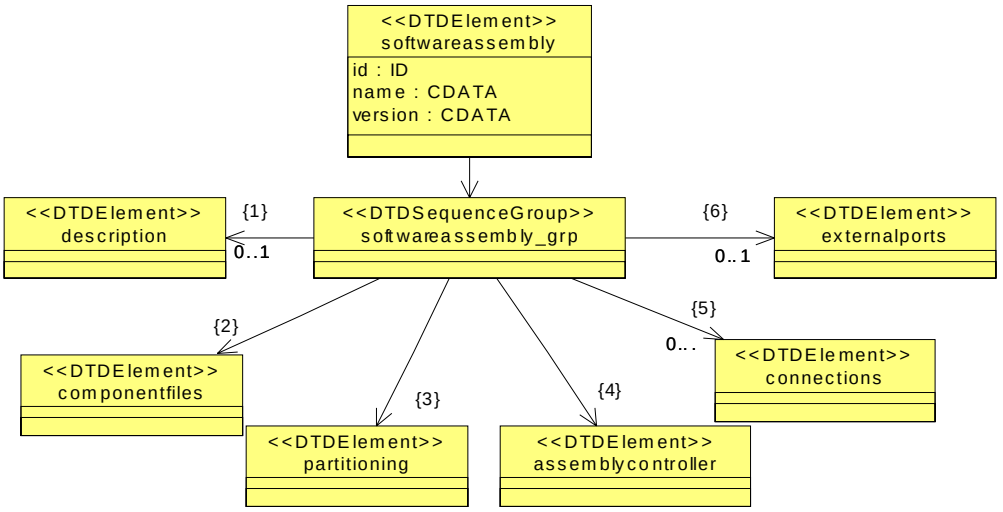

Figure 7.20 - *softwareassembly* Element Relationships

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

7.6.1 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)>
```

7.6.2 componentfiles

The *componentfiles* element is used to indicate that an assembly is made up of 1..n component files. The *componentfile* element contains a reference to a local file, which is a Software Package Descriptor file (see Section 7.2).

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

7.6.2.1 componentfile

The *componentfile* element is a reference to a local file. See Section 7.2.1.1.1 for the definition of the *localfile* element. The type attribute is “Software Package Descriptor.

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

7.6.3 partitioning

A component partitioning element (see Figure 7.21) specifies a deployment pattern of components and their components-to-hosts and process relationships. A component instantiation is captured inside a componentplacement element. The hostcollocation element allows the components to be placed on a common device. The processcollocation element allows components to be placed within the same process space. When the componentplacement is by itself and not inside a hostcollocation or processcollocation, it then has no collocation constraints.

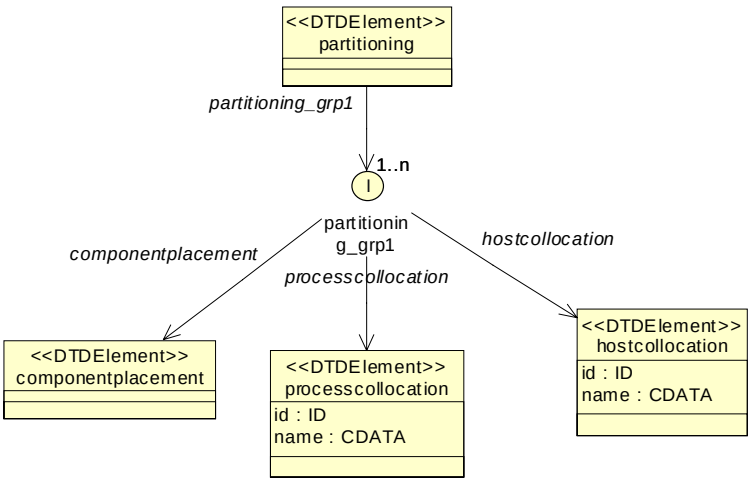

Figure 7.21 - *partitioning* Element Relationships

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

7.6.3.1 componentplacement

The *componentplacement* element (see Figure 7.22) defines a particular deployment of a component. The component can be deployed either directly or by using a ResourceFactoryComponent.

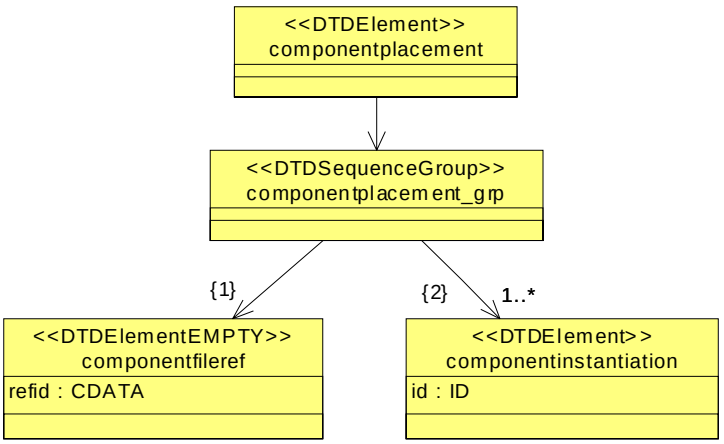

Figure 7.22 - *componentplacement* Element Relationships

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

7.6.3.1.1 componentfileref

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

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

7.6.3.1.2 componentinstantiation

The componentinstantiation element (see Figure 7.23) is intended to describe a particular instantiation of a component relative to a componentplacement element. The componentinstantiation's id attribute uniquely identifies the component. The componentinstantiation element's id may be referenced by the usesport and providesport elements within the SAD file.

The optional componentproperties element (see Figure 7.24) 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 componentproperty definitions as stated in the corresponding SCD.

The following sources will be searched in the given precedence order for initial values for "configure" kind of properties, whose modes are "readwrite" or "writeonly" and "execparam" kind of properties:

1. The componentproperties element of the componentinstantiation element in SAD.

The following sources will be searched initial values for the “factoryparam” kind of properties in the given precedence order:

1. The componentinstantiation element’s findcomponent element’s componentresourcefactoryref element’s resourcefactoryproperties element in the SAD.

The findcomponent element (see Figure 7.25) is used to obtain the object reference for the component instance. The two sources for obtaining an object reference are:

1. The componentresourcefactoryref element, which refers to a particular ResourceFactoryComponent componentinstantiation element found in the SAD, which is used to obtain a ResourceComponent instance for this componentinstantiation element. The refid attribute refers to a unique componentinstantiation id attribute. The componentresourcefactoryref element contains an optional resourcefactoryproperties element (see Figure 7.26), which specifies the properties “qualifiers,” for the ResourceFactoryComponent create call.

2. The optional findcomponent element should be specified except when there is no object reference for the component instance (e.g., non-CORBA code). The CORBA Naming Service, which is used to find the component’s object reference. The name specified in the namingservice element is a partial name that is used by the ApplicationFactoryComponent to form the complete context name.

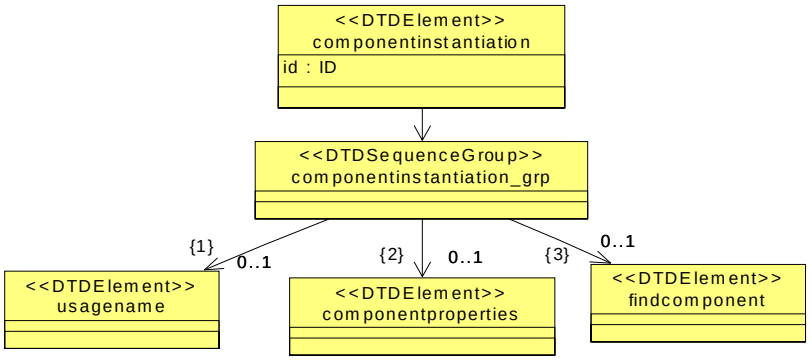

Figure 7.23 - *componentinstantiation* Element Relationships

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

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

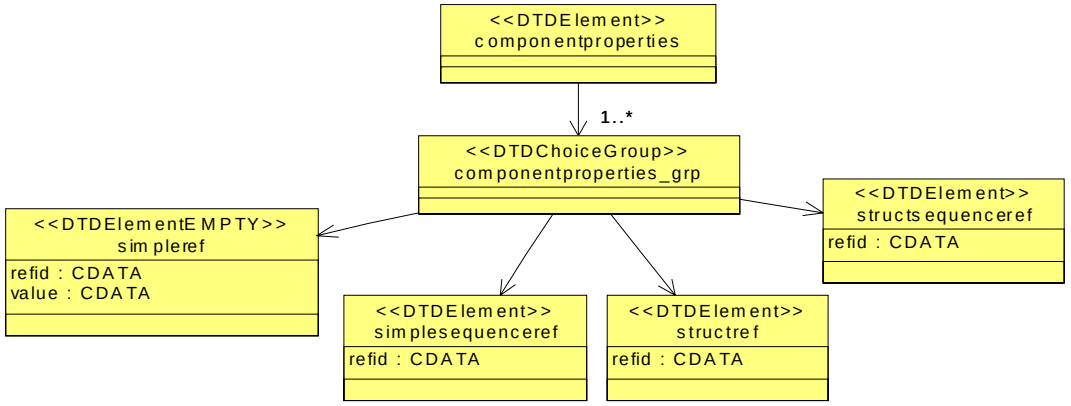

Figure 7.24 - *componentproperties* Element Relationships

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

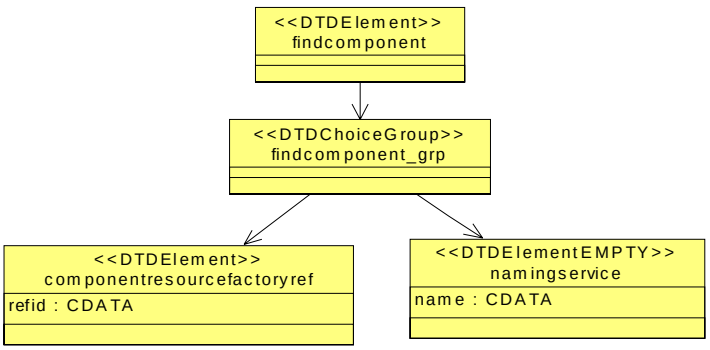

Figure 7.25 - *findcomponent* Element Relationships

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

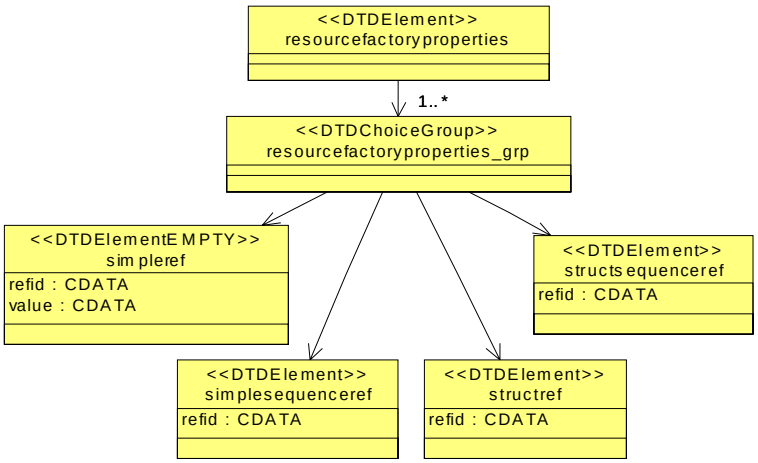

Figure 7.26 - *resourcefactoryproperties* Element Relationships

```
<!ELEMENT resourcefactoryproperties
        ( 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
        refidCDATA#REQUIRED>

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

<!ELEMENT structvalue
        (simpleref+
        )>
```

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

7.6.3.2 hostcollocation

The *hostcollocation* element specifies a group of component instances that are to be deployed together on a single host. The *componentplacement* element will be used to describe the 1...n components that will be collocated on the same host platform. The processcollocation element will be used to describe the processes that are to be collocated on the same host. 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
        | processcollocation
        )+>
<!ATTLIST hostcollocation
        id   ID   #IMPLIED
        name CDATA #IMPLIED>
```

7.6.3.2.1 componentplacement

See *componentplacement,* section 7.6.3.1.

7.6.3.3 processcollocation

The *processcollocation* element specifies a group of component instances that are to be deployed together in a single process. The *componentplacement* element will be used to describe the 1...n components that will be collocated in the process. The id and name attributes are optional but may be used to uniquely identify a set of collocated components within a SAD file.

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

7.6.4 assemblycontroller

The *assemblycontroller* element indicates the component that is the main ResourceComponent controller for the assembly. The ApplicationManager component delegates its *Resource configure*(), *query*(), *start(), stop(),* and *runTest()* operations to the *ResourceComponent's* Assembly Controller component.

```
<!ELEMENT assemblycontroller
        ( componentinstantiationref
        )>
```

7.6.5 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*
)>
```

7.6.5.1 connectinterface

The *connectinterface* element (see Figure 7.27) 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, componentsupportedinterface, or findby* element. These elements are intended to connect two compatible components.

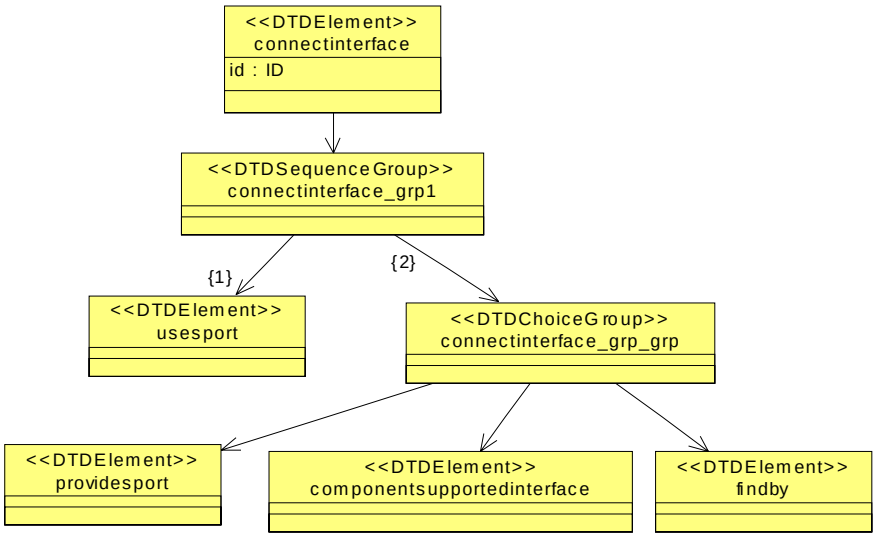

Figure 7.27 - *connectinterface* Element Relationships

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

7.6.5.1.1 usesport

The *usesport* element (see Figure 7.28) identifies, using the *usesidentifier* element, the component port that is using the provided interface from the *providesport* element. A component may be referenced by one of four elements. One element is the *componentinstantiationref* that refers to the *componentinstantiation* id attribute (*see componentinstantiation*) within the assembly; the other elements are *findby*, *devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref*.

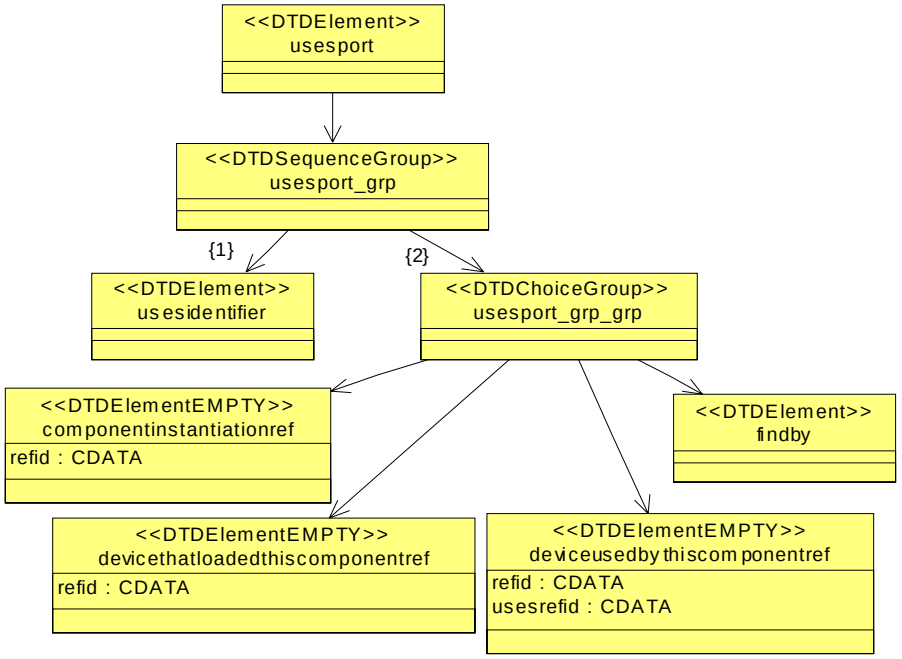

Figure 7.28 - *usesport* Element Relationships

```
<!ELEMENT usesport
        ( usesidentifier
        ,     ( componentinstantiationref
              | devicethatloadedthiscomponentref
        | deviceusedbythiscomponentref
              | findby
              )
        )>
```

7.6.5.1.1.1 usesidentifier

The *usesidentifier* 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 Software Component Descriptor (see Section 7.5).

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

7.6.5.1.1.2 componentinstantiationref

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

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

7.6.5.1.1.3 findby

The *findby* element (see Figure 7.29) is used to resolve a connection between two components. It tells the Domain Management function how to locate a component interface involved in a connection relationship. The *namingservice* element specifies a naming service name to search for the desired component interface.

The *domainfinder* element specifies an element within the domain that is known to the Domain Management function.

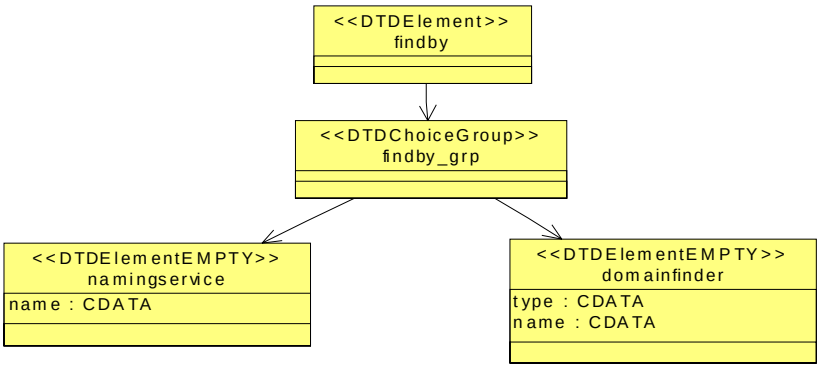

Figure 7.29 - *findby* Element Relationships

```
<!ELEMENT findby
        ( namingservice
        | domainfinder
        )>
```

7.6.5.1.1.3.1 namingservice

The *namingservice* element is a child element of the *findby* element. The *namingservice* element is used to indicate to the ApplicationFactoryComponent the requirement to find a component interface. The ApplicationFactoryComponent will use the *name* attribute to search the CORBA Naming Service for the appropriate interface.

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

7.6.5.1.1.3.2 domainfinder

The *domainfinder* element is a child element of the *findby* element. 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. At a minimum the following valid type attribute values need to be supported “filemanager,” “log,” “eventchannel,” “namingservice,” “application,” and “service.” If a name attribute is not supplied, then the component reference returned is the *DomainManagerComponent’s FileManager* or Naming Service

corresponding to the type attribute provided. If a name attribute is not supplied and the type attribute has a value of “application,” “service,” or “log,” then a null reference is returned. The type attribute value of “eventchannel” is used to specify the event channel to be used in the OE’s CORBA 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.

```
<!ELEMENT domainfinder EMPTY>
<!ATTLIST domainfinder
        type CDATA #REQUIRED
        name CDATA #IMPLIED>
```

7.6.5.1.1.4 devicethatloadedthiscomponentref

The *devicethatloadedthiscomponentref* element refers to a specific component found in the assembly, which is used to obtain the DeviceComponent that was used to load the referenced component from the ApplicationFactoryComponent. The DeviceComponent obtained is then associated with this component instance. This relationship is needed when a component (e.g., modem adapter) is pushing data and/or commands to a non-CORBA capable device such as modem.

```
<!ELEMENT devicethatloadedthiscomponentref EMPTY>
<!ATTLIST devicethatloadedthiscomponentref
        refidC DATA #REQUIRED>
```

7.6.5.1.1.5 deviceusedbythiscomponentref

The *deviceusedbythiscomponentref* element refers to a specific component, within the assembly, which is used to obtain the DeviceComponent 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>
```

7.6.5.1.2 providesport

The *providesport* element (see Figure 7.30) identifies, using the *providesidentifier* element, the component port that is provided to the *usesport* interface within the *connectinterface* element. A component may be referenced by one of four elements. One element is the *componentinstantiationref* that refers to the *componentinstantiation* id (*see componentinstantiation*) within the assembly; the other elements are *findby, devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref*. The *findby* element by itself is used when the object reference is not a ResourceComponent type.

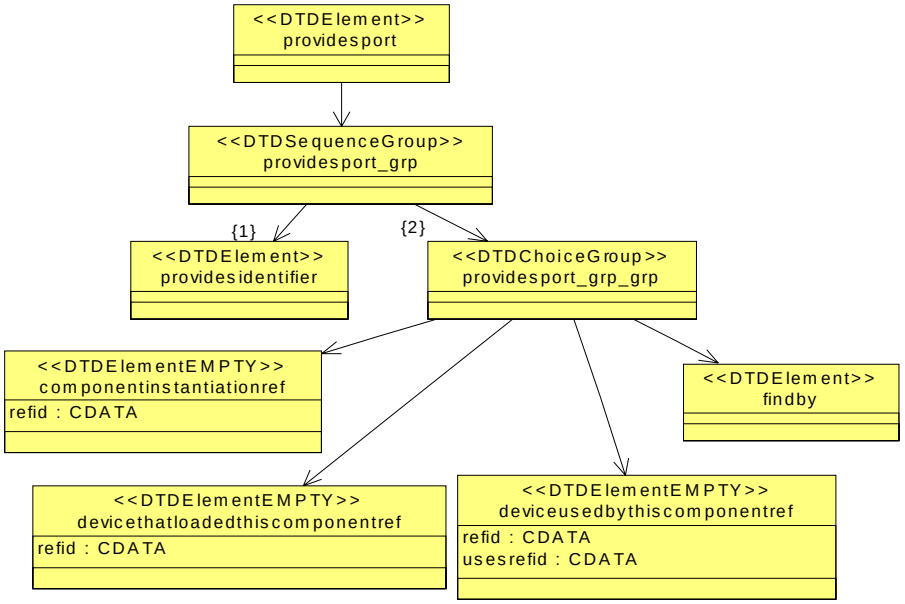

Figure 7.30 - *providesport* Element Relationships

```
<!ELEMENT providesport
        ( providesidentifier
        , ( componentinstantiationref
          | devicethatloadedthiscomponentref
          | deviceusedbythiscomponentref
          | findby
          )
        )>
```

7.6.5.1.2.1 providesidentifier

The *providesidentifier* 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 Software Component Descriptor (see Section 7.5).

<!ELEMENT providesidentifier (#PCDATA)>

7.6.5.1.2.2 componentinstantiationref

See Section 7.6.3.1.2, “componentinstantiation,” on page 37 for a description of the *componentinstantiationref* element.

7.6.5.1.2.3 findby

See Section 7.6.5.1.1.3, “findby” for a description of the *findby* element. The *namingservice* element’s name attribute denotes a complete naming context.

7.6.5.1.2.4 devicethatloadedthiscomponentref

See Section 7.6.5.1.1.4, “devicethatloadedthiscomponentref” for a description of the *devicethatloadedthiscomponentref* element.

7.6.5.1.2.5 deviceusedbythiscomponentref

See Section 7.6.5.1.1.5, “deviceusedbythiscomponentref” for a description of the *deviceusedbythiscomponentref* element.

7.6.5.1.2.6 componentsupportedinterface

The *componentsupportedinterface* element (see Figure 7.31) 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. This component is identified by a *componentinstantiationref* or a *findby* element. The *componentinstantiationref* identifies a component within the assembly. The *findby* element points to an existing component that can be found within a Naming Service.

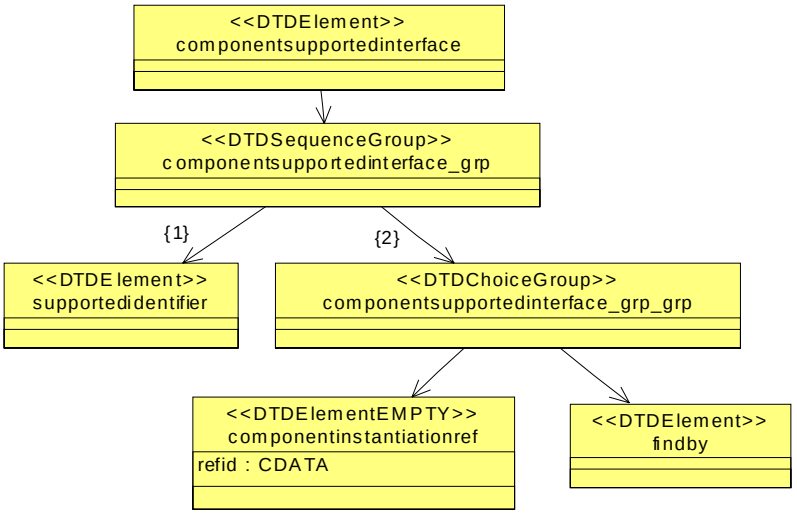

Figure 7.31 - *componentsupportedinterface* Element Relationships

```
<!ELEMENT componentsupportedinterface
( supportedidentifier
, ( componentinstantiationref
      | findby
          )
        )>
```

7.6.5.1.2.7 supportedidentifier

The *supportedidentifier* 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 Software Component Descriptor.

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

7.6.5.1.2.8 componentinstantiationref

See Section 7.6.3.1.2, “componentinstantiation,” on page 37 for a description of the *componentinstantiationref* element.

7.6.5.1.2.9 findby

See Section 7.6.5.1.1.3, “findby” for a description of the *findby* element.

7.6.6 externalports

The optional *externalports* element is a child element of the *softwareassembly* element (see Figure 7.32). The *externalports* element is used to identify the visible ports for the software assembly. The ApplicationManager *getport*() operation is used to access the assembly’s visible ports.

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

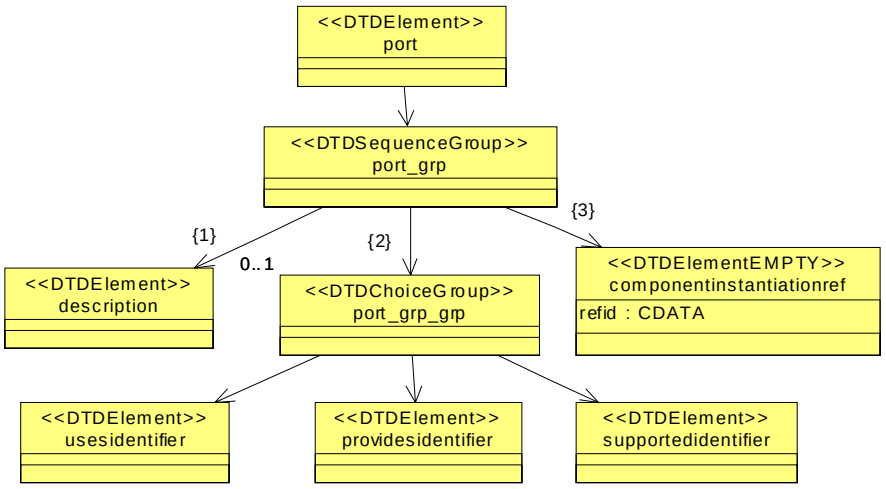

Figure 7.32 - *port* Element Relationships

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

7.7 Device Configuration Descriptor

This section describes the XML elements of the Device Configuration Descriptor (DCD) XML file; the *deviceconfiguration* element (see Figure 7.33). 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 object reference, connections of services to components (*ServiceComponent(*s), 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.

The *deviceconfiguration* element's id attribute is a unique identifier within the domain for the device configuration. The name attribute is the user-friendly name for the DeviceManagerComponent's label attribute. The name attribute is supplied when the id is not user-friendly such as a DCE UUID.

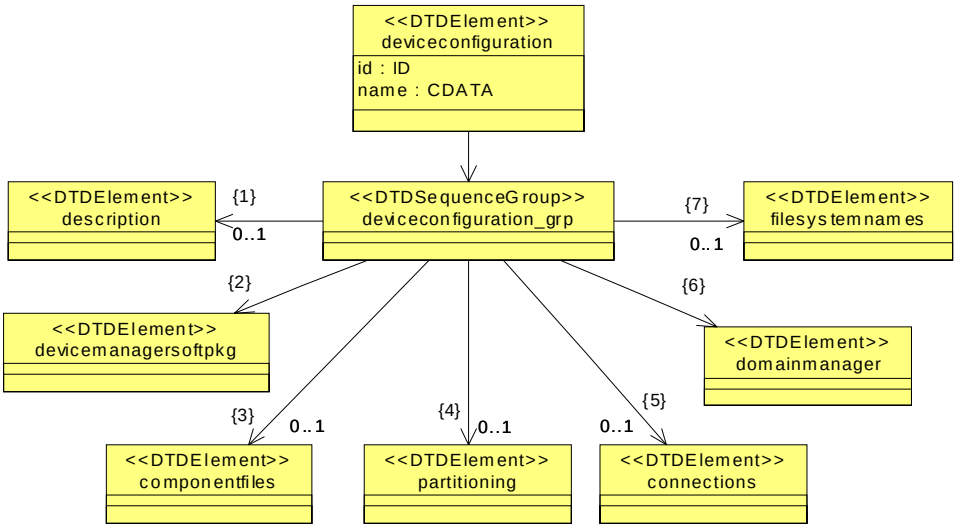

Figure 7.33 - *deviceconfiguration* Element Relationships

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

7.7.1 description

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

<!ELEMENT description (#PCDATA)>

7.7.2 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 DeviceManagerComponent implementation and to specify the *usesports* for the services (Log*(s)*, etc.) used by the DeviceManagerComponent. See Section 7.2.1.1.1 for description of the *localfile* element.

<!ELEMENT devicemanagersoftpkg
(localfile
)>

7.7.3 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 Software Package Descriptor (SPD). The SPD, for example, can be used to describe ServiceComponents, DeviceManagerComponents, a *DomainManagerComponent*, a Naming Service, and File Services. See Section 7.6.2 for the definition of the *componentfiles* element.

7.7.4 partitioning

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

<!ELEMENT partitioning
(componentplacement
)*>

7.7.5 componentplacement

The *componentplacement* element (see Figure 7.34) 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 and its id attribute is a DCE UUID value with the format as specified in Section 7.2.1, “Software Package.” 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 device that the *componentinstantiation* element is aggregated with to form an aggregate relationship. When the component is a logical *Device*, the *devicepkgfile* element indicates the hardware device information for the logical *Device*.

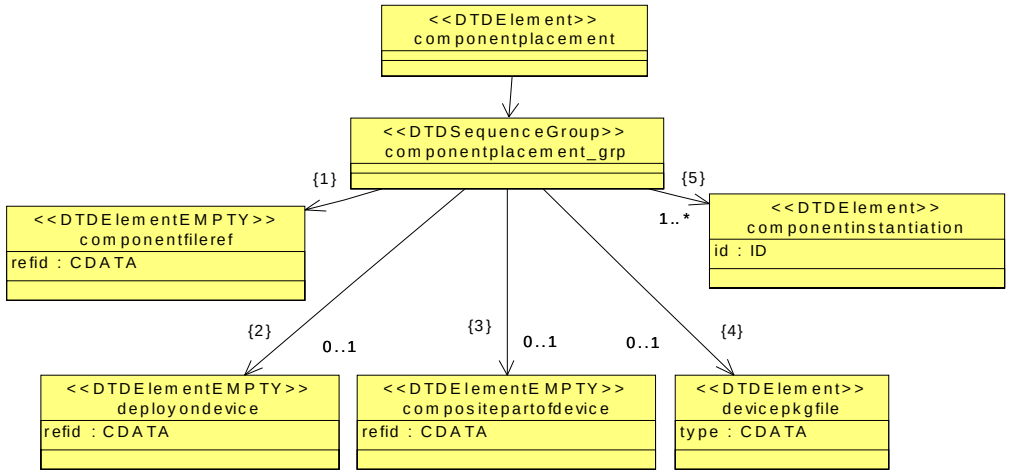

Figure 7.34 - *componentplacement* Element Relationships

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

7.7.5.1 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>
```

7.7.5.1.1 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>
```

7.7.5.1.2 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>

7.7.5.1.2.1 localfile

See 7.2.1.1.1 for a definition of the *localfile* element.

7.7.5.1.3 compositepartofdevice

The *compositepartofdevice* element is used when an aggregate relationship exists to reference the *componentinstantiation* element that describes the whole *Device* for which this *Device's componentinstantiation* element describes a part of the aggregate *Device.*

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

7.7.5.1.4 componentinstantiation

The *componentinstantiation* element (see Figure 7.35) is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation*'s id attribute is a DCE UUID that uniquely identifier the component. The id is a DCE UUID value as specified in section Software Package. The *componentinstantiation* contains a *usagename* element that is intended for an applicable name for the component. The optional *componentproperties* element (see Figure 7.36) is a list of property values that are used in configuring the component. Section 7.6.3.1.2, "componentinstantiation" defines the property list for the *componentinstantiation* element, which contains initial properties values. For a component service type (e.g., Log), the *usagename* element needs to be unique for each service type.

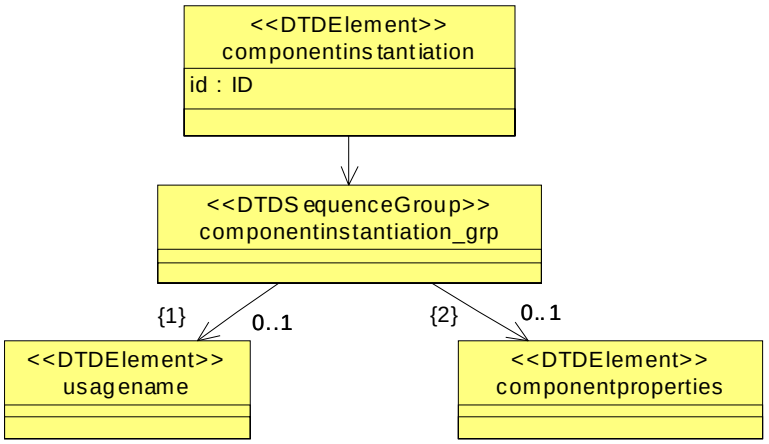

Figure 7.35 - *componentinstantiation* Element Relationships

<!ELEMENT componentinstantiation
(usagename?
,componentproperties?
)>
<!ATTLIST componentinstantiation
id ID #REQUIRED>

<!ELEMENT usagename (#PCDATA)>

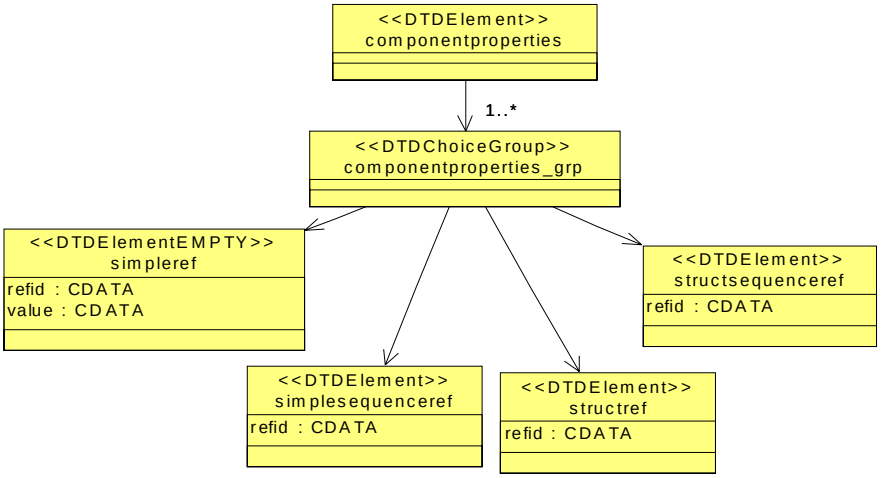

Figure 7.36 - *componentproperties* Element Relationships

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

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

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

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

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

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

7.7.6 connections

The *connections* element in the DCD is the same as the *connections* element in the SAD in section D.6.5. The *connections* element in the DCD is used to indicate the services (Log, etc.) instances that are used by the DeviceManagerComponent and ServiceComponent(s) in the DCD. The DomainManagerComponent will parse the connections element and make the connections when the DeviceManagerComponent registers with the DomainManagerComponent. 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.

7.7.7 domainmanager

The *domainmanager* element (see Figure 7.37) indicates how to obtain the DomainManagerComponent object reference. See section 7.6.5.1.1.3.1 for description of the namingservice.

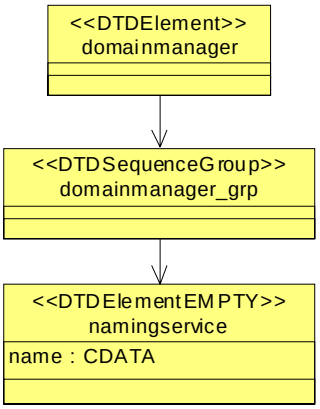

Figure 7.37 - *domainmanager* Element Relationships

```
<!ELEMENT domainmanager
( namingservice)>)>
<!ELEMENT namingservice EMPTY>
<!ATTLIST namingservice
    name CDATA #REQUIRED>
```

7.7.8 filesystemnames

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

```
<!ELEMENT filesystemnames
```

```
        ( filesystemname+
        )>

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

7.8 DomainManager Configuration Descriptor

This section describes the XML elements of the *DomainManagerComponent* Configuration Descriptor (DMD) XML file; the *domainmanagerconfiguration* element (see Figure 7.38). The *domainmanagerconfiguration* element id attribute is a DCE UUID that uniquely identifies the *DomainManagerComponent*. The id is a DCE UUID value as specified in section Software Package.

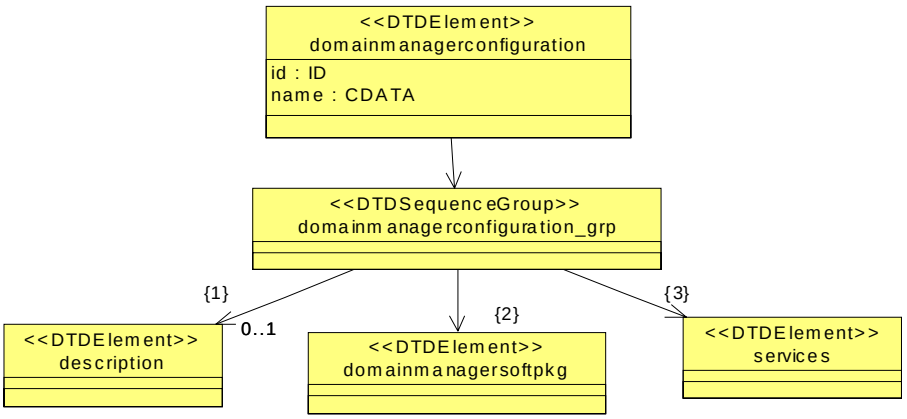

Figure 7.38 - *domainmanagerconfiguration* Element Relationships

```
<!ELEMENT domainmanagerconfiguration
    ( description?
    , domainmanagersoftpkg
    , services?
    )>
<!ATTLIST domainmanagerconfiguration
    id   ID   #required
    name #CDATA #required>
```

7.8.1 description

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

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

7.8.2 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 7.2.1.1.1 for description of the localfile element.

<!ELEMENT domainmanagersoftpkg
(localfile
) >

7.8.3 services

The *services* element in the DMD is used by the *DomainManagerComponent* to determine which service (Log, etc.) instances to use; it makes use of the *service* element (see Figure 7.39). See section 7.6.5.1.1.3 for a description of the *findby* element. See section 7.6.5.1.1.1 for a description of the *usesidentifier* element.

<!ELEMENT services
(service+
) >

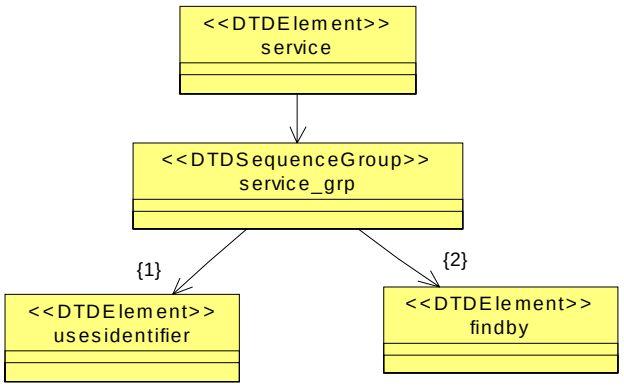

Figure 7.39 - *service* Element Relationships

<!ELEMENT service
(usesidentifier
, findby
)>

Annex A Software Radio Reference Sheet

The Software Radio specification responds to the requirements set by “Request for Proposals for a Platform Independent Model (PIM) and CORBA Platform Specific Model (PSM)” (swradio/02-06-02). The original specification (dtc/ 05-10-02) has been reorganized into 5 volumes, as follows:

Volume 1. Communication Channel and Equipment

This specification describes a UML profile for communication channel. The profile provides definitions for creating communication channel and communication equipment definitions. The specification also provides radio control facilities and physical layer facilities PIM for defining interfaces and components for managing communication channels and equipment for a radio set or radio system. Along with the profile and facilities is a platform specific model transformation rule set for transforming the communication channel into an XML representation and CORBA interfaces for the radio control facilities.

Volume 2. Component Document Type Definitions

This specification defines the content of a standard set of Data Type Definition (DTD) files for applications, components, and domain and device management. The complete DTD set is contained in Section 7, Document Type Definitions. XML files that are compliant with these DTD files will contain information about the service components to be started up when a platform is power on and information for deploying installed applications.

Volume 3. Component Framework

This specification describes a UML profile for component framework. The profile provides definitions for applications, components (properties, ports, interfaces, etc.), services, artifacts, logical devices, and infrastructure domain management components. In the profile are also library packages that contain interfaces for application, service, logical device, and infrastructure domain management components. Along with the profile is a platform specific model transformation rule set for transforming the profile model library interfaces into CORBA interfaces.

Volume 4. Common and Data Link Layer Facilities

This specification describes a set of facilities PIM for application and component definitions. The set of facilities are common layer facilities and data link layer facilities that can be utilized in developing waveforms and platform components, which promote the portability of waveforms across Software Defined Radios (SDR). Along with the facilities PIM is a platform specific model transformation rule set for transforming the facilities into CORBA interfaces.

Volume 5. POSIX Profiles

This specification defines the application environment profiles for embedded constraint systems, based on Standardized Application Environment Profile - POSIX® Realtime Application Support (AEP), IEEE Std 1003.13-1998.

INDEX