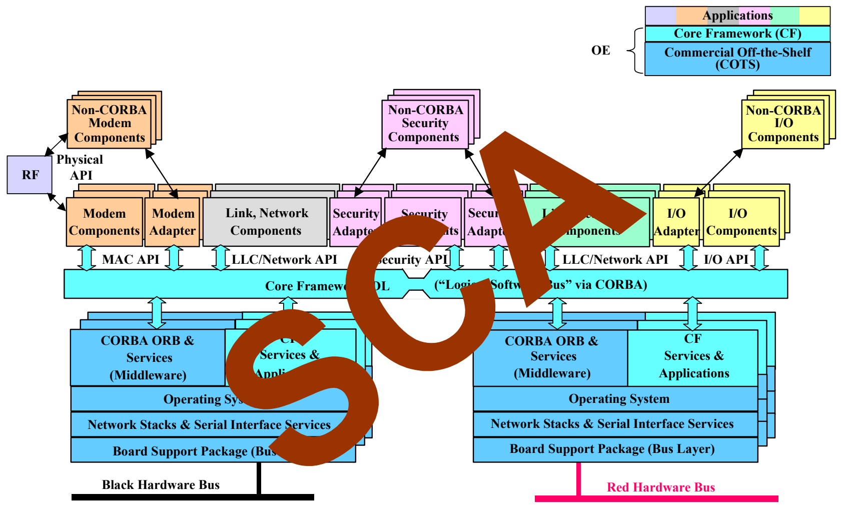
Software Communications Architecture



Neli Hayes
Principal Software Architect
SCA Core Framework Team
JTRS Cluster 1 Program
The Boeing Company, Anaheim, CA
persnaz.n.hayes@boeing.com
(714) 762-8768

Objectives

Provide an overview of reasons for creation of the SCA.

Describe what the SCA is, what it is not, and what it enables.

Describe SCA's maturity, and where it is being used today.

Introduce the SCA specification, supplements, accompanying documents, available formal training, and emerging SCA-based standards.

Provide a comprehensive overview of the SCA core architecture rule set.

Depict example sequences of how the SCA interacts with SCA-compliant application components.

Introduce SCA's CORBA IDL modules.

Outline

Outline (cont.)

Outline (cont.)

Outline (cont.)

SW Architecture Overview (cont.)

Outline (cont.)

Outline (cont.)

Why was the SCA Created?

Tactical Communication Systems Limitations

Evolved to meet service-specific and mission-specific requirements.

- Cannot communicate with each other.
- Cannot take advantage of rapid advances in technology in a timely and cost-effective manner.

Hence, the Joint Tactical Radio System (JTRS) Program was formed.

BOEING®

JTRS Program – Why?

DOD Complaints to Radio Manufacturers:

"We can't support all your discrete radios"

"We can't fight the way we want to with your discrete radios"

"We can't introduce new technology at the pace of commercial industry"

JTRS Program

A DOD Joint Program for an information warfare communication system built on a family of HF-to-L-band radios with a common open architecture with these…

Requirements

- Software controlled and reprogrammable
- Modular and scaleable
- Extensive use of COTS technology
- Simplified applications engineering
- Rapid deployment of system improvements

SOFTWARE DEFINED RADIO

BOEING®

What is a Software Defined Radio?

Collection of hardware and software technologies that enable reconfigurable system architectures for wireless networks and user terminals.

Efficient and comparatively inexpensive solution for building multimode, multi-band, multifunctional wireless devices that can be adapted, updated, or enhanced using software upgrades.

SDR Concepts Allow

- Standard, open, flexible architectures.
- Enhanced wireless roaming by extending capabilities of current and emerging commercial air interface standards.
- Over-the-air downloads of new services, features, software patches.
- Advanced networking capabilities allowing truly portable networks.
- Unified communication across commercial, civil, federal and military organizations.
- And, therefore, significant life-cycle reduction costs.

http://www.sdrforum.org

JTRS Target – A Family of Software Defined Radios

Radio functionality Provided through Software (versus Hardware)

- Mostly software applications (rather than hardware components) provide waveform generation and processing, encryption, signal processing, and other major communications functions.

Programmable

- Can accommodate various physical layer formats and protocols.
- Multiple software modules allow implementation of different standards in the same radio system.
- Can be altered to implement both legacy communication systems as well as those yet to be defined.

Reconfigurable

- Support dynamic partitioning of radio resources and re-characterization of radio in response to user needs.

Easily Upgradeable

- New/improved functionality easily incorporated, w/o need to upgrade/replace hardware components.

Responsive

- Can dynamically modify their operation in response to changes in environment.

Less Costly Maintenance

- Over-the-air downloads of features, services, software patches.

Inception of the SCA

Four of the top defense radio manufacturers get together and define a standard software architecture--the ***Software Communications Architecture (SCA)***--that can support the JTRS Requirements.

- The Modular Software-programmable Radio Consortium (MSRC)
  - Raytheon, BAE SYSTEMS, Rockwell-Collins, ITT

BOEING®

What is the SCA?

The SCA is…

An Open Architecture Framework

Based on

- CORBA
- CORBAservices
- CORBA Component Model
- POSIX

Expressed in

- CORBA IDL
- UML
- XML

Series of Mandated Interfaces, Behavioral Specifications and Requirements

Promote component portability, interchangeability, and interoperability, software reuse, and architecture scalability.

Standard

For communication platform hardware & software design engineers to ensure a system's SCA-compliancy just as a building architect or planner uses a local building code to design and build homes to ensure sound construction and safety.

The SCA Is Not…

<u>Is not</u> a system specification.

- It is a set of rules that constrain the design of systems, to achieve SCA objectives.

<u>Is not</u> a design specification.

- Does not tell hardware and software designers <u>how</u> to design their equipment and programs.
- SCA requirements limited to those necessary to meet SCA-compliant system criteria, w/o restricting innovation or domain-specific requirements.

<u>Is not</u> an implementation architecture.

- There are many valid definitions of an SCA architecture.
- A particular SCA implementation architecture is a result of joining between the SCA, the particular domain and project technical specifications, and other procurement requirements.

BOEING®

SCA Maturity
Where is it Today?

Phased Development to Prove the Architecture Works

JTRS Step 1 – Architecture Development

- MSRC members develop initial architecture

JTRS Step 2a – Architecture Feasibility

- MSRC members deliver prototype radios -- SCA Core Framework and waveform implementations
- Specification maturation
- Pursue commercial acceptance (OMG/SDR Forum)

JTRS Step 2b – Independent Architecture Validation

- Non-MSRC members deliver prototype radios
  - Harris/General Dynamics deliver prototype radios
  - Boeing – Delivers SCA Core Framework
  - Thales/Vanu provides handheld radios
  - Rockwell-Collins provides prototype waveform software

JTRS Step 2c – Ruggedized Fieldable Radio

- BAE SYSTEMS chosen to deliver SCA compliant radios for field evaluation

SCA Compliance Mandatory on JTRS Clusters

As well as any DoD Program involved with Network Centric Operations (NCO) and Network Centric Warfare (NCW)

Cluster 1

- Multi-channel software programmable, hardware-configurable digital radio networking system for the Army.

Cluster 2

- Handheld and man-pack radios for the Army, Navy Marine Corps and Air Force.

AMF (JTRS Airborne and Maritime/Fixed Station, Mergers of Clusters 3 & 4)

- Acquisition of JTR sets for Air Force and Navy platforms.

Cluster 5

- JTRS hand-held and man pack units for embedded platforms suitable for Small Form Fit (SFF) radios.

Future Clusters

- Will respond to interests from the Homeland and Security operational communities and evolving DoD requirements

Public Domain Detail at http://jtrs.army.mil/

BOEING®

Standardization Efforts

JTRS SCA Technical Architecture Group (TAG)

- An SCA discussion forum, invites the JTRS community members to propose and develop change proposals to the SCA and its related appendices and supplements. Oversees submitted changes and leads effort to research, accept or reject each change proposal.
- https://jtel-sca.spawar.navy.mil/

Software Defined Radio Forum (SDRF)

- Working with JTRS JPO, adopted the SCA in 2000 as a body of work mature enough to move out to a formal standards body, the Object Management Group (OMG). Involved with development of SCA-based software radio standards.
- http://www.sdrforum.org/

OMG Software-Based Communication Domain Task Force (DTF)

- Former OMG Software Radio Domain Special Interest Group (DSIG)
- With JPO sponsorship, works toward building an international commercial standard based on the SCA. Most recent efforts include the *PIM & PSM for Software Radio Components Joint Revised Submission*.
- http://sbc.omg.org

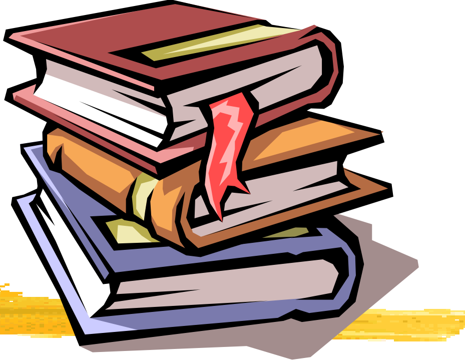
International Acceptance

- Individual nations, such as France and Sweden have adopted or are considering adopting forms of the SCA.
- The US and others are also working within NATO to have the SCA adopted.

BOEING®

SCA Specifications, Supporting Documents, Training, Emerging SCA-Based Standards

SCA Specifications



SCA Specification V2.2.1

Basic architecture definition & rule sets governing SCA-compliant hardware and software development.

SCA Application Program Interface Supplement V2.2.1

Common APIs beyond the SCA core rule set & standards for defining app-specific common APIs to promote further app portability.

SCA Security Supplement V2.2.1

Security requirements for SCA-compliant systems.

Available at http://jtrs.army.mil/

BOEING®

SCA Specifications

Specialized HW Supplement to the SCA V3.0

Portability for specialized hardware such as

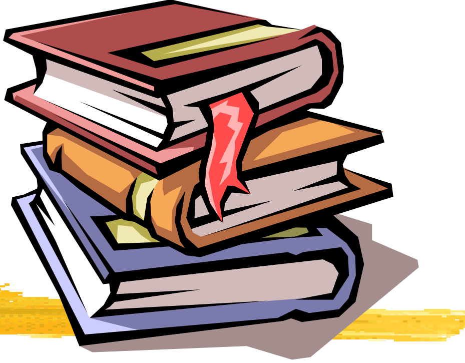
- *Field Programmable Gate Arrays (FPGA)*
- *Digital Signal Processors (DSP)*
- *Application Specific Integrated Circuits (ASIC))*

In particular, it specifies

- *a Hardware Abstraction Layer Connectivity (HAL-C) specification*
- *a reduced POSIX AEP for DSP environments,*
- *Waveform functional blocks to be provided as part of each radio set*
- *an Application Interface for antenna interfaces.*

Draft July 9, 2004 available at http://jtrs.army.mil/

SCA Supporting Documents & Training



SCA
Support
&
Rationale
Document
V2.2

Rationale behind architectural rule set decisions accompanied with supporting material.

SCA
Developers
Guide
V1.1

Design guidelines for developing SCA-compliant applications and devices.

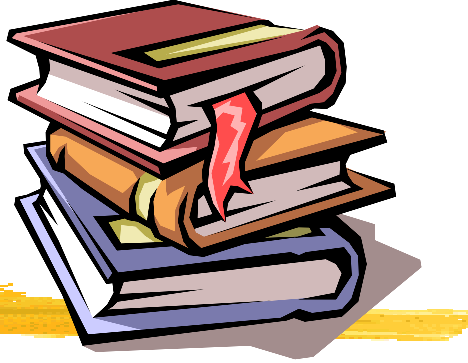
SCA
Training
Course

- *SCA Overview*
- *Security & APIs*
- *Application Design*
- *Application XML Files*
- *Porting and Test*

Available at http://jtrs.army.mil/

BOEING®

Emerging SCA-Based Standards



Based on the SCA.
Highly recommended reading for SCA software radio system and software engineers and architects, specially on JTRS and related programs.
Sheds a lot of light on the SCA CF infrastructure, how it relates to software radio concepts.

PIM & PSM for Software Radio Components
April 2004

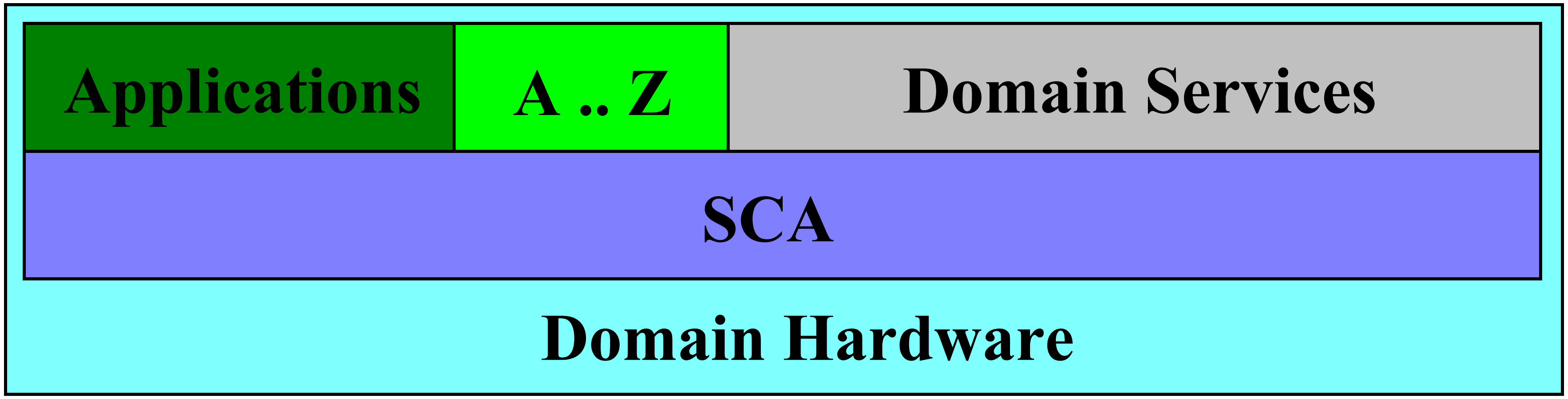
Defines platform independent model and a UML Software Profile for software radios, describing in great detail, all common aspects of a software radio (i.e. the entire radio management infrastructure, radio devices and services, and …), explaining concepts such as communication channels, and how these concepts map to the SCA infrastructure (for example exactly what is a communication channel and how does it map to the SCA infrastructure for CF, devices, services, and the operating environment).

Available at http://www.omg.org/cgi-bin/doc?sbc/2004-04-01

Software Architecture Overview

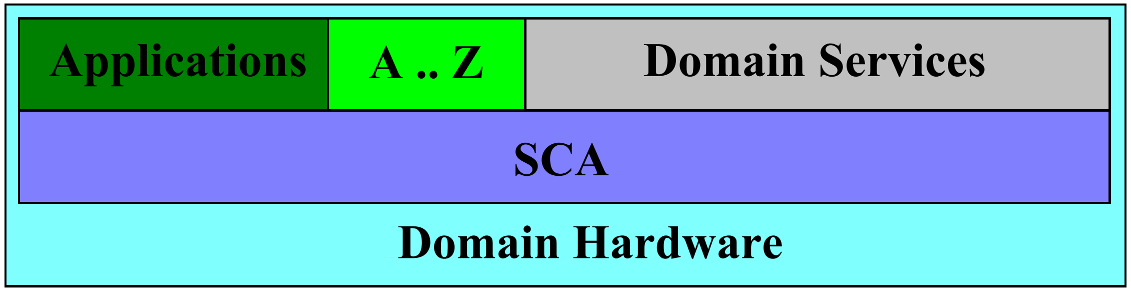
What does the SCA Enable for Communication Systems?

- Maximum independence of software from specific hardware solutions.
- Portability of applications software across diverse SCA-compliant platforms in the same domain.
- Interoperability of applications software.
- Reuse of common software across these platforms
- Scalability across platforms – same architecture from handheld to base station.

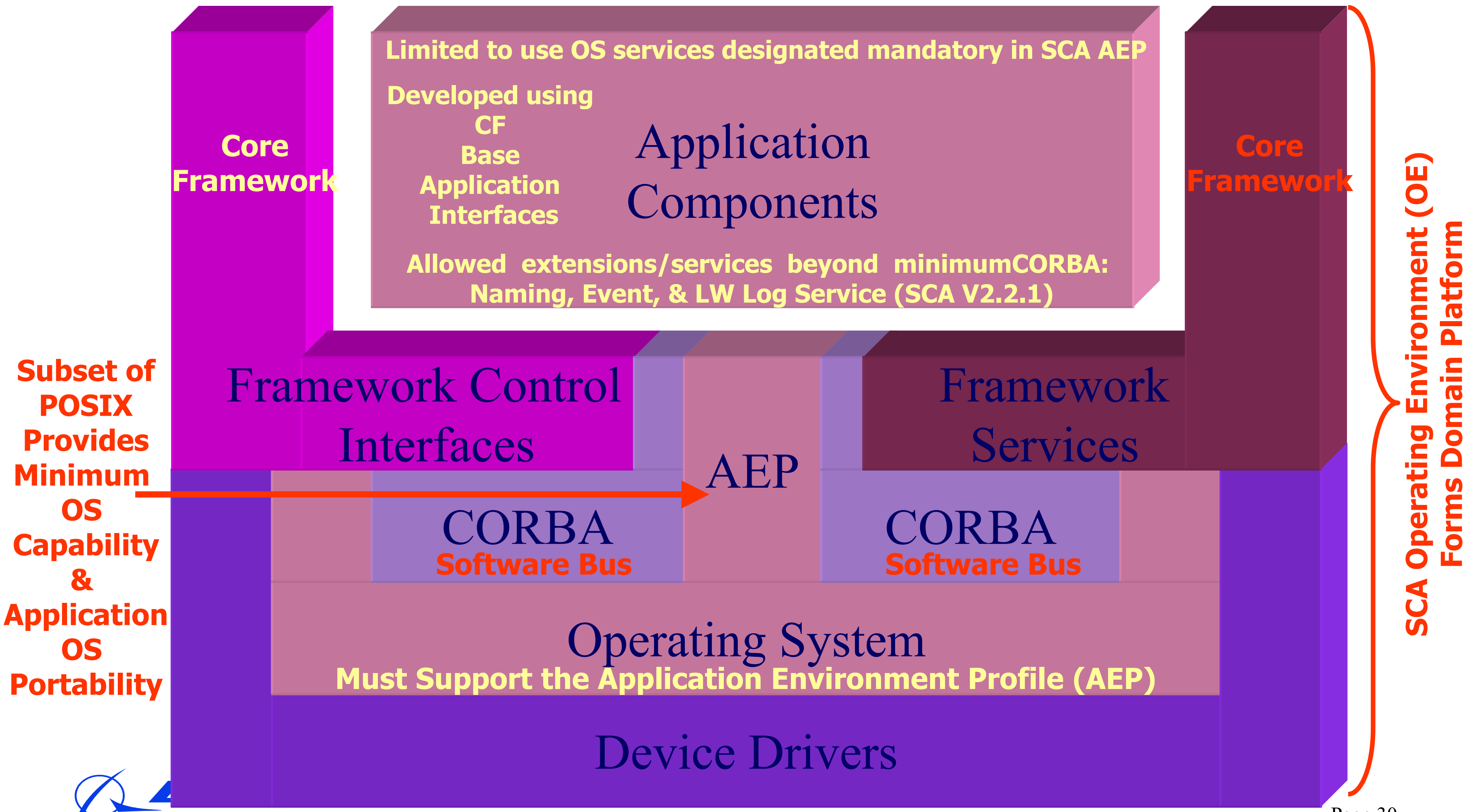


How does the SCA Enable Such Things?

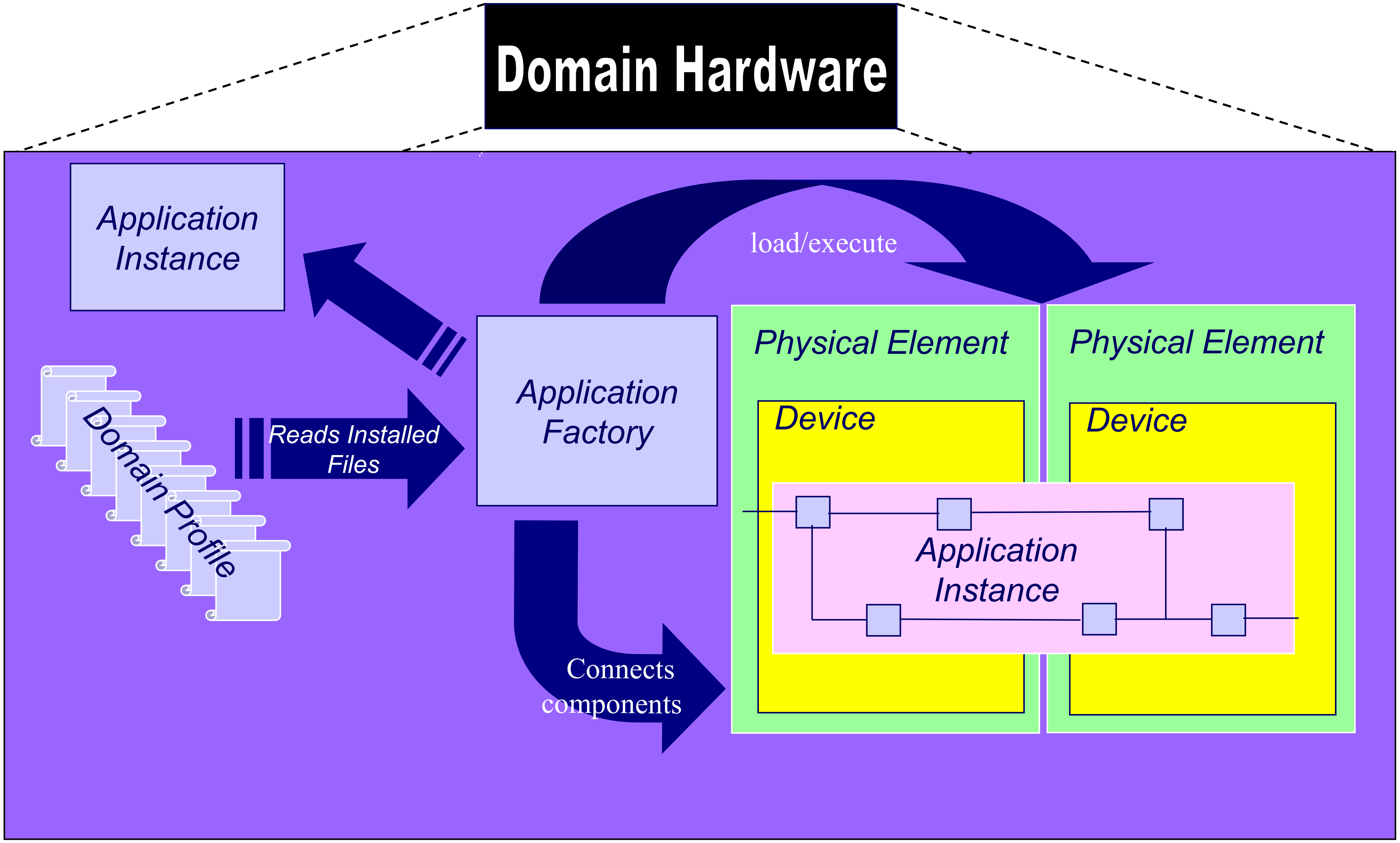
- Defines a distributed, object-oriented, language-independent, and platform-independent operating environment with common software interfaces and framework.
  - For deployment, configuration, and tear down of applications, and management of the domain that hosts these applications.
- Defines a standard way to package and deploy application components.
- Separates applications from the operating environment.
- Segments application functionality.
- Defines and provides the means for defining application-specific common services and APIs.
  - To promote further application portability.



SCA Layering


Limited to use OS services designated mandatory in SCA AEP
Developed using CF Base Application Interfaces
Application Components
Allowed extensions/services beyond minimumCORBA: Naming, Event, & LW Log Service (SCA V2.2.1)
Core Framework
Core Framework
Framework Control Interfaces
AEP
Framework Services
CORBA Software Bus
CORBA Software Bus
Operating System
Must Support the Application Environment Profile (AEP)
Device Drivers
Subset of POSIX Provides Minimum OS Capability & Application OS Portability
SCA Operating Environment (OE) Forms Domain Platform

SCA Application Instantiation



SCA Operating Environment

Operating Environment (OE) – forms domain platform

- Core Framework
  - Domain Profile
- CORBA middleware (software bus)
- Application Environment Profile
  - Defined by SCA to provide minimum operating system functionality
- Operating System
- Device Drivers

OE's Job

- Impose design constraints on applications to provide increased portability of applications from one SCA-compliant platform to another
  - Specified interfaces between the CF and application software.
  - Restrictions on applications' usage of OS and CORBA middleware APIs.

CF and Domain Profile

Core Framework (CF)

- Set of software interfaces and profiles that provide for the deployment, management, interconnection, and intercommunication of software application components in embedded communication systems, using CORBA to communicate between entities.

Domain Profile

- XML files that describe:
  - Individual components of a software application.
  - How the components are interconnected.
  - Component properties.
  - Properties of hardware device abstractions.

SCA OE Mandates

RTOS

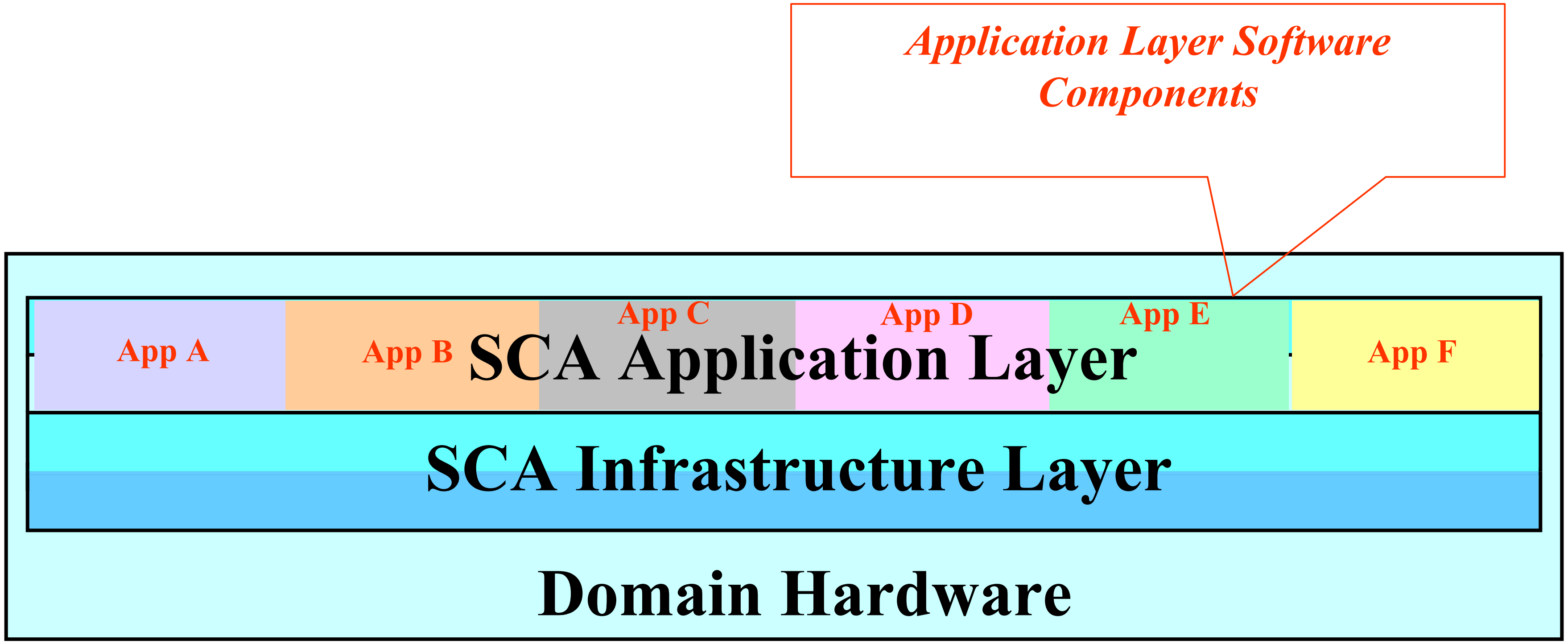
- RTOS must Support SCA Application Environment Profile (AEP).
- The SCA AEP is a subset of the POSIX.13 Real-time Controller System Profile (PSE52).
- Applications are limited to using the RTOS services that are designated as mandatory in the SCA AEP.

CORBA

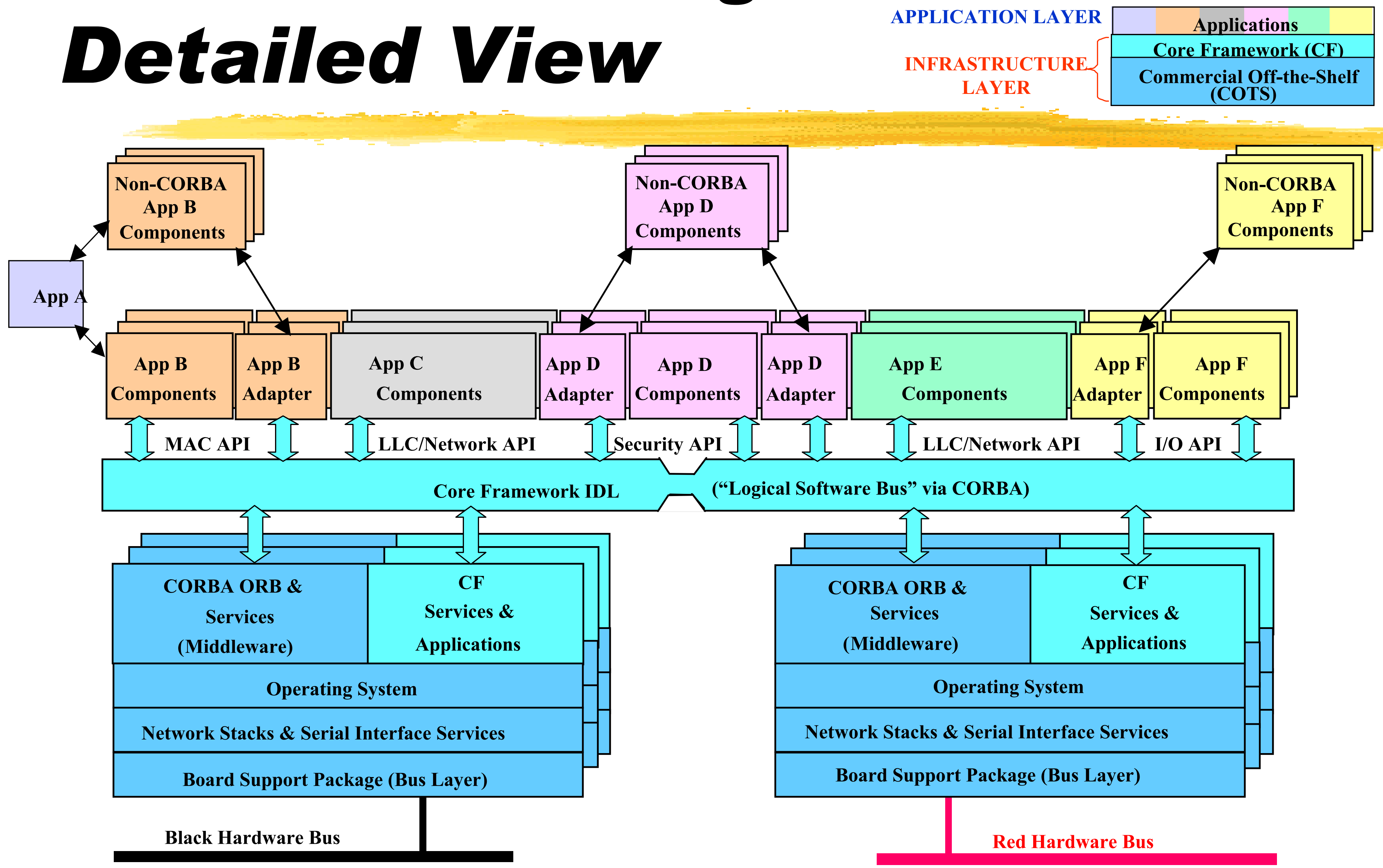
- No extensions and/or services beyond Minimum CORBA can be used except as specified in the SCA.
  - CORBA Naming Service
  - CORBA Event Service
  - OMG Light Weight Log Service (SCA V2.2.1)
- Desired extensions listed as optional, pending commercial availability
  - Interoperable Naming Service
  - Real-Time CORBA
  - CORBA Messaging

BOEING®

SCA Partitioning Birds Eye View

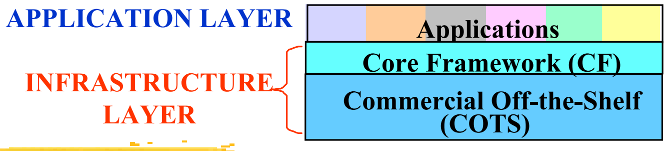


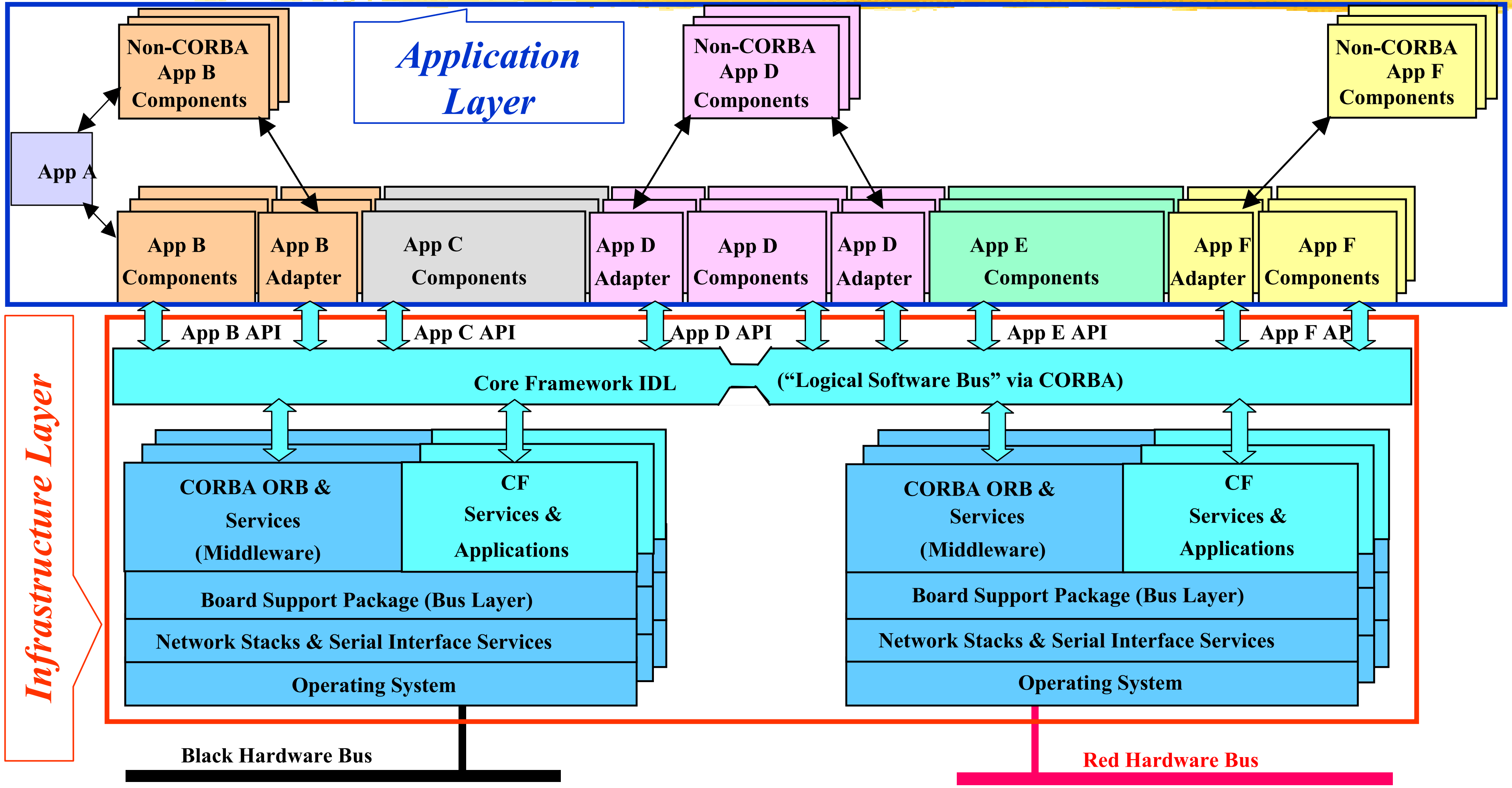
SCA Partitioning Detailed View


APPLICATION LAYER
Applications
INFRASTRUCTURE LAYER
Core Framework (CF)
Commercial Off-the-Shelf (COTS)
Non-CORBA App B Components
Non-CORBA App D Components
Non-CORBA App F Components
App A
App B Components
App B Adapter
App C Components
App D Adapter
App D Components
App D Adapter
App E Components
App F Adapter
App F Components
MAC API
LLC/Network API
Security API
LLC/Network API
I/O API
Core Framework IDL
("Logical Software Bus" via CORBA)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

BOEING®

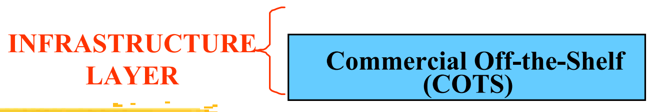
SCA Partitioning Major Divisions



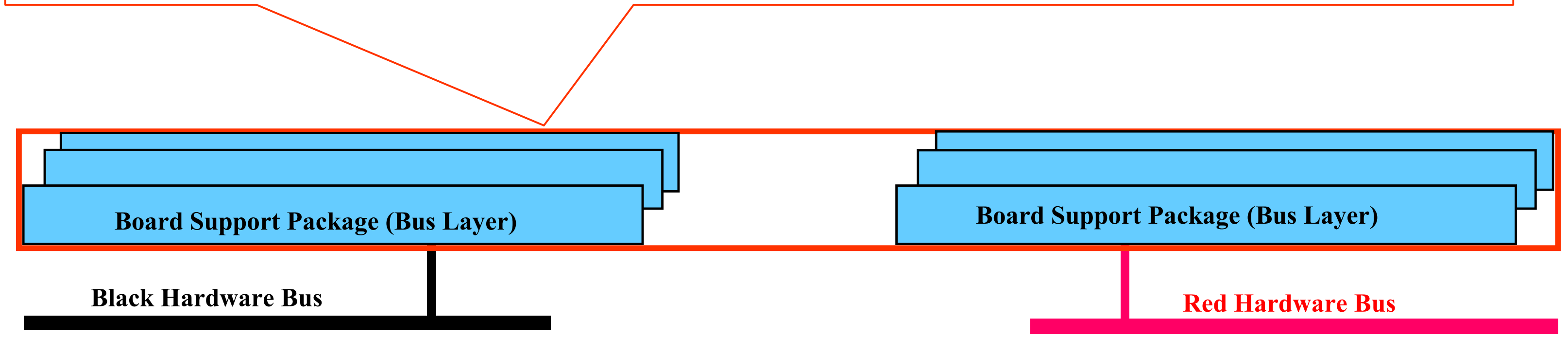


Application Layer
Non-CORBA App B Components
Non-CORBA App D Components
Non-CORBA App F Components
App A
App B Components
App B Adapter
App C Components
App D Adapter
App D Components
App D Adapter
App E Components
App F Adapter
App F Components
App B API
App C API
App D API
App E API
App F API
Core Framework IDL
("Logical Software Bus" via CORBA)
Infrastructure Layer
CORBA ORB & Services (Middleware)
CF Services & Applications
Board Support Package (Bus Layer)
Network Stacks & Serial Interface Services
Operating System
CORBA ORB & Services (Middleware)
CF Services & Applications
Board Support Package (Bus Layer)
Network Stacks & Serial Interface Services
Operating System
Black Hardware Bus
Red Hardware Bus

Infrastructure Bus Layer

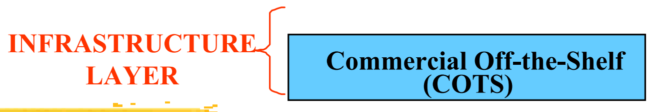


- *Transport mechanisms that perform error checking and correction at the bus support level.*
- *Possible commercial bus architectures: VME, PCI, CompactPCI, Firewire, Ethernet, etc..*
- *Different bus architectures could be used on the Red and Black Subsystems.*

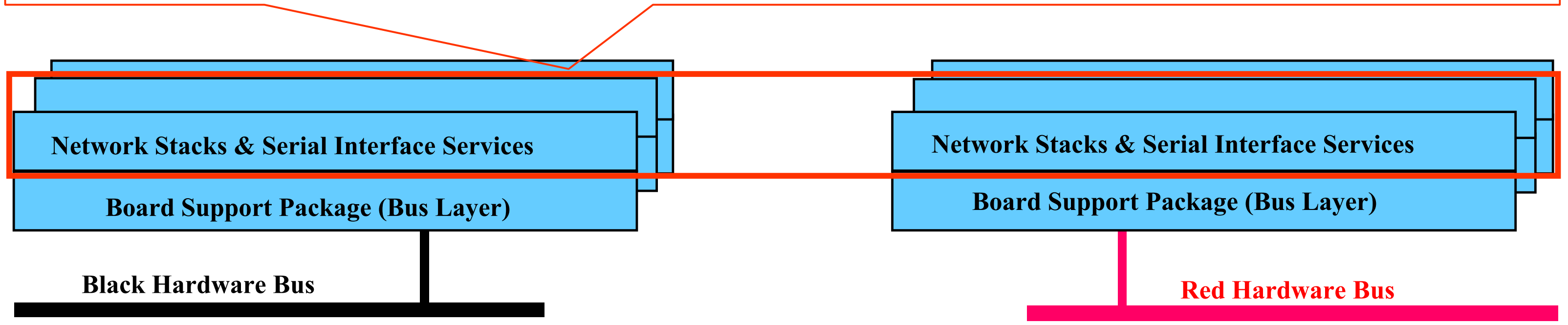


Board Support Package (Bus Layer)
Board Support Package (Bus Layer)
Black Hardware Bus
Red Hardware Bus

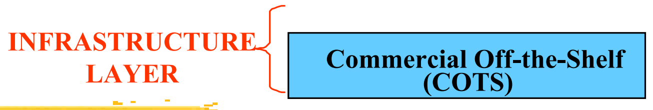
Infrastructure Network & Serial Interface Services



- *Commercial components support multiple unique serial and network interfaces.*
  - *RS-232, RS-422, RS-423, RS-485, Ethernet, 802.x, etc.).*
- *To support these interfaces, various low-level network protocols may be used (e.g. PPP, SLIP, LAPx, and others).*


Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

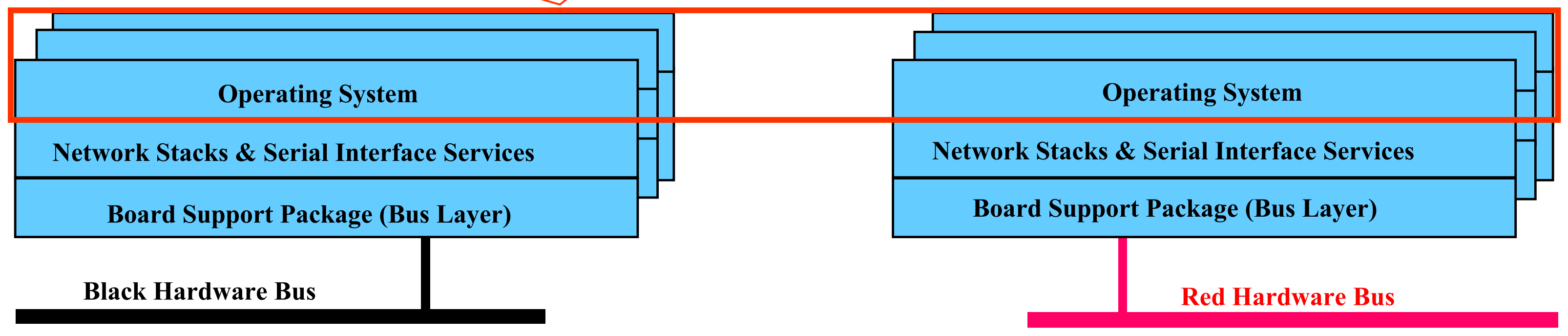
Infrastructure OS Layer



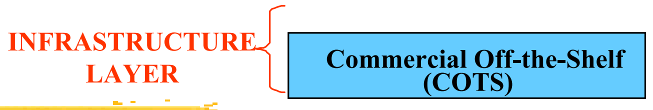
Real-time embedded OS functions provide needed support for Infrastructure and Application layers. For example:

- *Booting the processor*
- *Multi-threading support*
- *I/O support*
- *Inter-process and inter-device support*
- *Other general-purpose capabilities required for real-time embedded applications*

As such, may be present on all CORBA-capable processors in a system.

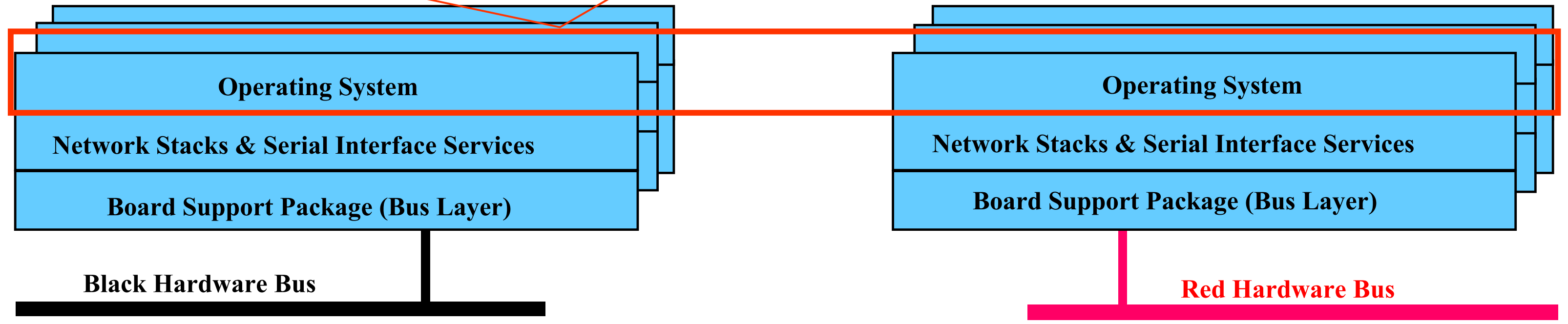

Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

Infrastructure OS Layer (cont.)

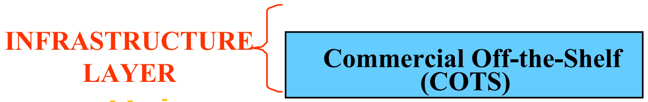


May be tailored to support specific target/board environments. For example:

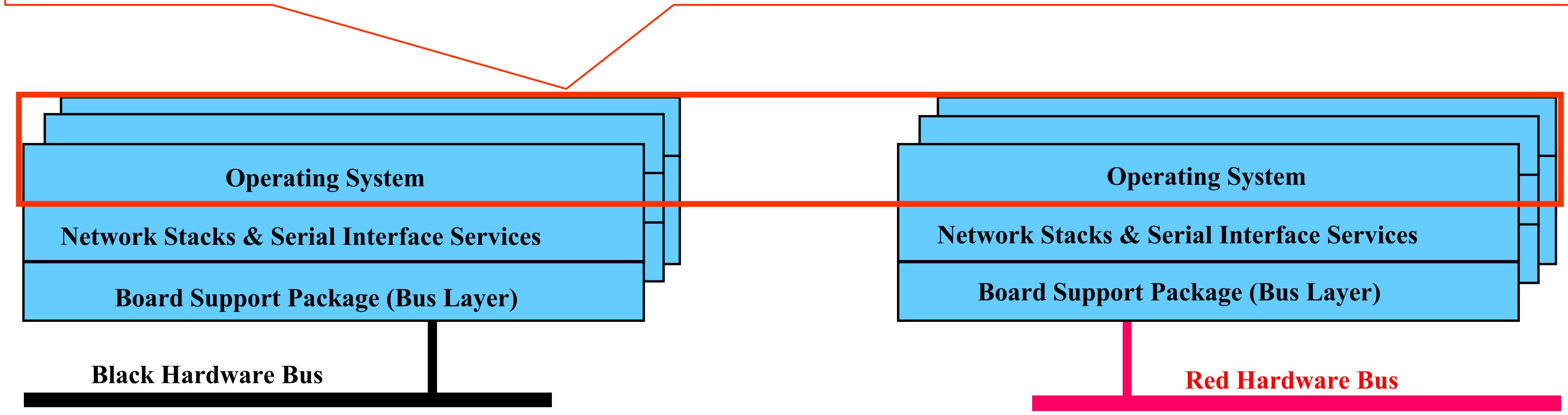
- *A Board Support Package (BSP), can tailor the RTOS for specific processor board elements, including the device drivers that support the specific devices/chip sets resident on the board.*
- *Device drivers to support the necessary inter-processor communication required by the ORB.*


Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

Infrastructure OS Layer (cont.)



A standard OS interface is required to facilitate portability of applications. As POSIX is an accepted industry standard, and POSIX and its real-time extensions are compatible with the requirements to support the OMG CORBA specification, the SCA defines a minimal POSIX profile, known as the SCA Application Environment Profile (AEP; SCA Appendix A), based on the Real-time Controller System Profile (PSE52) as defined in POSIX 1003.13, to meet SCA requirements. More on OS usage restrictions enforced on the Infrastructure and Application Layers later.


Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

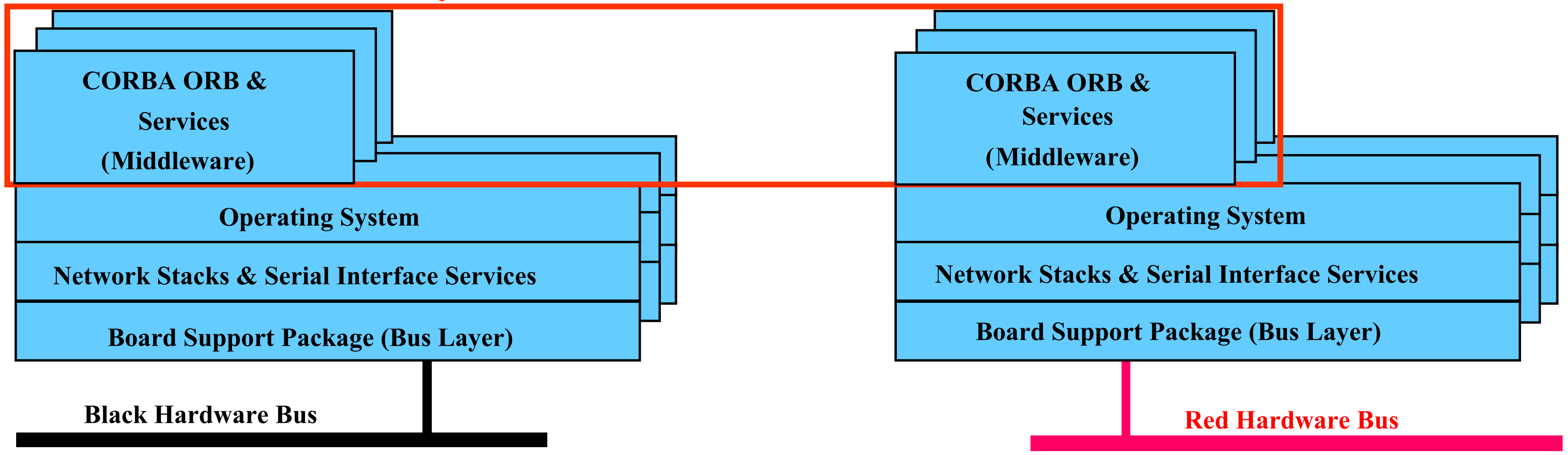
Infrastructure CORBA Middleware

INFRASTRUCTURE LAYER

Commercial Off-the-Shelf (COTS)

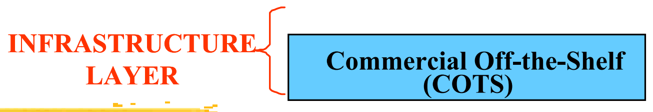
Distributed processing is a fundamental aspect of the system architecture and is based on the OMG CORBA specification. It is used in the Infrastructure Layer as the message passing technique for the distributed processing environment.

Why CORBA?



CORBA ORB & Services (Middleware)
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
CORBA ORB & Services (Middleware)
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

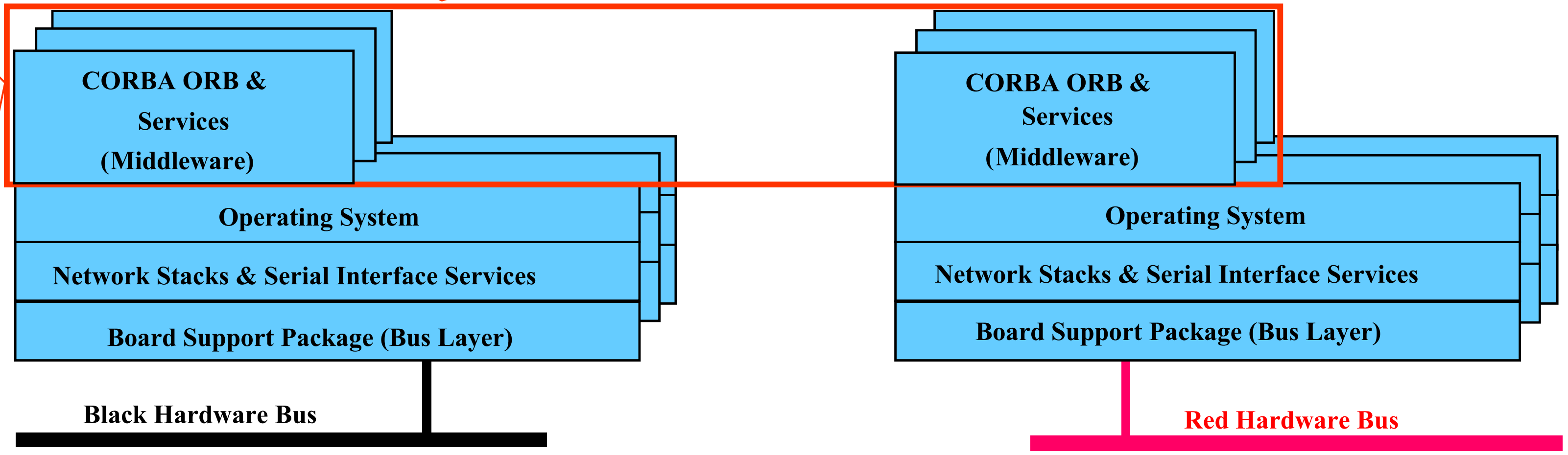
Infrastructure CORBA Middleware (cont.)



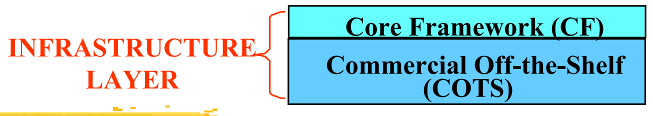
OMG's CORBA specification: a specification for a "software bus" or "software back plane".

- *Middleware for establishing relationships between clients and servers in a distributed system.*
- *Provides interoperability between applications on different machines, difference OSs, different networking protocols, and different programming languages.*
- *Supports location transparent invocation of server objects by client objects.*
- *Components are "plugged into" the software bus, and are visible to other components.*
- *More later on the ORB and CORBA services used in the Infrastructure Layer.*

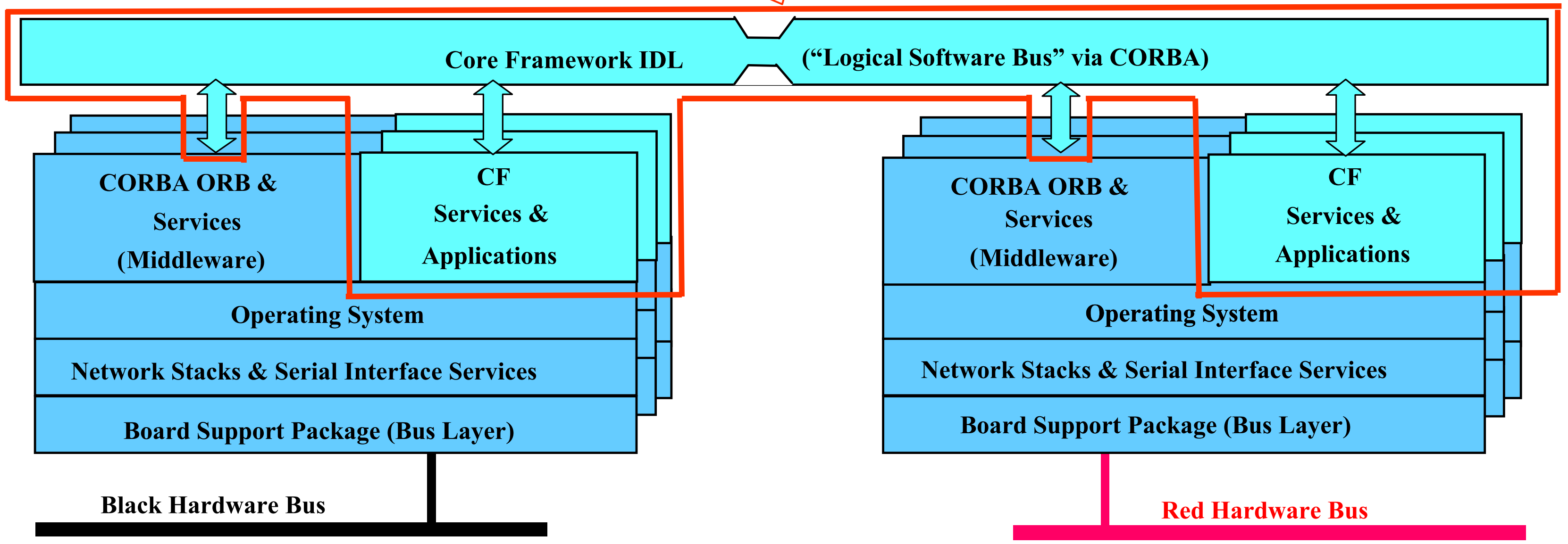
The "software bus" and the middleware for the SCA OE.


CORBA ORB & Services (Middleware)
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
CORBA ORB & Services (Middleware)
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

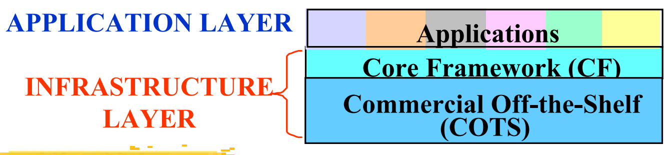
Infrastructure Core Framework

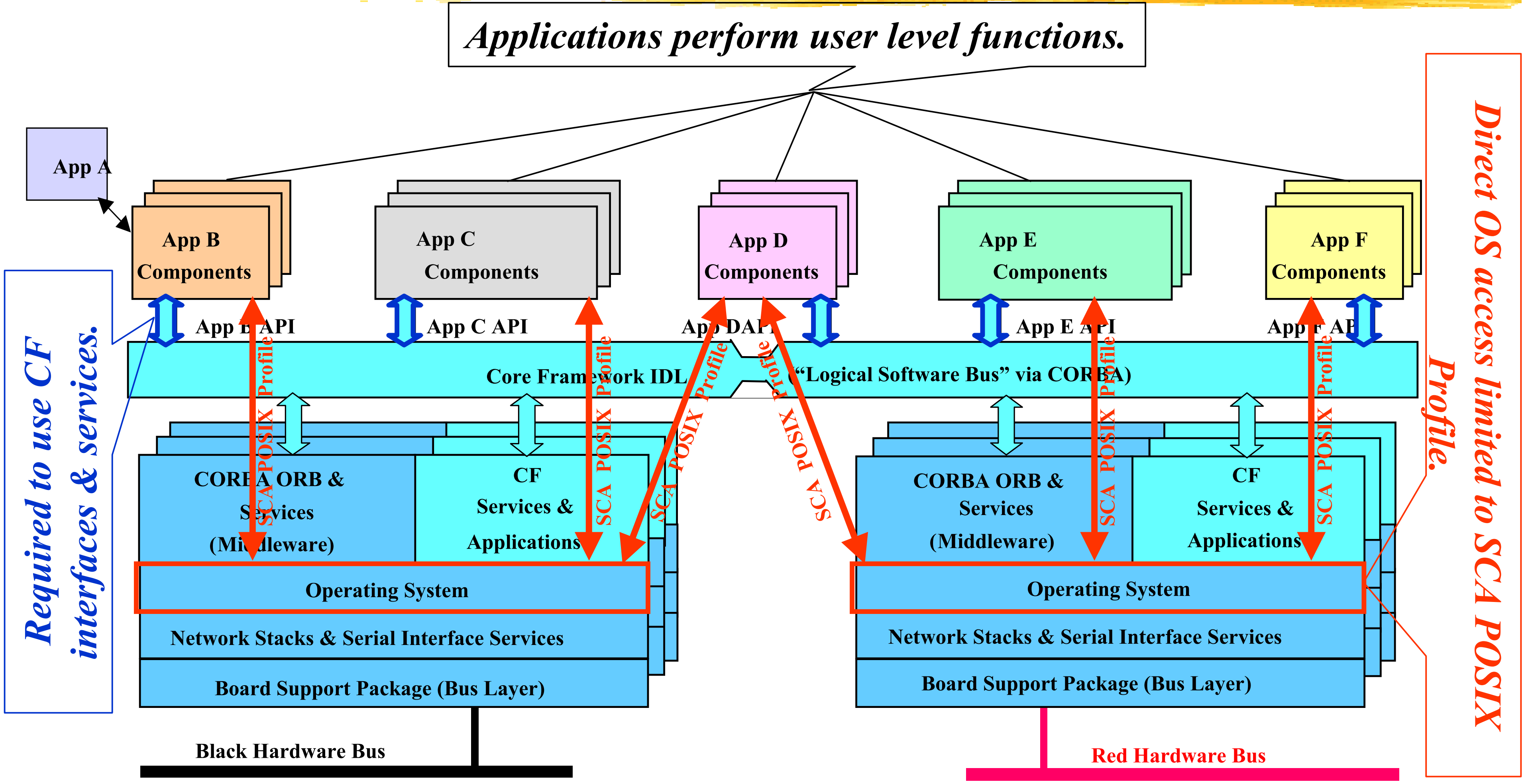


The CF is the essential ("core") set of open application-layer interfaces and services to provide an abstraction of the underlying software and hardware layers for software application designers. More on CF later.

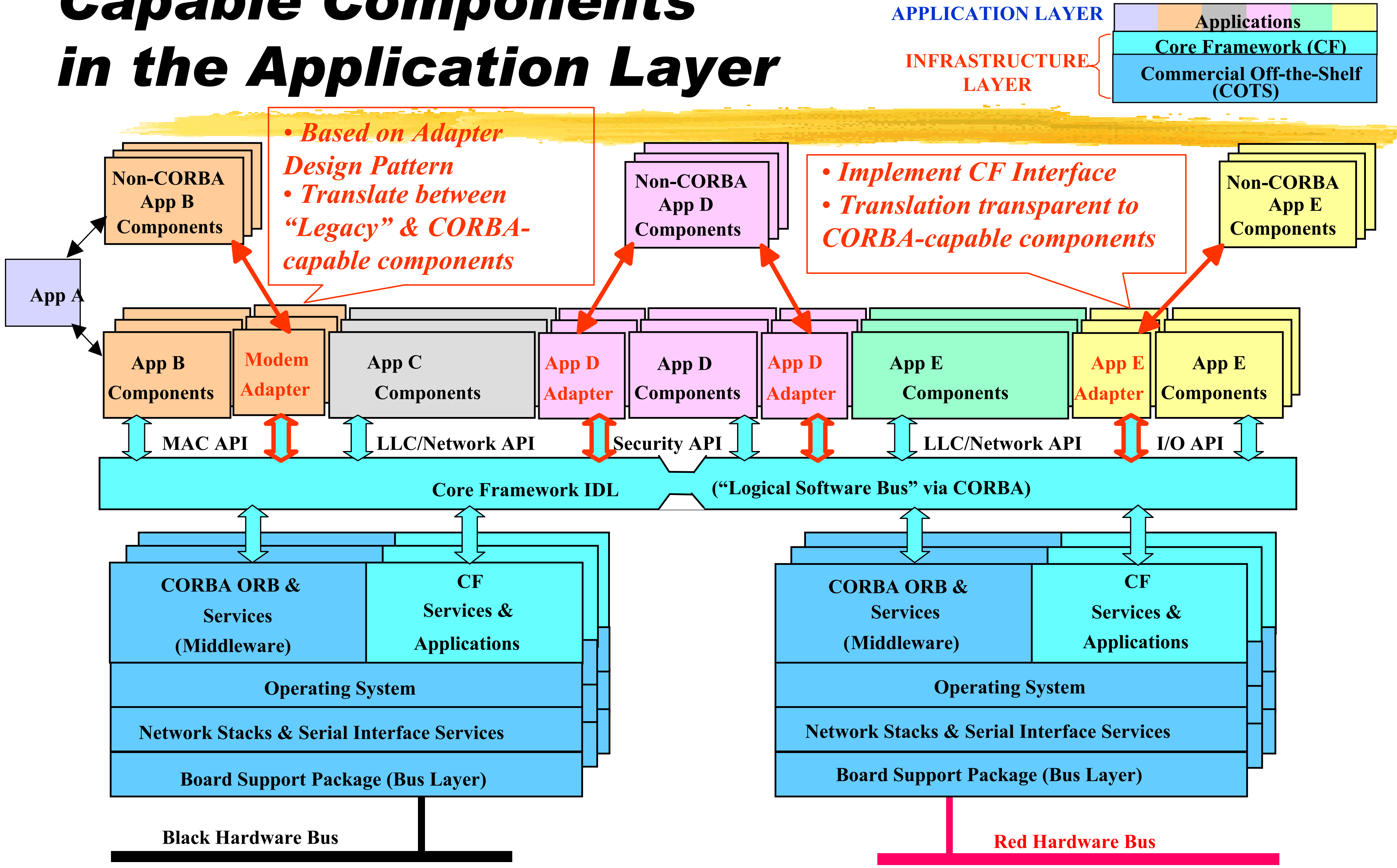

Core Framework IDL
("Logical Software Bus" via CORBA)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Red Hardware Bus

Application Layer CORBA Capable Components




Applications perform user level functions.
App A
App B Components
App C Components
App D Components
App E Components
App F Components
App B API
App C API
App D API
App E API
App F API
Core Framework IDL ("Logical Software Bus" via CORBA)
SCA POSIX Profile
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
Red Hardware Bus
Required to use CF interfaces & services.
Direct OS access limited to SCA POSIX Profile.

Adapters Allowing Use of Non-CORBA Capable Components in the Application Layer



APPLICATION LAYER
INFRASTRUCTURE LAYER
Applications
Core Framework (CF)
Commercial Off-the-Shelf (COTS)
• Based on Adapter Design Pattern
• Translate between "Legacy" & CORBA-capable components
• Implement CF Interface
• Translation transparent to CORBA-capable components
Non-CORBA App B Components
Non-CORBA App D Components
Non-CORBA App E Components
App A
App B Components
Modem Adapter
App C Components
App D Adapter
App D Components
App D Adapter
App E Components
App E Adapter
App E Components
MAC API
LLC/Network API
Security API
LLC/Network API
I/O API
Core Framework IDL
("Logical Software Bus" via CORBA)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Black Hardware Bus
Red Hardware Bus

SCA Partitioning Benefits

Maximizes use of commercial protocols and products.

Isolates core and non-core applications from underlying hardware through multiple layers of open, commercial software infrastructure.

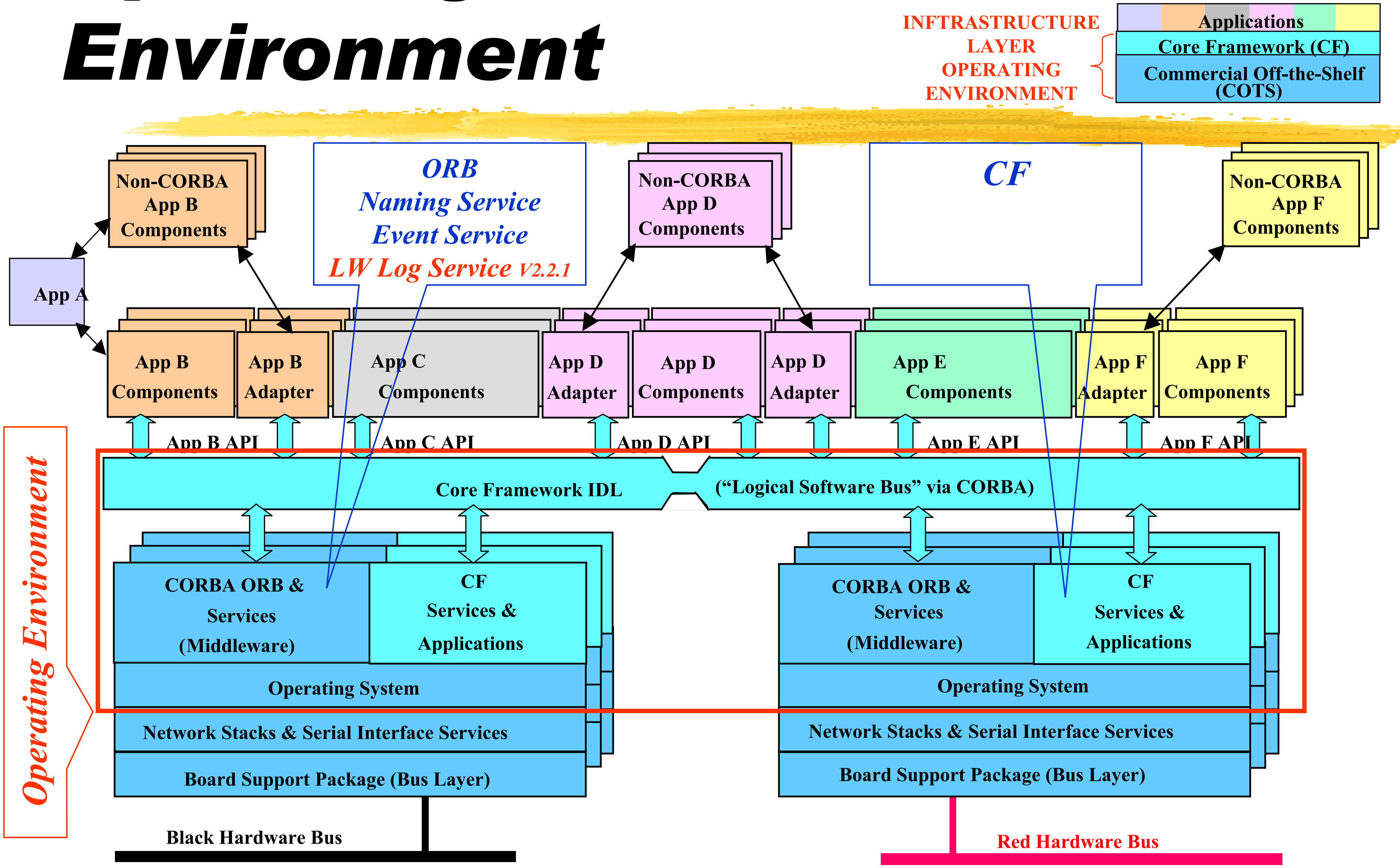
Provides for a distributed processing environment through CORBA.

Hence, provides software application portability, reusability, and scalability.

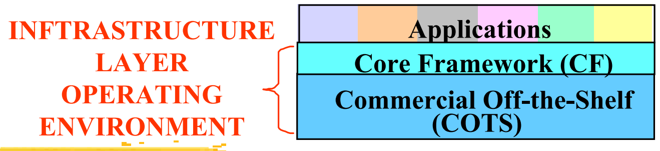
Infrastructure Layer Operating Environment

BOEING®

Operating Environment


INFTRASTRUCTURE LAYER OPERATING ENVIRONMENT
Applications
Core Framework (CF)
Commercial Off-the-Shelf (COTS)
ORB
Naming Service
Event Service
LW Log Service V2.2.1
CF
Non-CORBA App B Components
Non-CORBA App D Components
Non-CORBA App F Components
App A
App B Components
App B Adapter
App C Components
App D Adapter
App D Components
App D Adapter
App E Components
App F Adapter
App F Components
App B API
App C API
App D API
App E API
App F API
Core Framework IDL
("Logical Software Bus" via CORBA)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
CORBA ORB & Services (Middleware)
CF Services & Applications
Operating System
Network Stacks & Serial Interface Services
Board Support Package (Bus Layer)
Operating Environment
Black Hardware Bus
Red Hardware Bus

Operating Environment's Job



Impose design constraints on applications.

– To provide increased portability of applications from one SCA-compliant platform to another.

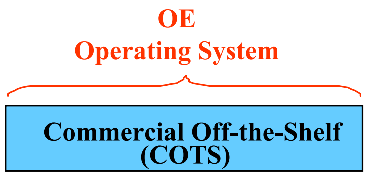
These design constraints include:

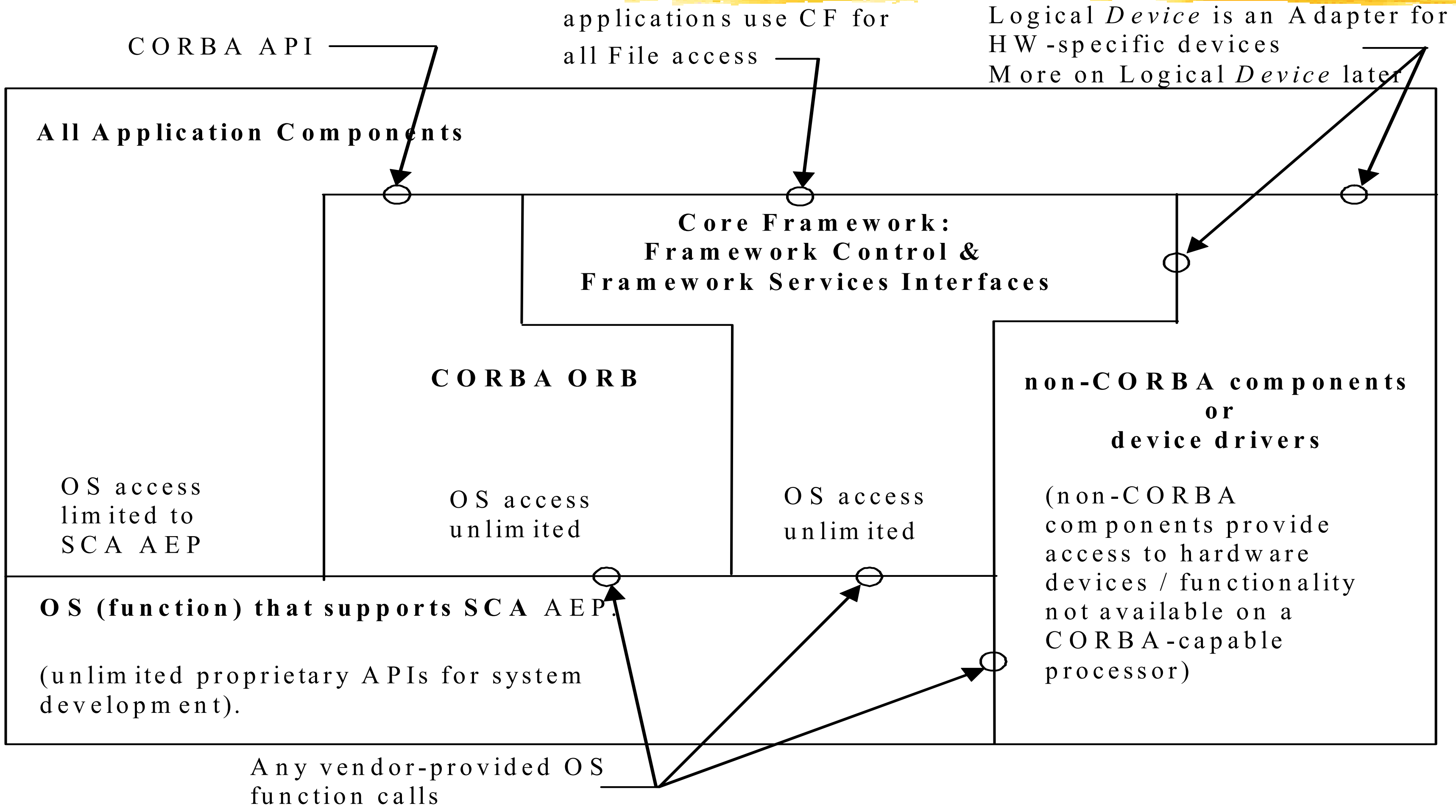
– Specified interfaces between the CF and application software.

– Restrictions on applications' usage of OS and CORBA middleware APIs.

Operating Environment Operating System

OS Usage Restrictions




CORBA API
applications use CF for all File access
Logical *Device* is an Adapter for HW-specific devices
More on Logical *Device* later
All Application Components
Core Framework: Framework Control & Framework Services Interfaces
CORBA ORB
non-CORBA components or device drivers
OS access limited to SCA AEP
OS access unlimited
OS access unlimited
(non-CORBA components provide access to hardware devices / functionality not available on a CORBA-capable processor)
OS (function) that supports SCA AEP.
(unlimited proprietary APIs for system development).
Any vendor-provided OS function calls

Operating Environment Object Request Broker

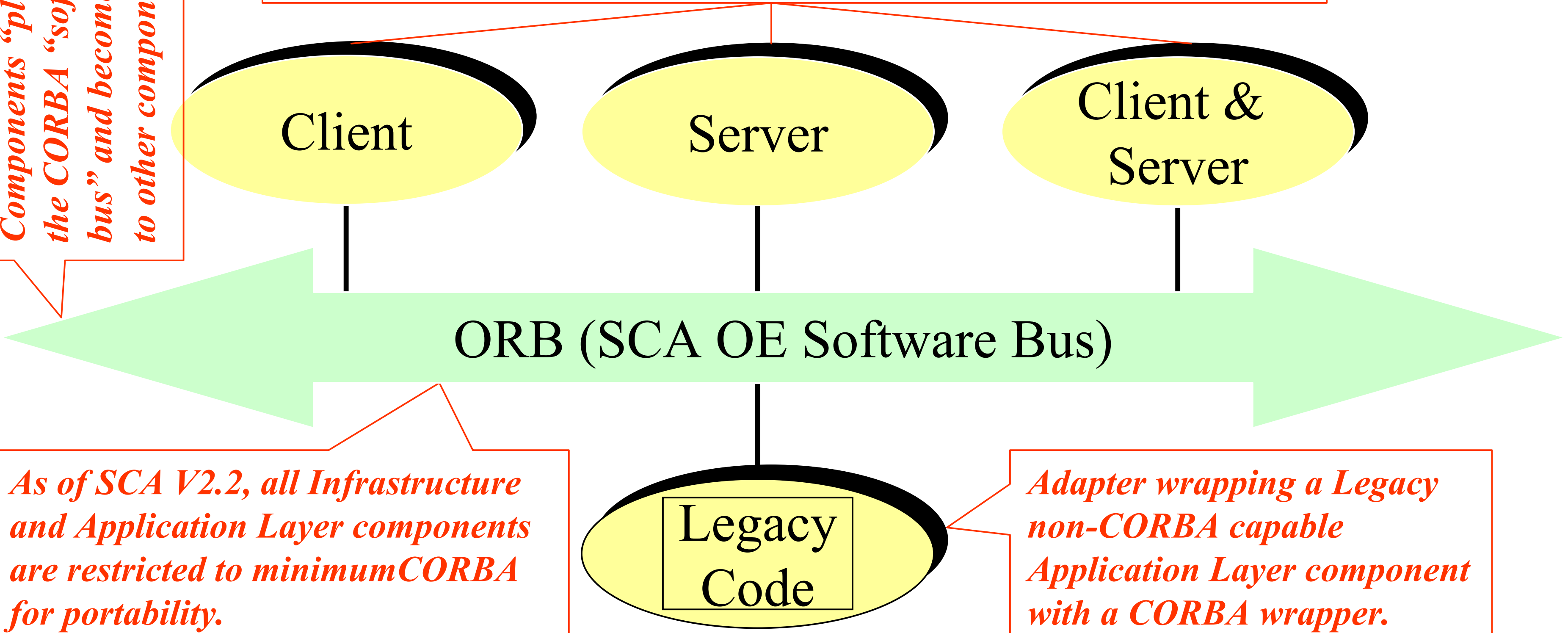
OE
CORBA Middleware
ORB

Commercial Off-the-Shelf (COTS)

Object Request Broker

Components "plug" into the CORBA "software bus" and become visible to other components.

- *Infrastructure Layer OE components (e.g. CF)*
- *Application Layer components*



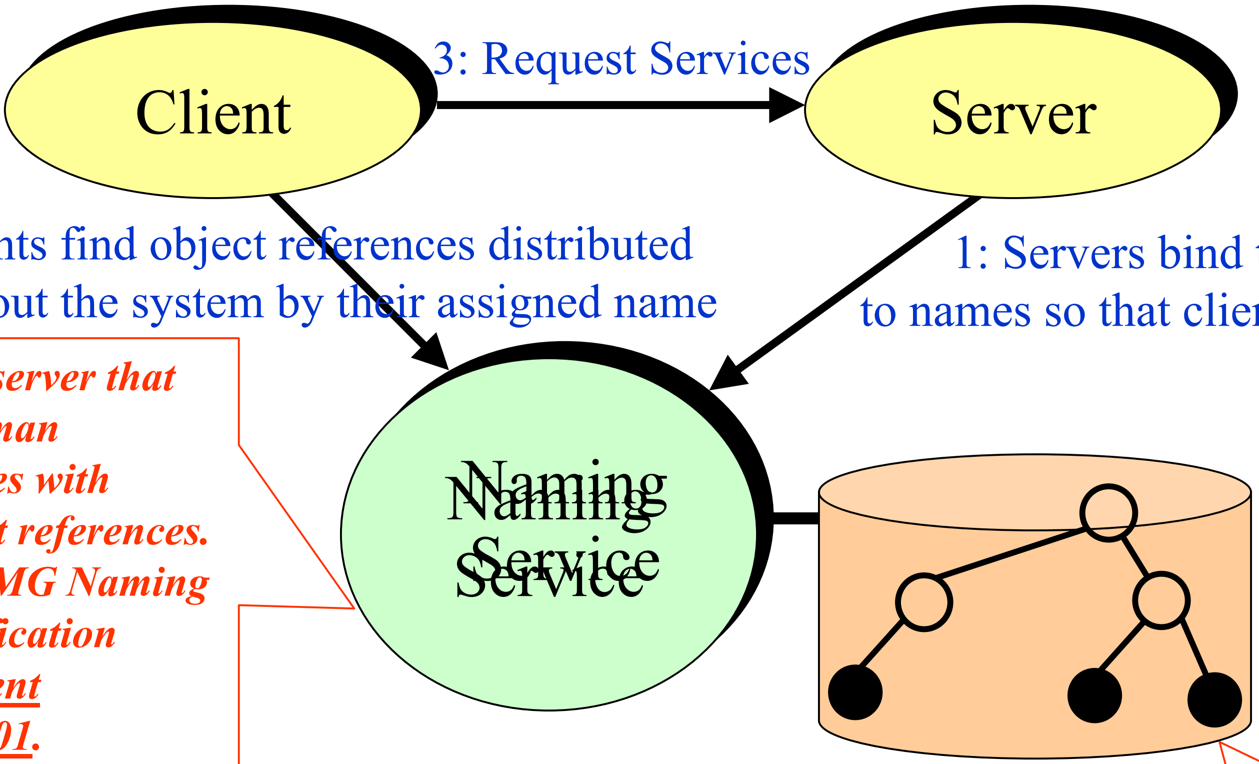
As of SCA V2.2, all Infrastructure and Application Layer components are restricted to minimumCORBA for portability.

Adapter wrapping a Legacy non-CORBA capable Application Layer component with a CORBA wrapper.

BOEING®

Operating Environment CORBA Naming Service

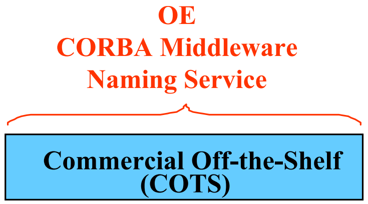
Naming Service


Client
3: Request Services
Server
2: Clients find object references distributed throughout the system by their assigned name
1: Servers bind their objects to names so that clients can find them
Naming Service

- *A CORBA server that associates human readable names with CORBA object references.*
- *Based on OMG Naming Service Specification* *__OMG Document formal/00-11/01__.*
- *Must support bind(), bind_new_context(), unbind(), destroy(), resolve().*

- *Centralized repository of object references in the CORBA environment.*
- *Server Bindings organized in Naming Contexts.*

Naming Service Usage in the OE

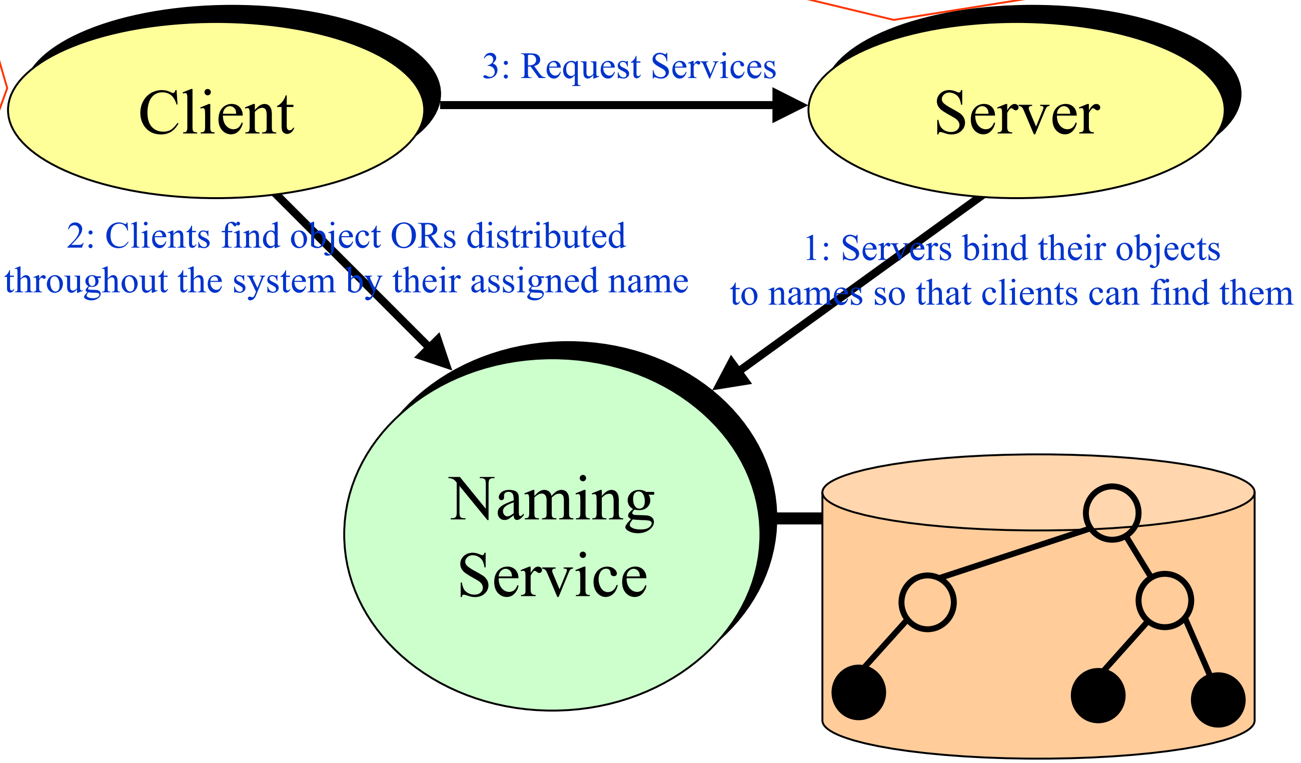


Typical OE Naming Service Servers

- ***CF component that installs applications in the CF domain***
- ***Deployed domain components***
- ***Instantiated application components***

Typical OE Naming Service Clients

- ***CF: connect deployed components to services in CF domain***
- ***Application Layer components: installation of application software through CF***
- ***CF: create, setup, connect, and tear down instantiated application components.***
- ***More on this later.***


Client
Server
3: Request Services
2: Clients find object ORs distributed throughout the system by their assigned name
1: Servers bind their objects to names so that clients can find them
Naming Service

Operating Environment CORBA Event Service

Event Service

Based on OMG's Event Service specification.
- Push interfaces (*PushConsumer* and *PushSupplier*) of the *CosEventComm* CORBA module as described in OMG Document formal/01-03-01: Event Service, v1.1.
- Compatible IDL for the *CosEventComm* CORBA module in the OMG Document formal/01-03-02: Event Service IDL, v1.1.

Decoupled, asynchronous communication model between clients and servers.
- Event producers and consumers

Suppliers "publish" information without knowing who the consumers are.

Consumers "subscribe" to information without regard to the information source.

Suppliers are shielded from exceptions resulting from any of the consumer objects being unreachable or poorly behaved.

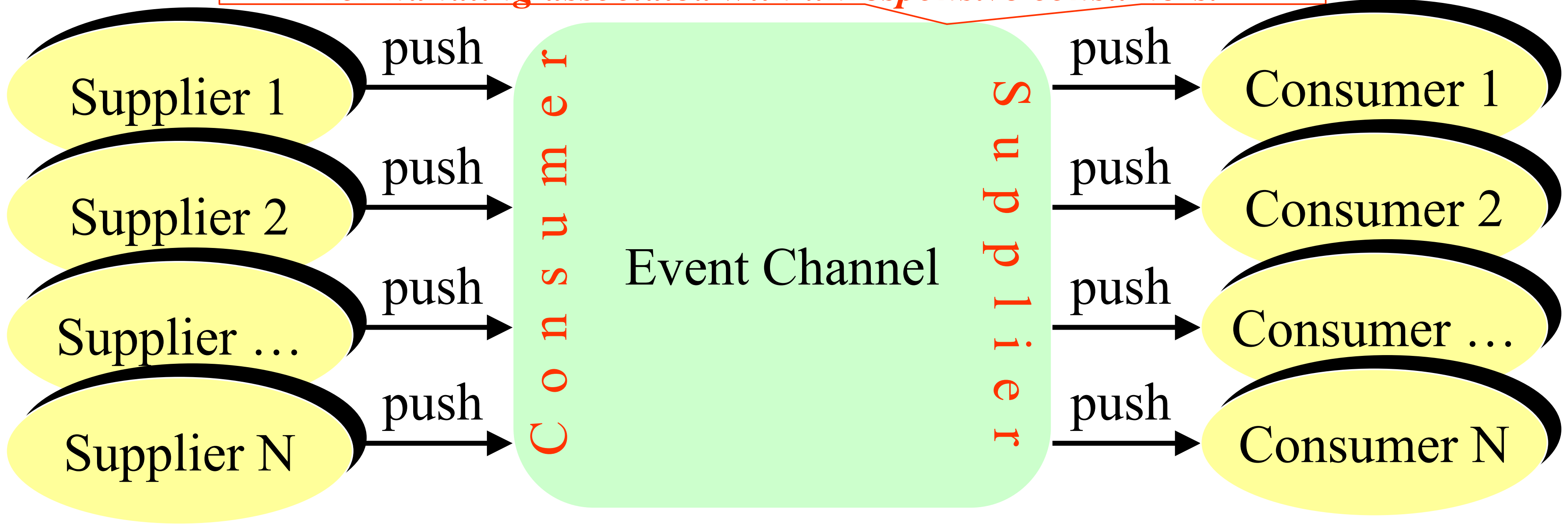
Event Service (cont.)

OE
CORBA Middleware
Event Service

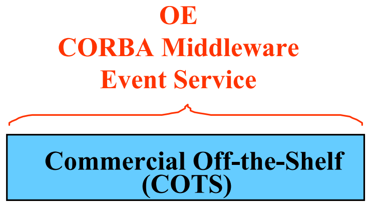
Commercial Off-the-Shelf (COTS)

Central Role in Event Service Specification

- *Supplier(s)/consumer(s) intermediary.*
- *Acts as consumer to supplier(s) and supplier to consumer(s).*
- *Supplier and consumer registrations.*
- *Timely and reliable event delivery to all registered consumers.*
- *Error handling associated with unresponsive consumers.*

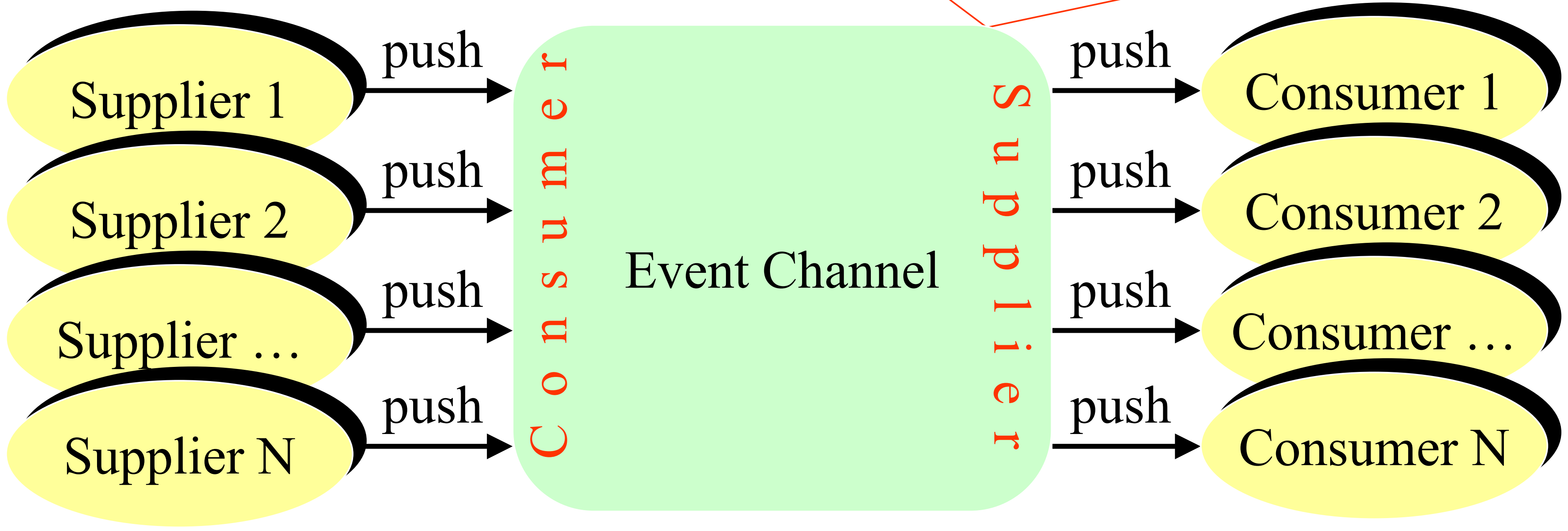


Event Service Usage in the OE



Used by CF for

- ***Connection of CF domain components to required event channels, and***
- ***Connection/disconnection of application components to required event channels in the CF domain, during instantiation/tear down of an application in the CF domain.***
- ***Logging SCA-specified events to SCA-specified event channels.***



Event Service Usage in the OE (cont.)

OE
CORBA Middleware
Event Service

Commercial Off-the-Shelf (COTS)

OE provides two standard event channels:

- **Incoming Domain Management Event Channel**
  - Allows the CF domain to become aware of changes in the domain.
    - e.g. device state changes.
- **Outgoing Domain Management Event Channel**
  - Allows CF domain clients to become aware of changes in the domain.
    - e.g. Human Computer Interface (HCI) application receiving events pertaining to device/service/application software/instantiated application additions/removals from domain.

Application developers allowed to setup other “non-standard” event channels.

Operating Environment Log Service

Log Service

Replaced with OMG Light Weight Log Service in SCA V2.2.1…

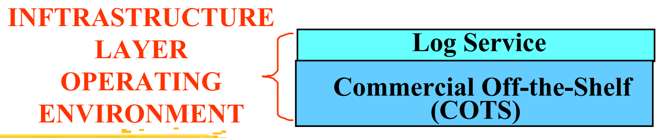
SCA V2.2 provides one interface (*Log*) and types for log producers, consumers, and administrators to generate standard SCA log records, consume them, get status from the log, and control the logging output of a log producer.

…which divides the Log interface into Four separate interfaces for…

Log producers are required to implement configure properties that configure the producer's log record output.

…Log Consumers, Producers, Administrators, and Status Retrievers

Log Interface



<<Interface>>
Log

clearLog() : void
destroy() : void
setAdministrativeState(state : in AdministrativeStateType) : void
setLogFullAction(action : in LogFullActionType) : void
setMaxSize(size : in unsigned long long) : void

SCA V2.2.1 CosLwLog::LogAdministrator

getAdministrativeState() : AdministrativeStateType
getAvailabilityStatus() : AvailabilityStatusType
getCurrentSize() : unsigned long long
getLogFullAction() : LogFullActionType
getMaxSize() : unsigned long long
getNumRecords() : unsigned long long
getOperationalState() : OperationalStateType

SCA V2.2.1 CosLwLog::LogStatus

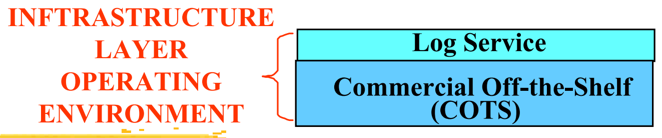
getRecordIdFromTime(fromTime : in LogTimeType) : RecordIdType
retrieveById(currentId : inout RecordIdType, howMany : in unsigned long) : LogRecordSequence

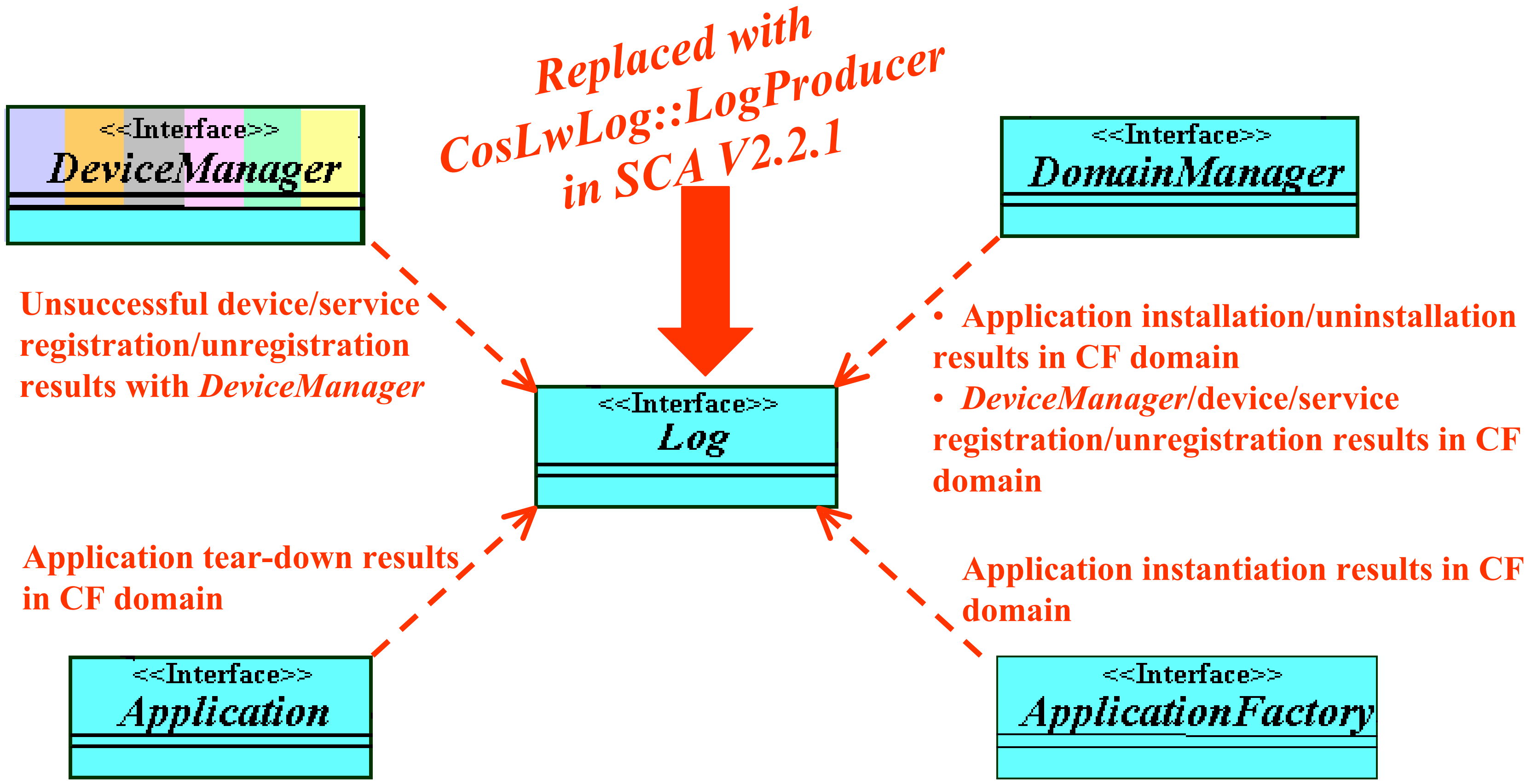
SCA V2.2.1 CosLwLog::LogConsumer

writeRecords(records : in ProducerLogRecordSequence) : void

SCA V2.2.1 CosLwLog::LogProducer

Log Service Usage by CF




Replaced with CosLwLog::LogProducer in SCA V2.2.1
<<Interface>> DeviceManager
<<Interface>> DomainManager
<<Interface>> Log
<<Interface>> Application
<<Interface>> ApplicationFactory

Unsuccessful device/service registration/unregistration results with *DeviceManager*

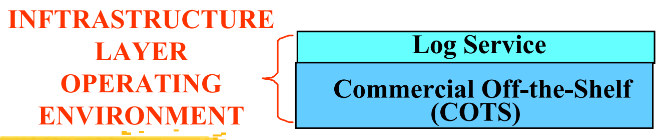
- Application installation/uninstallation results in CF domain
- *DeviceManager*/device/service registration/unregistration results in CF domain

Application tear-down results in CF domain

Application instantiation results in CF domain

BOEING

Log Service Implementation and Deployment Optional in the OE



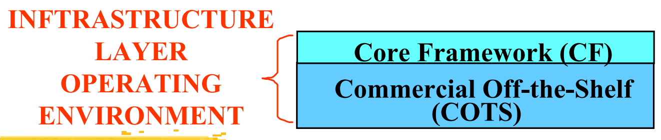
A CF provider may deliver an SCA-compliant product without a Log Service implementation.

An SCA-compliant installation (e.g., a handheld platform with limited resources) may choose not to deploy a Log Service as part of its domain.

CF components that are required to write log records are also required to account for the absence of a log service and otherwise operate normally.

Operating Environment Core Framework

Core Framework



The essential ("core") set of open application-layer interfaces and services that

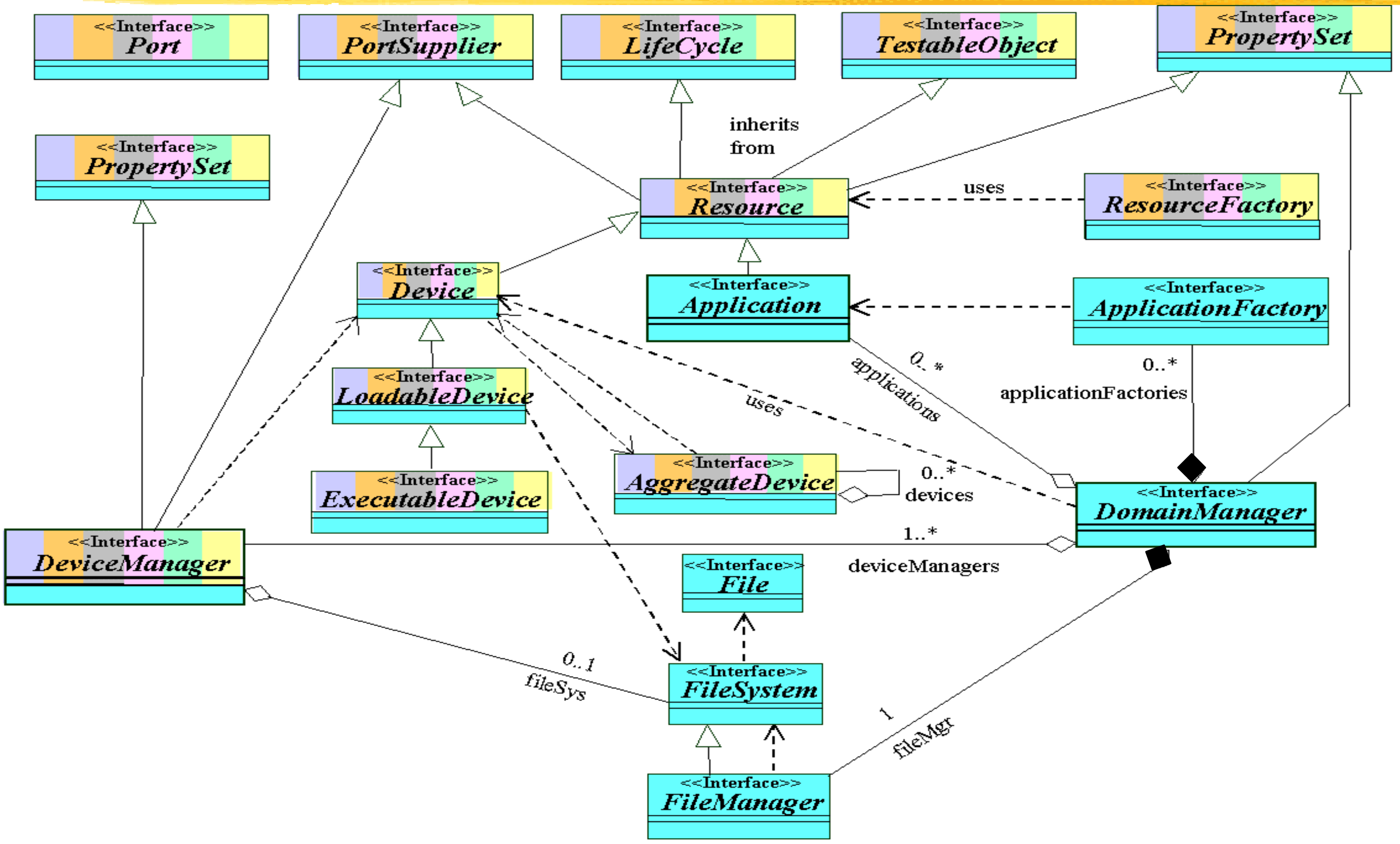
- Provide an abstraction of the underlying software and hardware layers for software application designers.
- Provide for deployment, management, interconnection, and intercommunication of software application components in a distributed embedded communication system.
- Through imposed design constraints provide increased portability of these application components from one SCA-compliant platform to another.
  - Specified interfaces between CF and application software.
  - Specified behavioral requirements for application software.

BOEING®

CF Interfaces Top-Level View

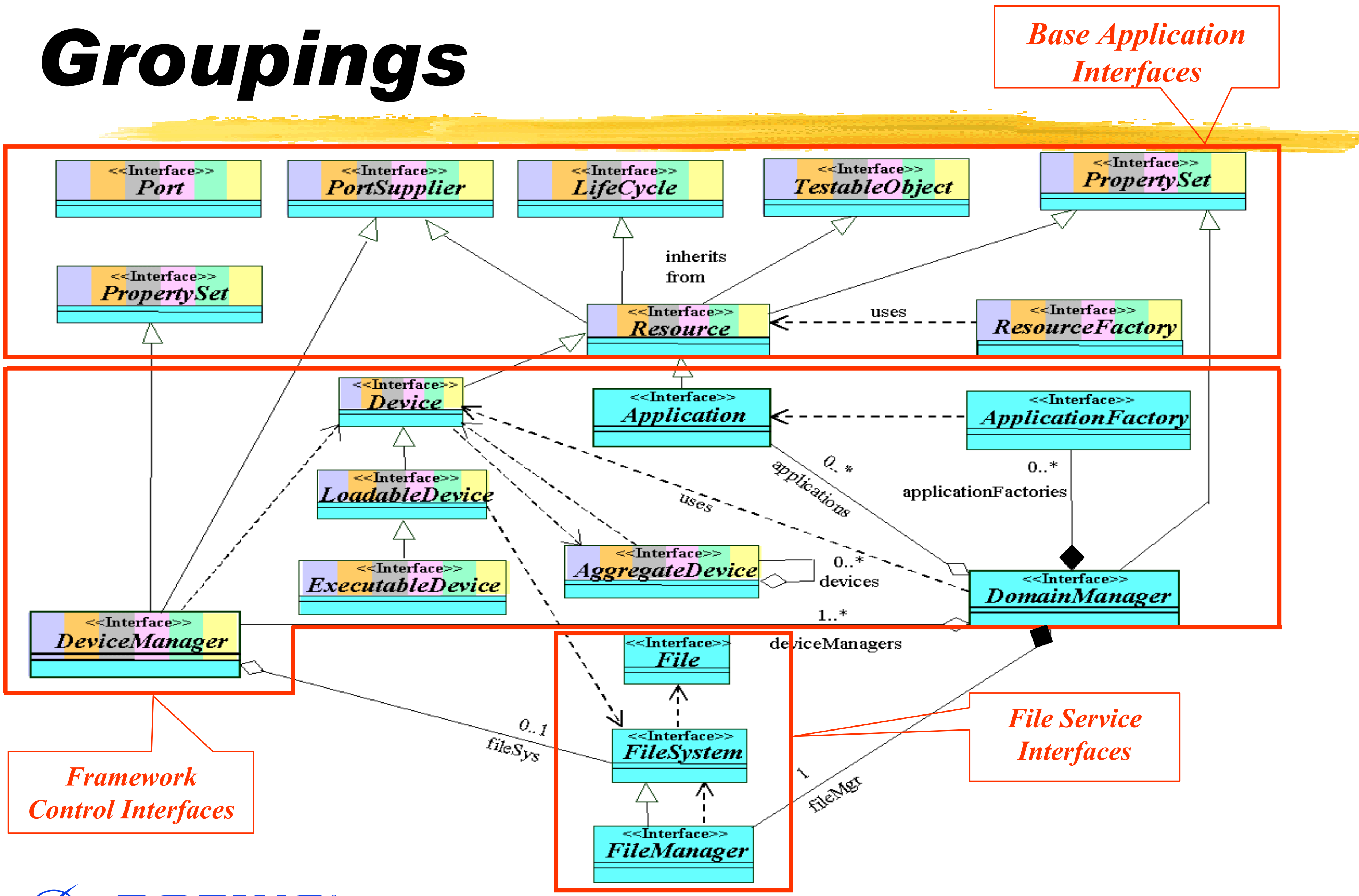
Implemented as Core Application Services
Core Framework Interface

Implemented by Non-Core Applications
Core Framework Interface



<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
<<Interface>> PropertySet
inherits from
<<Interface>> Resource
uses
<<Interface>> ResourceFactory
<<Interface>> Device
<<Interface>> Application
<<Interface>> ApplicationFactory
<<Interface>> LoadableDevice
uses
0..* applications
0..* applicationFactories
<<Interface>> ExecutableDevice
<<Interface>> AggregateDevice
0..* devices
<<Interface>> DomainManager
<<Interface>> DeviceManager
1..* deviceManagers
<<Interface>> File
0..1 fileSys
<<Interface>> FileSystem
1 fileMgr
<<Interface>> FileManager

CF Interface Groupings



Base Application Interfaces
<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
<<Interface>> PropertySet
inherits from
<<Interface>> Resource
uses
<<Interface>> ResourceFactory
<<Interface>> Device
<<Interface>> Application
<<Interface>> ApplicationFactory
<<Interface>> LoadableDevice
uses
0..*
applications
0..*
applicationFactories
<<Interface>> ExecutableDevice
<<Interface>> AggregateDevice
0..*
devices
<<Interface>> DomainManager
<<Interface>> DeviceManager
1..*
deviceManagers
<<Interface>> File
<<Interface>> FileSystem
0..1
fileSys
<<Interface>> FileManager
1
fileMgr
File Service Interfaces
Framework Control Interfaces

BOEING®

CF Interfaces and Services

Base Application Interfaces (***Port, LifeCycle, TestableObject, PropertySet, PortSupplier, ResourceFactory,*** and ***Resource***) that provide a common set of interfaces for developing software application components and exchanging information between them.

Framework Control Interfaces that provide the means for control and management of hardware assets, applications, and domain (system).

- **Device Interfaces** (***Device, LoadableDevice, ExecutableDevice, AggregateDevice***)
- **Device Management Interfaces** (***DeviceManager***)
- **Domain Management Interfaces** (***Application, ApplicationFactory, DomainManager***)

File Service Interfaces (***File, FileSystem, FileManager***) that provide the means for distributed file access services.

CF Component Packaging and Deployment

The **Domain Profile**, is a series of eXtensible Markup Language (XML) files, based on…

the CORBA Component Specification, and

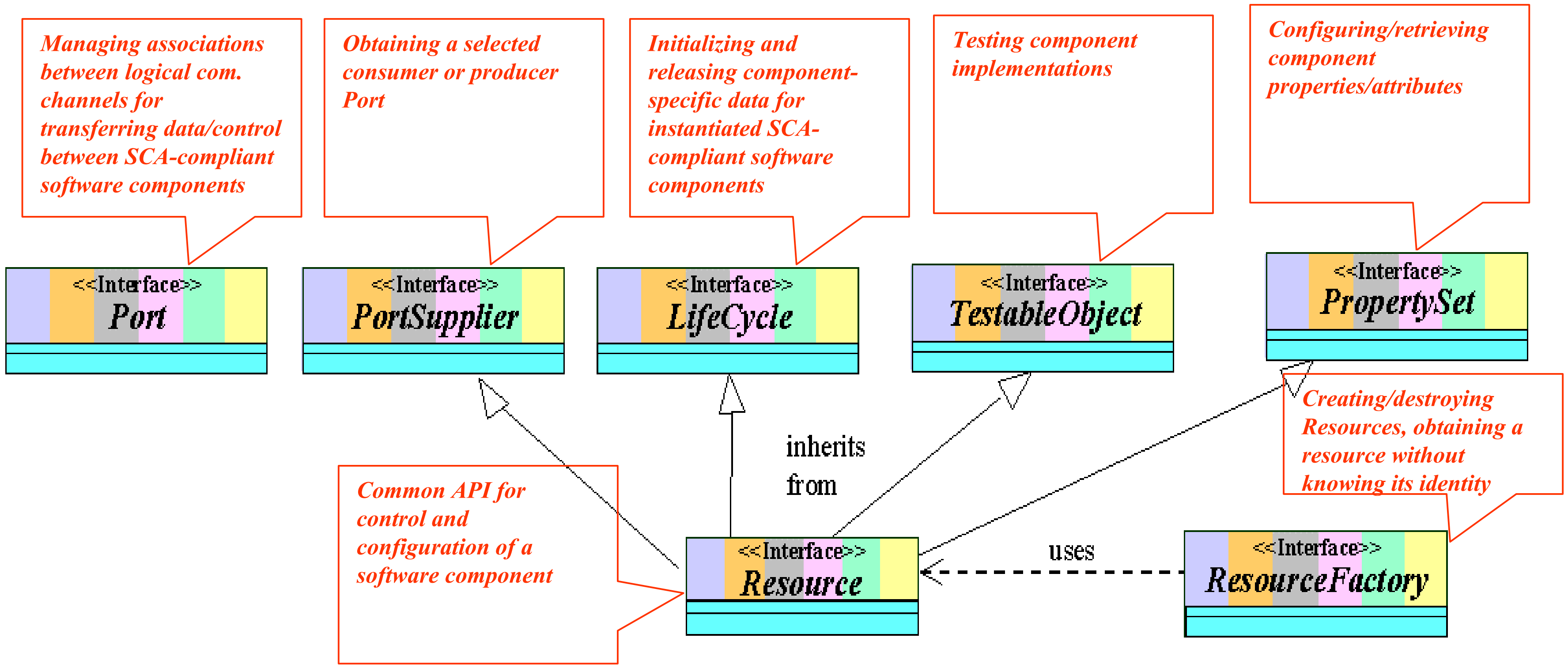
SCA-defined XML Document Type Definitions (DTDs)…

…expressing the packaging and deployment of software application components and related hardware assets into an SCA-compliant system.

CF Base Application Interfaces

CF Base Application Interfaces

Defined by CF requirements, implemented by application developers.

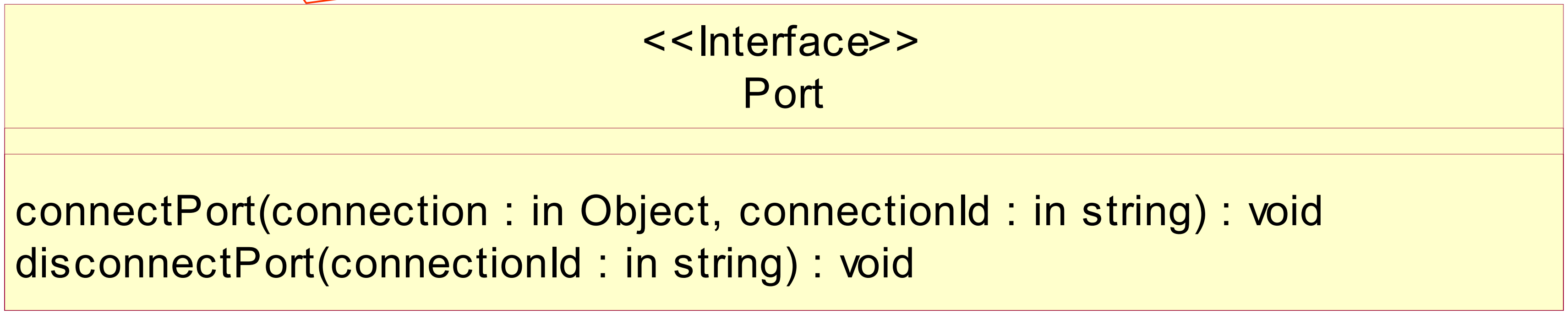

Managing associations between logical com. channels for transferring data/control between SCA-compliant software components
Obtaining a selected consumer or producer Port
Initializing and releasing component-specific data for instantiated SCA-compliant software components
Testing component implementations
Configuring/retrieving component properties/attributes
<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
inherits from
Creating/destroying Resources, obtaining a resource without knowing its identity
Common API for control and configuration of a software component
<<Interface>> Resource
uses
<<Interface>> ResourceFactory

Unimplemented framework of operations for creation, initialization, configuration, control, test, and tear down of application components in the system.

BOEING®

Port Interface

Provides a specialized connectivity to a component. Used to setup and tear down connections between CORBA components (mainly application components) in the CF domain. How?



```
connectPort(connection : in Object, connectionId : in string) : void
disconnectPort(connectionId : in string) : void
```

A Component can have a Set of Ports

A component may have 0 or more ports as needed.

The implementation for each port is specific to the component.

A Component Defines a Component-Specific Port by...

- Creating a specific port type by defining an interface that inherits from *Port*.
  - Port Types
    - Command and Control, Data, Status.
    - Basic push and pull *Ports* defined in the SCA.
- Establishing the operations for transferring data and control.
- Establishing the meaning of data and control values.
- Specifying whether the port is a “uses” or a “provides” port.

“Uses” and “Provides” Ports

Uses Port

- A port that uses some set of services provided by a provider component.
- All uses ports implement the *Port* interface.

Provides Port

- A port that provides some set of services at an interface.

How do Component Ports get Connected?

Connection Definitions

- Definition of how the component ports get connected is described in the following Domain Profile files:
  - (An application's) Software Assembly Descriptor
  - (A CF domain node's) Device Configuration Descriptor
  - Covered later when we go over the Domain Profile.

By Whom?

- Ports are connected by CF's Framework Control Interfaces in charge of deploying and managing components in the CF domain (i.e. *DomainManager*), and instantiating applications in the CF domain (i.e. *ApplicationFactory*).

How?

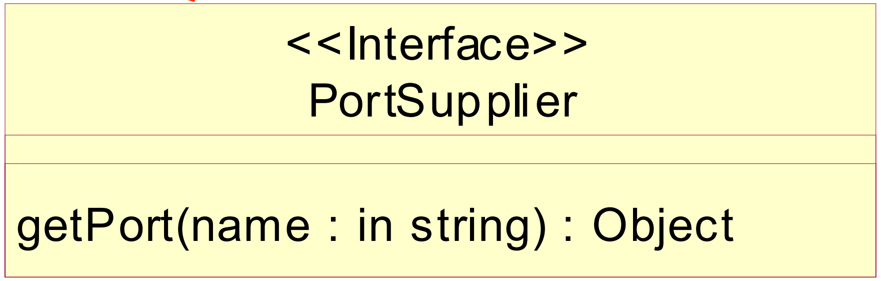
- Covered later when we cover CF's Framework Control Interfaces (*DomainManager* and *ApplicationFactory*).

Port Interface Advantages

- Supports the connection concepts in the CORBA Components Specification where any provides port type can be connected to another uses port, as long the uses port understands the port, which it is connecting.
- The uses port can be behave either as a push producer or a pull consumer. Likewise, the provides port type can be behave either as a push consumer or pull producer.
- The fan-in, fan-out, or on-one-on implementation of a port is allowed, and is component dependent.

PortSupplier Interface

Provides the means for a component that has a Port to make the Port available to those who need it. Mainly, those in charge of connecting the Port to other(s).

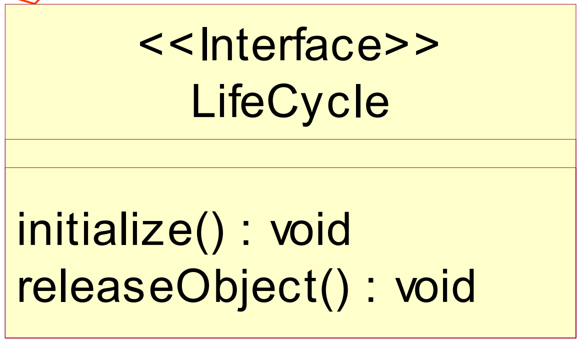


Use of PortSupplier Interface in CF

- The main *PortSupplier* interface users in the CF are the Framework Control interfaces *DomainManager* (during deploying components to the CF domain) and *ApplicationFactory* (during instantiation of an application in the CF domain).
- They use a component's *getPort* operation, as directed by the Domain Profile, to retrieve uses and provides ports in order to connect the component to services or other components in the domain.

LifeCycle Interface

Provides a generic means for initializing and releasing a resource, application, or device in the CF domain.



LifeCycle Users

The component initialization *LifeCycle* interface users in the CF are the Framework Control interfaces *DeviceManager* (during deploying components to the CF domain) and *ApplicationFactory* (during instantiation an application in the CF domain), after a component has been deployed and its object reference has been obtained.

The component tear-down *LifeCycle* interface users in the CF are the Human Control Interface (HCI) or other domain client to release an application or device, and the CF Application interface to release an application's components as a part of application termination.

TestableObject Interface

Provides a generic means for black box testing a resource, application, or device in the CF domain.

<<Interface>>
TestableObject

runTest(testid : in unsigned long, testValues : inout Properties) : void

<u>Location of Component Test IDs, Input, and Output Values</u>

Component's Properties Descriptor referenced from the component's Software Package Descriptor in the CF Domain Profile.

TestableObject Interface Uses and Advantages

May be used by an HCI to test an *Application* or *Device* component within the domain.

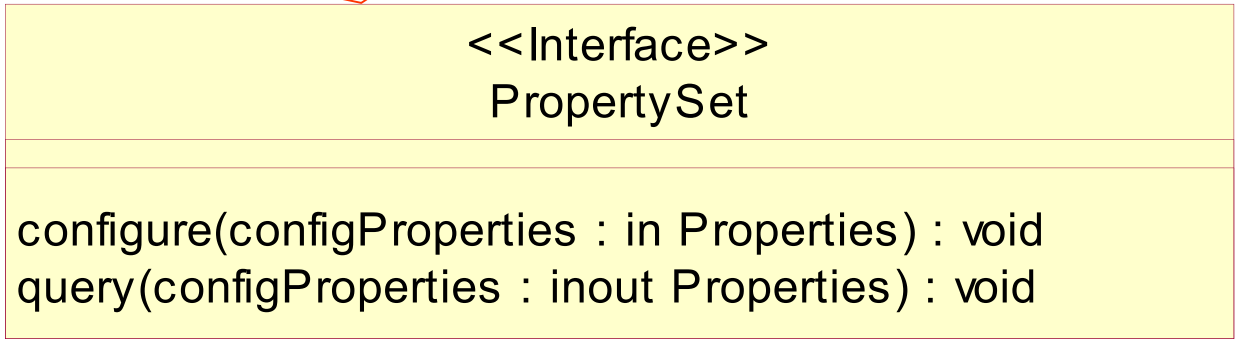
Using this interface, it would be possible to implement a generic HCI that could work for all Applications and *Devices* since this interface is based upon a CF Domain Profile XML element.

Currently there is no requirement to perform initial testing of a deployed domain component *DeviceManager* or an application's components by the *ApplicationFactory*.

BOEING®

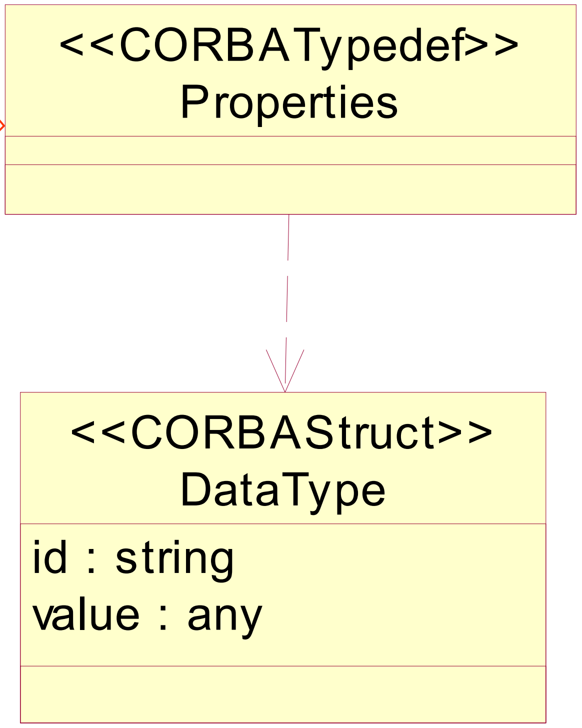
PropertySet Interface

Provides a generic means of configuring and retrieving a component's properties.


<<Interface>>
PropertySet
configure(configProperties : in Properties) : void
query(configProperties : inout Properties) : void

Properties Sequence

Properties or id (name)/value pairs, is a concept from the CORBA Components, Telecommunications Information Networking Architecture (TINA), and CORBA property service specifications.



Use of PropertySet Interface in CF

If the component has any configure properties defined in the CF Domain Profile, this interface is used to configure the component by *DeviceManager* (during deploying components to the CF domain) and *ApplicationFactory* (during instantiation of an application in the CF domain), after the component has been deployed and its object reference has been obtained.

Can also be used by a HCI to configure an *Application, Device, DeviceManager, and DomainManager* within the domain.

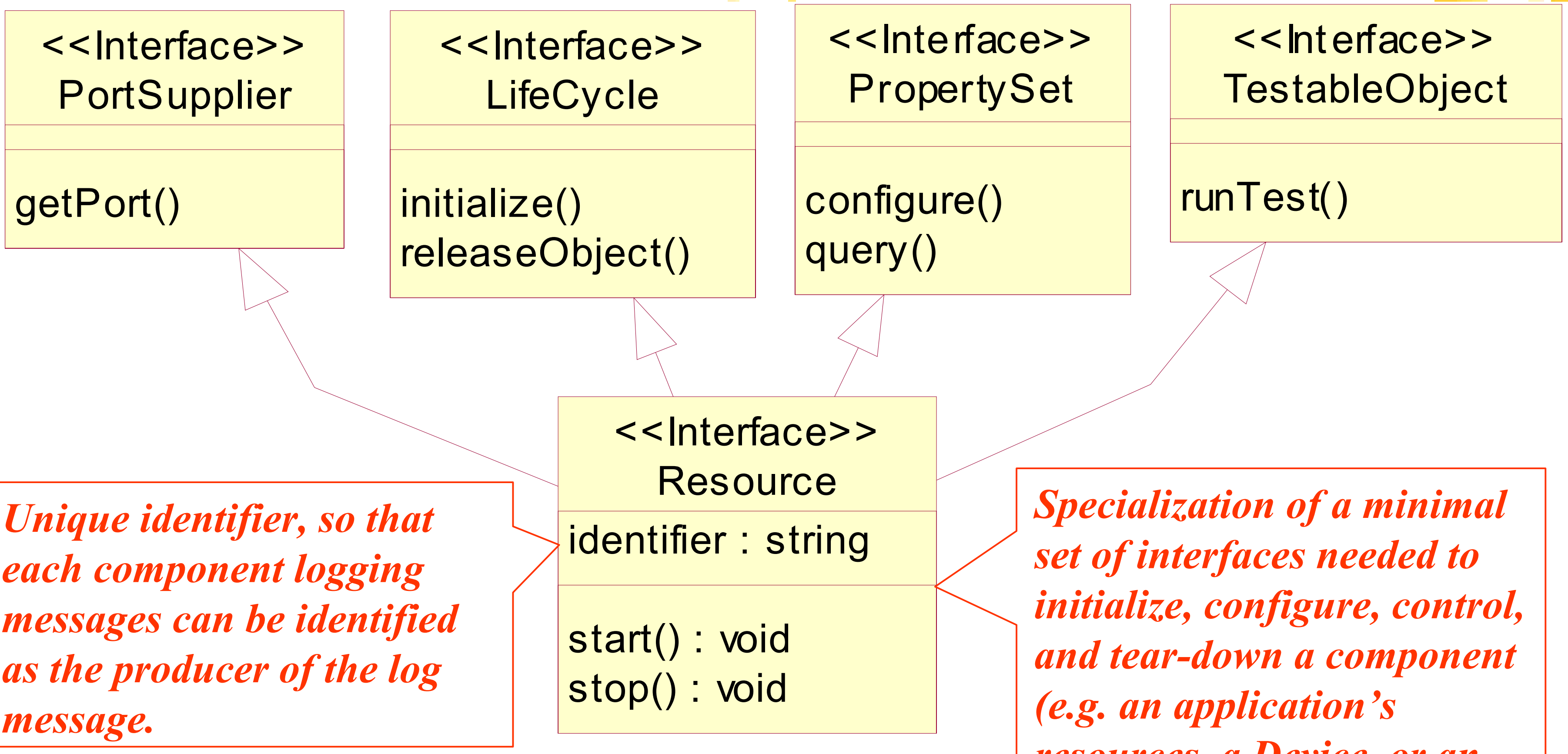
PropertySet Interface Advantages

Promotes the possibility of having a generic HCI based on the corresponding CF Domain Profile XML element to configure and query components.

Provides the capability of adding configure and query properties to a component’s implementation without having to change an IDL interface definition.

- In this case, the component’s appropriate Domain Profile XML file would have to be updated.

Resource Interface



Unique identifier, so that each component logging messages can be identified as the producer of the log message.

Specialization of a minimal set of interfaces needed to initialize, configure, control, and tear-down a component (e.g. an application's resources, a Device, or an Application).

Resource Interface Advantages

Resource supports a set of behavior that enables *Application* delegation and teardown and consistent component deployment between *ApplicationFactory* (during application instantiations) and *DeviceManager* (during domain component deployment).

Resource is used by an HCI (or other domain management clients) to configure an *Application* or *Device* within the domain.

Makes it possible to have a generic HCI that could work for all *Applications* and *Device*s.

ResourceFactory Interface

Modeled after the Factory design pattern
Provides a generic framework of operations for creating and destroying Resources and obtaining a resource without knowing its identity

```
<<Interface>>
ResourceFactory
---------------------------------------------------------------
identifier : string
---------------------------------------------------------------
createResource(resourceId : in string, qualifiers : in Properties) : Resource
releaseResource(resourceId : in string) : void
shutdown() : void
```

Resource Factory Can Create Resources of Different Types

- *The resource's Software Package Descriptor in the Domain Profile specifies properties dictating the resource type.*
- *These properties are passed in as "qualifiers" to createResource(), dictating the type of resource to be created.*

ResourceFactory Resource Creation and Destruction

Creation
- If the *Resource* has not yet been created, it is created in the same process space as the *ResouceFactory*.
- If the *Resource* has already been created, its reference count is incremented and a copy of it is returned.

Destruction
- If the specified *Resource's* reference count is greater than one, it is decremented.
- Otherwise, the specified *Resource* is torn down.

ResourceFactory Interface Advantages

The reference counting feature of the ResourceFactory is beneficial for managing *Resources* that are used by multiple applications.

Eliminates the constant redeployment of the shared components across application boundaries.

BOEING®

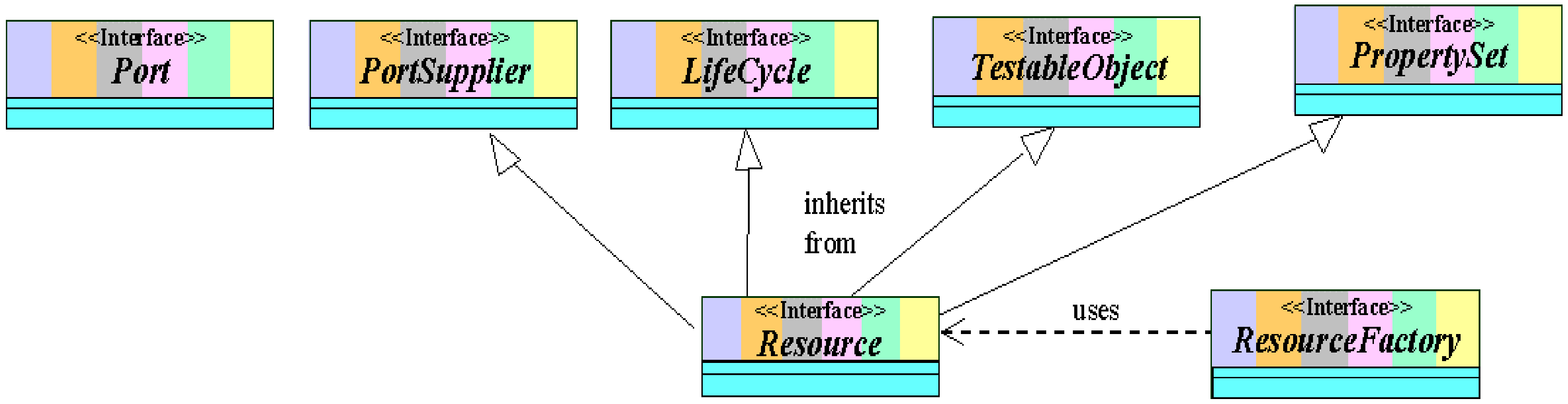
Implementation of a ResourceFactory is Optional

An application is not required to use *ResourceFactories* to obtain, create, or tear down resources.

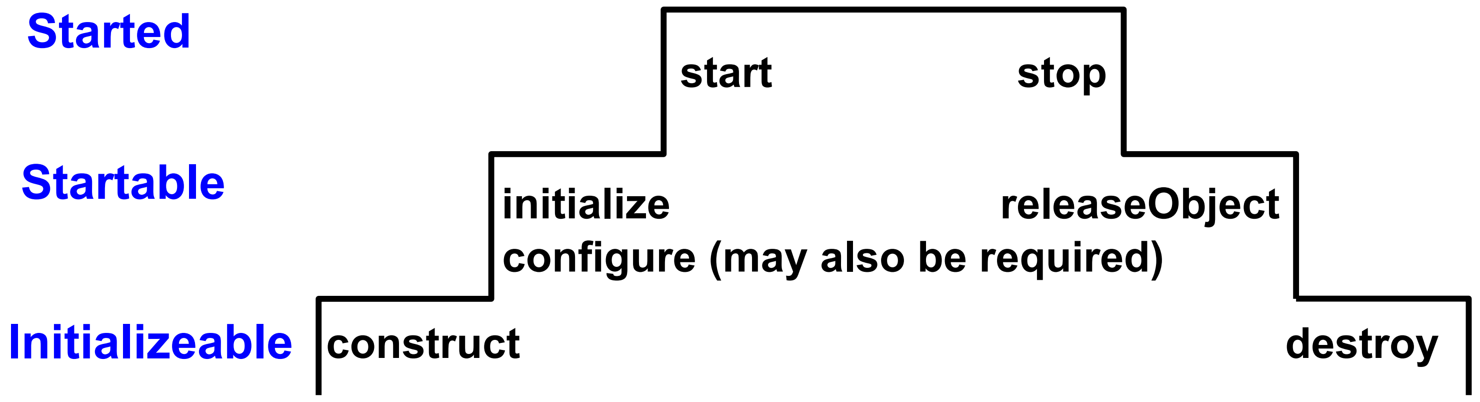
The application's Software Profile specifies which application components require *ResourceFactories*.

- That is,
  - for which application's components does the CF *ApplicationFactory* create and use a *ResourceFactory* (instead of directly executing the application's component's entry point) during instantiation of the application, and correspondingly,
  - for which application components does the CF *Application* interface use a *ResourceFactory* to destroy the application component, (instead of directly terminating the component's running task or process).

Base Application Interfaces Model A Component's Maturity Cycle

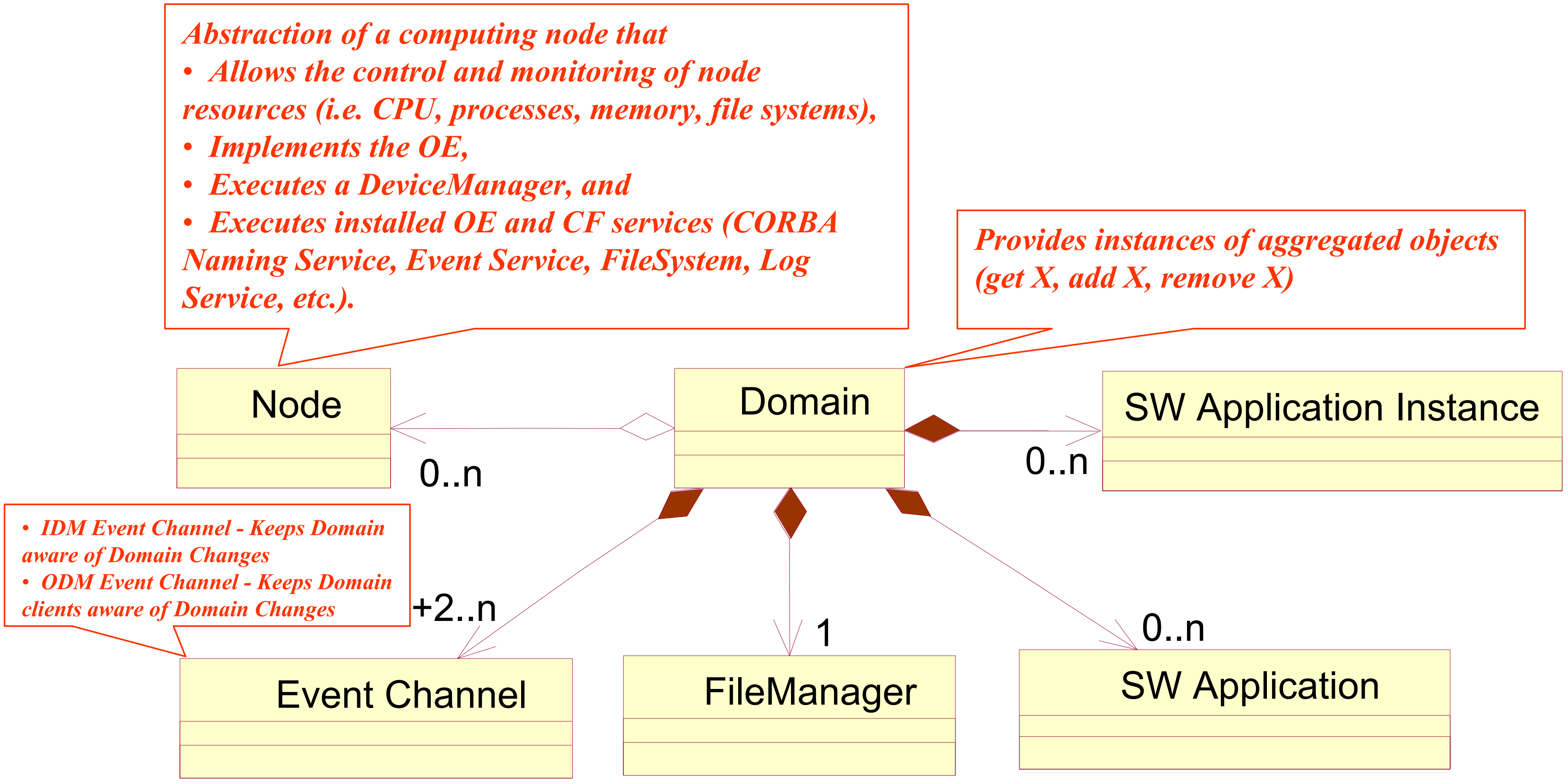


<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
inherits from
<<Interface>> Resource
uses
<<Interface>> ResourceFactory



CF Framework Control Interfaces

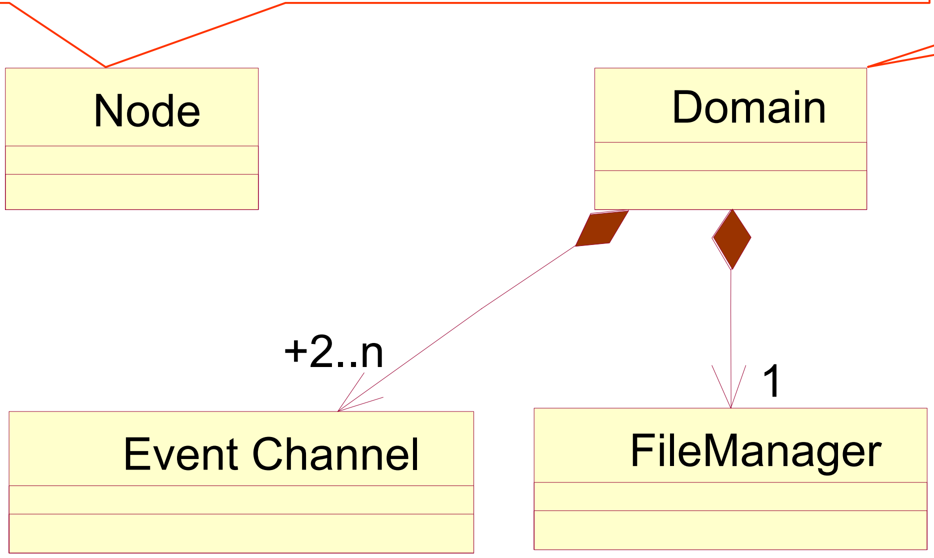
Definition of "Domain" in the CF Problem Space


Abstraction of a computing node that
• Allows the control and monitoring of node resources (i.e. CPU, processes, memory, file systems),
• Implements the OE,
• Executes a DeviceManager, and
• Executes installed OE and CF services (CORBA Naming Service, Event Service, FileSystem, Log Service, etc.).
Provides instances of aggregated objects (get X, add X, remove X)
Node
Domain
SW Application Instance
0..n
0..n
• IDM Event Channel - Keeps Domain aware of Domain Changes
• ODM Event Channel - Keeps Domain clients aware of Domain Changes
+2..n
1
0..n
Event Channel
FileManager
SW Application

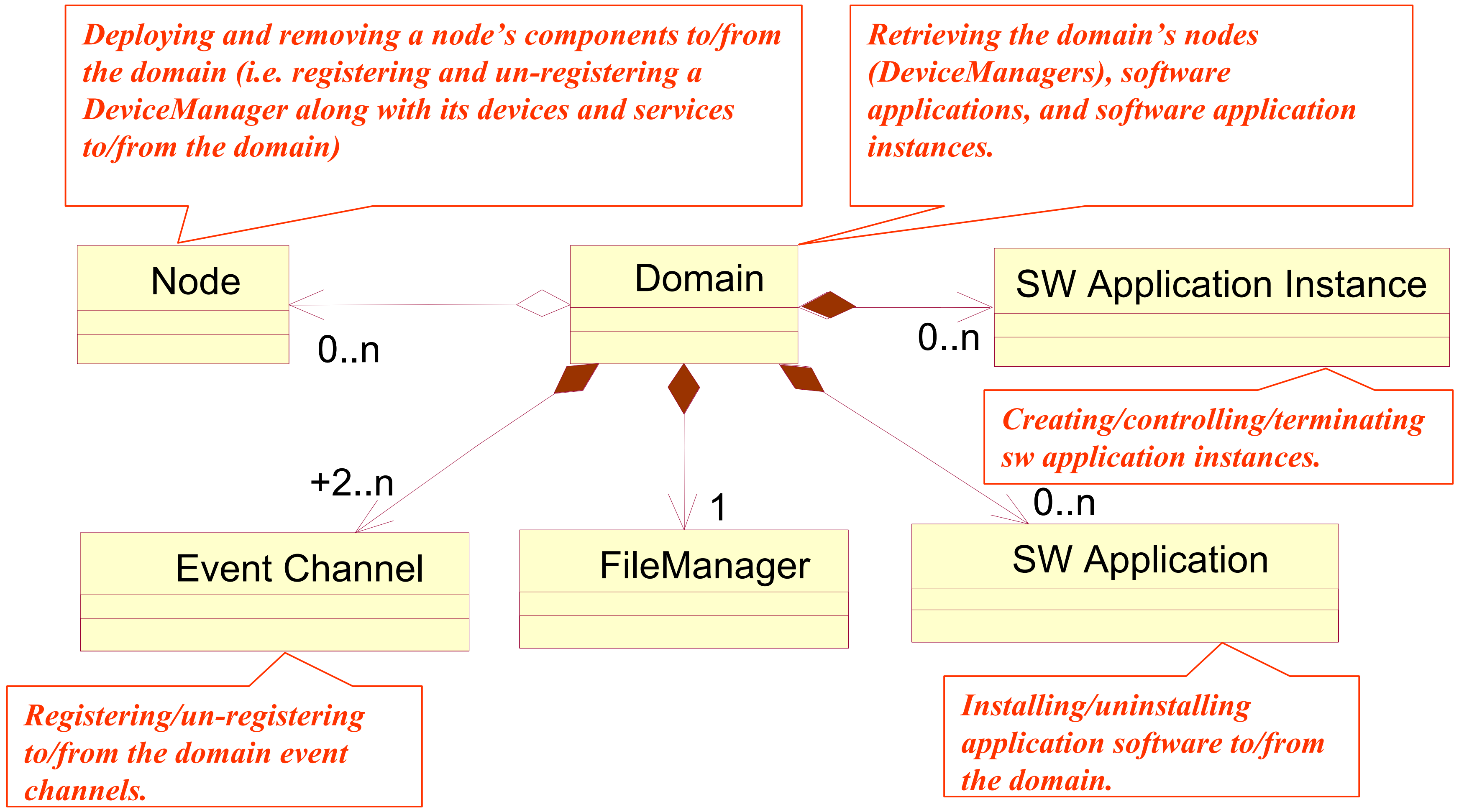
How does the CF Domain Get "Created"?

Each node and its resources are created (i.e. OS booted, file systems created, installed OE and CF services started (CORBA Naming Service, Event Service, FileSystem, Log Service), DeviceManager representing the Node created, DeviceManager components (devices, services, additional file systems, etc.) created and deployed on the Node.

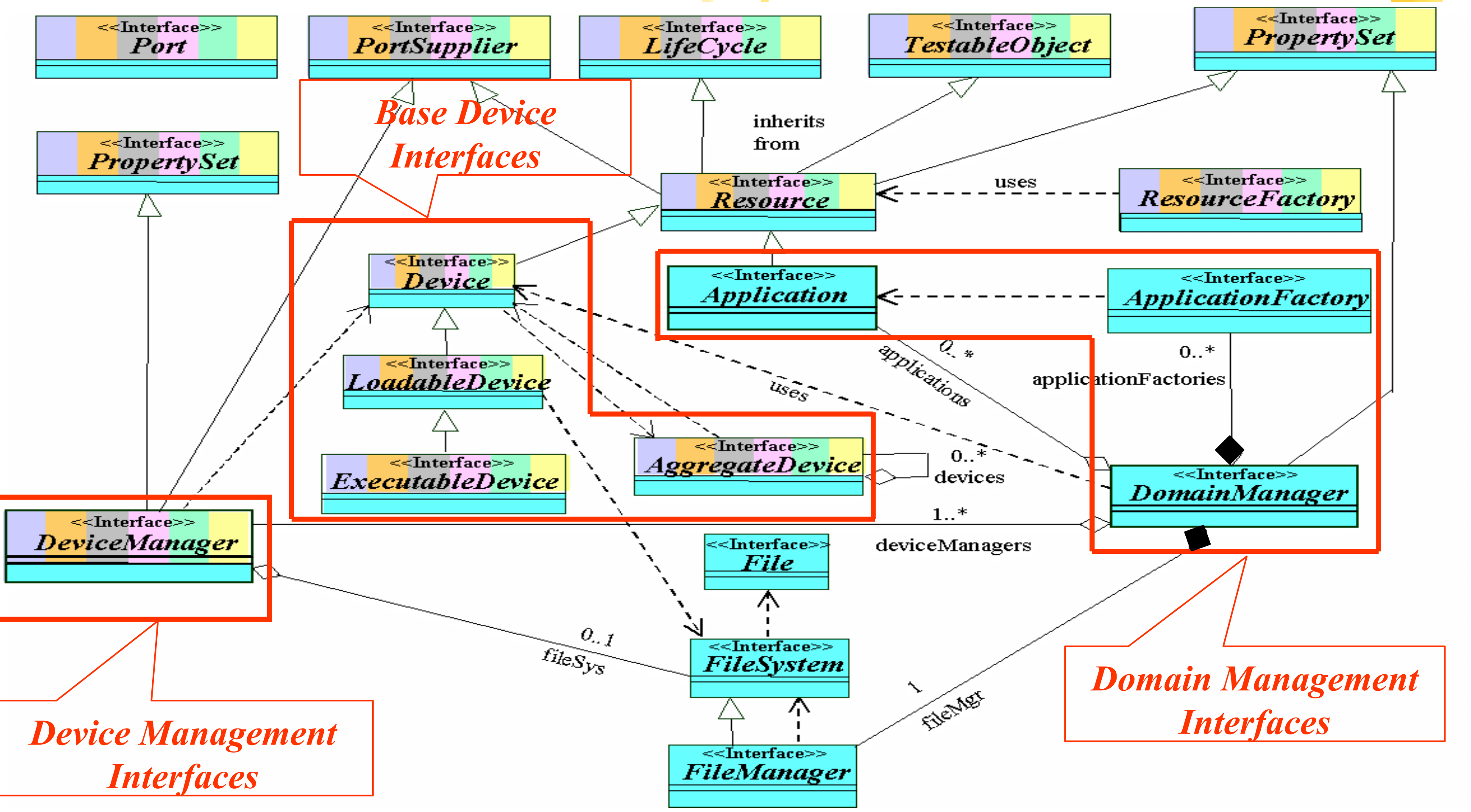
The entity in charge of managing the domain is created.



Services Provided by the CF Domain

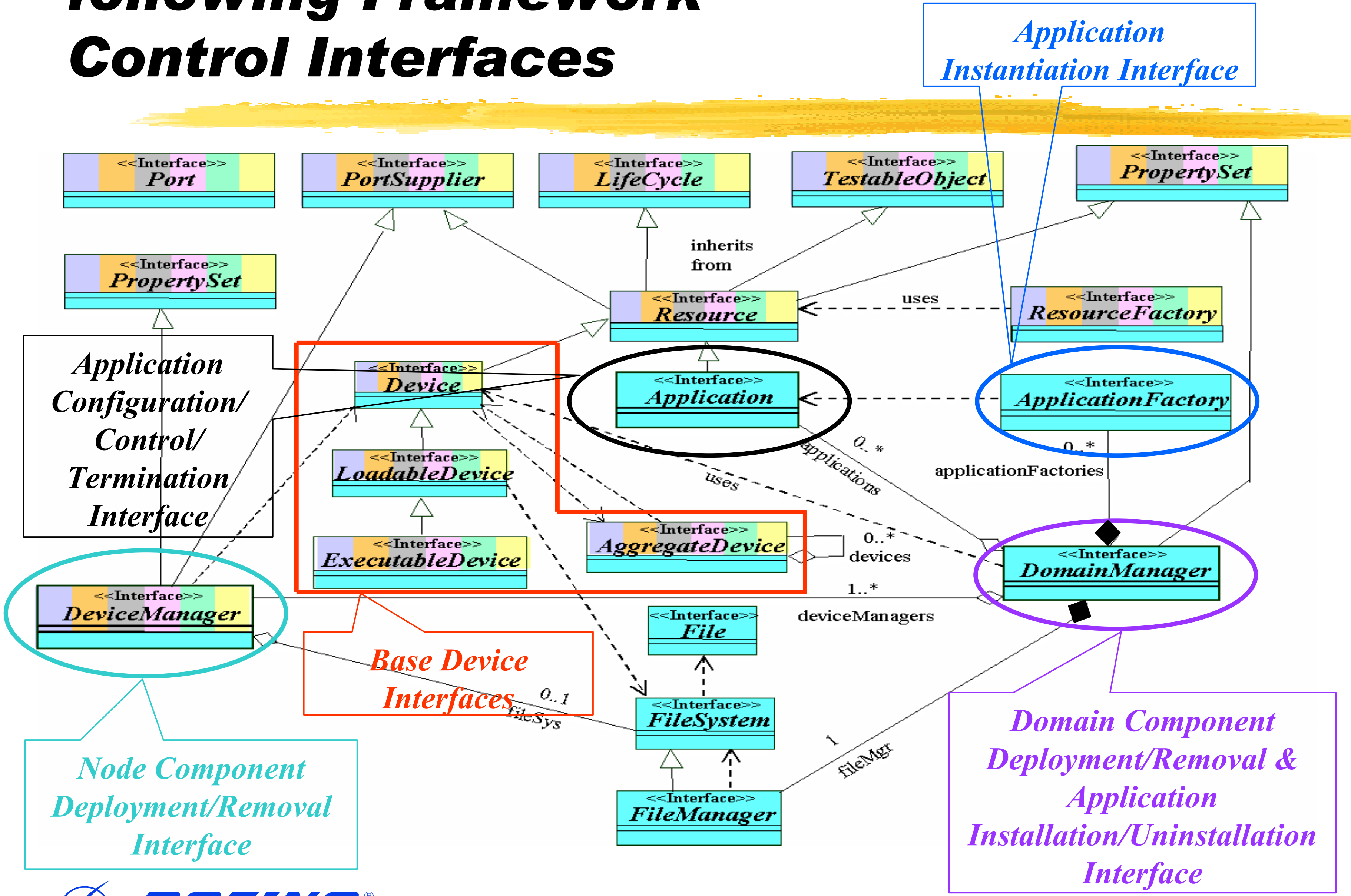

Deploying and removing a node's components to/from the domain (i.e. registering and un-registering a DeviceManager along with its devices and services to/from the domain)
Retrieving the domain's nodes (DeviceManagers), software applications, and software application instances.
Node
Domain
SW Application Instance
0..n
0..n
Creating/controlling/terminating sw application instances.
+2..n
1
0..n
Event Channel
FileManager
SW Application
Registering/un-registering to/from the domain event channels.
Installing/uninstalling application software to/from the domain.

CF Domain Services are Provided by the following Framework Control Interface Groupings



<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
<<Interface>> PropertySet
Base Device Interfaces
inherits from
<<Interface>> Resource
uses
<<Interface>> ResourceFactory
<<Interface>> Device
<<Interface>> Application
<<Interface>> ApplicationFactory
<<Interface>> LoadableDevice
0..* applications
0..*
applicationFactories
uses
<<Interface>> ExecutableDevice
<<Interface>> AggregateDevice
0..* devices
<<Interface>> DomainManager
<<Interface>> DeviceManager
1..*
deviceManagers
<<Interface>> File
0..1 fileSys
<<Interface>> FileSystem
1 fileMgr
<<Interface>> FileManager
Domain Management Interfaces
Device Management Interfaces

CF Domain Services are Mapped to the following Framework Control Interfaces


Application Instantiation Interface
<<Interface>> Port
<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> TestableObject
<<Interface>> PropertySet
<<Interface>> PropertySet
inherits from
<<Interface>> Resource
uses
<<Interface>> ResourceFactory
Application Configuration/ Control/ Termination Interface
<<Interface>> Device
<<Interface>> Application
<<Interface>> ApplicationFactory
<<Interface>> LoadableDevice
0..* applications
uses
0..*
applicationFactories
<<Interface>> ExecutableDevice
<<Interface>> AggregateDevice
0..* devices
<<Interface>> DomainManager
<<Interface>> DeviceManager
1..*
deviceManagers
<<Interface>> File
Base Device Interfaces
0..1 fileSys
<<Interface>> FileSystem
1 fileMgr
<<Interface>> FileManager
Node Component Deployment/Removal Interface
Domain Component Deployment/Removal & Application Installation/Uninstallation Interface

BOEING

CF Framework Control
Base Device Interfaces

CF Base Device Interfaces

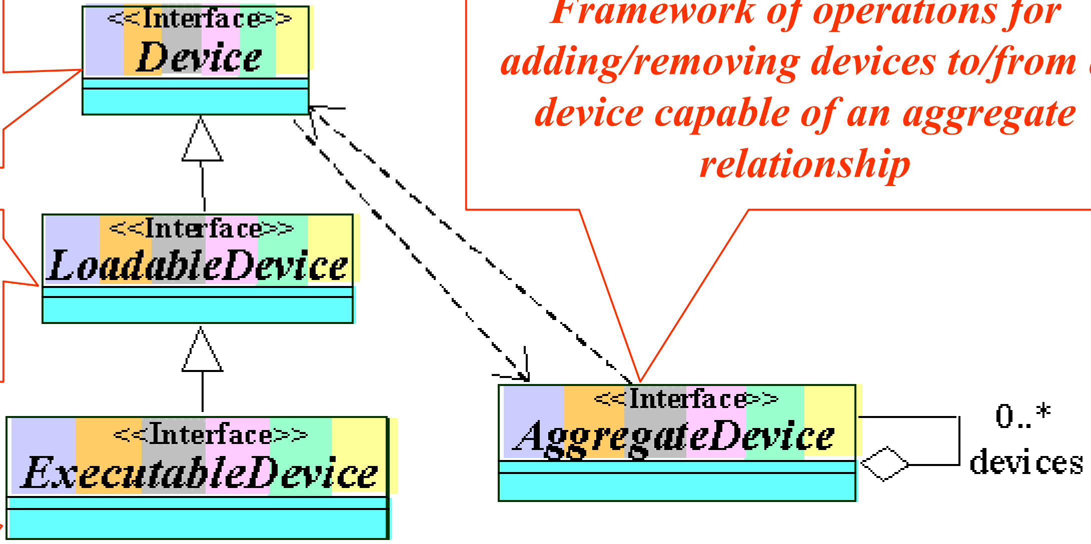
Defined by CF requirements, implemented by application developers.

Framework for a device's software profile attribute, state management and status attributes, and capacity allocation/deallocation operations

Framework of operations for loading/unloading software modules onto a device processor's memory

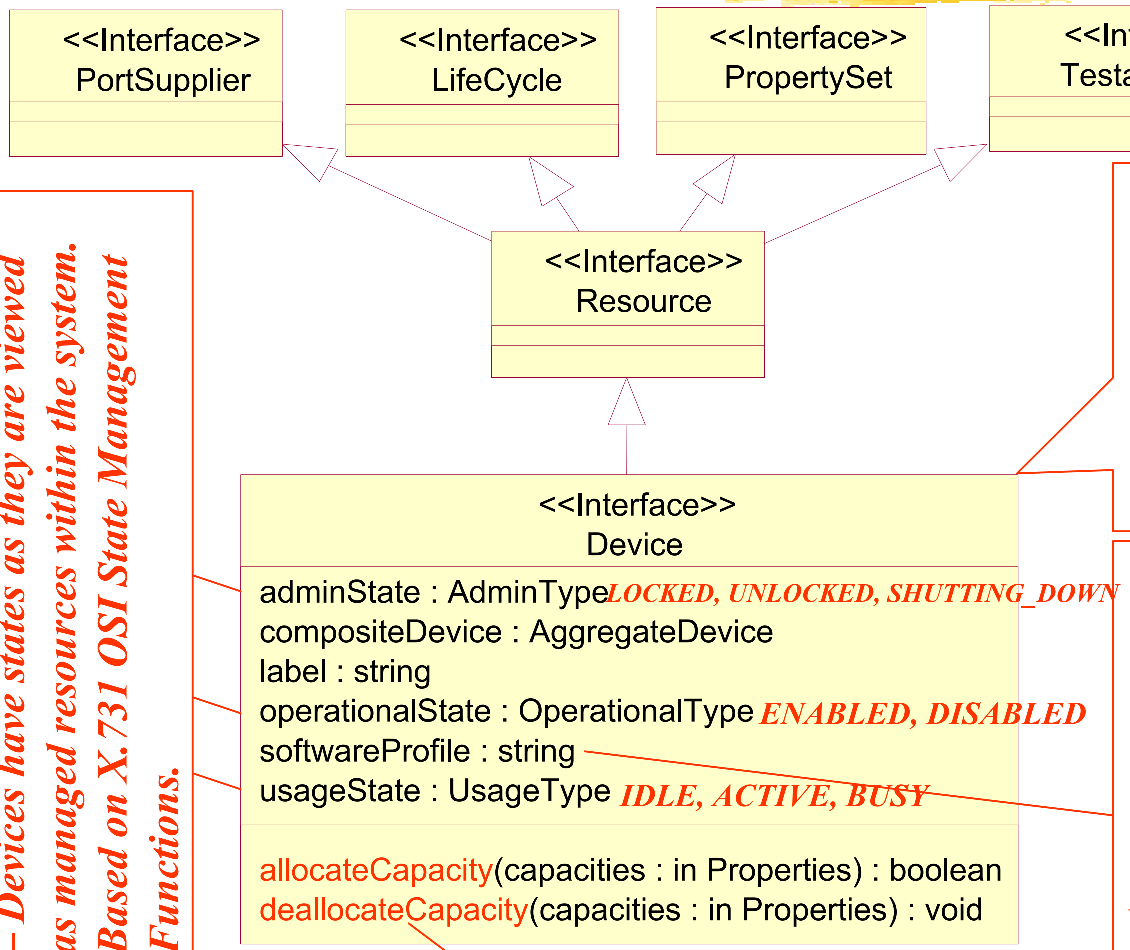
Framework of operations for executing/terminating processes/threads on a device

Framework of operations for adding/removing devices to/from a device capable of an aggregate relationship



Framework of attributes and unimplemented operations representing the functional abstraction of a logical device (zero or more hardware devices).

Device Interface


<<Interface>> PortSupplier
<<Interface>> LifeCycle
<<Interface>> PropertySet
<<Interface>> TestableObject
<<Interface>> Resource
<<Interface>> Device
adminState : AdminType
LOCKED, UNLOCKED, SHUTTING_DOWN
compositeDevice : AggregateDevice
label : string
operationalState : OperationalType
ENABLED, DISABLED
softwareProfile : string
usageState : UsageType
IDLE, ACTIVE, BUSY
allocateCapacity(capacities : in Properties) : boolean
deallocateCapacity(capacities : in Properties) : void

State Management and Status Attributes – Devices have states as they are viewed as managed resources within the system. Based on X.731 OSI State Management Functions.

Basic definition of all Logical Devices in a system.
Logical Device - an abstraction of 0 or more hardware devices.
Since it's a Resource, it can be created, controlled, and terminated like a Resource.

Software Profile attribute – Each device has a Software Package Descriptor (SPD) in the Domain Profile that defines the logical Device capabilities (data/command uses and provides ports, configure/query properties, capacity properties, status properties, etc.).

Implementation depends on the device's capacity model.

Device Interface Advantages

The use of the Domain Profile Software Profile Descriptor allows a generic HMI to be written to interface with any vendor's *Device* and to be used by a CF *DomainManager* and *ApplicationFactory* for deployment of components onto devices.

Since the *Device* interface merely provides an abstraction of hardware, it may be the case that many *Devices* can exist per physical hardware element.

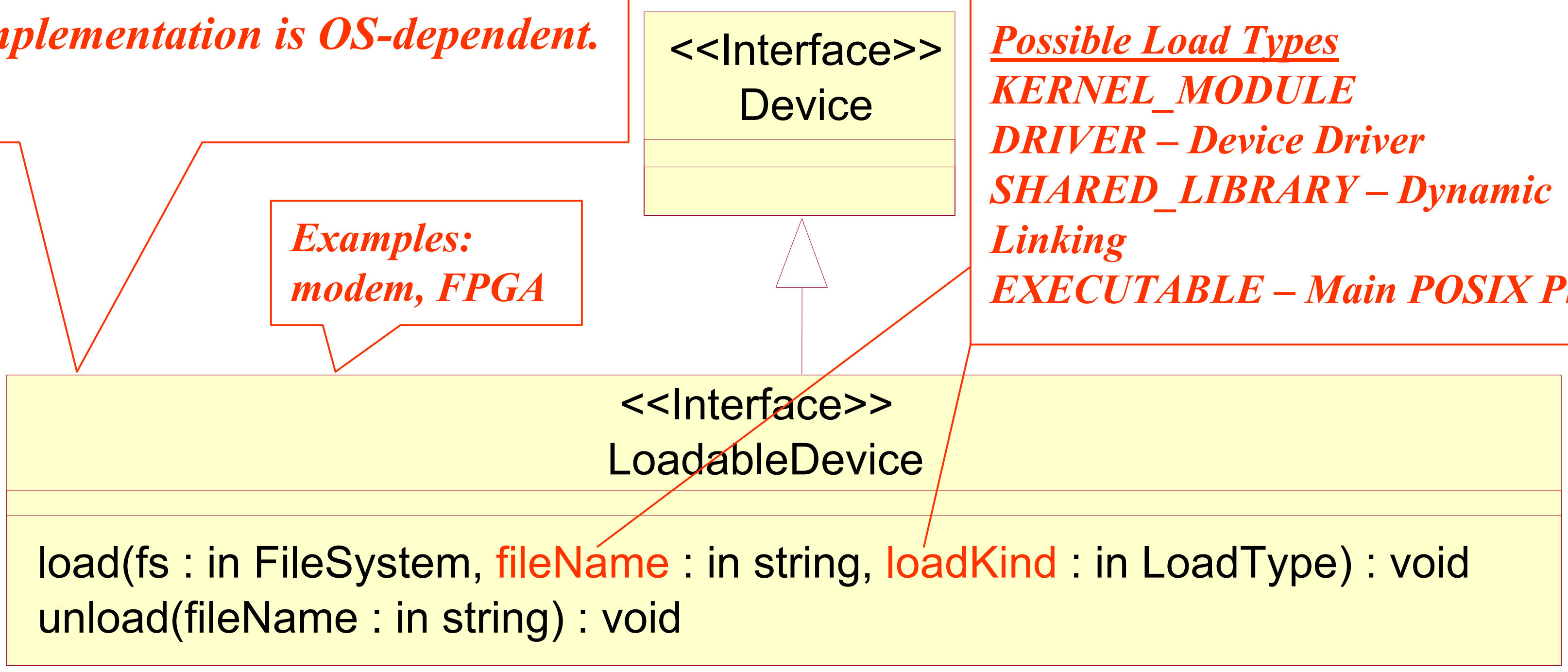
- A device such as a programmable modem could present itself as both a Code Division Multi-Access (CDMA) and a Time Division Multiple Access (TDMA) modem *Device* simultaneously.

This abstraction also allows physical devices to be emulated.

- For example, a *Device* such as a printer could be emulated on a general-purpose processor. Instead of providing a printed page of the information sent to it, the printer *Device* could present the page graphically on a display.

LoadableDevice Interface

- *Extends the Device interface by adding software loading and unloading behavior for a particular device.*
- *Implementation is OS-dependent.*

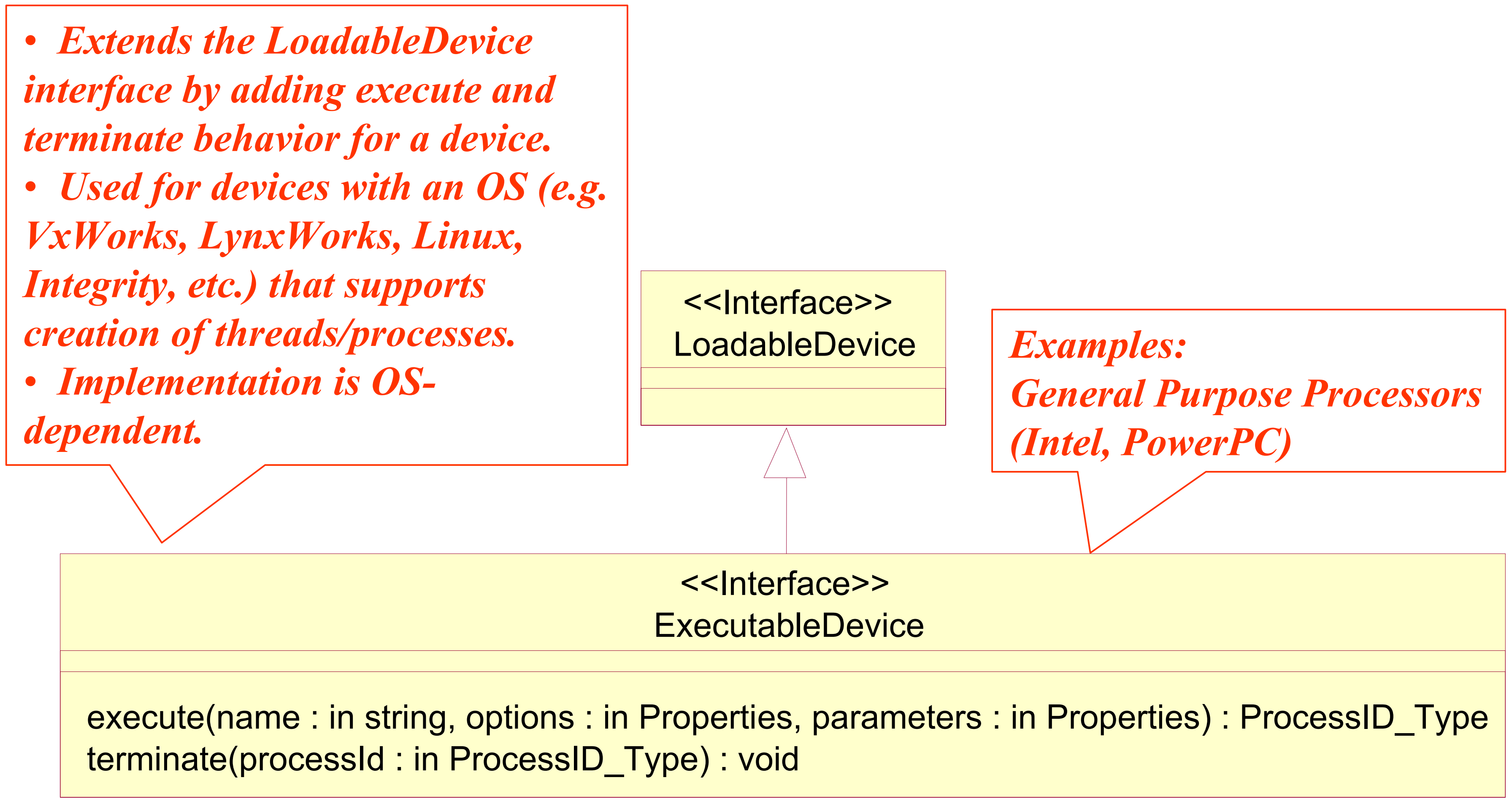


```
load(fs : in FileSystem, fileName : in string, loadKind : in LoadType) : void
unload(fileName : in string) : void
```

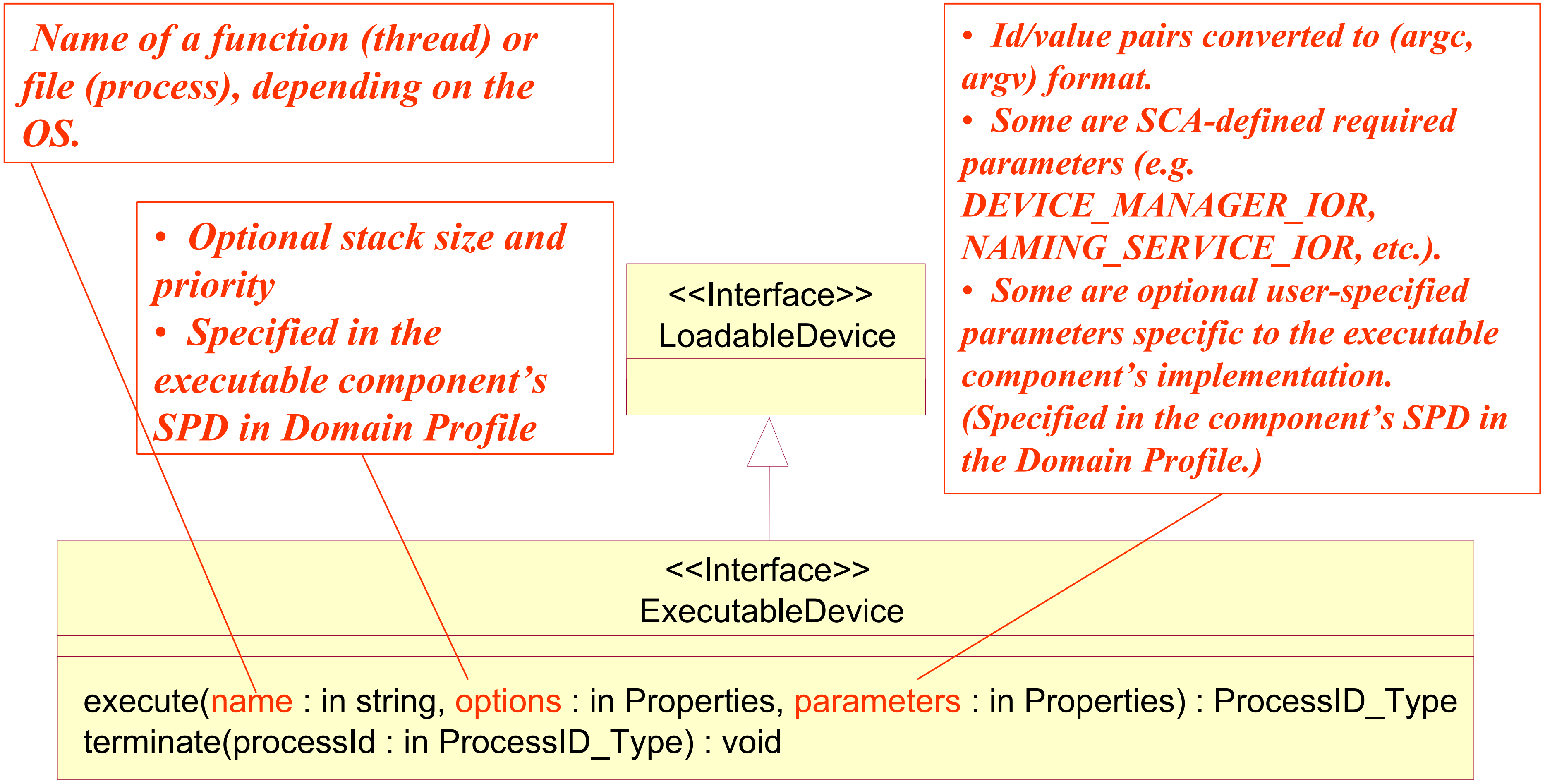
The load module file name and load type are located in the device component's Software Package Descriptor in the Domain Profile.

Possible Load Types
KERNEL_MODULE
DRIVER – Device Driver
SHARED_LIBRARY – Dynamic Linking
EXECUTABLE – Main POSIX Process

ExecutableDevice Interface


• Extends the LoadableDevice interface by adding execute and terminate behavior for a device.
• Used for devices with an OS (e.g. VxWorks, LynxWorks, Linux, Integrity, etc.) that supports creation of threads/processes.
• Implementation is OS-dependent.
<<Interface>> LoadableDevice
Examples: General Purpose Processors (Intel, PowerPC)
<<Interface>> ExecutableDevice
execute(name : in string, options : in Properties, parameters : in Properties) : ProcessID_Type
terminate(processId : in ProcessID_Type) : void

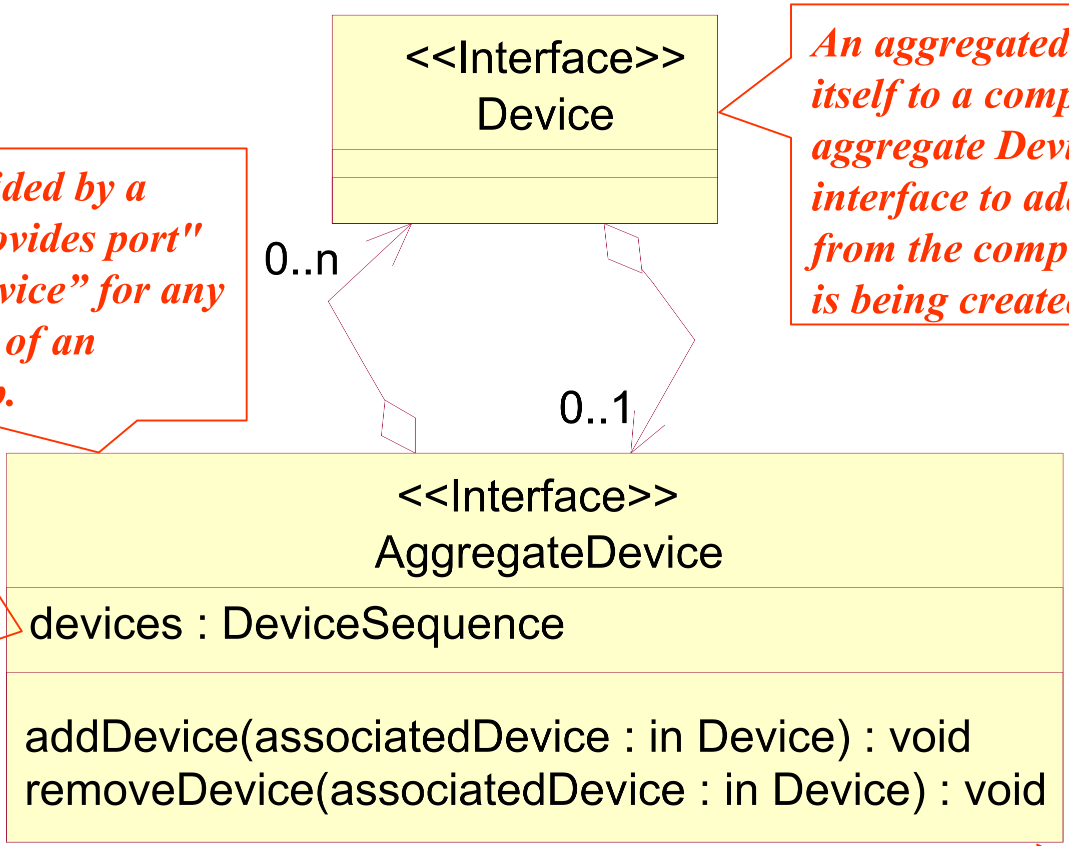
ExecutableDevice Interface (cont.)


Name of a function (thread) or file (process), depending on the OS.
• Optional stack size and priority
• Specified in the executable component's SPD in Domain Profile
• Id/value pairs converted to (argc, argv) format.
• Some are SCA-defined required parameters (e.g. DEVICE_MANAGER_IOR, NAMING_SERVICE_IOR, etc.).
• Some are optional user-specified parameters specific to the executable component's implementation. (Specified in the component's SPD in the Domain Profile.)
<<Interface>> LoadableDevice
<<Interface>> ExecutableDevice
execute(name : in string, options : in Properties, parameters : in Properties) : ProcessID_Type
terminate(processId : in ProcessID_Type) : void

AggregateDevice Interface

This Interface is provided by a Device through a "provides port" named "CompositeDevice" for any Device that is capable of an aggregate relationship.

An aggregated logical Device adds itself to a composite Device. An aggregate Device uses this interface to add or remove itself from the composite Device when it is being created or torn-down.



<<Interface>>
Device
0..n
0..1
<<Interface>>
AggregateDevice
devices : DeviceSequence
addDevice(associatedDevice : in Device) : void
removeDevice(associatedDevice : in Device) : void

List of Devices that have been added to this Device through the addDevice( ) operation and have not yet been removed through the removeDevice( ) operation.

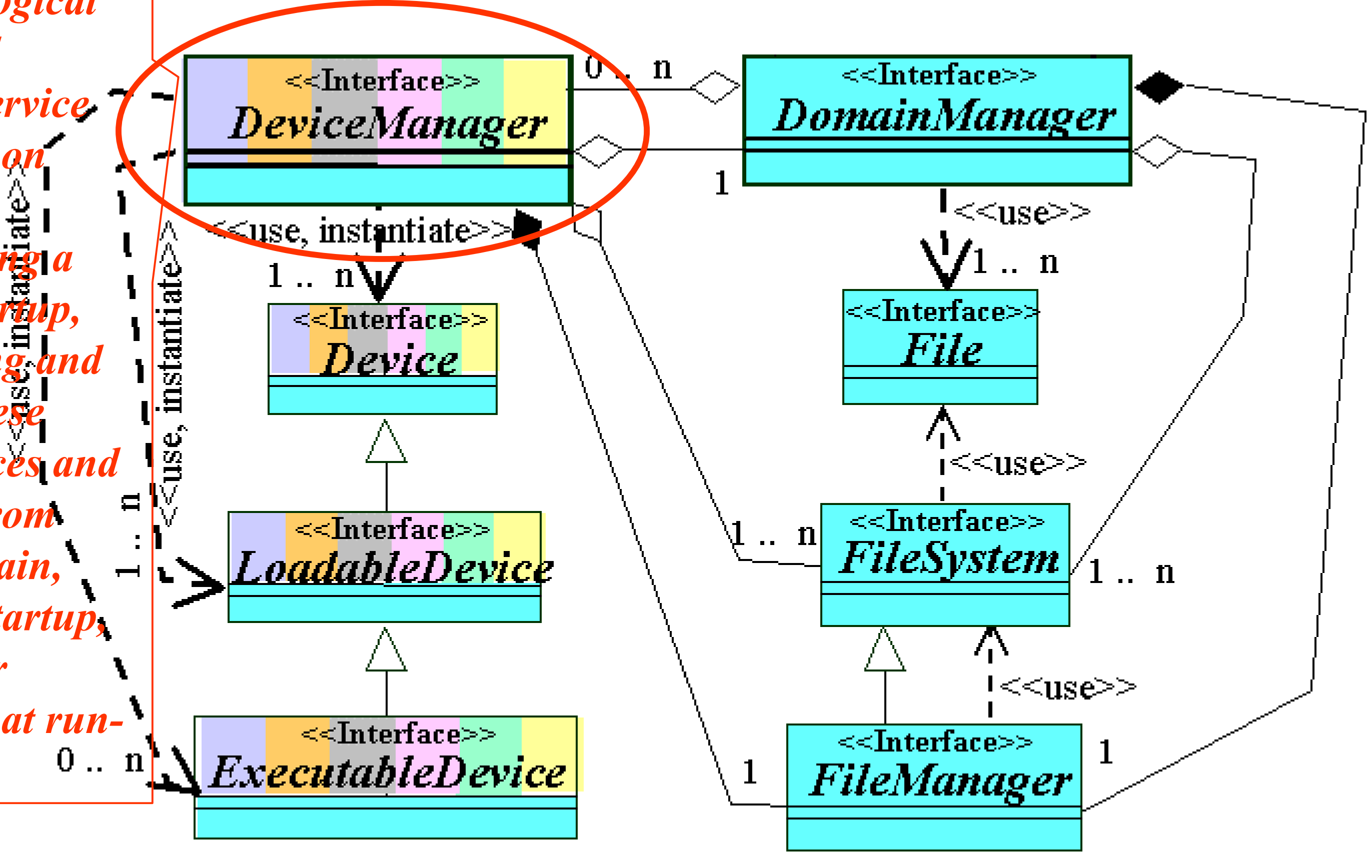
Examples:

- *INFOSEC Device with "n" crypto channel Devices*
- *I/O Device with "n" ports*

CF Framework Control Interface for Node Component Deployment and Removal

Node Component Deployment and Removal Interface

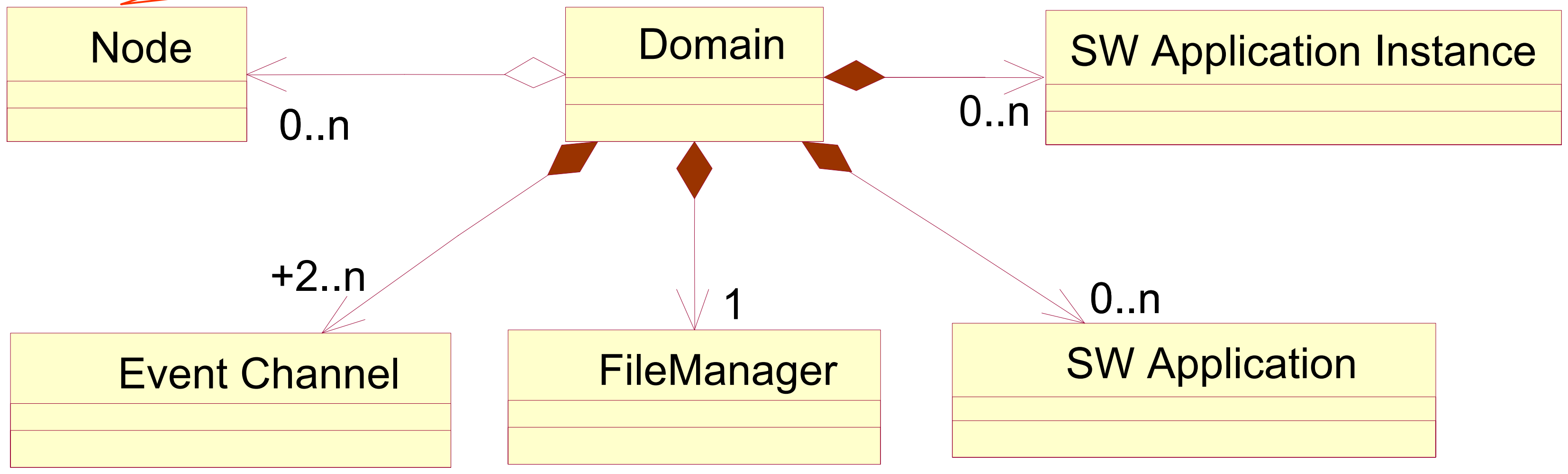
Responsible for creation of logical Devices, and launching service applications on these logical Devices during a CF Node startup, and deploying and removing these logical Devices and services to/from the CF Domain, during CF startup, shutdown, or dynamically at run-time.



What the DeviceManager Represents in the CF Domain

A given system is composed of a set of nodes (CORBA-capable boards) with persistent components deployed and running on them.

Each node is associated with a DeviceManager. And each node is viewed as an independent object within the system.



DeviceManager’s Job

The *DeviceManager* manages a set of persistent components for a given node within a domain.

- Logical *Devices* and services (e.g., *Lo*g, Event Service, Naming Service, domain-specific services and devices).

It manages these persistent components by

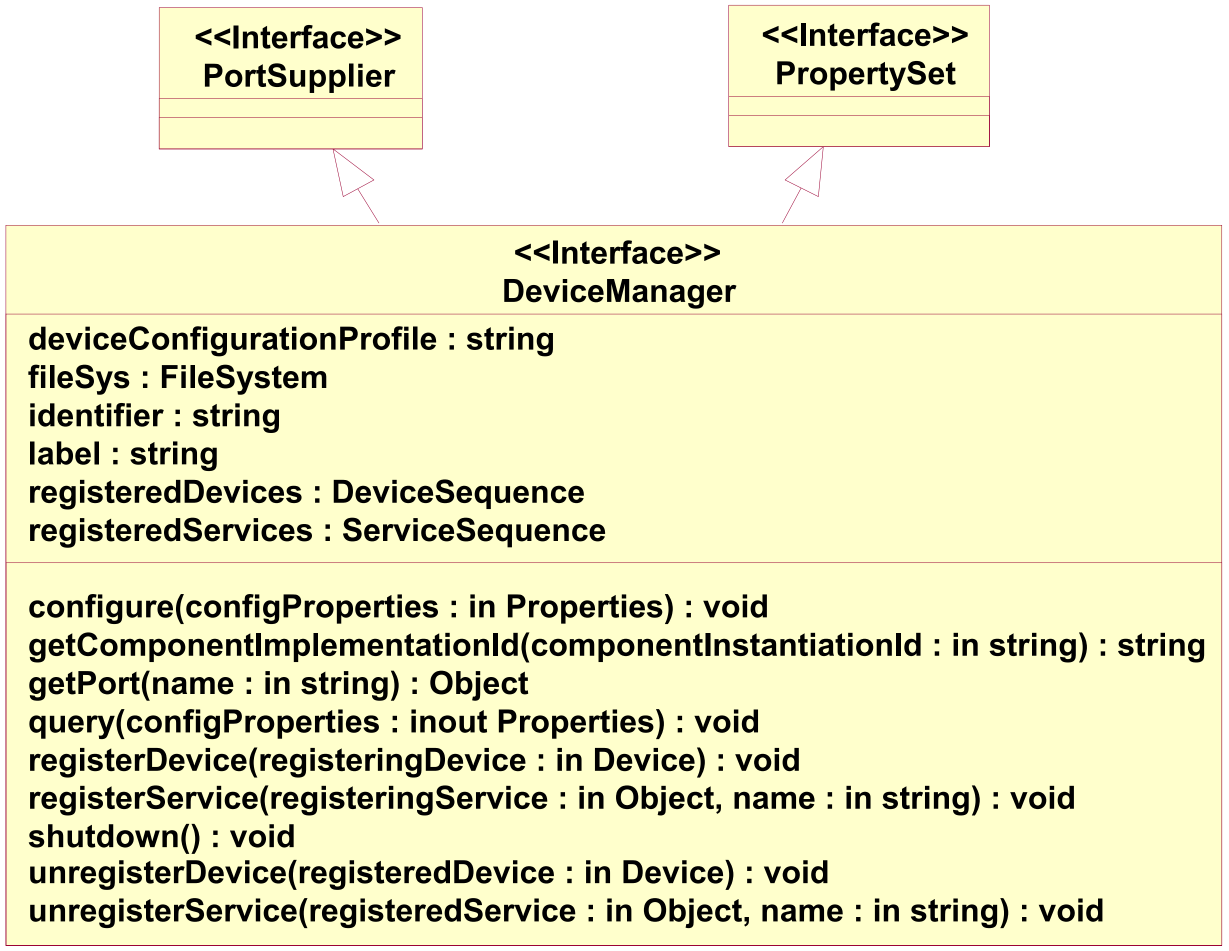
- Starting them up as specified in the Domain Profile file, known as the Device Configuration Descriptor (DCD) file, AKA DeviceManager Profile
- Provides operations for deploying them to the CF domain as directed by the DCD during *DeviceManager* startup,
- And removing them from the CF domain upon *DeviceManager* shutdown.

DeviceManager's Job (cont.)

During startup of Logical Devices and service components, the *DeviceManager* supplies a set of well-defined executable parameters to the component to promote portability of logical *Devices* and services and consistent behavior from one SCA-complaint system to another SCA-complaint system that hosts these components.

- *DEVICE_MGR_IOR*
- *PROFILE_NAME*
- *DEVICE_ID*
- *DEVICE_LABEL*
- *Composite_DEVICE_IOR*
- *SERVICE_NAME*
- Component-specific startup parameters defined in the component's Software Profile (SPD) processed by *DeviceManager*

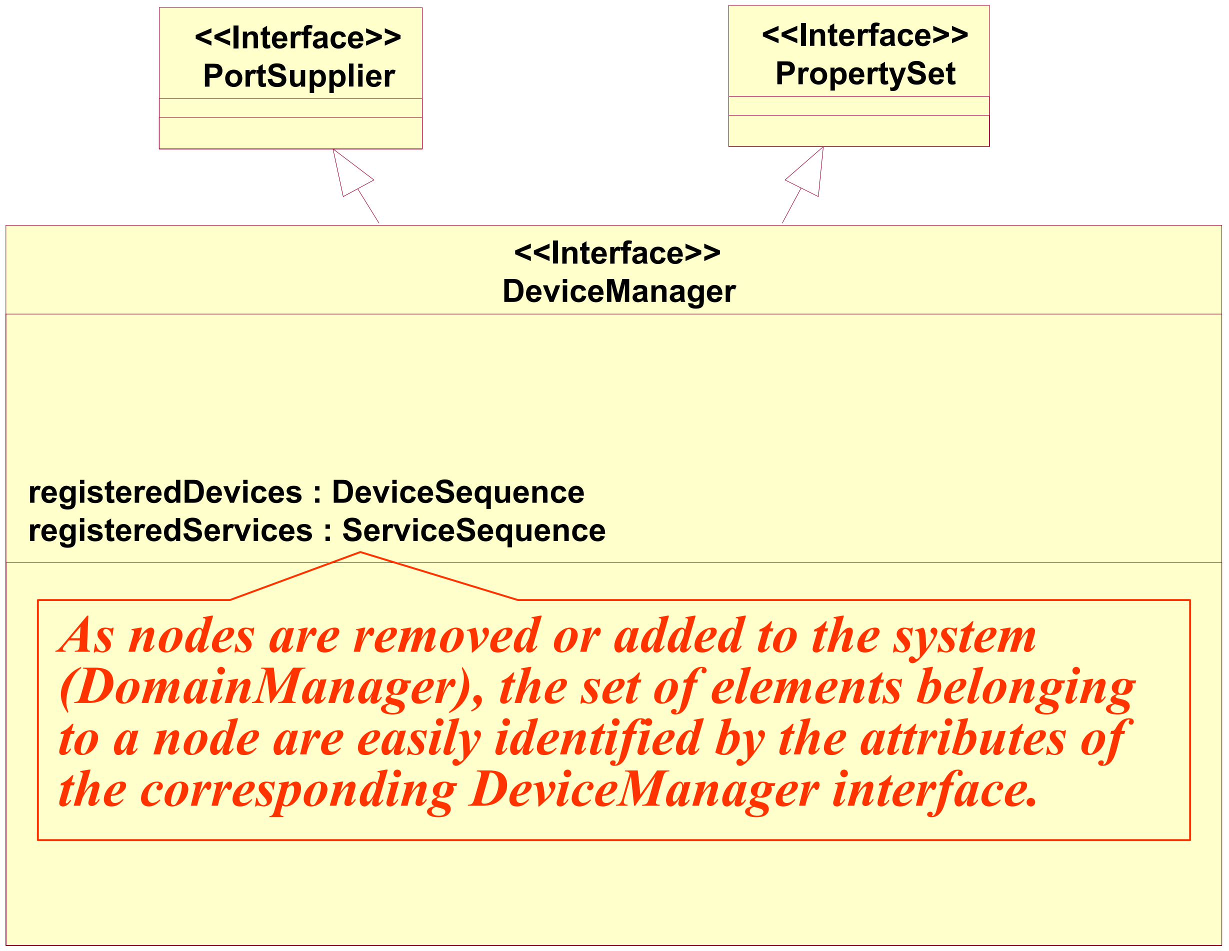
DeviceManager Interface



deviceConfigurationProfile : string
fileSys : FileSystem
identifier : string
label : string
registeredDevices : DeviceSequence
registeredServices : ServiceSequence

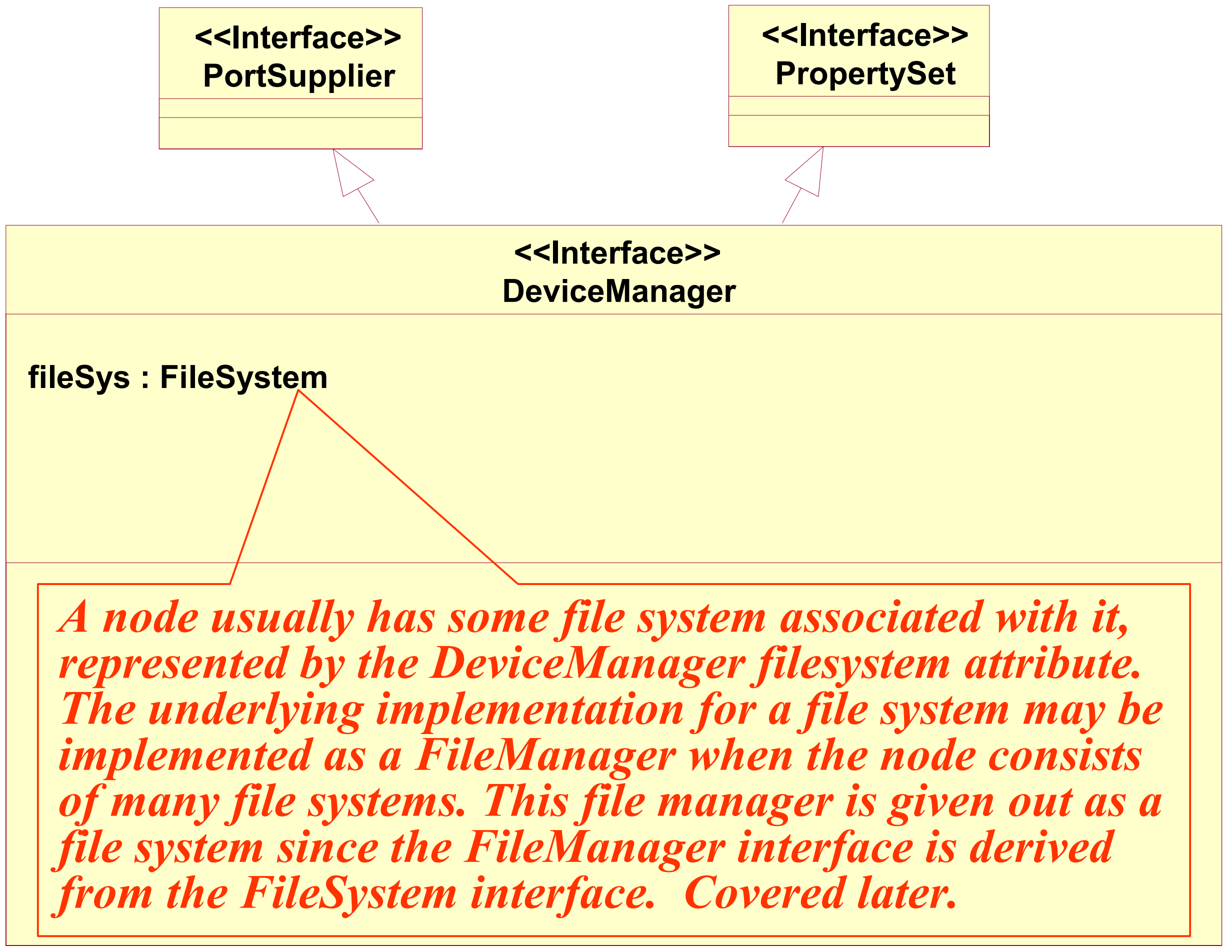
configure(configProperties : in Properties) : void
getComponentImplementationId(componentInstantiationId : in string) : string
getPort(name : in string) : Object
query(configProperties : inout Properties) : void
registerDevice(registeringDevice : in Device) : void
registerService(registeringService : in Object, name : in string) : void
shutdown() : void
unregisterDevice(registeredDevice : in Device) : void
unregisterService(registeredService : in Object, name : in string) : void

Elements of a DeviceManager Node



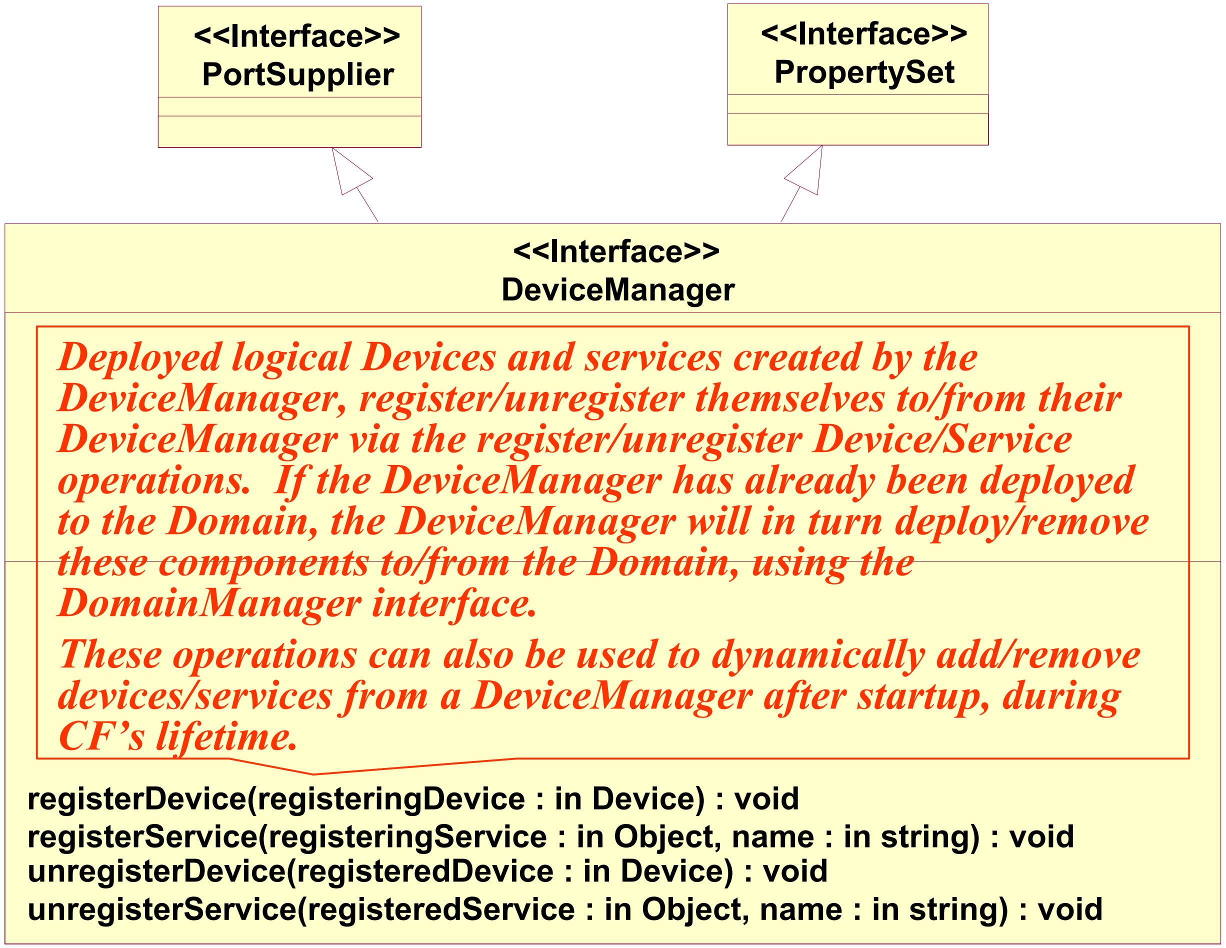
As nodes are removed or added to the system (DomainManager), the set of elements belonging to a node are easily identified by the attributes of the corresponding DeviceManager interface.

DeviceManager's FileSystem



A node usually has some file system associated with it, represented by the DeviceManager filesystem attribute. The underlying implementation for a file system may be implemented as a FileManager when the node consists of many file systems. This file manager is given out as a file system since the FileManager interface is derived from the FileSystem interface. Covered later.

Registration of Logical Devices and Services with DeviceManager

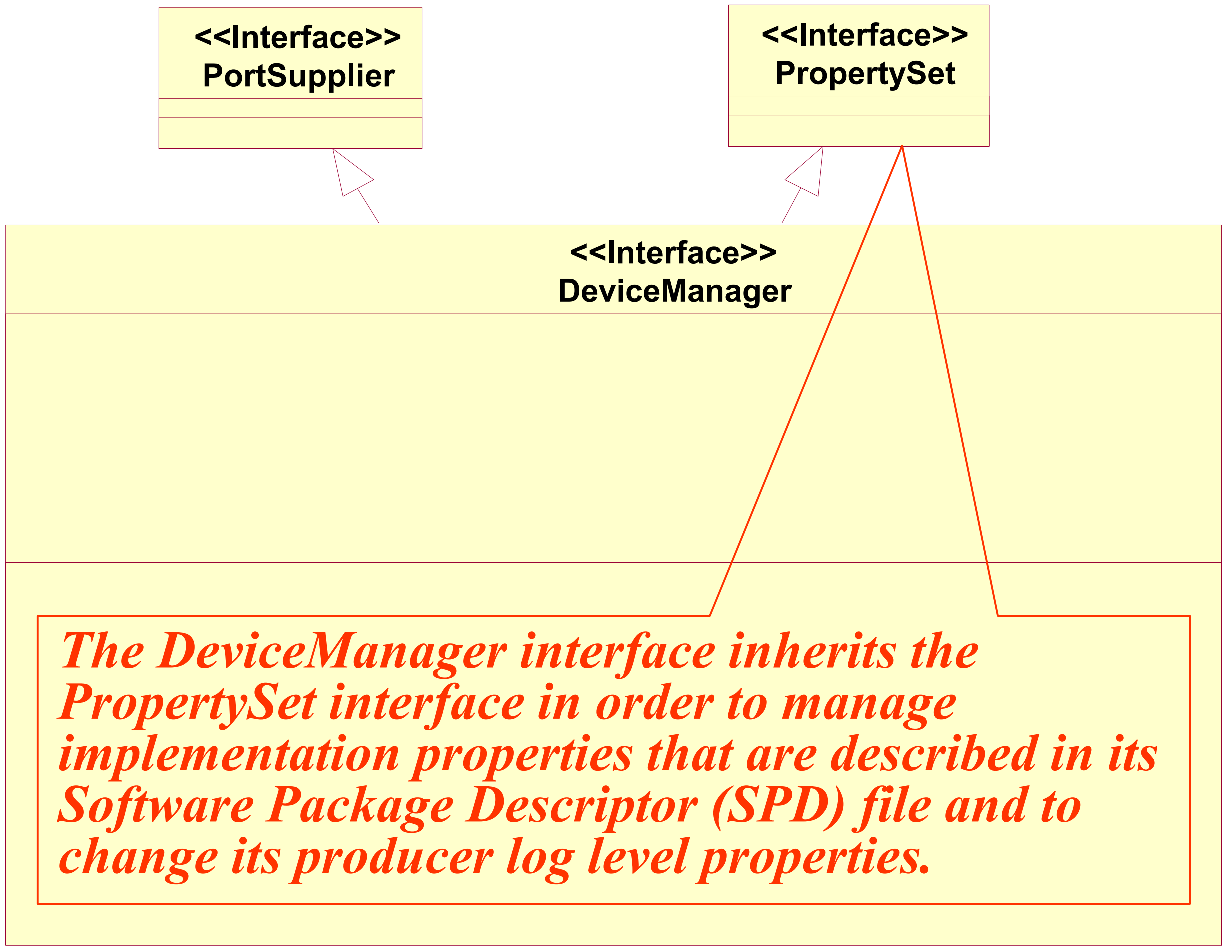


Deployed logical Devices and services created by the DeviceManager, register/unregister themselves to/from their DeviceManager via the register/unregister Device/Service operations. If the DeviceManager has already been deployed to the Domain, the DeviceManager will in turn deploy/remove these components to/from the Domain, using the DomainManager interface.

These operations can also be used to dynamically add/remove devices/services from a DeviceManager after startup, during CF's lifetime.

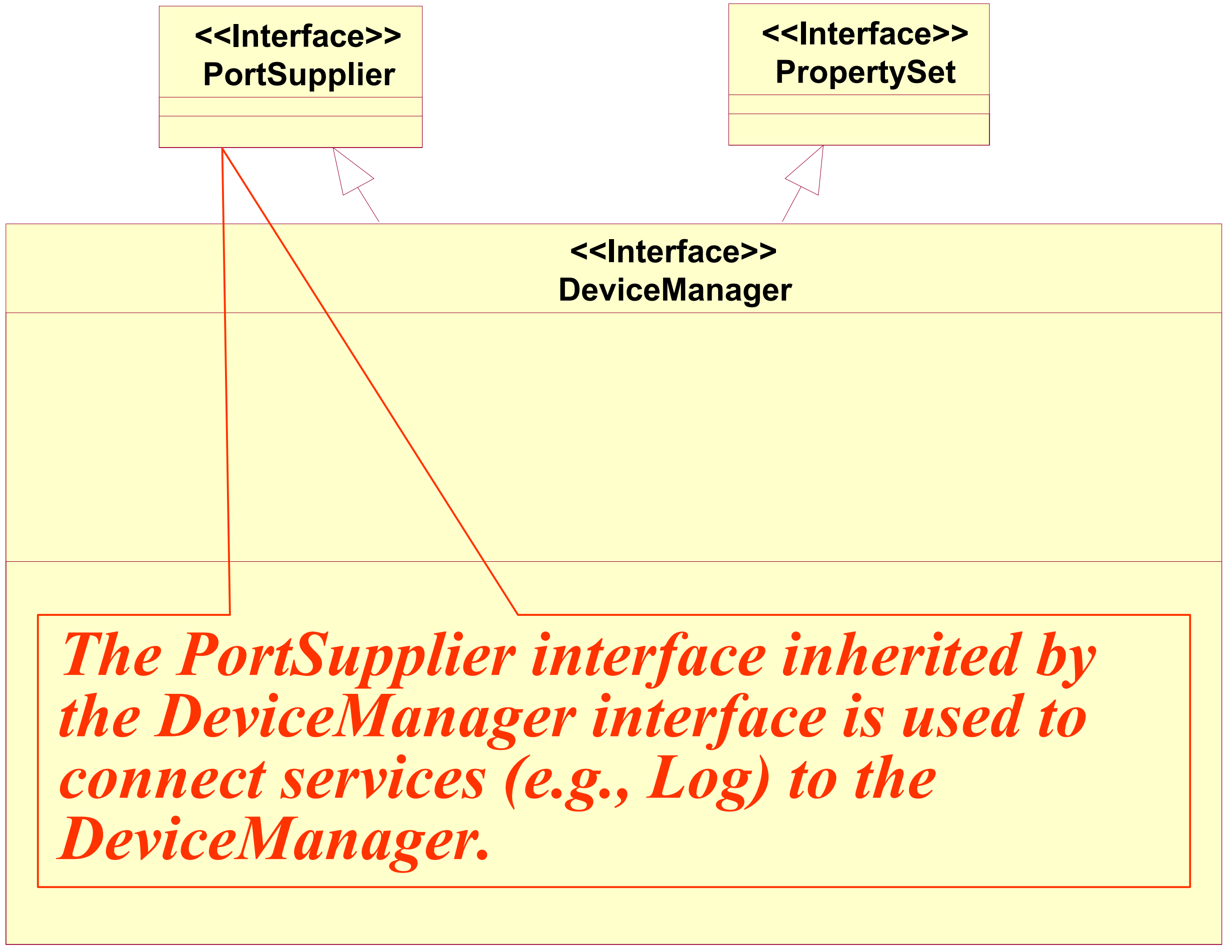
```
registerDevice(registeringDevice : in Device) : void
registerService(registeringService : in Object, name : in string) : void
unregisterDevice(registeredDevice : in Device) : void
unregisterService(registeredService : in Object, name : in string) : void
```

DeviceManager Configure/Query Properties



The DeviceManager interface inherits the PropertySet interface in order to manage implementation properties that are described in its Software Package Descriptor (SPD) file and to change its producer log level properties.

Services for DeviceManager's Use



The PortSupplier interface inherited by the DeviceManager interface is used to connect services (e.g., Log) to the DeviceManager.

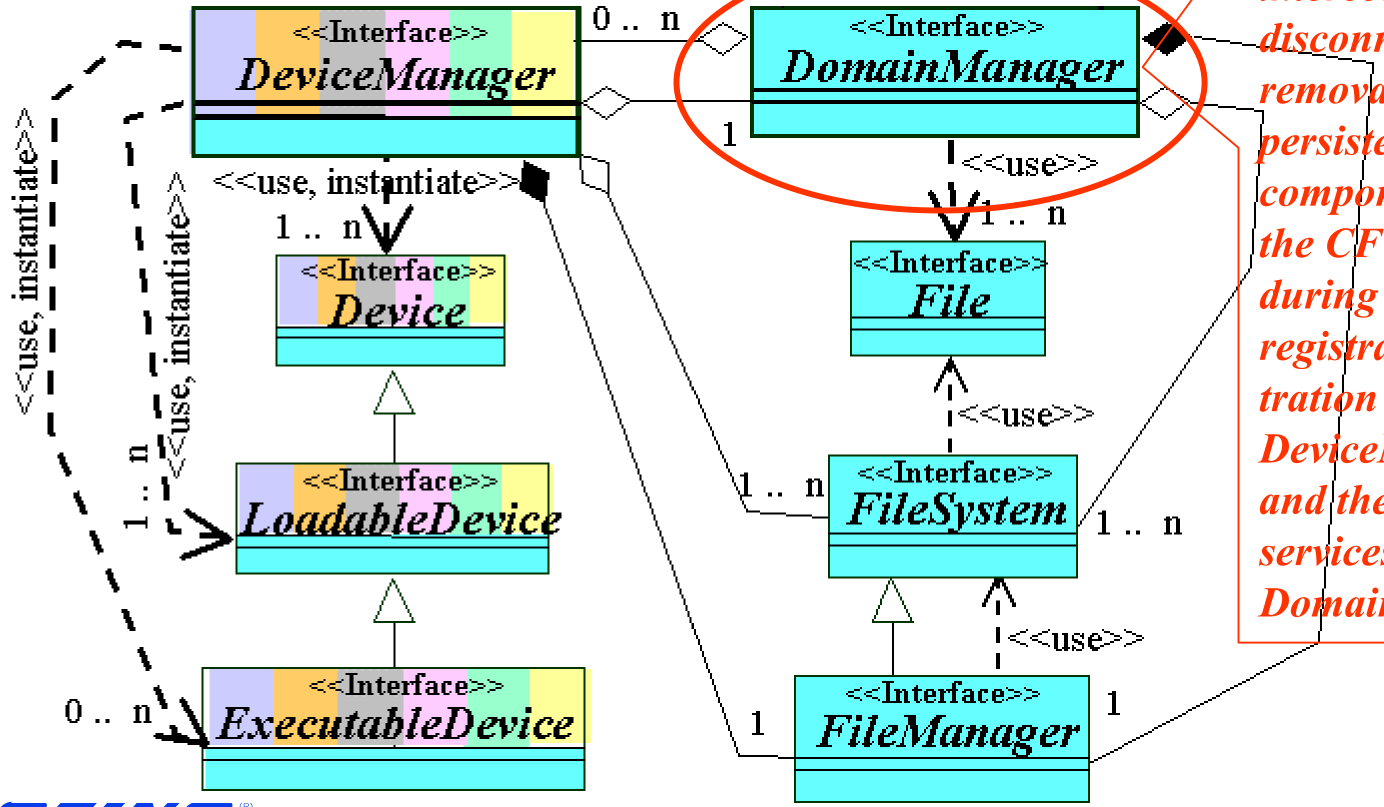
CF Framework Control Interface for Domain Component Deployment and Removal

&

Application Installation and Un-installation

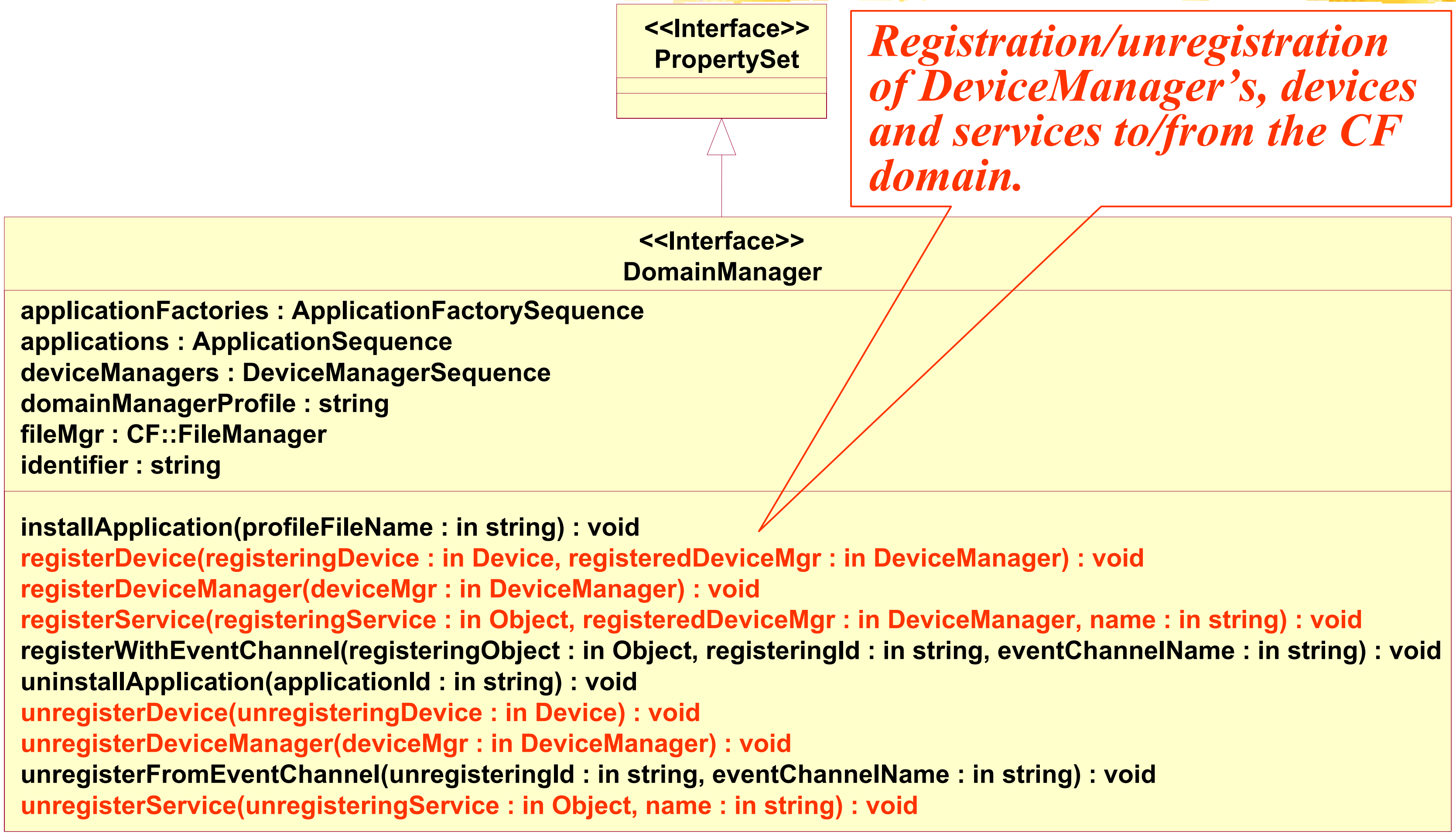
DomainManager's Job

To support the instantiation and tear-down of applications in the CF domain, the DomainManager provides 2 kinds of services:



First, it provides the means for deployment, interconnection, disconnection, and removal of persistent components to/from the CF domain during registration/unregistration of DeviceManagers and their devices, services, with/from DomainManager.

DomainManager Provides Operations for...

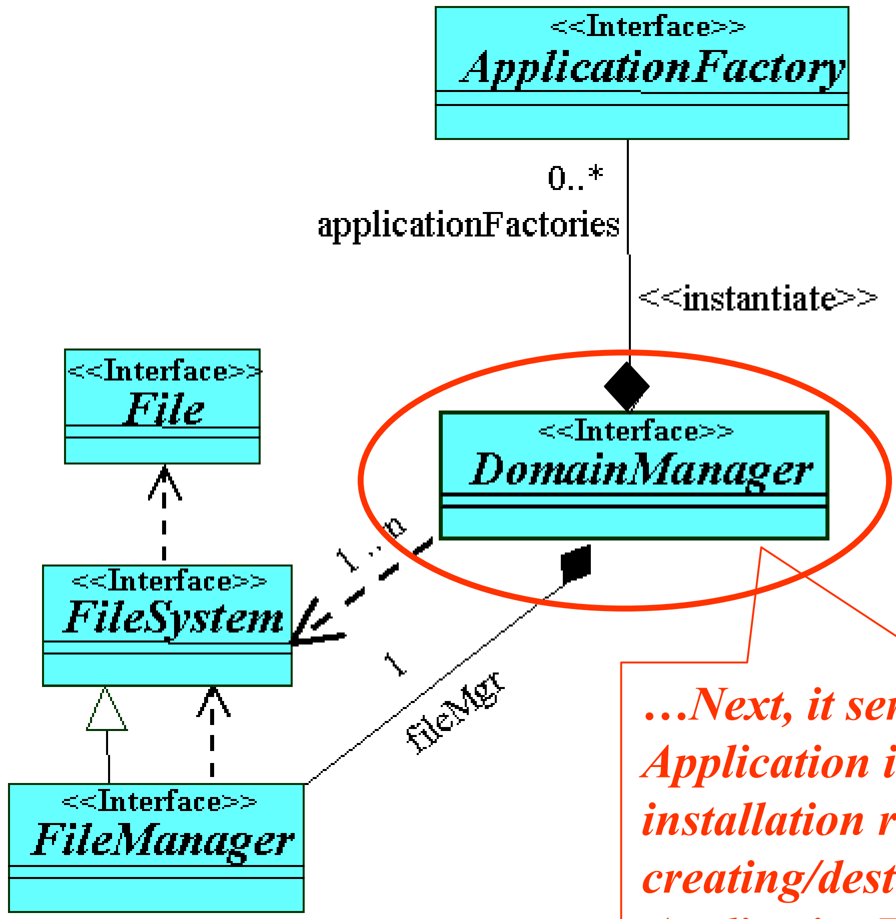


<<Interface>>
PropertySet
Registration/unregistration of DeviceManager's, devices and services to/from the CF domain.
<<Interface>>
DomainManager

applicationFactories : ApplicationFactorySequence
applications : ApplicationSequence
deviceManagers : DeviceManagerSequence
domainManagerProfile : string
fileMgr : CF::FileManager
identifier : string

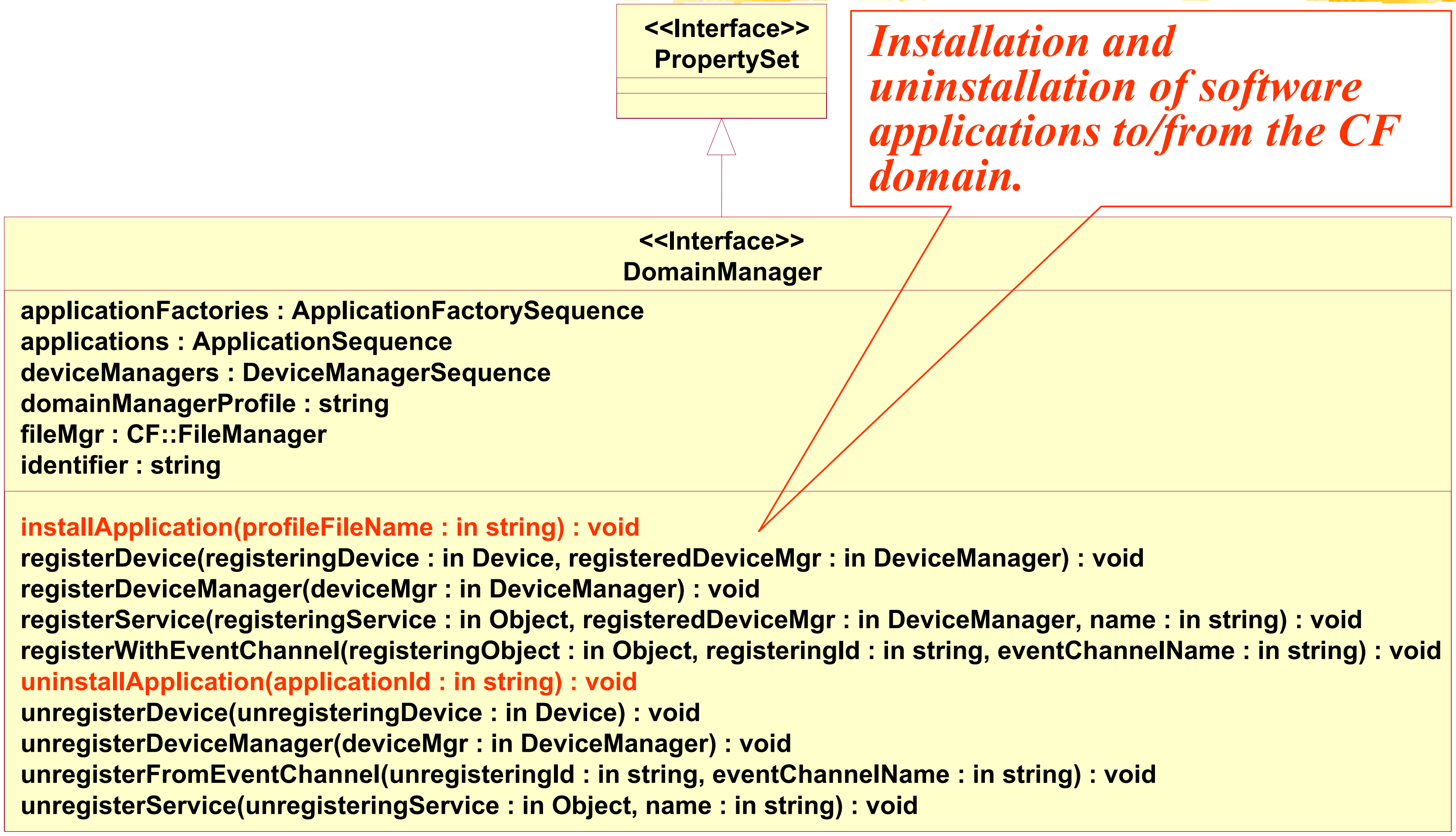
installApplication(profileFileName : in string) : void
registerDevice(registeringDevice : in Device, registeredDeviceMgr : in DeviceManager) : void
registerDeviceManager(deviceMgr : in DeviceManager) : void
registerService(registeringService : in Object, registeredDeviceMgr : in DeviceManager, name : in string) : void
registerWithEventChannel(registeringObject : in Object, registeringId : in string, eventChannelName : in string) : void
uninstallApplication(applicationId : in string) : void
unregisterDevice(unregisteringDevice : in Device) : void
unregisterDeviceManager(deviceMgr : in DeviceManager) : void
unregisterFromEventChannel(unregisteringId : in string, eventChannelName : in string) : void
unregisterService(unregisteringService : in Object, name : in string) : void

DomainManager's Job (cont.)



…Next, it services Domain Application installation and un-installation requests by creating/destroying a corresponding ApplicationFactory for each installed Application.

DomainManager Provides Operations for...

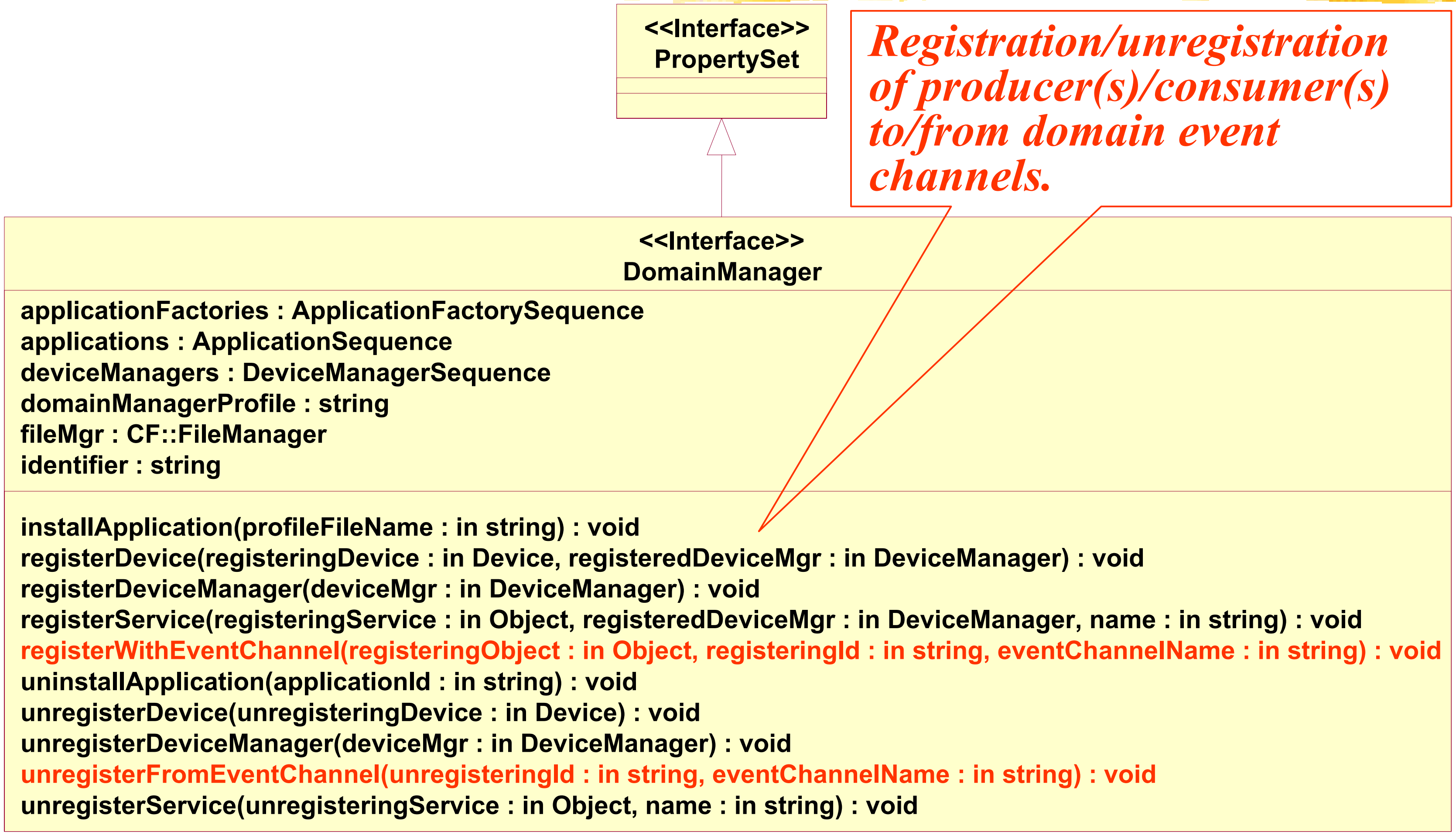


<<Interface>>
PropertySet
Installation and uninstallation of software applications to/from the CF domain.
<<Interface>>
DomainManager

applicationFactories : ApplicationFactorySequence
applications : ApplicationSequence
deviceManagers : DeviceManagerSequence
domainManagerProfile : string
fileMgr : CF::FileManager
identifier : string

installApplication(profileFileName : in string) : void
registerDevice(registeringDevice : in Device, registeredDeviceMgr : in DeviceManager) : void
registerDeviceManager(deviceMgr : in DeviceManager) : void
registerService(registeringService : in Object, registeredDeviceMgr : in DeviceManager, name : in string) : void
registerWithEventChannel(registeringObject : in Object, registeringId : in string, eventChannelName : in string) : void
uninstallApplication(applicationId : in string) : void
unregisterDevice(unregisteringDevice : in Device) : void
unregisterDeviceManager(deviceMgr : in DeviceManager) : void
unregisterFromEventChannel(unregisteringId : in string, eventChannelName : in string) : void
unregisterService(unregisteringService : in Object, name : in string) : void

DomainManager Provides Operations for...

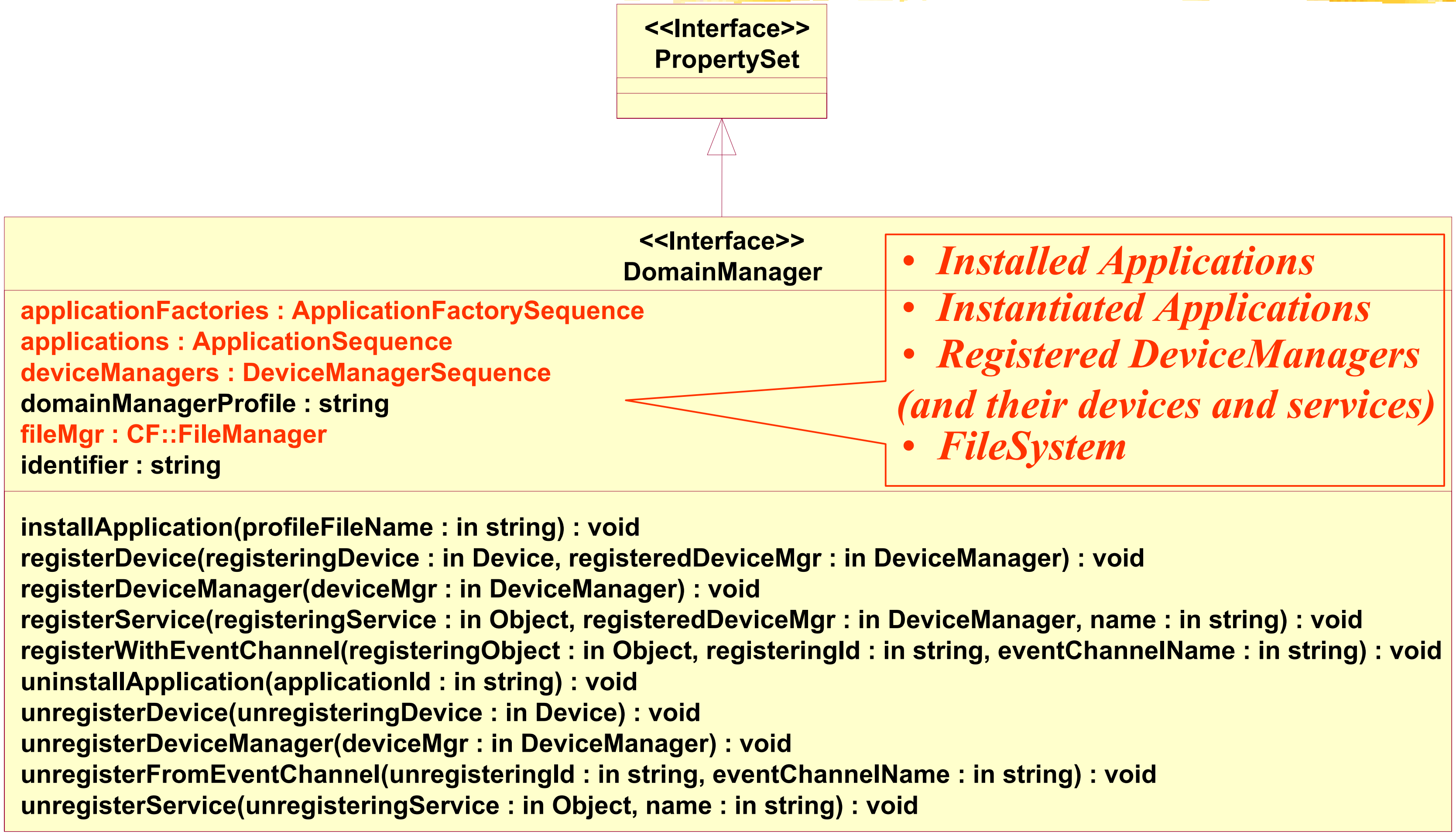


<<Interface>>
PropertySet
Registration/unregistration of producer(s)/consumer(s) to/from domain event channels.
<<Interface>>
DomainManager

applicationFactories : ApplicationFactorySequence
applications : ApplicationSequence
deviceManagers : DeviceManagerSequence
domainManagerProfile : string
fileMgr : CF::FileManager
identifier : string

installApplication(profileFileName : in string) : void
registerDevice(registeringDevice : in Device, registeredDeviceMgr : in DeviceManager) : void
registerDeviceManager(deviceMgr : in DeviceManager) : void
registerService(registeringService : in Object, registeredDeviceMgr : in DeviceManager, name : in string) : void
registerWithEventChannel(registeringObject : in Object, registeringId : in string, eventChannelName : in string) : void
uninstallApplication(applicationId : in string) : void
unregisterDevice(unregisteringDevice : in Device) : void
unregisterDeviceManager(deviceMgr : in DeviceManager) : void
unregisterFromEventChannel(unregisteringId : in string, eventChannelName : in string) : void
unregisterService(unregisteringService : in Object, name : in string) : void

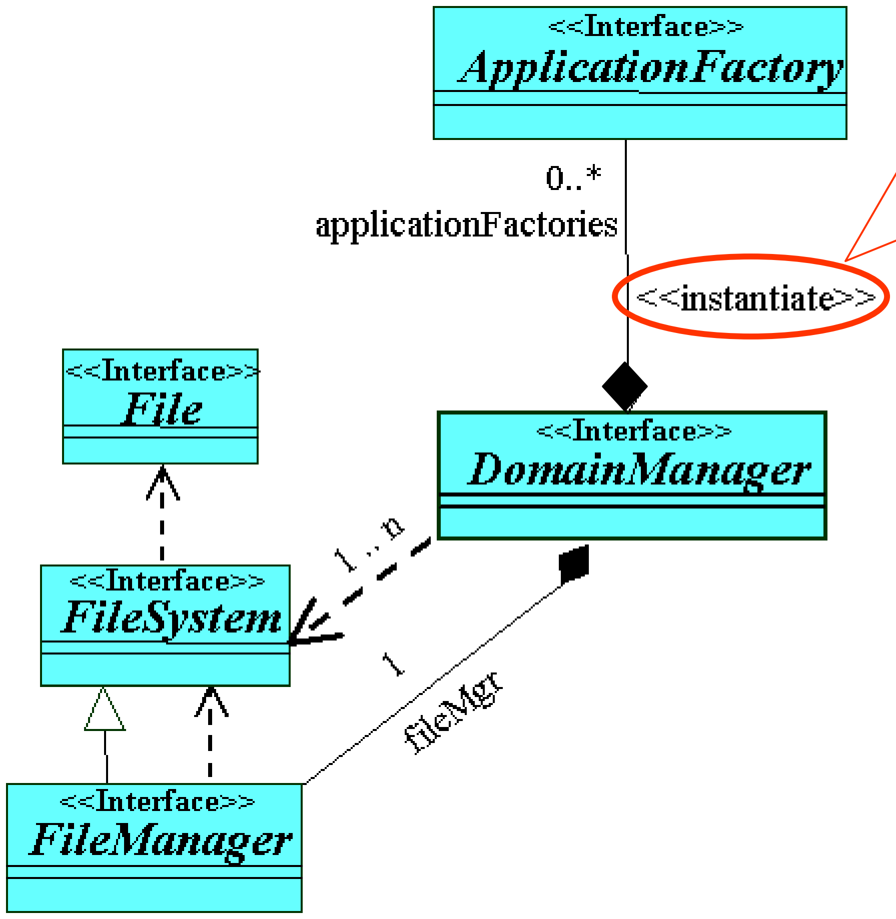
DomainManager Provides Access to the Domain's...



<<Interface>>
PropertySet
<<Interface>>
DomainManager
applicationFactories : ApplicationFactorySequence
applications : ApplicationSequence
deviceManagers : DeviceManagerSequence
domainManagerProfile : string
fileMgr : CF::FileManager
identifier : string
installApplication(profileFileName : in string) : void
registerDevice(registeringDevice : in Device, registeredDeviceMgr : in DeviceManager) : void
registerDeviceManager(deviceMgr : in DeviceManager) : void
registerService(registeringService : in Object, registeredDeviceMgr : in DeviceManager, name : in string) : void
registerWithEventChannel(registeringObject : in Object, registeringId : in string, eventChannelName : in string) : void
uninstallApplication(applicationId : in string) : void
unregisterDevice(unregisteringDevice : in Device) : void
unregisterDeviceManager(deviceMgr : in DeviceManager) : void
unregisterFromEventChannel(unregisteringId : in string, eventChannelName : in string) : void
unregisterService(unregisteringService : in Object, name : in string) : void
• Installed Applications
• Instantiated Applications
• Registered DeviceManagers (and their devices and services)
• FileSystem

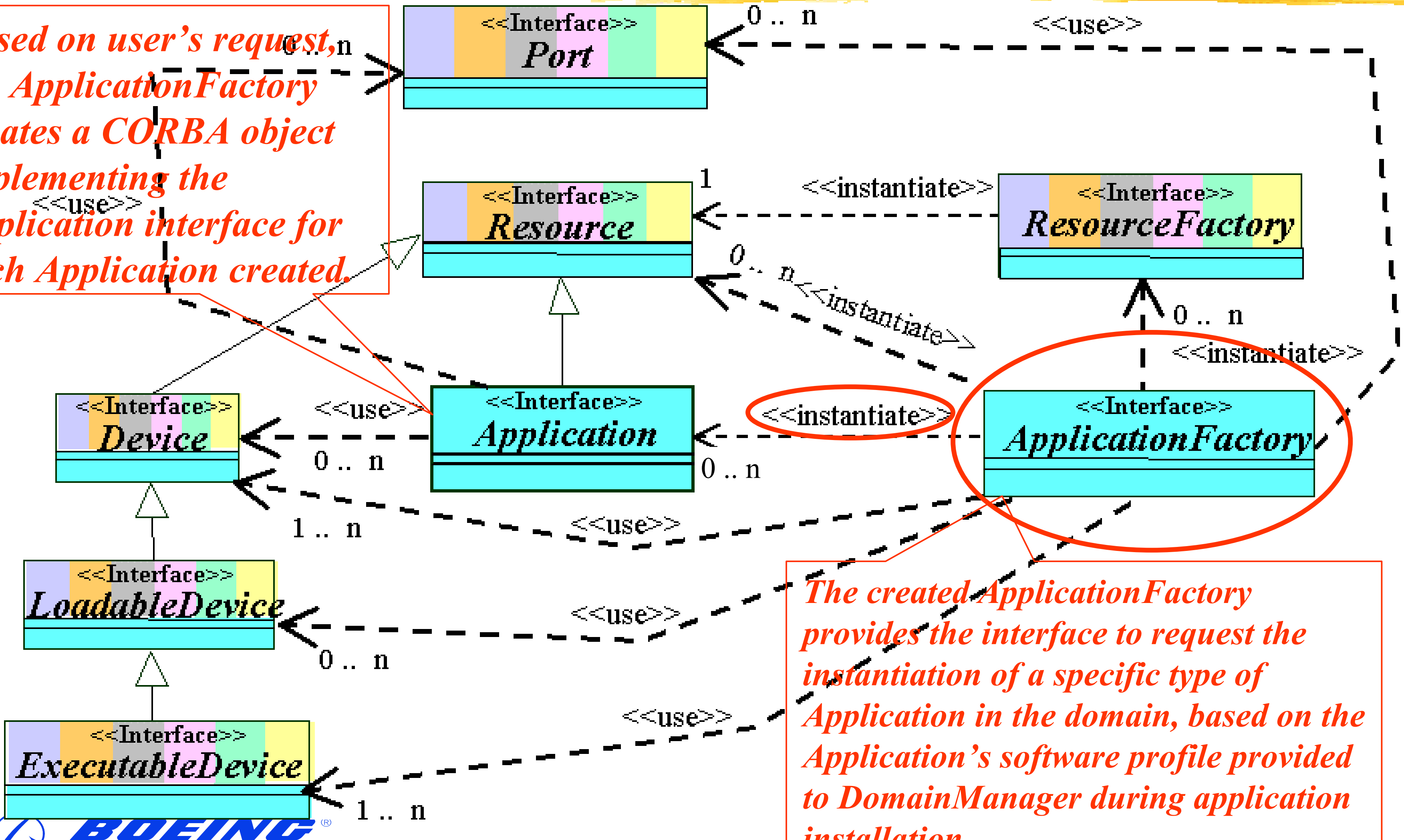
CF Framework Control Application Creation Interface

Installation of an Application…



…results in creation of a corresponding ApplicationFactory for that type of Application.

Application Creation Interface



Based on user's request, the ApplicationFactory creates a CORBA object implementing the Application interface for each Application created.

The created ApplicationFactory provides the interface to request the instantiation of a specific type of Application in the domain, based on the Application's software profile provided to DomainManager during application installation.

ApplicationFactory Interface

Based on the Domain Profile, creates instances of software applications of a specific type by:

- *Allocating software (Resources) to hardware (Devices)*
  - *Allocating capacities against Devices*
- *Connecting Resources that make up an Application*
- *Performing initial configuration on these Resources*

```
<<Interface>>
ApplicationFactory
---------------------------------
identifier : string
name : string
softwareProfile : string
---------------------------------
create(name : in string,
        initConfiguration : in Properties,
        deviceAssignments : in DeviceAssignmentSequence) : Application
```

ApplicationFactory Interface (cont.)

User-friendly name for the application type, identifier, and the software profile (Domain Profile Software Assembly Descriptor (SAD)) used for creating applications of that type.

```
<<Interface>>
ApplicationFactory

identifier : string
name : string
softwareProfile : string

create(name : in string,
        initConfiguration : in Properties,
        deviceAssignments : in DeviceAssignmentSequence) : Application
```

SAD ID – Used to identify the corresponding ApplicationFactory when logging messages or sending events to the Outgoing Domain Management Event Channel.

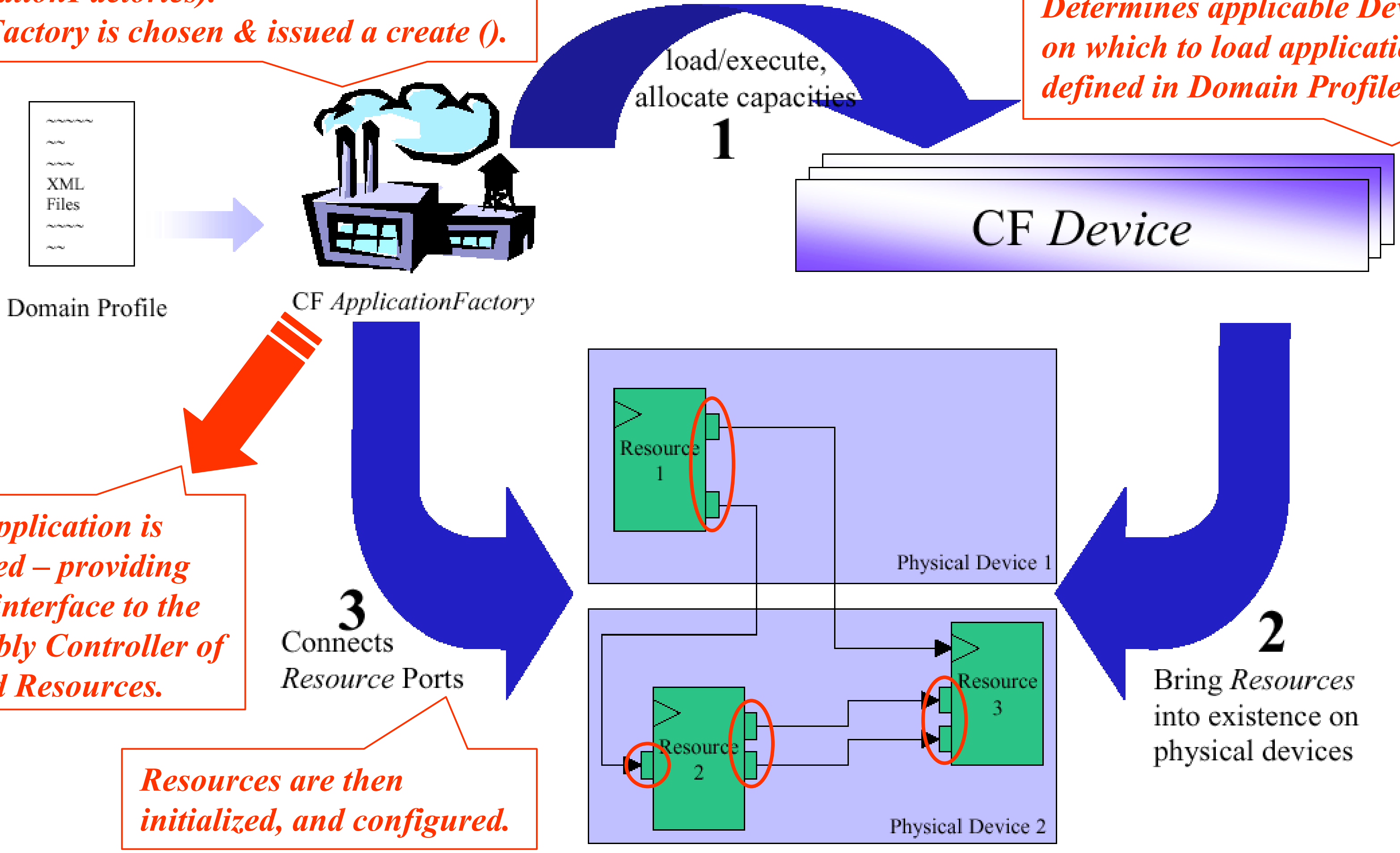
The create operation allows the user to request the creation of an application, provide it with pre-selected devices, and specify its configuration properties.

BOEING®

How an Application Instance is Brought into Existence

- *UI asks for all installed Apps (ApplicationFactories).*
- *AppFactory is chosen & issued a create ().*

Determines applicable Device(s) on which to load application code defined in Domain Profile.



~~~~~
~~
~~~
XML
Files
~~~~
~~
Domain Profile
CF ApplicationFactory
load/execute,
allocate capacities
1
CF Device
Resource 1
Physical Device 1
Resource 2
Resource 3
Physical Device 2
2
Bring Resources into existence on physical devices
3
Connects Resource Ports

*CF::Application is returned – providing proxy interface to the Assembly Controller of created Resources.*

*Resources are then initialized, and configured.*

BOEING®
~~~~~

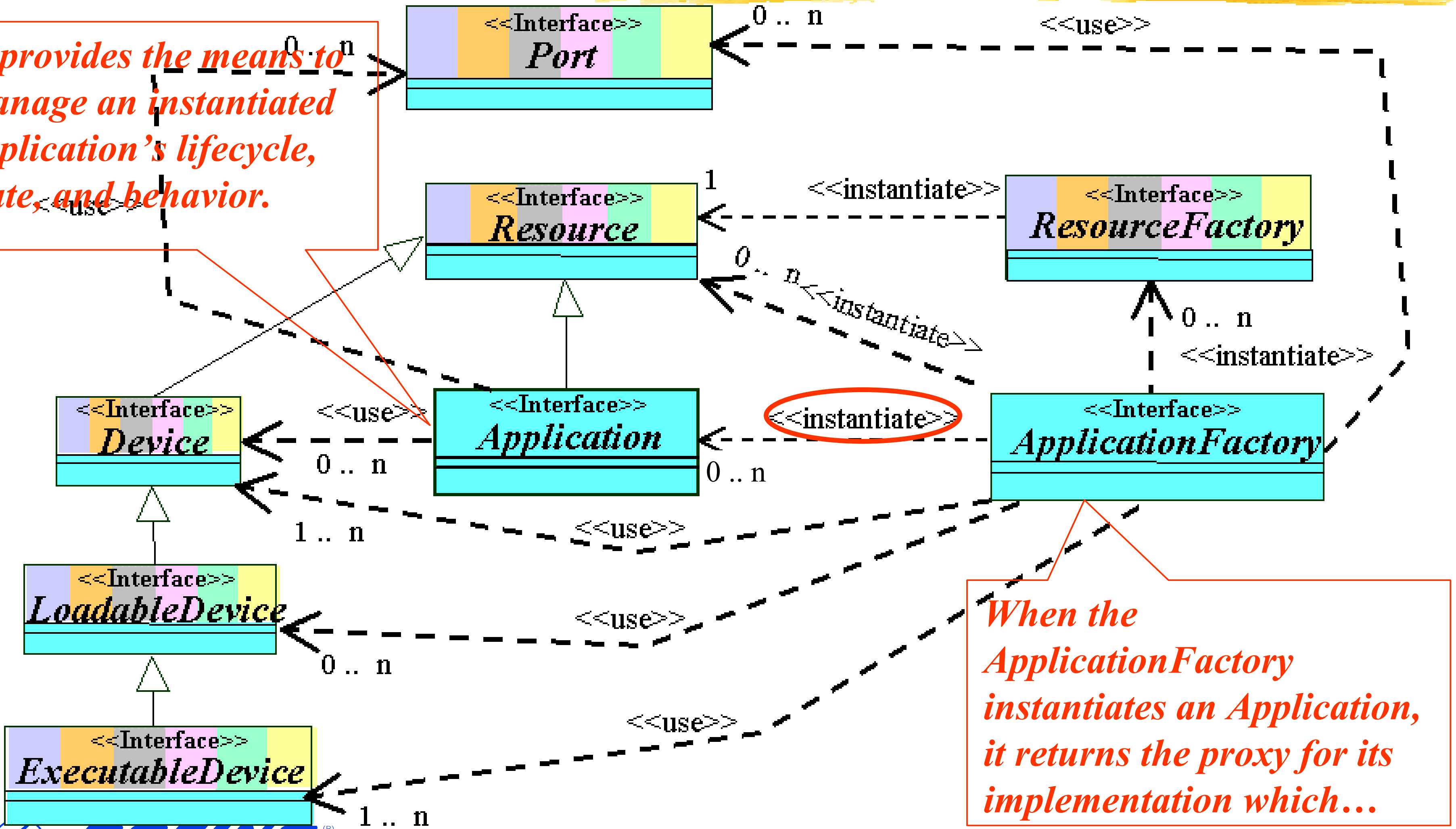
ApplicationFactory Interface Advantages

- The *ApplicationFactory* provides an interface to request the creation of a specific type of application in the domain.
  - This abstraction allows a user interface to manipulate a group of application types as objects.
- There is one *ApplicationFactory* object for each type of application and one implementation for all *ApplicationFactory* objects.
  - Each *ApplicationFactory* object has its own instance variables based upon the same servant class definition and therefore is its own unique instance.
- The *ApplicationFactory* interface provides a standard interface for creating applications within the domain, and one common implementation within the domain for creating applications.
  - Each application developer doesn't have to develop code to parse their own software profiles and retrieve domain profile (devices, etc.) to create their application within a domain.

CF Framework Control

Application Configuration, Control and Termination Interface

CF Application Instantiation Interfaces



<<Interface>> Port
<<Interface>> Resource
<<Interface>> ResourceFactory
<<Interface>> Application
<<Interface>> ApplicationFactory
<<Interface>> Device
<<Interface>> LoadableDevice
<<Interface>> ExecutableDevice
<<use>>
<<instantiate>>
0 .. n
1
1 .. n

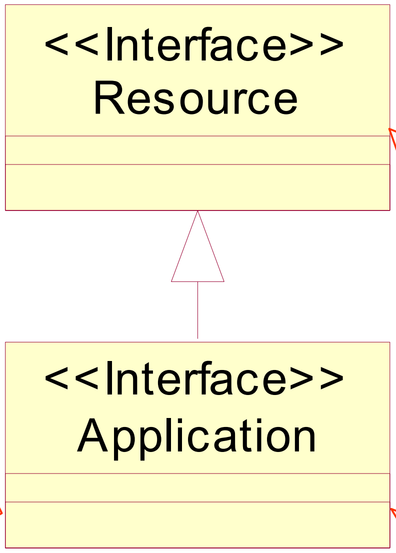
…provides the means to manage an instantiated application's lifecycle, state, and behavior.

When the ApplicationFactory instantiates an Application, it returns the proxy for its implementation which…

Application Interface – A Specialized Resource

From an end-user perspective, software applications are the primary functional element of a system.

An Application has characteristics very similar to a Resource. For example, an application has properties and communicates through ports.



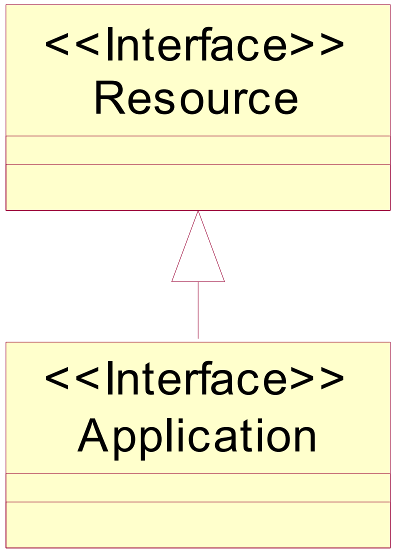
Since users need to control the lifespan of applications, configure them, and control their behavior, the Application interface is derived from the Resource interface, so that it can provides the necessary operations for managing application lifecycle, state, and behavior.

The implementation of an Application is comprised of a set of Resource component implementations and their interconnections.

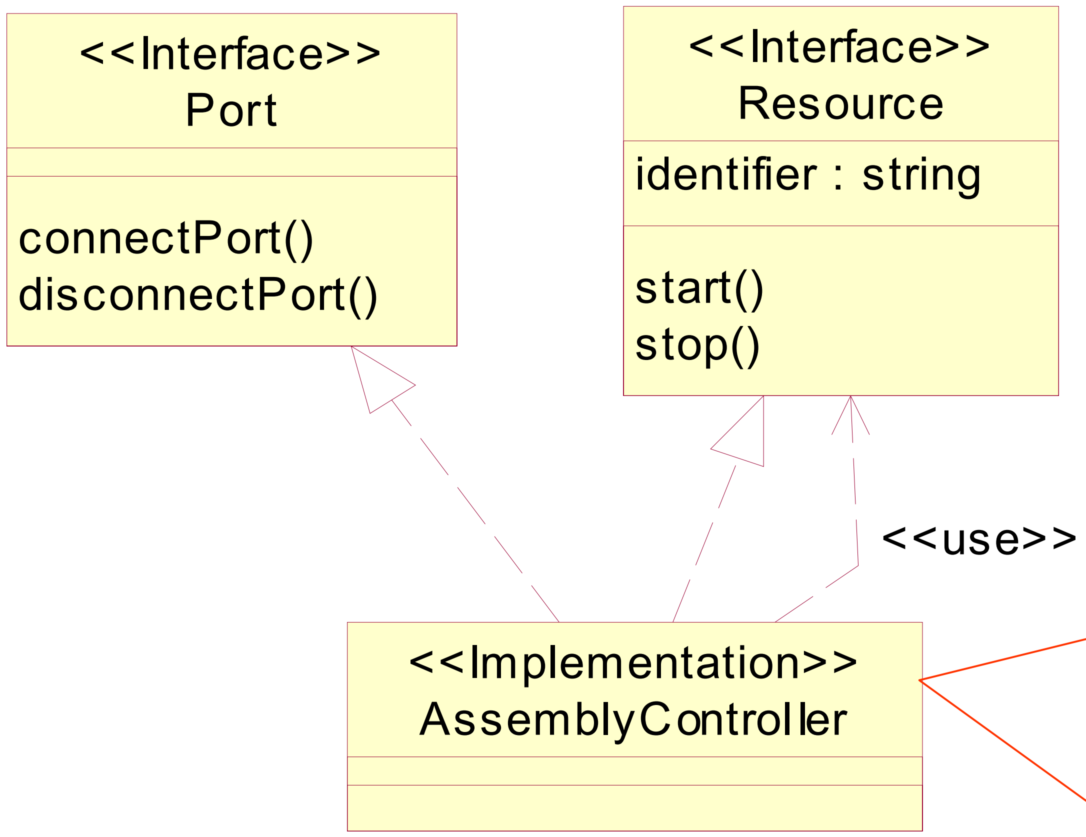
Application Interface – A Proxy for an Instantiated SW Assembly

The Application delegates its inherited Resource operations to a Resource component that has been identified as the "assembly controller for the software assembly".

Since the Application interface is derived from Resource and an assembly controller is also a Resource, this makes the delegation very simple and elegant.

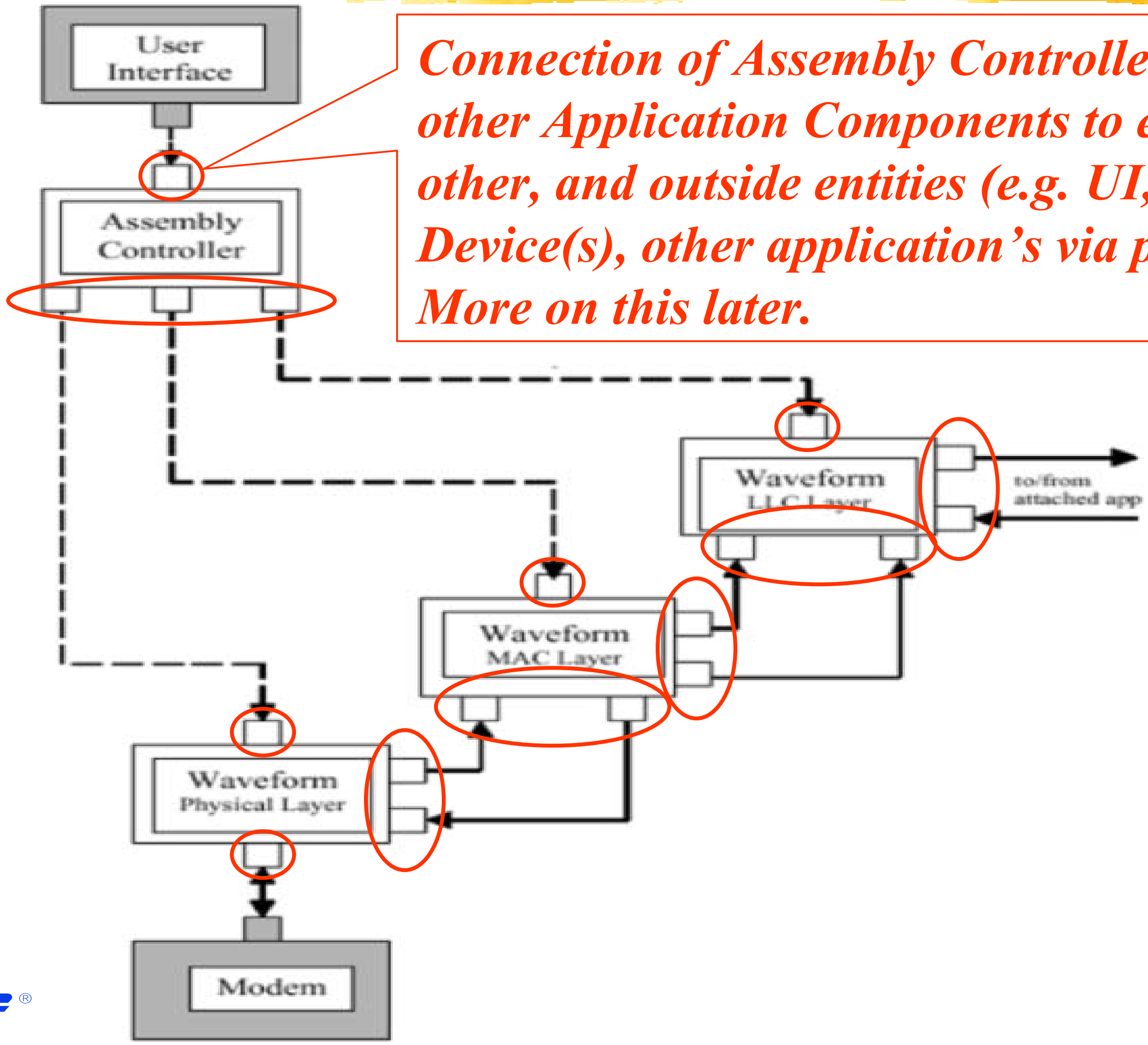


What is an Assembly Controller?



- *Acts like the POC for the CF to communicate with the Application.*
- *All applications are required to have one component that acts like an assembly controller.*
- *Identified in the Application's Software Assembly Descriptor file in the Domain Profile.*

Assembly Controller Example



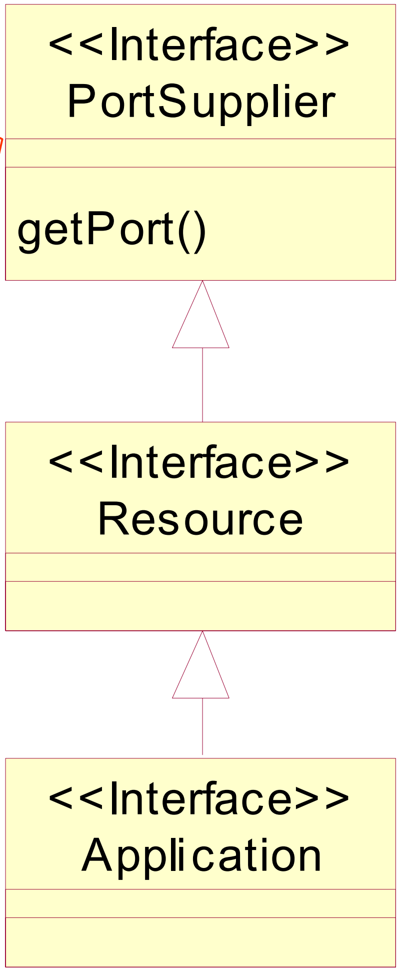
Connection of Assembly Controller and other Application Components to each other, and outside entities (e.g. UI, Device(s), other application's via ports. More on this later.

Providing Access to an Application's Ports

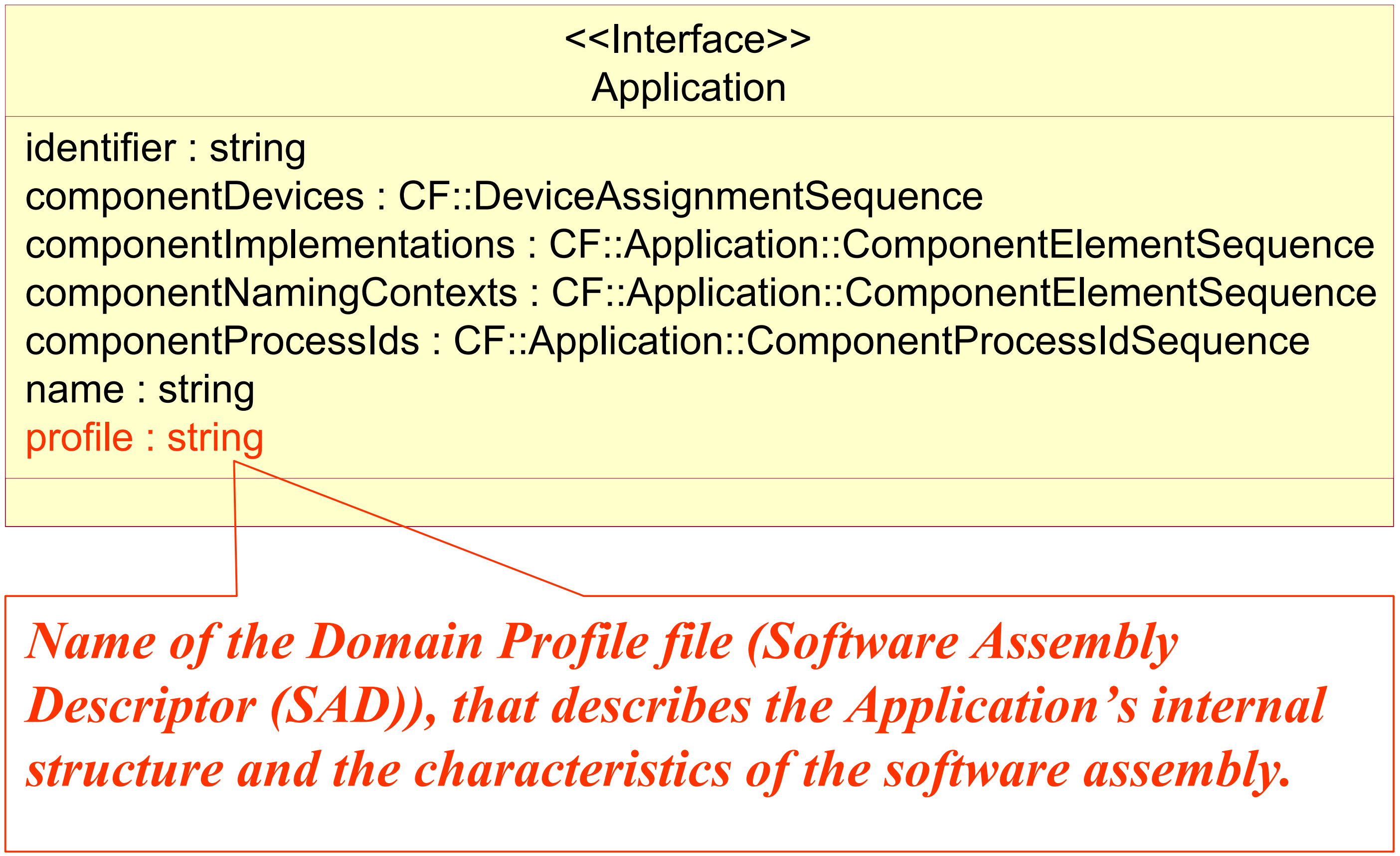
The Application's ports provide the capability of connecting the Application up to other components such as an Application.

The ports can also be used to provide additional command/control and/or status interfaces for an Application.

The ports for an Application are flexible allowing developers to determine which of their Application's components' ports should be made visible via the Application's software profile definition.



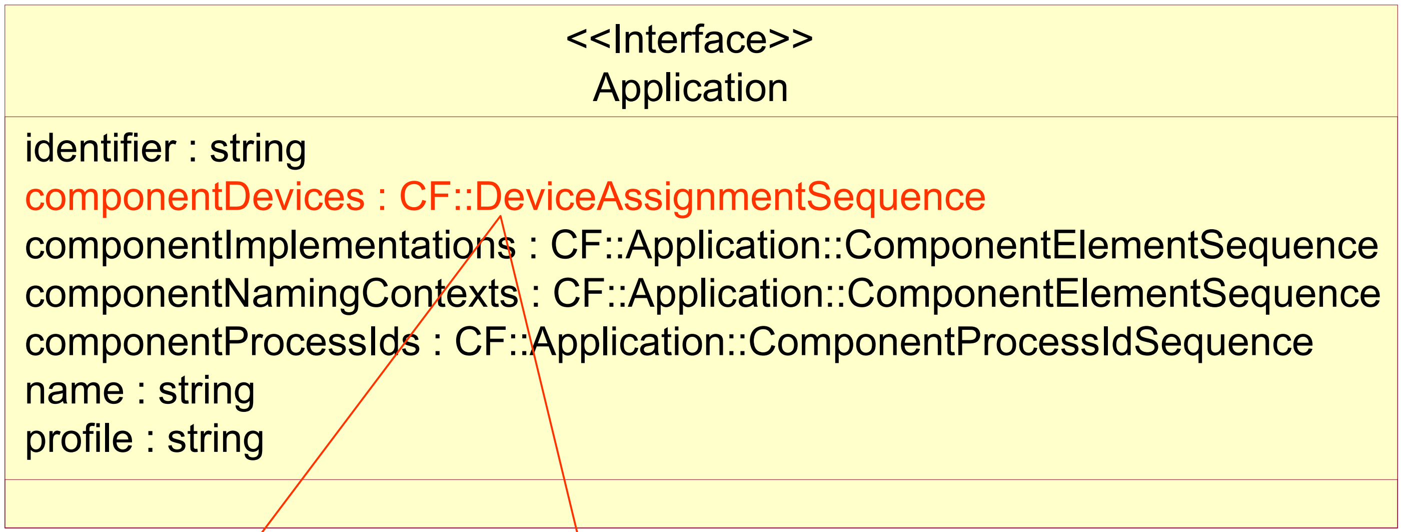
Application Interface



<<Interface>>
Application
identifier : string
componentDevices : CF::DeviceAssignmentSequence
componentImplementations : CF::Application::ComponentElementSequence
componentNamingContexts : CF::Application::ComponentElementSequence
componentProcessIds : CF::Application::ComponentProcessIdSequence
name : string
profile : string

Name of the Domain Profile file (Software Assembly Descriptor (SAD)), that describes the Application's internal structure and the characteristics of the software assembly.

Application Interface (cont.)



<<Interface>>
Application
identifier : string
componentDevices : CF::DeviceAssignmentSequence
componentImplementations : CF::Application::ComponentElementSequence
componentNamingContexts : CF::Application::ComponentElementSequence
componentProcessIds : CF::Application::ComponentProcessIdSequence
name : string
profile : string

List of object references of all Logical Device(s) on which Application's component's are loaded and/or are running. Used by Application for unloading and terminating component modules and processes during Application tear-down.

Application Interface (cont.)

<<Interface>>
Application

identifier : string
componentDevices : CF::DeviceAssignmentSequence
componentImplementations : CF::Application::ComponentElementSequence
componentNamingContexts : CF::Application::ComponentElementSequence
componentProcessIds : CF::Application::ComponentProcessIdSequence
name : string
profile : string

List of identifiers for chosen (running) component implementations from the Application's software profile for each created component . More on this later during the Domain Profile lecture.

Application Interface (cont.)

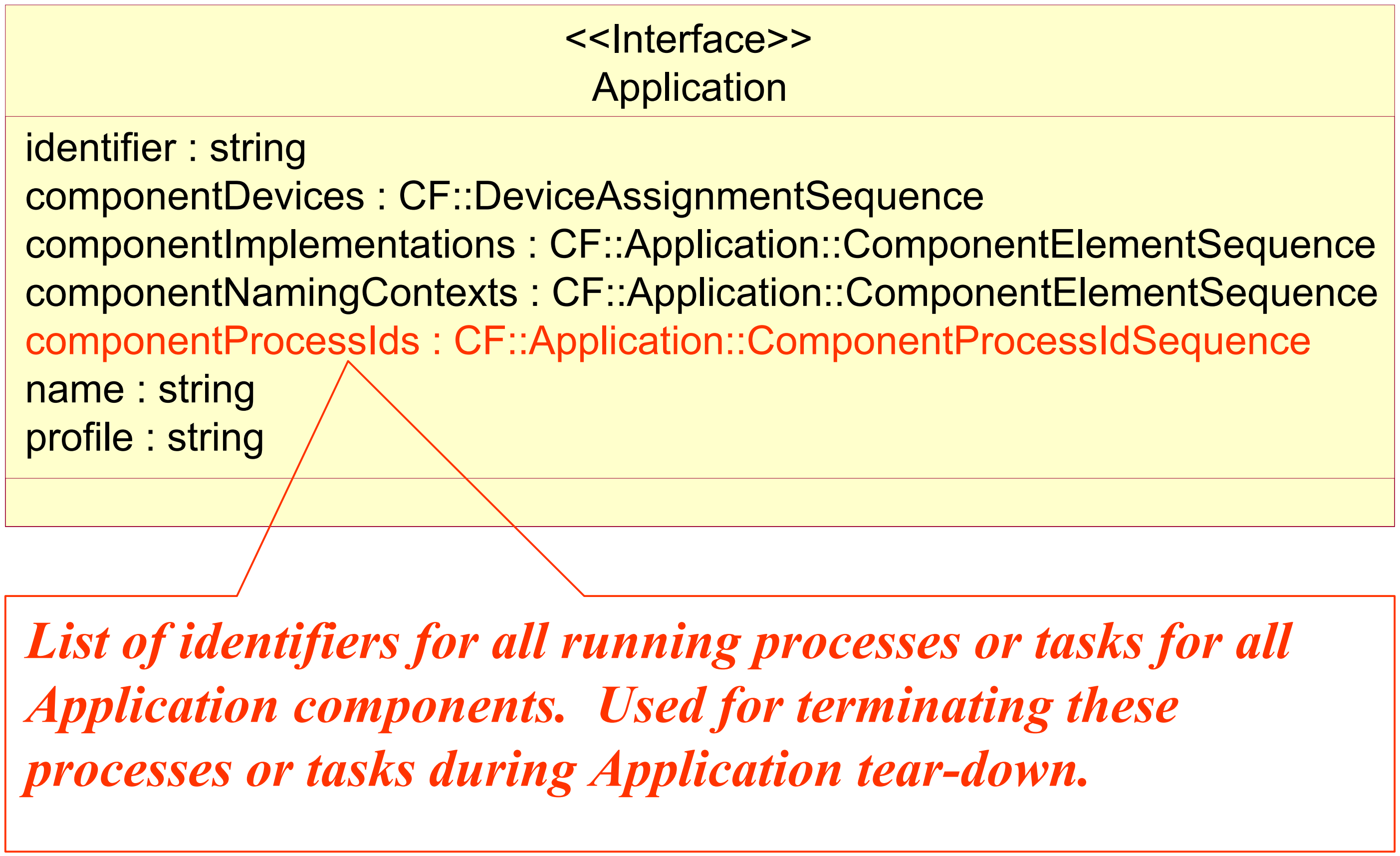
<<Interface>>
Application

identifier : string
componentDevices : CF::DeviceAssignmentSequence
componentImplementations : CF::Application::ComponentElementSequence
componentNamingContexts : CF::Application::ComponentElementSequence
componentProcessIds : CF::Application::ComponentProcessIdSequence
name : string
profile : string

List of all Naming Service Naming Contexts to which Application components are bound. When a component is created, it binds itself to the Naming Service under a predefined naming context defined in the Application's software profile. During Application tear-down, the component's object ref along with this naming context are removed from the Naming Service.

BOEING®

Application Interface *(cont.)*



<<Interface>>
Application

identifier : string
componentDevices : CF::DeviceAssignmentSequence
componentImplementations : CF::Application::ComponentElementSequence
componentNamingContexts : CF::Application::ComponentElementSequence
componentProcessIds : CF::Application::ComponentProcessIdSequence
name : string
profile : string

List of identifiers for all running processes or tasks for all Application components. Used for terminating these processes or tasks during Application tear-down.

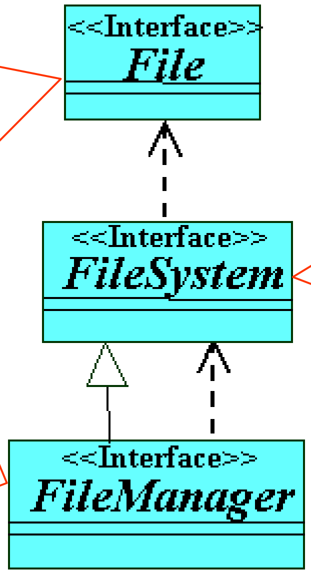
Application Interface Advantages

- The *Application* interface provides a standard interface for all applications within the domain, and one common implementation within the domain for all applications.
- Application developers don't have to develop code to tear down their application and to behave according to the *Application's* software profile (SAD).
- Having a generic *Application* abstraction supports general purpose user interface components for controlling a system.

CF File Service Interfaces

CF File Services Interfaces

Provides the ability to read and write files within a SCA-compliant distributed file system.



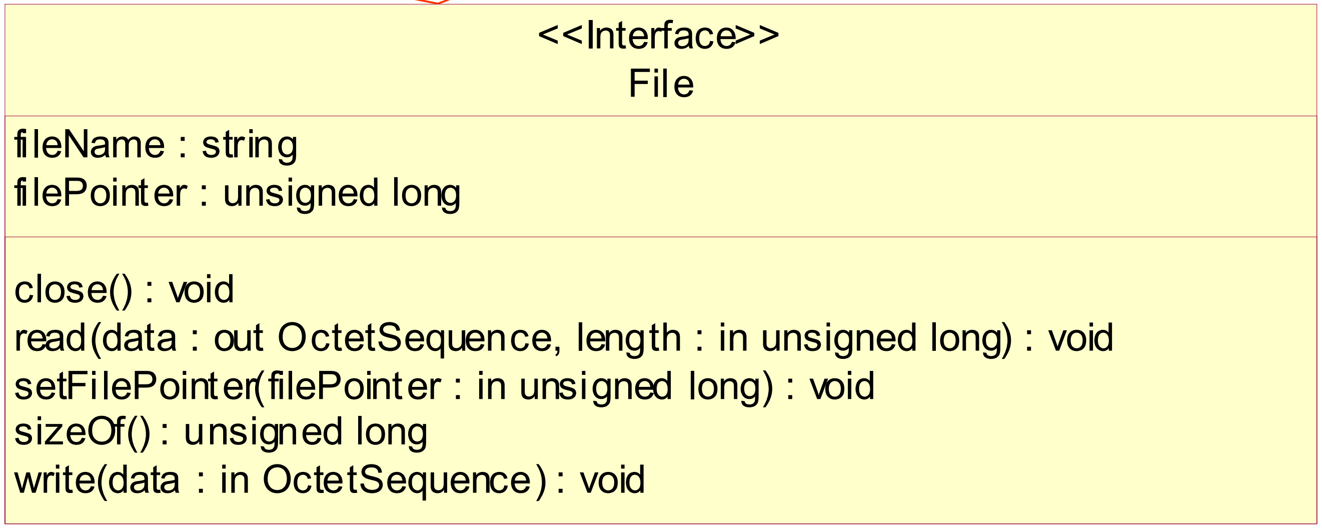
Defines CORBA operations that enable remote access to a physical file system.

Appearing as a single FileSystem, provides access to multiple distributed file systems, whose file storage may span multiple physical file systems (AKA a federated file system).

Used by CF clients for all file accesses.

File Interface

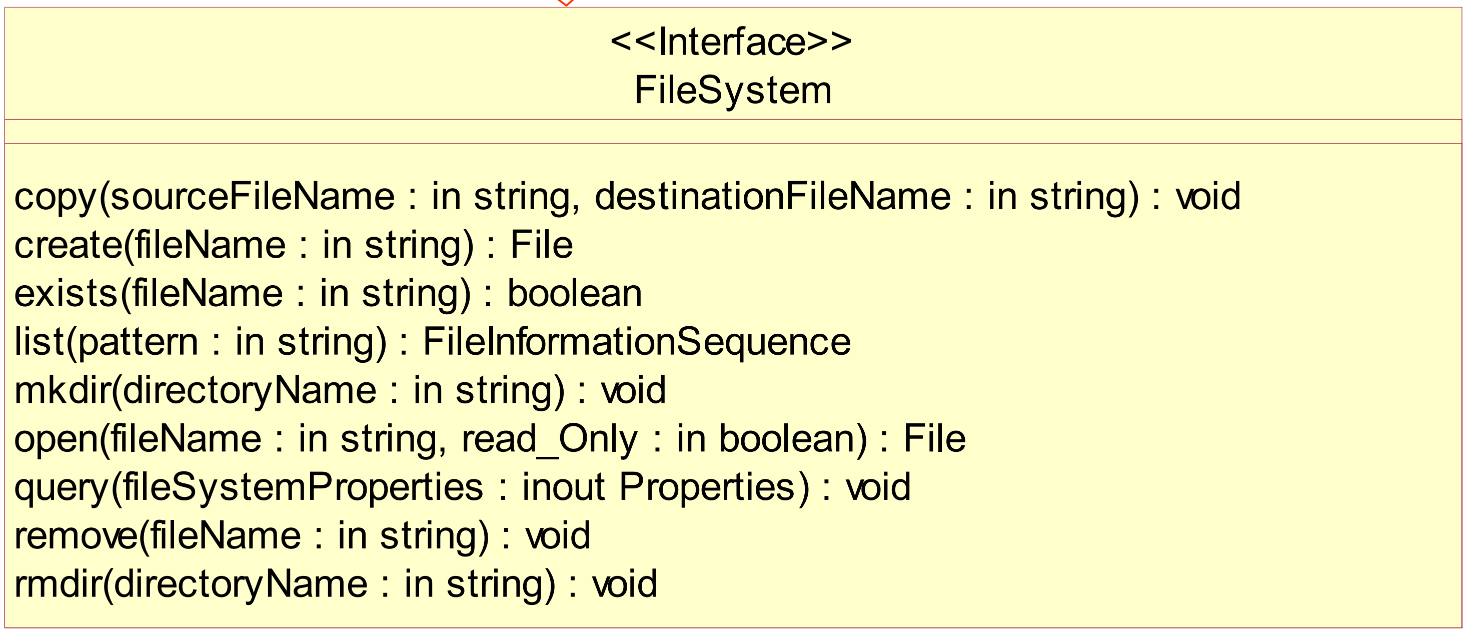
- *Provides access to files within the system*
- *Allows access across processor boundaries (distributed FileSystems)*



```
<<Interface>>
File
-----------------------------------------------------------------------
fileName : string
filePointer : unsigned long
-----------------------------------------------------------------------
close() : void
read(data : out OctetSequence, length : in unsigned long) : void
setFilePointer(filePointer : in unsigned long) : void
sizeOf() : unsigned long
write(data : in OctetSequence) : void
```

FileSystem Interface

Allows creation, deletion, copying of files



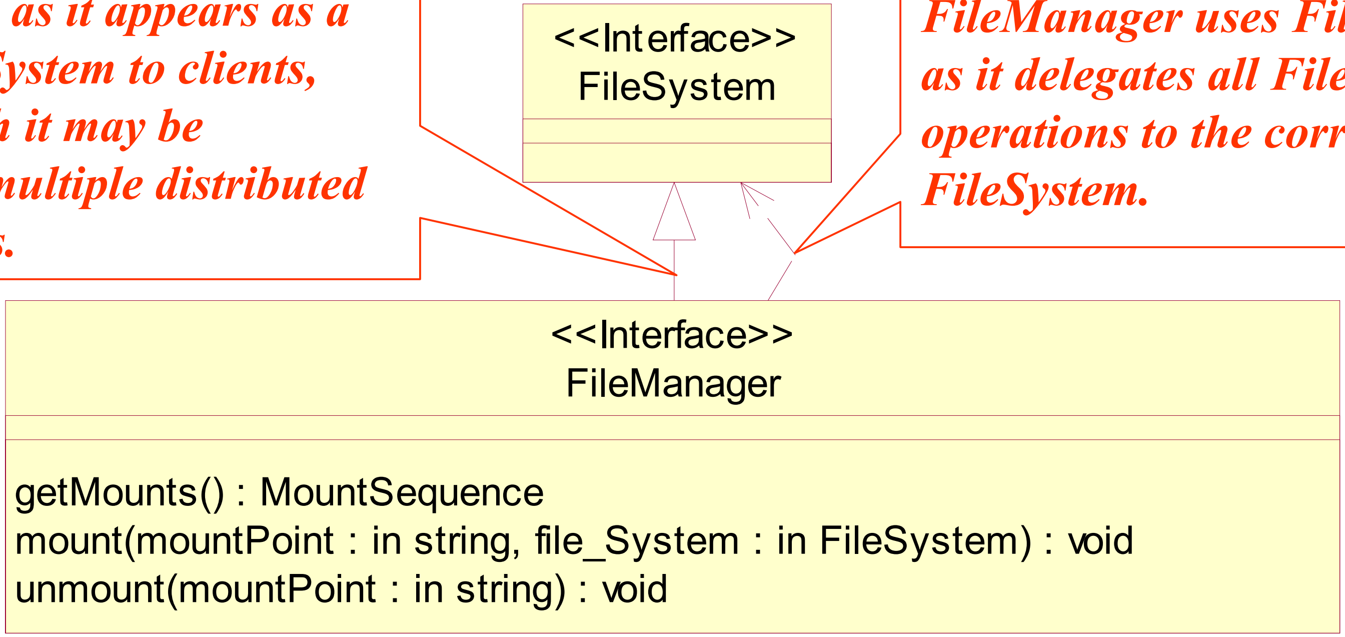
```
<<Interface>>
FileSystem

copy(sourceFileName : in string, destinationFileName : in string) : void
create(fileName : in string) : File
exists(fileName : in string) : boolean
list(pattern : in string) : FileInformationSequence
mkdir(directoryName : in string) : void
open(fileName : in string, read_Only : in boolean) : File
query(fileSystemProperties : inout Properties) : void
remove(fileName : in string) : void
rmdir(directoryName : in string) : void
```

FileManager Interface

FileManager inherits from FileSystem, as it appears as a single FileSystem to clients, even though it may be managing multiple distributed FileSystems.

FileManager uses FileSystem, as it delegates all FileSystem operations to the correct FileSystem.



<<Interface>>
FileSystem
<<Interface>>
FileManager
getMounts() : MountSequence
mount(mountPoint : in string, file_System : in FileSystem) : void
unmount(mountPoint : in string) : void

CF Domain Profile

Domain Profile

A set of files that express the packaging and deployment of SCA-complaint components.

- Specific characteristics of software components or hardware devices.
  - Interfaces, functional capabilities, logical location, interconnections, inter-dependencies, and other pertinent properties and parameters.
    - e.g. Description of applications, startup requirements of devices, etc.

Defined in XML and Based on XML DTDs

Defined in eXtensible Markup Language (XML) format.

Based upon SCA-defined XML Document Type Definitions (DTDs).

- A DTD defines the meaning and compliant syntax for a XML file.

BOEING®

Based on OMG CORBA Components Specification

- Based upon the OMG's draft CORBA Components specification (orbos/99-07-01).
- Customized from the OMG CORBA Component Specification.
  - Addition, or customization of existing elements to better address the needs of an SCA-compliant system.
  - Elimination of unneeded elements in order to reduce the associated XML parsing and aid in XML file content validation.
  - Advantageous, as it builds upon work already completed by OMG and does not duplicate effort.

Domain Profile Files

- Software Package Descriptor (SPD)
  - Describes a component's (CORBA and non-CORBA) implementation(s).
- Property File (PRF)
  - Describes properties for a component.
- Software Component Descriptor (SCD)
  - Describes a CORBA component's characteristics.
- Software Assembly Descriptor (SAD)
  - Describes an application's deployment characteristics.
- Device Configuration Descriptor (DCD)
  - Describes configuration characteristics for a *DeviceManager*.

Domain Profile Files (cont.)

DomainManager Configuration Descriptor (DMD)

- Describes configuration characteristics for a *DomainManager*.

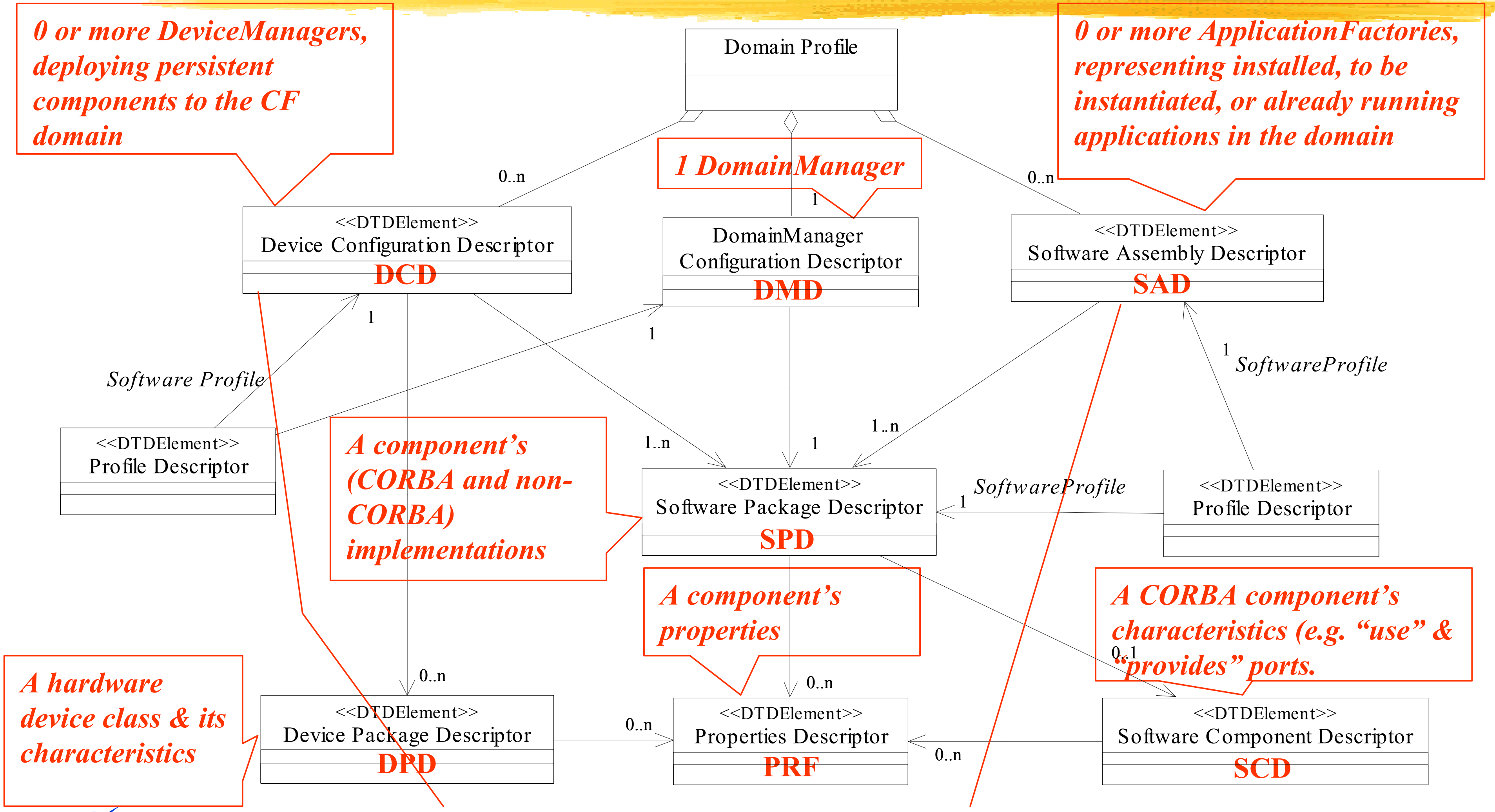
Device Package Descriptor (DPD)

- Identifies a class of hardware device and its characteristics.

Profile Descriptor

- Describes a type of file (SAD, SPD, DCD, DMD) along with the file name.

Domain Profile XML DTD Relationships



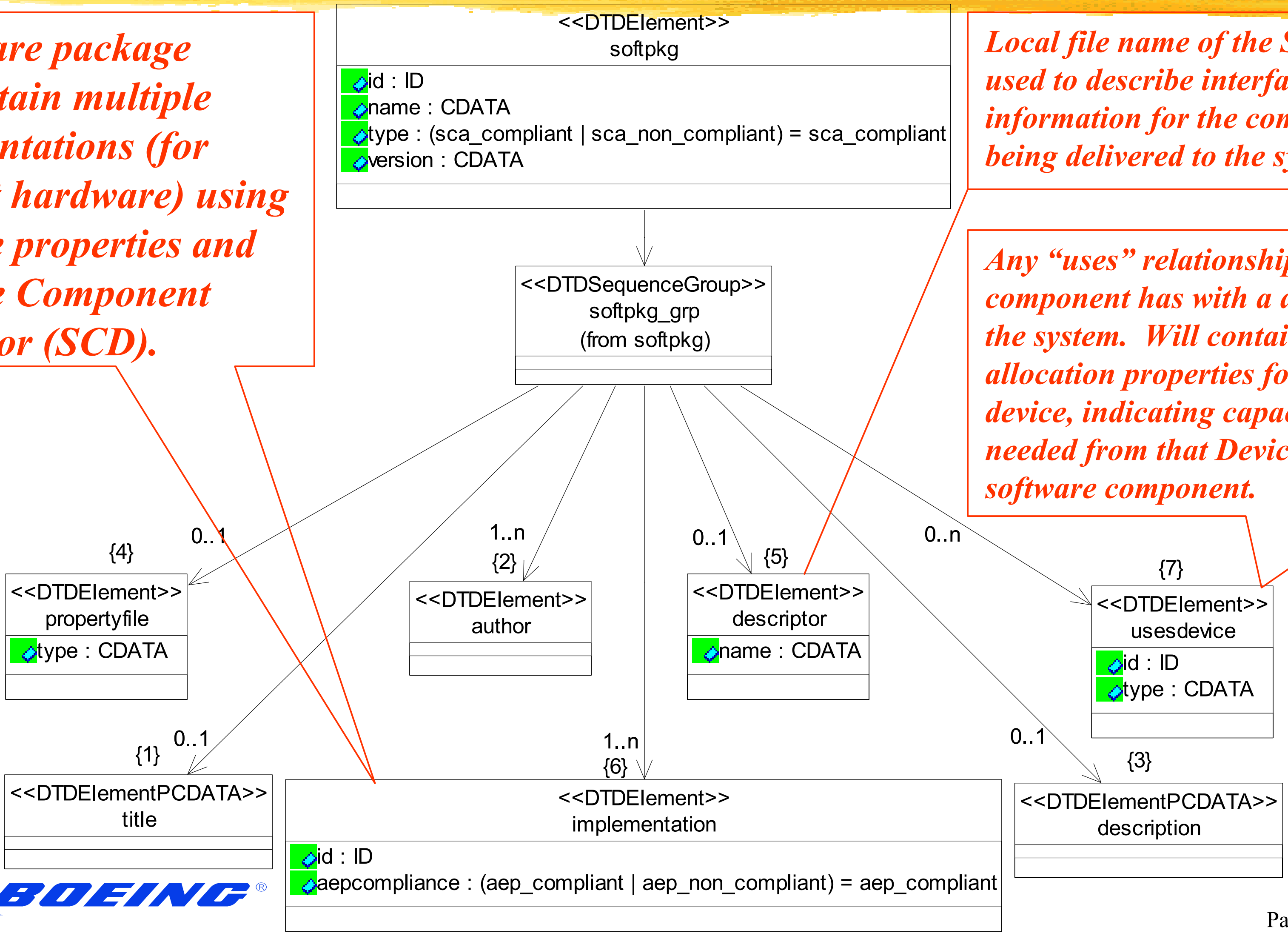
0 or more DeviceManagers, deploying persistent components to the CF domain
Domain Profile
1 DomainManager
0 or more ApplicationFactories, representing installed, to be instantiated, or already running applications in the domain
0..n
1
0..n
<<DTDElement>> Device Configuration Descriptor DCD
DomainManager Configuration Descriptor DMD
<<DTDElement>> Software Assembly Descriptor SAD
1
1
Software Profile
1
SoftwareProfile
<<DTDElement>> Profile Descriptor
A component's (CORBA and non-CORBA) implementations
1..n
1
1..n
<<DTDElement>> Software Package Descriptor SPD
SoftwareProfile
1
<<DTDElement>> Profile Descriptor
A component's properties
A CORBA component's characteristics (e.g. "use" & "provides" ports.
0..1
A hardware device class & its characteristics
0..n
0..n
<<DTDElement>> Device Package Descriptor DPD
0..n
<<DTDElement>> Properties Descriptor PRF
0..n
<<DTDElement>> Software Component Descriptor SCD

Both DCD and SAD are assembly types that deploy component implementations.

Software Package Descriptor (SPD)

- Based on CORBA Components Software Package Descriptor.
- Used to describe any software component.
  - CORBA or non-CORBA.
- An application’s software profile (Software Assembly Descriptor), contains one SPD for each software component of that application.

SPD Major Elements



<<DTDElement>> softpkg
id : ID
name : CDATA
type : (sca_compliant | sca_non_compliant) = sca_compliant
version : CDATA
<<DTDSequenceGroup>> softpkg_grp (from softpkg)
0..1 {4}
<<DTDElement>> propertyfile
type : CDATA
1..n {2}
<<DTDElement>> author
0..1 {5}
<<DTDElement>> descriptor
name : CDATA
0..n {7}
<<DTDElement>> usesdevice
id : ID
type : CDATA
0..1 {1}
<<DTDElementPCDATA>> title
1..n {6}
<<DTDElement>> implementation
id : ID
aepcompliance : (aep_compliant | aep_non_compliant) = aep_compliant
0..1 {3}
<<DTDElementPCDATA>> description

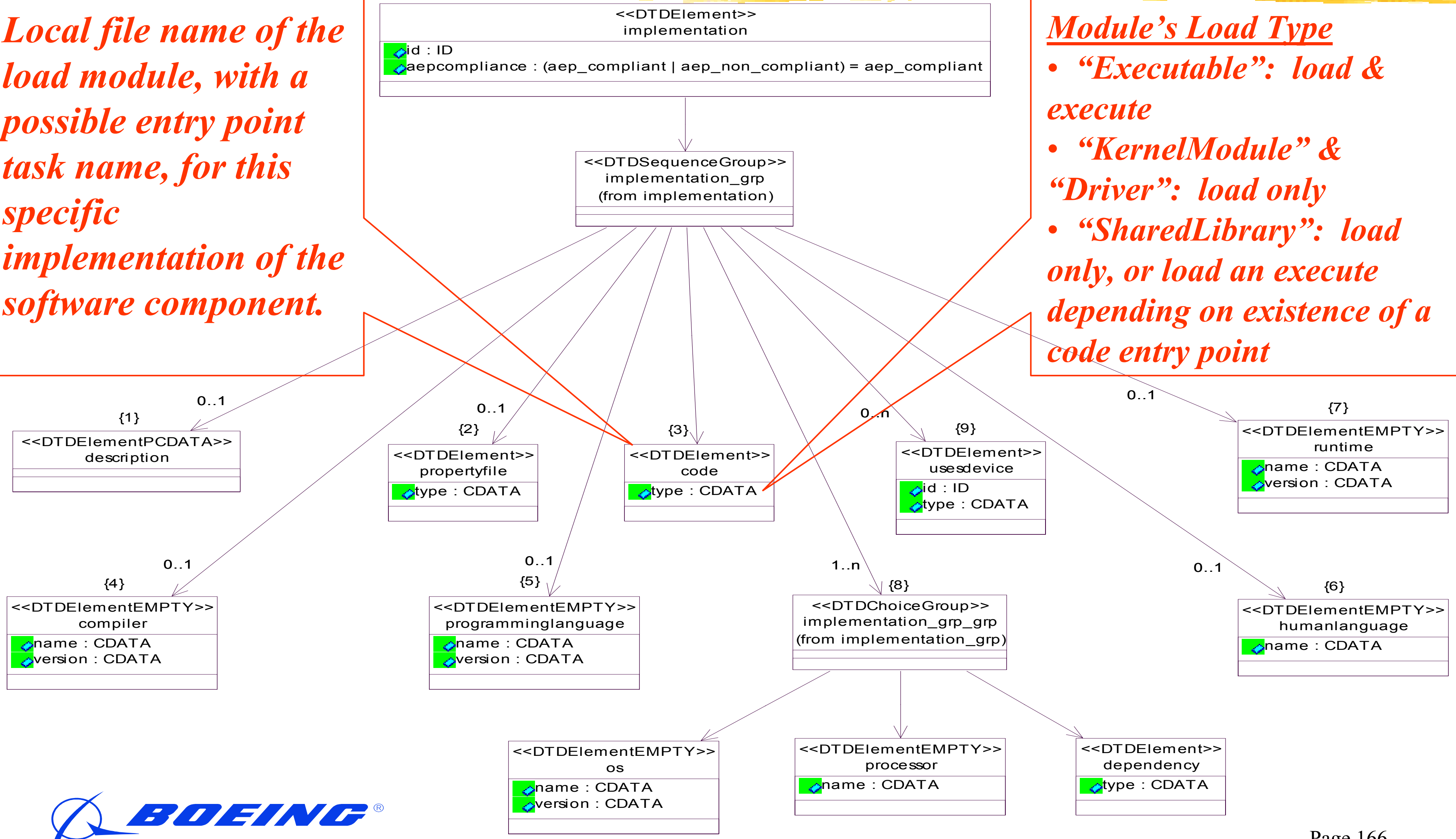
A Software package may contain multiple implementations (for different hardware) using the same properties and Software Component Descriptor (SCD).

Local file name of the SCD file used to describe interface information for the component being delivered to the system.

Any "uses" relationship this component has with a device in the system. Will contain all allocation properties for that device, indicating capacities needed from that Device by the software component.

SPD implementation Element

Local file name of the load module, with a possible entry point task name, for this specific implementation of the software component.



<<DTDElement>> implementation
id : ID
aepcompliance : (aep_compliant | aep_non_compliant) = aep_compliant
<<DTDSequenceGroup>> implementation_grp (from implementation)
0..1
{1}
<<DTDElementPCDATA>> description
0..1
{2}
<<DTDElement>> propertyfile
type : CDATA
{3}
<<DTDElement>> code
type : CDATA
0..n
{9}
<<DTDElement>> usesdevice
id : ID
type : CDATA
0..1
{7}
<<DTDElementEMPTY>> runtime
name : CDATA
version : CDATA
0..1
{4}
<<DTDElementEMPTY>> compiler
name : CDATA
version : CDATA
0..1
{5}
<<DTDElementEMPTY>> programminglanguage
name : CDATA
version : CDATA
1..n
{8}
<<DTDChoiceGroup>> implementation_grp_grp (from implementation_grp)
0..1
{6}
<<DTDElementEMPTY>> humanlanguage
name : CDATA
<<DTDElementEMPTY>> os
name : CDATA
version : CDATA
<<DTDElementEMPTY>> processor
name : CDATA
<<DTDElement>> dependency
type : CDATA

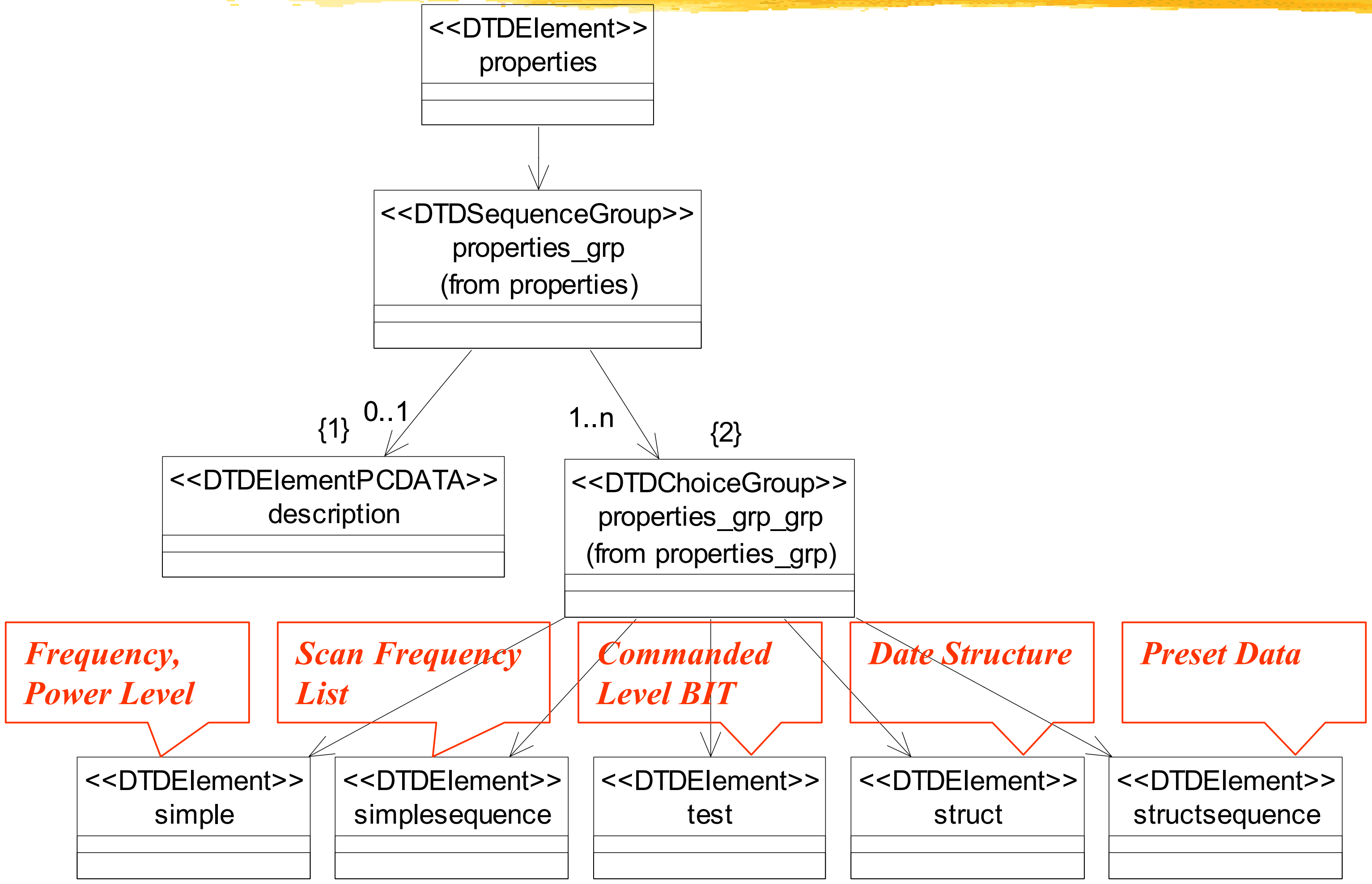
Module's Load Type

- *"Executable": load & execute*
- *"KernelModule" & "Driver": load only*
- *"SharedLibrary": load only, or load an execute depending on existence of a code entry point*

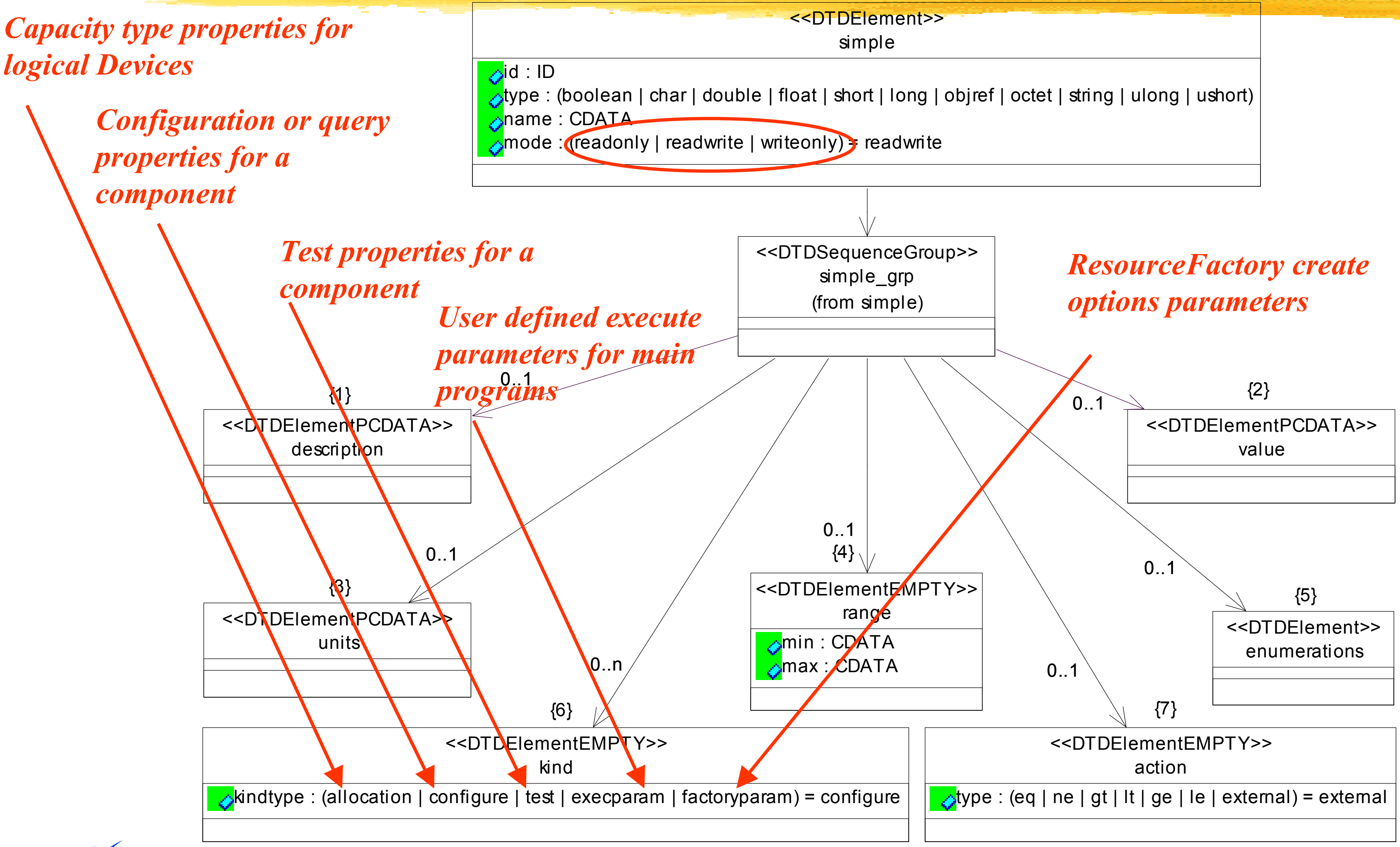
Properties Descriptor

- Derived from the CORBA Components *properties* element.
- Contains information about properties applicable to a software package or a device package.
- It is a sequence of 1 or more properties, where…
- Each property is of a particular property type and property kind.

Properties Descriptor
Property Types



Properties Descriptor
Property Kinds



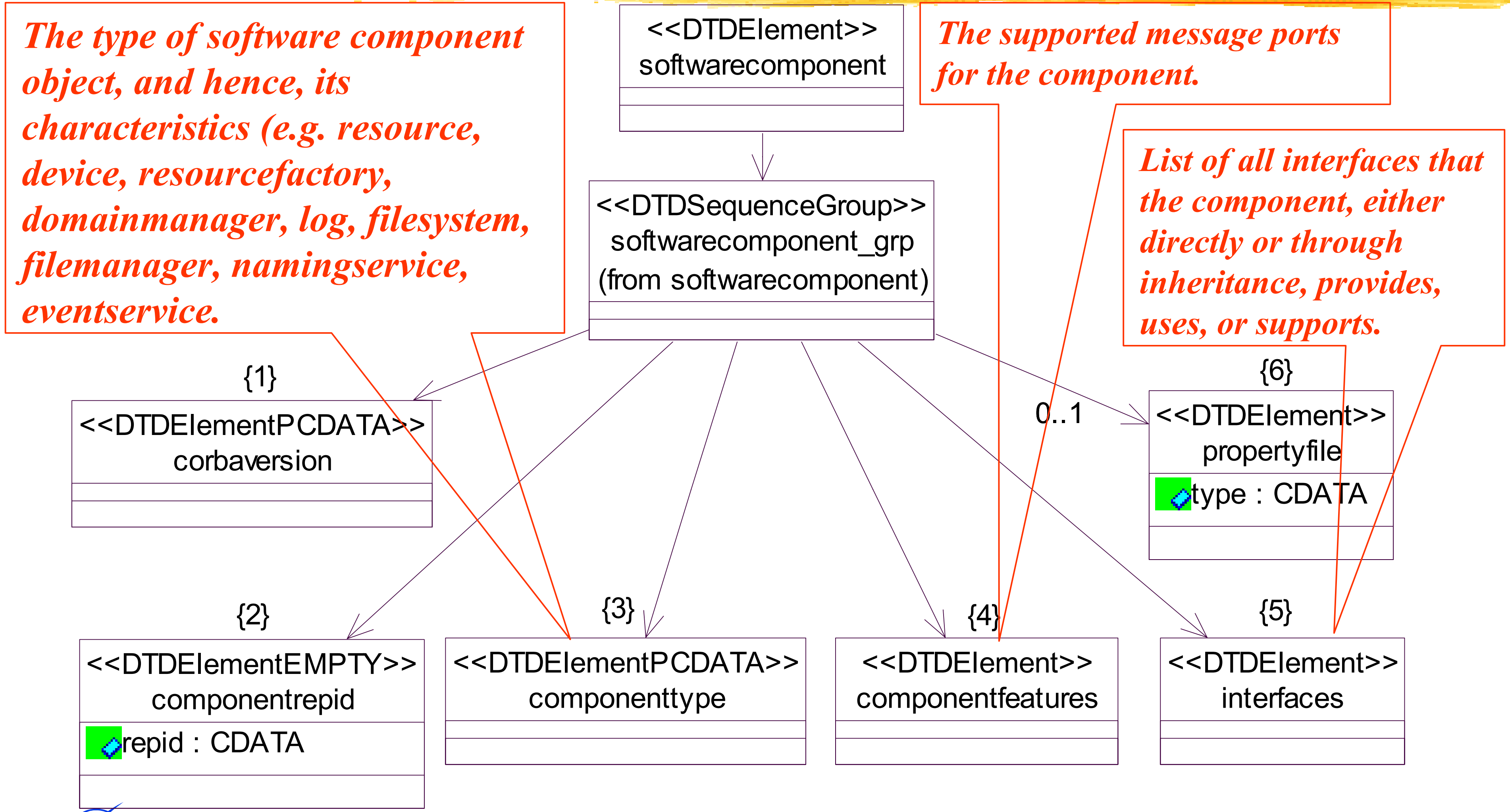
Capacity type properties for logical Devices
Configuration or query properties for a component
Test properties for a component
User defined execute parameters for main programs
ResourceFactory create options parameters
<<DTDElement>>
simple
id : ID
type : (boolean | char | double | float | short | long | objref | octet | string | ulong | ushort)
name : CDATA
mode : (readonly | readwrite | writeonly) = readwrite
<<DTDSequenceGroup>>
simple_grp
(from simple)
{1}
<<DTDElementPCDATA>>
description
0..1
{2}
<<DTDElementPCDATA>>
value
0..1
{3}
<<DTDElementPCDATA>>
units
0..1
{4}
<<DTDElementEMPTY>>
range
min : CDATA
max : CDATA
0..1
{5}
<<DTDElement>>
enumerations
0..1
{6}
<<DTDElementEMPTY>>
kind
kindtype : (allocation | configure | test | execparam | factoryparam) = configure
0..n
{7}
<<DTDElementEMPTY>>
action
type : (eq | ne | gt | lt | ge | le | external) = external
0..1

BOEING®

Software Component Descriptor (SCD)

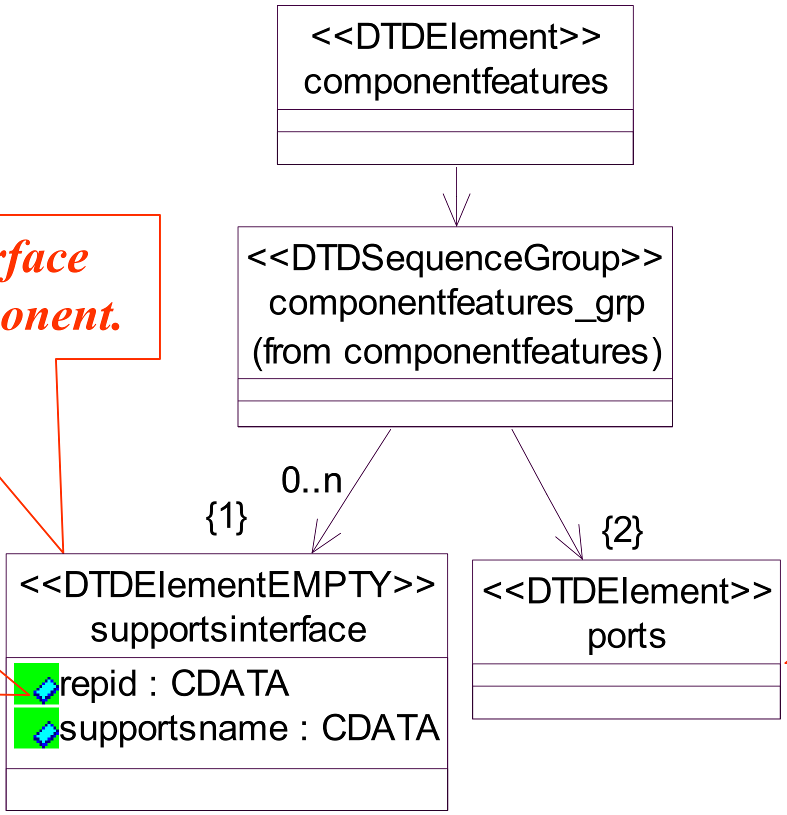
- Based on the CORBA Component Descriptor (CCD) file definition.
  - interfaces, uses port, provides port, supporting interfaces.
- Provides the means for a component to describe its interfaces in two ways.
  - Supporting interfaces which are IDL interfaces inherited by the component IDL interface.
  - Provides interfaces or “facets” - Distinct CORBA interfaces that a component implementation supports in addition to the component’s interface.

SCD Elements



The type of software component object, and hence, its characteristics (e.g. resource, device, resourcefactory, domainmanager, log, filesystem, filemanager, namingservice, eventservice.
<<DTDElement>> softwarecomponent
The supported message ports for the component.
<<DTDSequenceGroup>> softwarecomponent_grp (from softwarecomponent)
List of all interfaces that the component, either directly or through inheritance, provides, uses, or supports.
{1}
<<DTDElementPCDATA>> corbaversion
0..1
{6}
<<DTDElement>> propertyfile
type : CDATA
{2}
<<DTDElementEMPTY>> componentrepid
repid : CDATA
{3}
<<DTDElementPCDATA>> componenttype
{4}
<<DTDElement>> componentfeatures
{5}
<<DTDElement>> interfaces

SCD Supported Message Ports Element



Identifies an IDL interface supported by the component.

Refers to the supported interface in the SCD "interfaces" element.

Interfaces "provided" and/or "used" by the component.

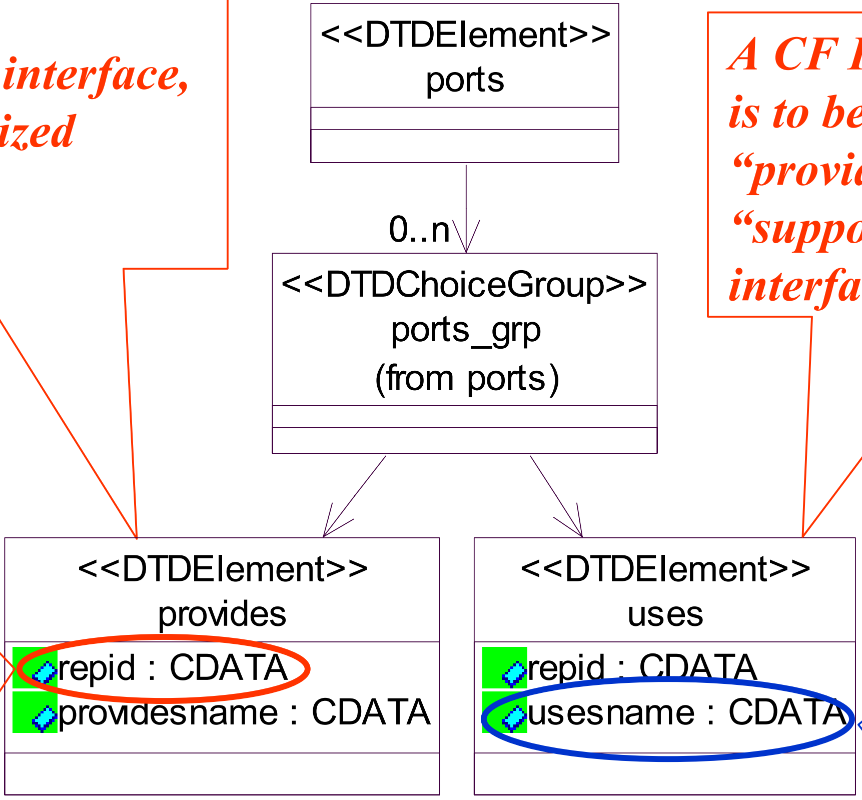
SCD Ports Element

A "Facet" – An independent interface from the component's main interface, providing a specialized connectivity to the component.

Each "uses" and "provides" port references its corresponding interface (i.e. the corresponding SCD interface element) by its id.

A CF Port interface type that is to be connected to a "provides" or "supportinterfaces" interface.

Each "uses" and "provides" port references its corresponding interface (i.e. the corresponding SCD interface element) by its id.



Device Package Descriptor (DPD)

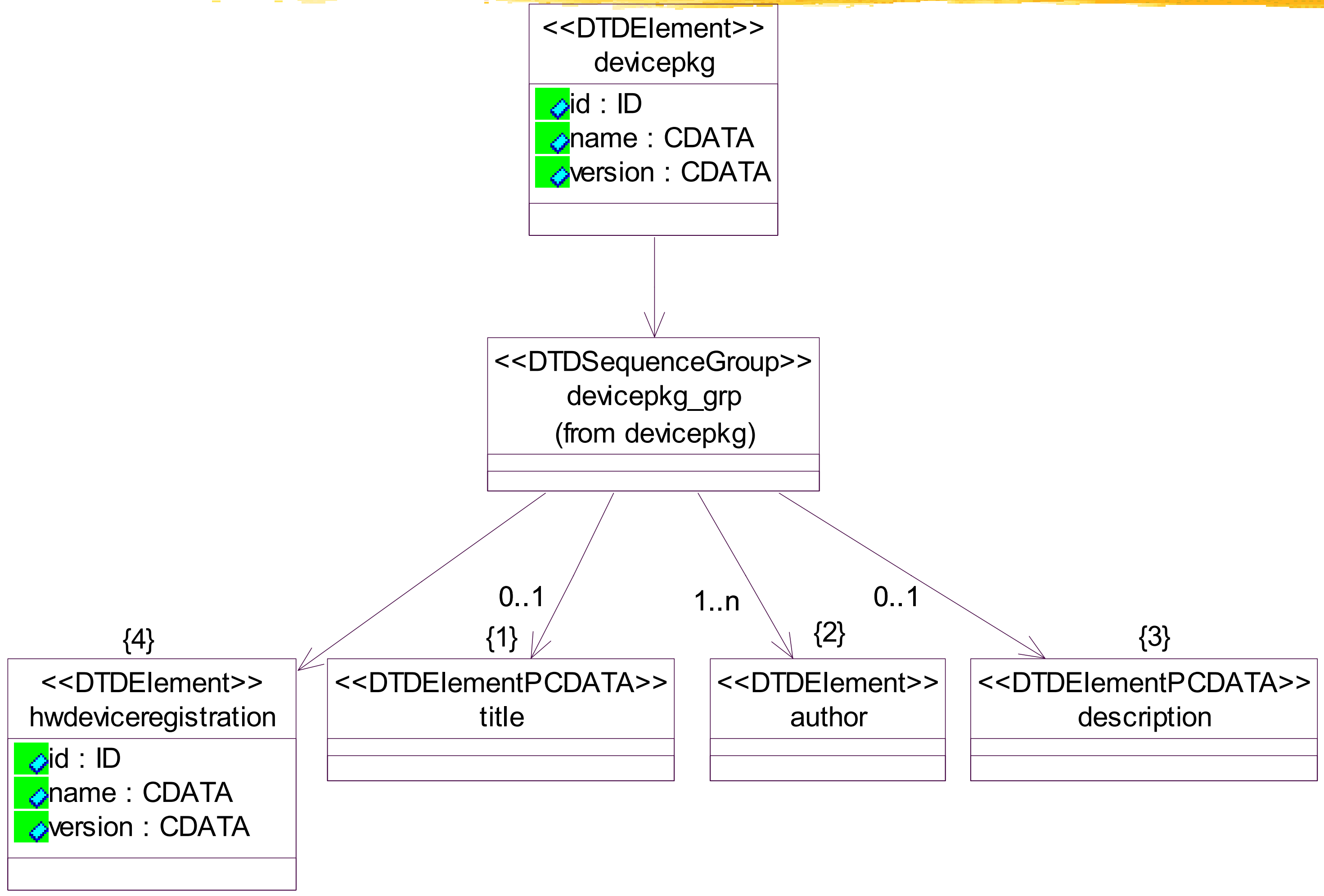
Not derived from the CORBA Component Specification.

Identifies a class of a hardware device along with its properties.

- Typically used by an HCI application to display information about the device(s) resident on an SCA-compliant system.

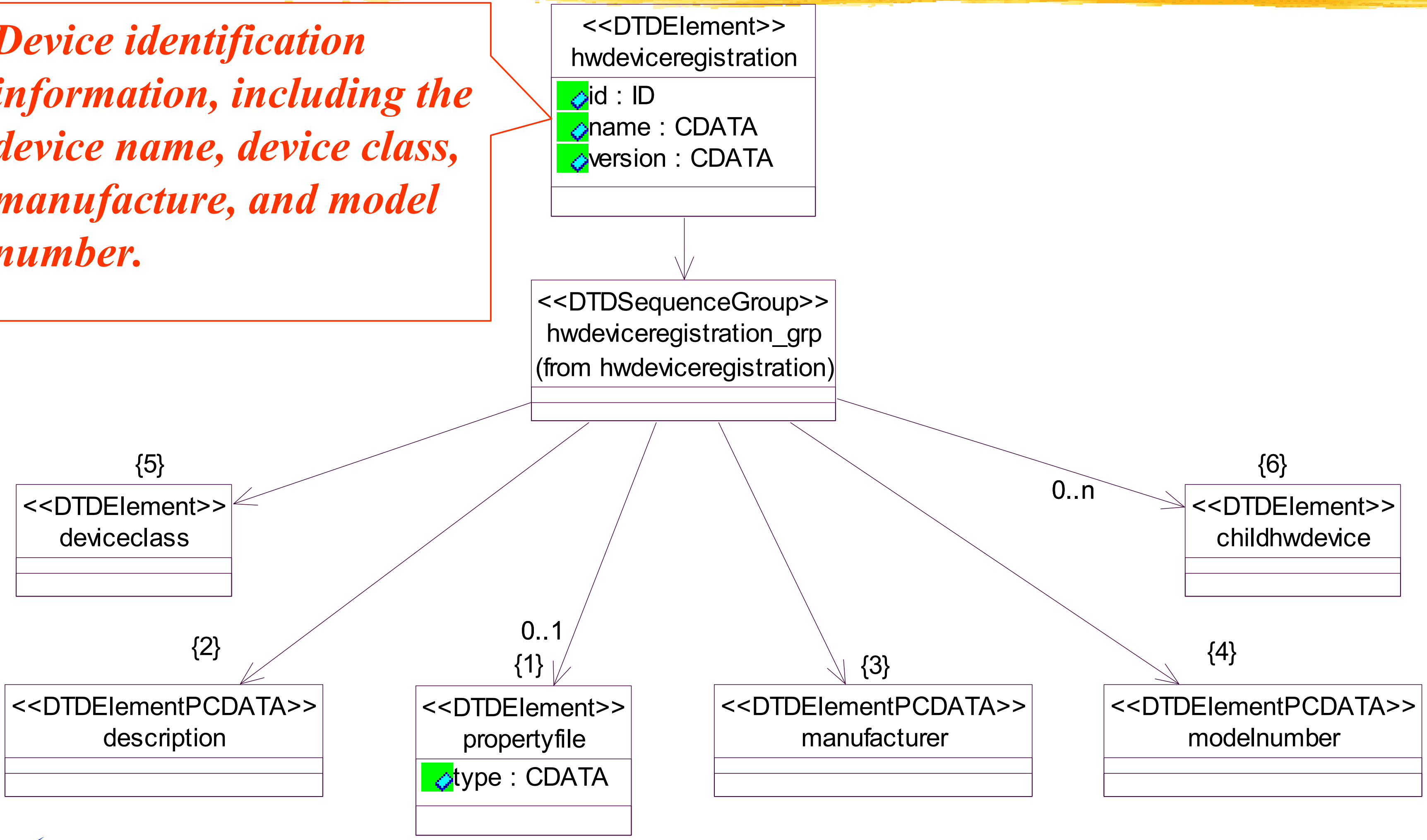
Based on the SCA Hardware Architecture definition.

DPD Elements



DPD Elements (cont.)

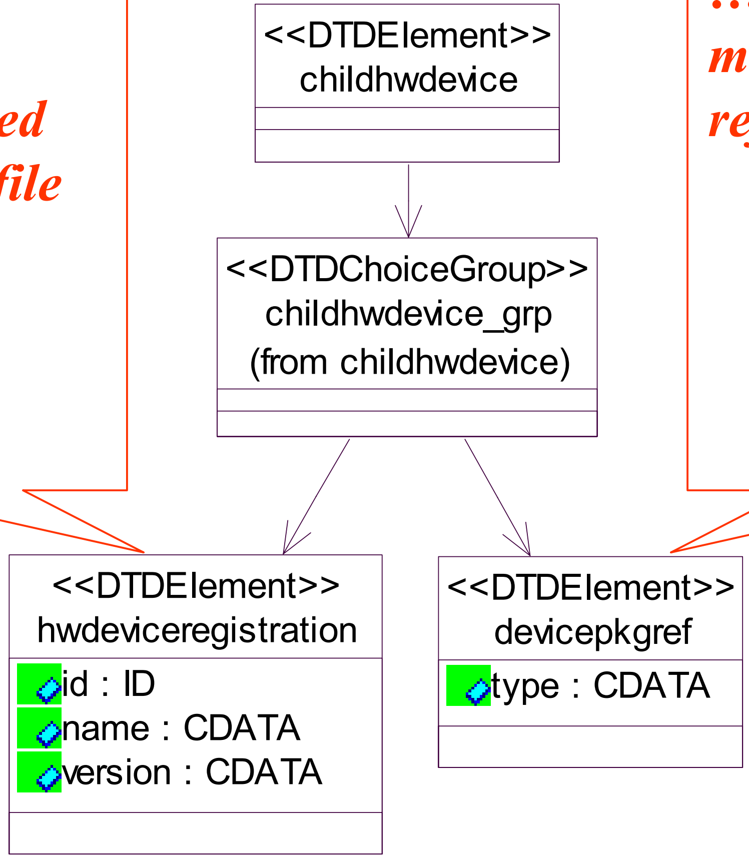
Device identification information, including the device name, device class, manufacture, and model number.



DPD Elements (cont.)

The DPD allows for composition device definitions to be formed up completely in one file …

… or in multiple files (DPD can reference other DPDs).



Software Assembly Descriptor (SAD)

- Derived from the CORBA Component Assembly Descriptor (CAD) file definition.
- Describes what makes up an application, how to create the application's pieces, configure them, and interconnect them.
- Processed by *CF::ApplicationFactory* during application instantiation.

BOEING®

SAD Contents

Based on the CORBA Components Assembly Descriptor concepts of component files, partitioning, and connections, it describes:

- What component instances makeup an assembly.
- Which components are located together.
- How to find components (e.g., naming service).
- How to connect components together.

SAD Contents *(cont.)*

- The required identification of a component as the main assembly controller.
  - Simplifies the behavior of the CF *Application* that acts a proxy for the assembly controller.
  - By having a mandated assembly controller, the *Application* proxy simply delegates its *Resource* operations to the assembly controller *Resource*.
- The optional visible ports for a software assembly.
  - Allows the assembly to be treated like a component that can be connected to for its data ports, or additional status and command/control ports.

SAD Top-Level Elements

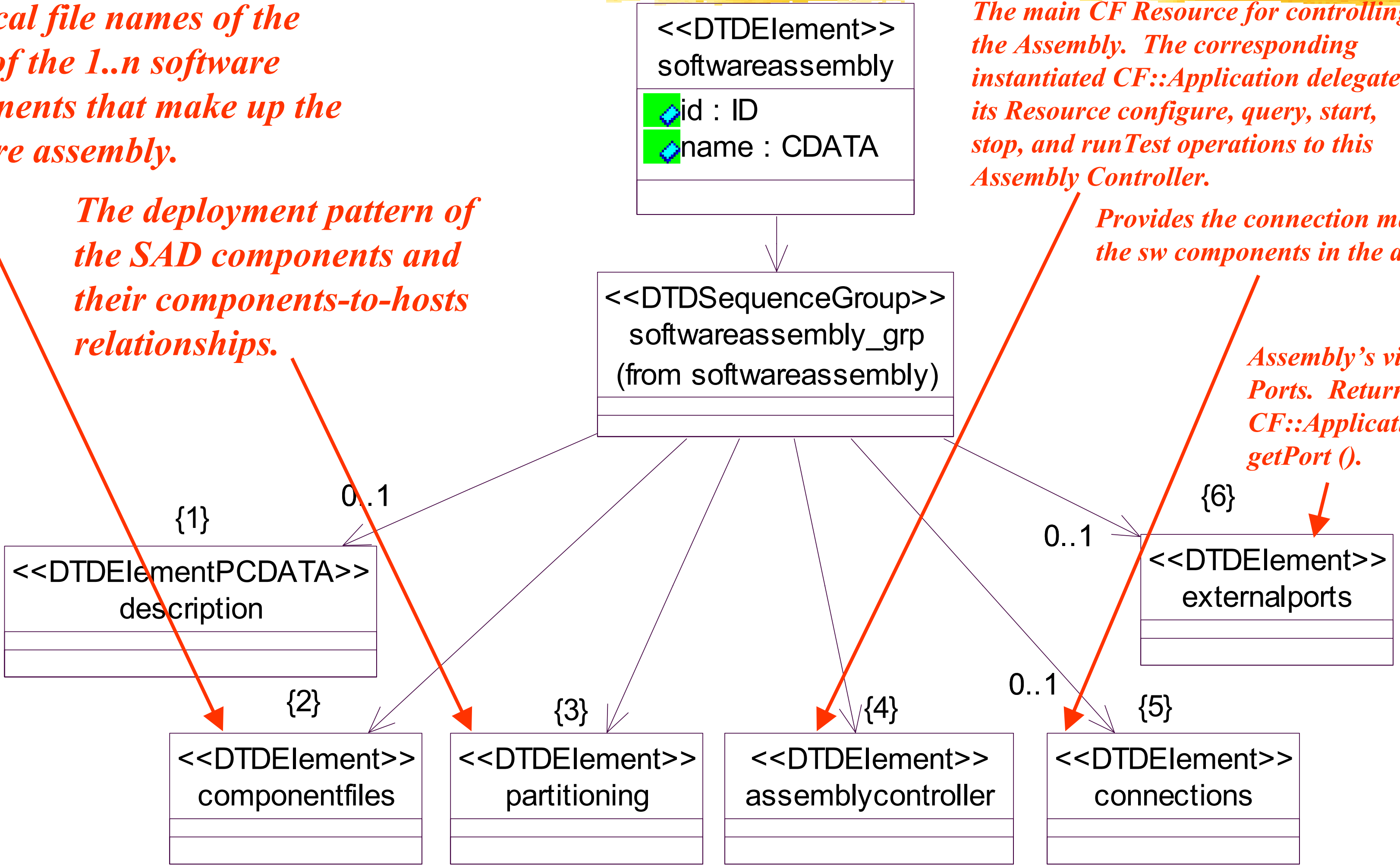
The local file names of the SPDs of the 1..n software components that make up the software assembly.

The deployment pattern of the SAD components and their components-to-hosts relationships.

The main CF Resource for controlling the Assembly. The corresponding instantiated CF::Application delegates its Resource configure, query, start, stop, and runTest operations to this Assembly Controller.

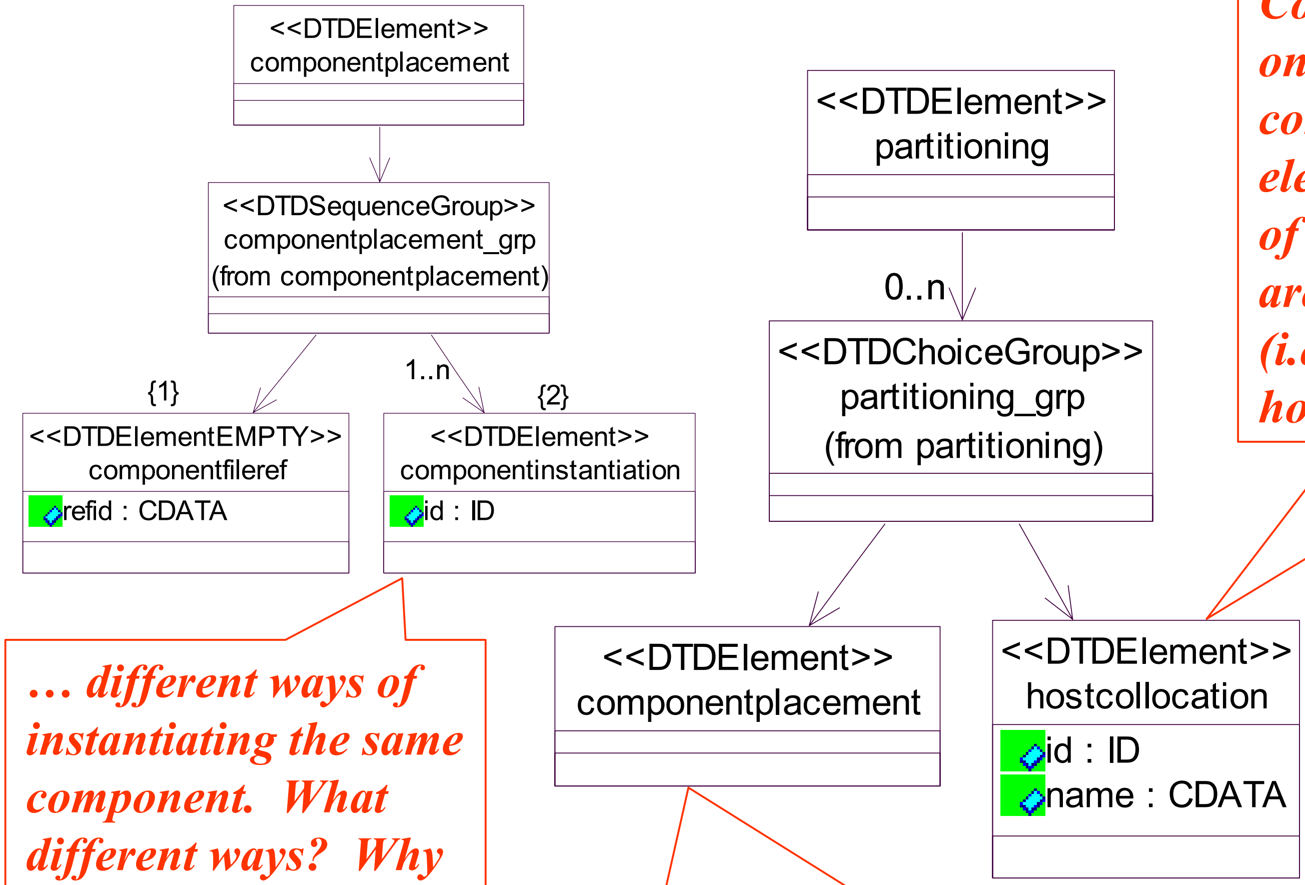
Provides the connection map between the sw components in the assembly.

Assembly's visible Ports. Returned by CF::Application getPort ().



<<DTDElement>> softwareassembly
id : ID
name : CDATA
<<DTDSequenceGroup>> softwareassembly_grp (from softwareassembly)
0..1
{1}
<<DTDElementPCDATA>> description
{2}
<<DTDElement>> componentfiles
{3}
<<DTDElement>> partitioning
{4}
<<DTDElement>> assemblycontroller
0..1
{5}
<<DTDElement>> connections
0..1
{6}
<<DTDElement>> externalports

SAD Component Deployment Pattern



<<DTDElement>> componentplacement
<<DTDSequenceGroup>> componentplacement_grp (from componentplacement)
{1}
1..n
{2}
<<DTDElementEMPTY>> componentfileref
refid : CDATA
<<DTDElement>> componentinstantiation
id : ID
<<DTDElement>> partitioning
0..n
<<DTDChoiceGroup>> partitioning_grp (from partitioning)
<<DTDElement>> componentplacement
<<DTDElement>> hostcollocation
id : ID
name : CDATA

Comprised of a grouping of one or more componentplacement elements, describing a group of component instances that are to be deployed together (i.e collocated) on a single host.

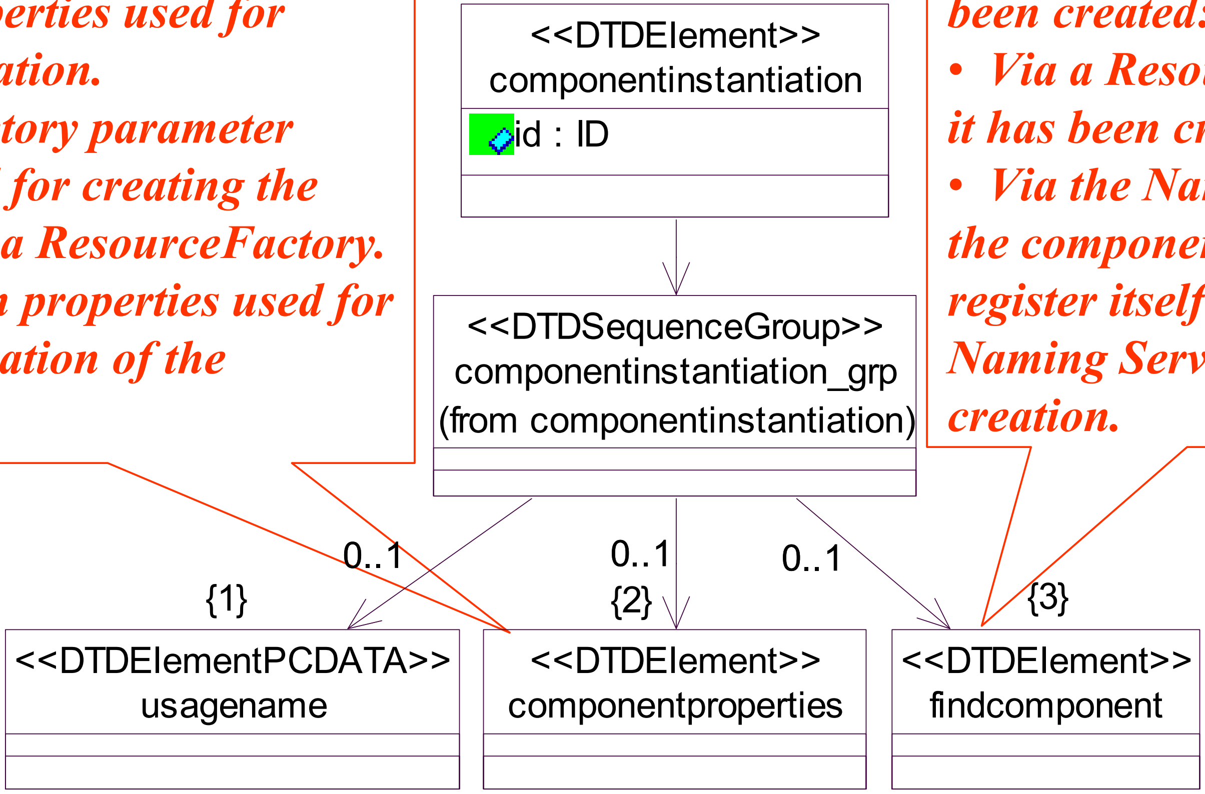
… different ways of instantiating the same component. What component. What different ways? Why do this?

Defines one or more deployments of a component. Meaning…

SAD Component Deployment Pattern (cont.)

Optional list of
- *Component-specific execute parameter properties used for component creation.*
- *ResourceFactory parameter properties used for creating the component via a ResourceFactory.*
- *Configuration properties used for initial configuration of the component*

Optional means of finding the component after it has been created:
- *Via a ResourceFactory, if it has been created by one.*
- *Via the Naming Service, if the component is directed to register itself with the Naming Service upon creation.*



BOEING®

Device Configuration Descriptor (DCD)

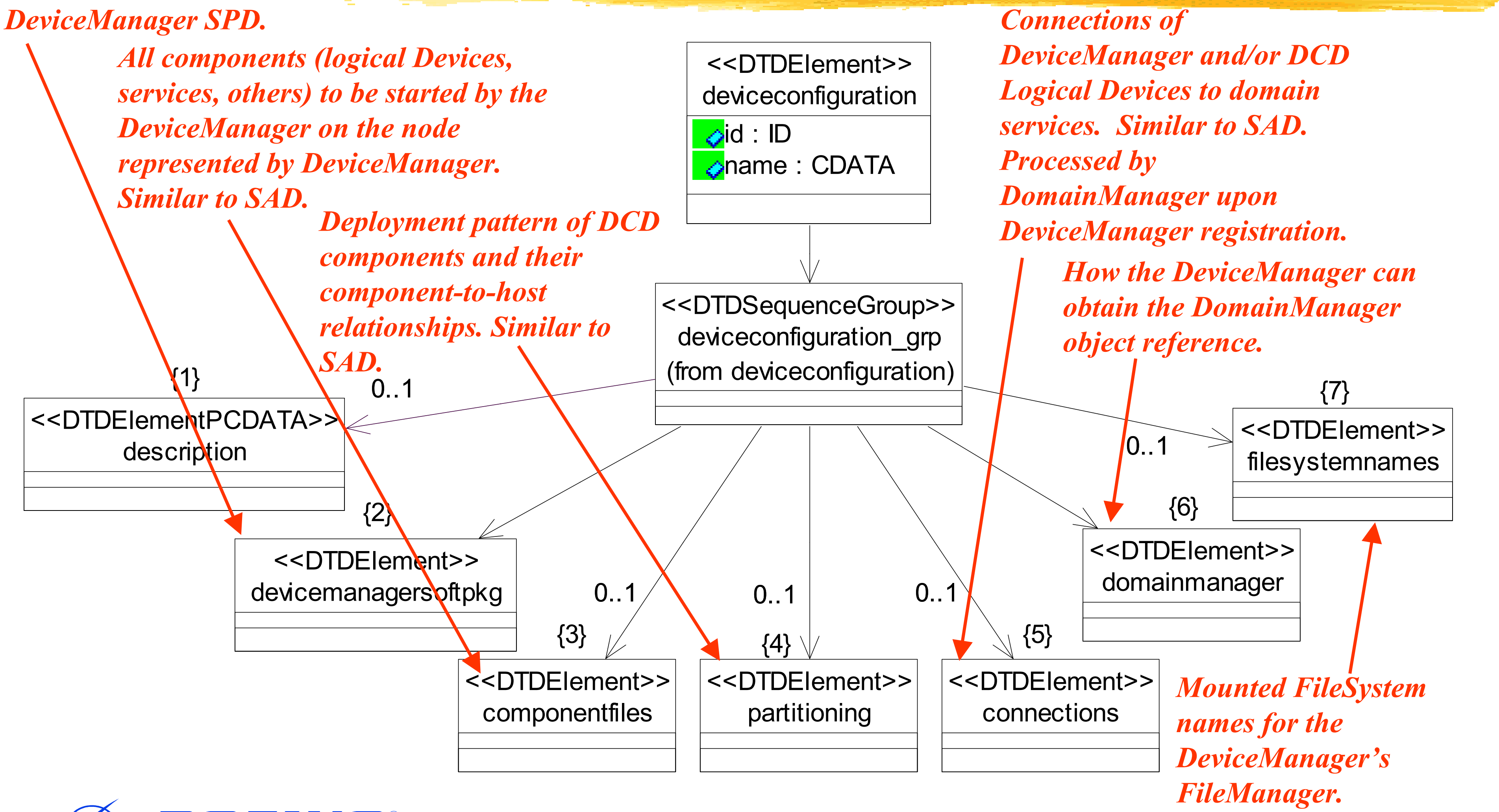
Derived from the Software Assembly Descriptor file definition.

Processed by each *DeviceManager* and by *DomainManager* upon *DeviceManager* registration with CF domain.

Provides a generic means of deploying persistent components (logical Device(s) and services) to the CF domain during each node (*DeviceManager*) startup.

Allows a System Administrator flexibility to decide what nodes within the system services should be deployed on and how many services should be within a system.

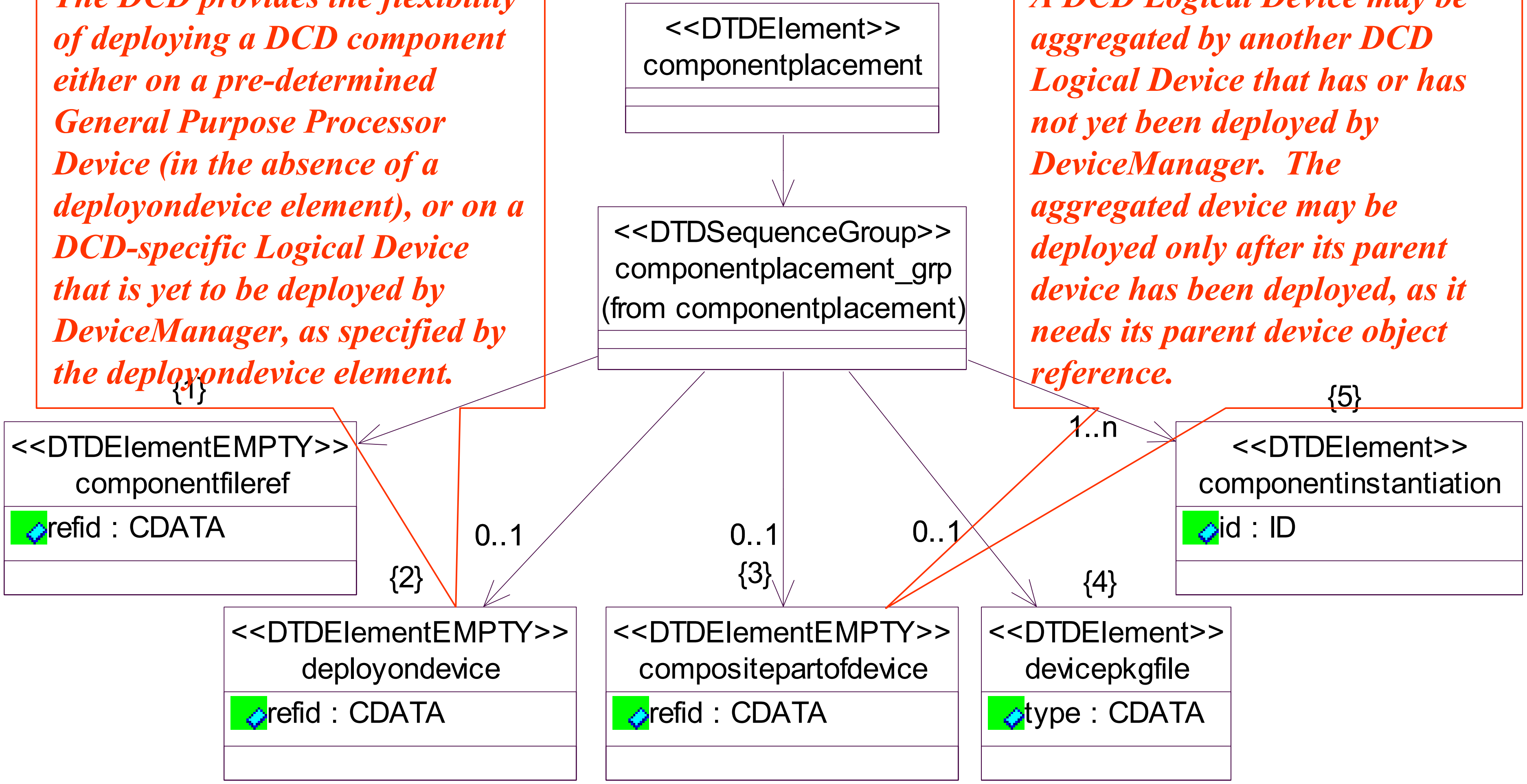
DCD Elements


DeviceManager SPD.
All components (logical Devices, services, others) to be started by the DeviceManager on the node represented by DeviceManager. Similar to SAD.
Deployment pattern of DCD components and their component-to-host relationships. Similar to SAD.
Connections of DeviceManager and/or DCD Logical Devices to domain services. Similar to SAD. Processed by DomainManager upon DeviceManager registration.
How the DeviceManager can obtain the DomainManager object reference.
Mounted FileSystem names for the DeviceManager's FileManager.
<<DTDElement>> deviceconfiguration
id : ID
name : CDATA
<<DTDSequenceGroup>> deviceconfiguration_grp (from deviceconfiguration)
{1}
0..1
<<DTDElementPCDATA>> description
{2}
<<DTDElement>> devicemanagersoftpkg
0..1
{3}
<<DTDElement>> componentfiles
0..1
{4}
<<DTDElement>> partitioning
0..1
{5}
<<DTDElement>> connections
{6}
<<DTDElement>> domainmanager
{7}
0..1
<<DTDElement>> filesystemnames

DCD Component Deployment Pattern

The DCD provides the flexibility of deploying a DCD component either on a pre-determined General Purpose Processor Device (in the absence of a deployondevice element), or on a DCD-specific Logical Device that is yet to be deployed by DeviceManager, as specified by the deployondevice element.

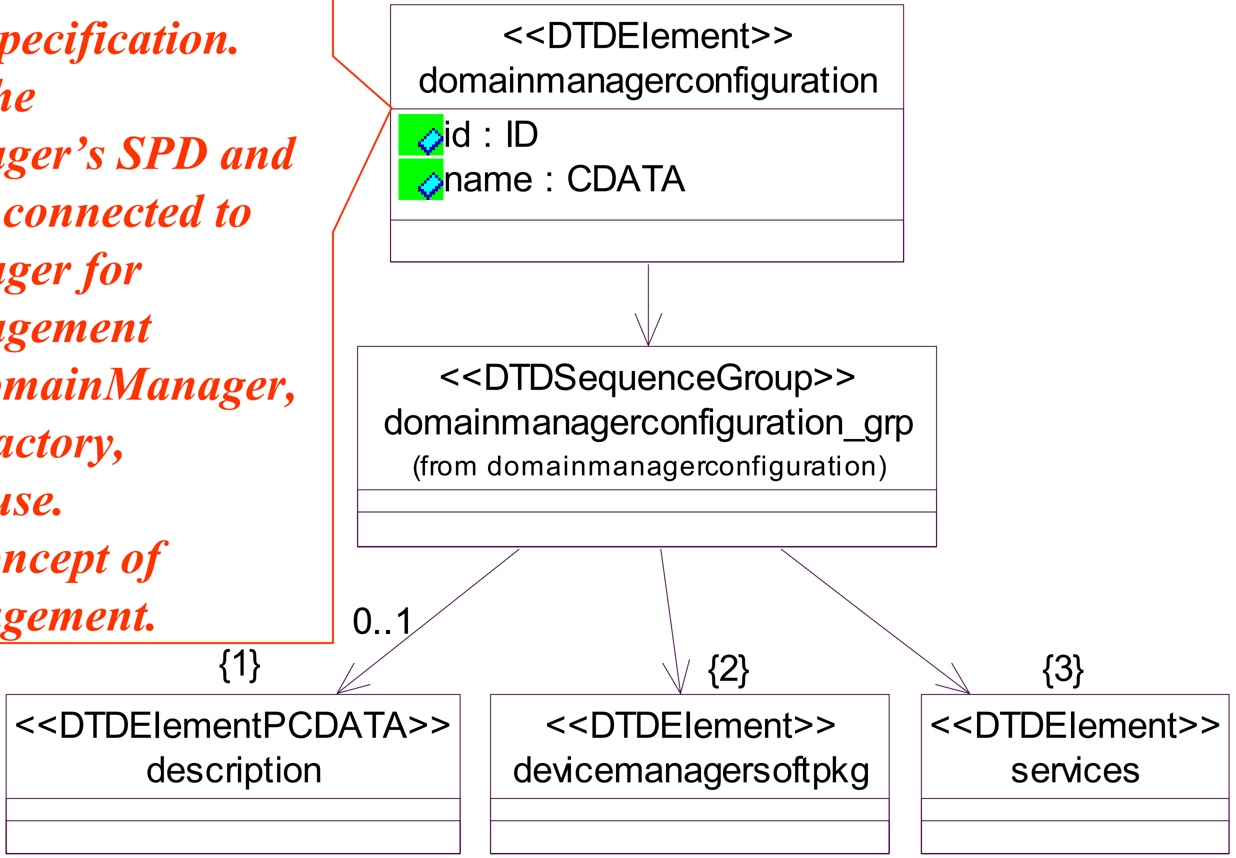
A DCD Logical Device may be aggregated by another DCD Logical Device that has or has not yet been deployed by DeviceManager. The aggregated device may be deployed only after its parent device has been deployed, as it needs its parent device object reference.



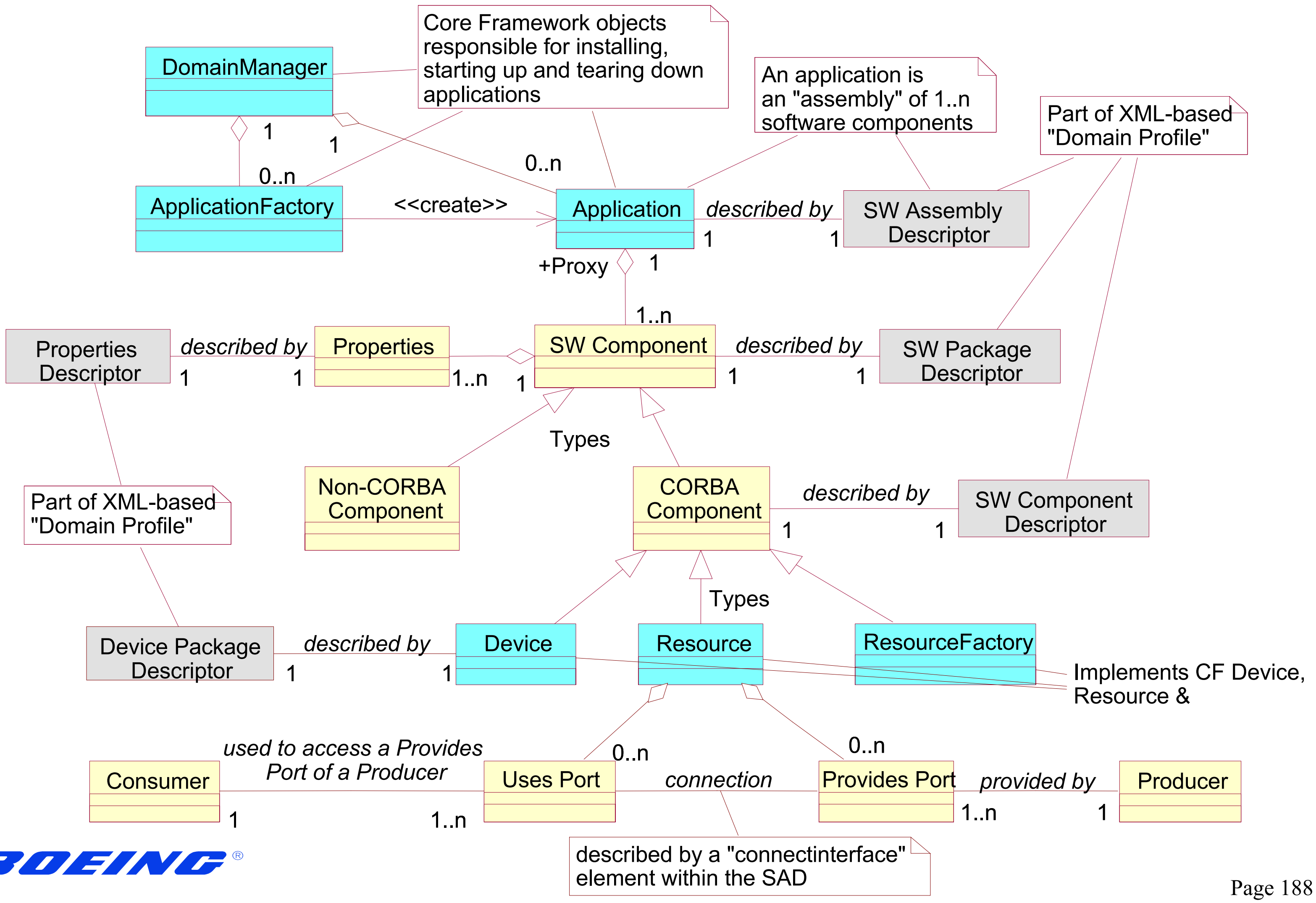
<<DTDElement>> componentplacement
<<DTDSequenceGroup>> componentplacement_grp (from componentplacement)
{1}
<<DTDElementEMPTY>> componentfileref
refid : CDATA
0..1
{2}
<<DTDElementEMPTY>> deployondevice
refid : CDATA
0..1
{3}
<<DTDElementEMPTY>> compositepartofdevice
refid : CDATA
0..1
{4}
<<DTDElement>> devicepkgfile
type : CDATA
1..n
{5}
<<DTDElement>> componentinstantiation
id : ID

DomainManager Configuration Descriptor

- *Not based on CORBA Component Specification.*
- *Describes the DomainManager's SPD and services to be connected to DomainManager for DomainManagement functions (DomainManager, ApplicationFactory, Application) use.*
- *No other concept of domain management.*



SCA Concept of Components & Domain Profile File Types



How Does CF Interact with Application Components?

Some Examples of...

Starting a node (*DeviceManager*)

- DeviceManager startup
- Creating persistent devices and services on that node (*DeviceManager*)
- Deploying persistent devices and services to the CF domain to service domain application instantiations (*DeviceManager* and *DomainManager*)

Installing an application (*DomainManager*)

Retrieving the list of installed applications (*DomainManager*)

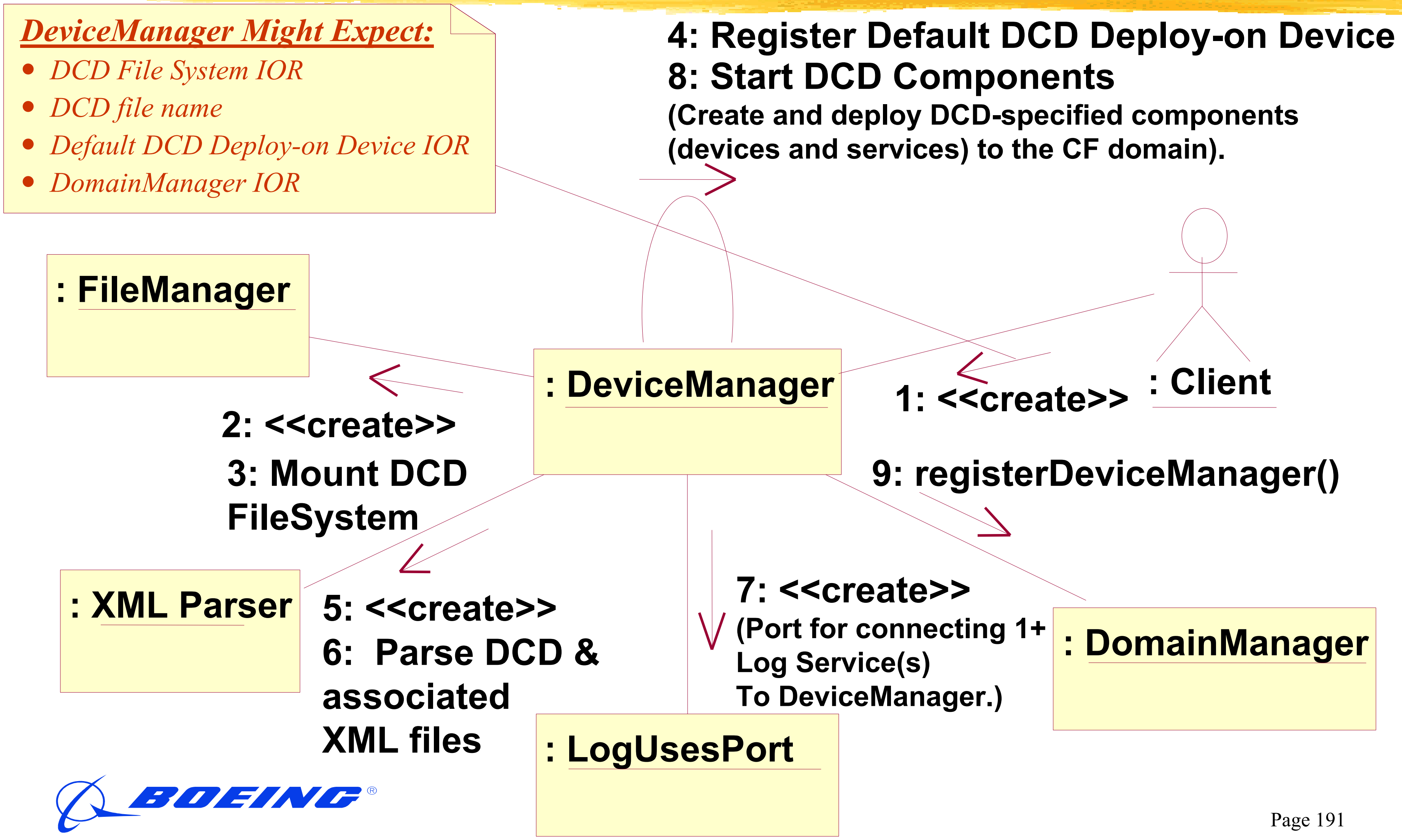
Creating an application (*ApplicationFactory*)

- Creating a single application component
- Connecting an application's components together

Retrieving the list of running applications (*DomainManager*)

Communicating with one of those running applications (*Application*)

Node (DeviceManager) Startup

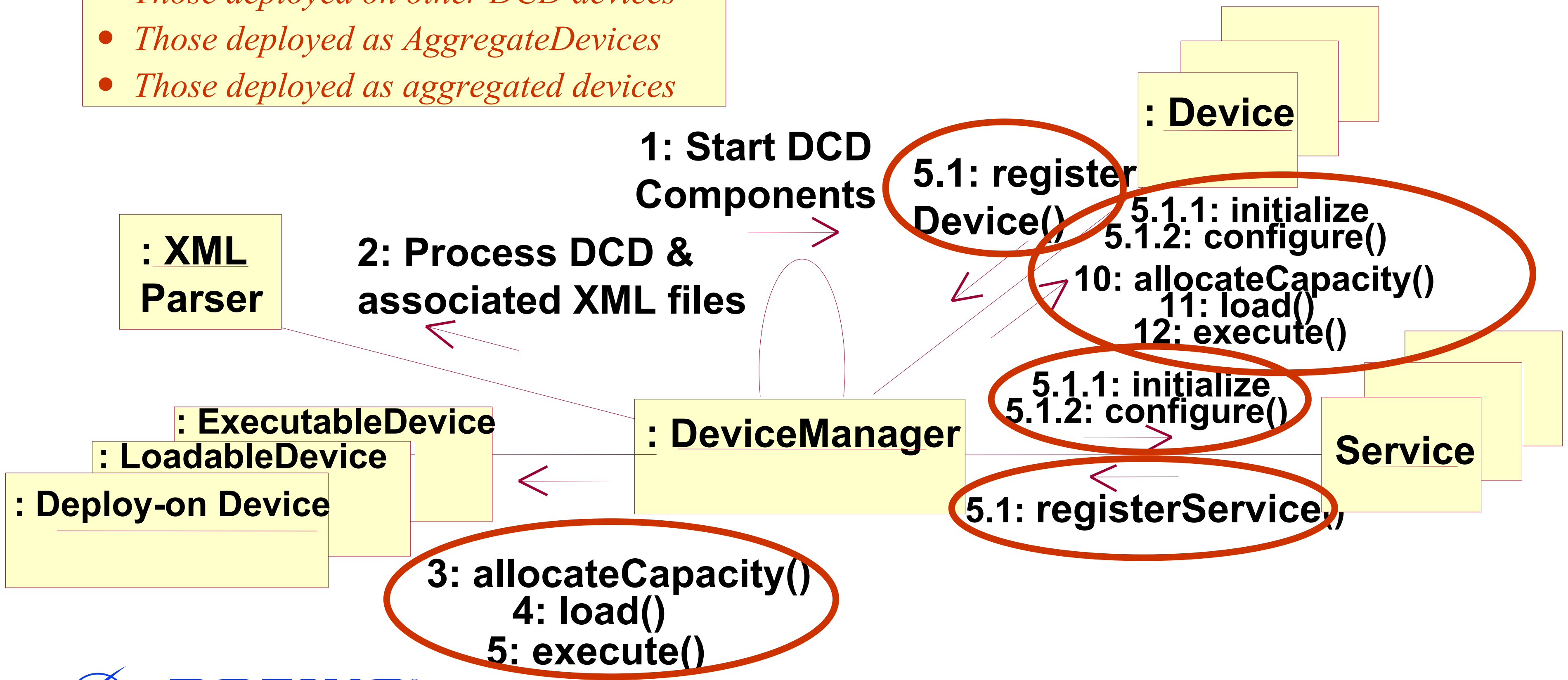

DeviceManager Might Expect:
• DCD File System IOR
• DCD file name
• Default DCD Deploy-on Device IOR
• DomainManager IOR
4: Register Default DCD Deploy-on Device
8: Start DCD Components
(Create and deploy DCD-specified components (devices and services) to the CF domain).
: FileManager
: DeviceManager
: Client
1: <<create>>
2: <<create>>
3: Mount DCD FileSystem
9: registerDeviceManager()
: XML Parser
5: <<create>>
6: Parse DCD & associated XML files
7: <<create>>
(Port for connecting 1+ Log Service(s) To DeviceManager.)
: DomainManager
: LogUsesPort

BOEING

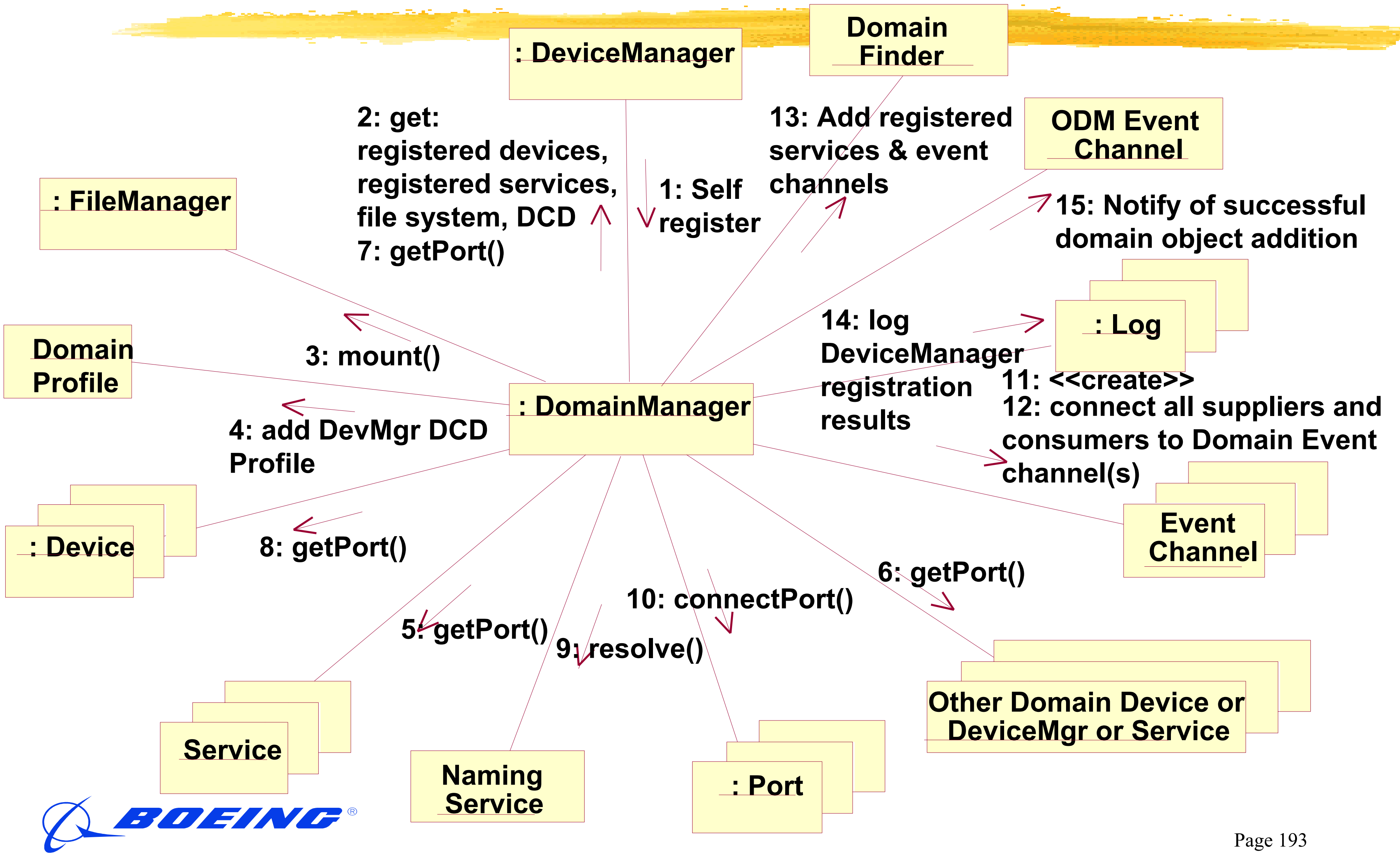
Node (DCD) Component Creation

Iterative process, until all DCD devices & services are created

- *Those deployed on deploy-on device*
- *Those deployed on other DCD devices*
- *Those deployed as AggregateDevices*
- *Those deployed as aggregated devices*



DCD Component (Device/Service) Deployment to CF Domain


: DeviceManager
Domain Finder
2: get: registered devices, registered services, file system, DCD
7: getPort()
1: Self register
13: Add registered services & event channels
ODM Event Channel
: FileManager
15: Notify of successful domain object addition
14: log DeviceManager registration results
: Log
Domain Profile
3: mount()
: DomainManager
11: <<create>>
12: connect all suppliers and consumers to Domain Event channel(s)
4: add DevMgr DCD Profile
Event Channel
: Device
8: getPort()
6: getPort()
10: connectPort()
5: getPort()
9: resolve()
Other Domain Device or DeviceMgr or Service
Service
Naming Service
: Port

BOEING®

Application Installation

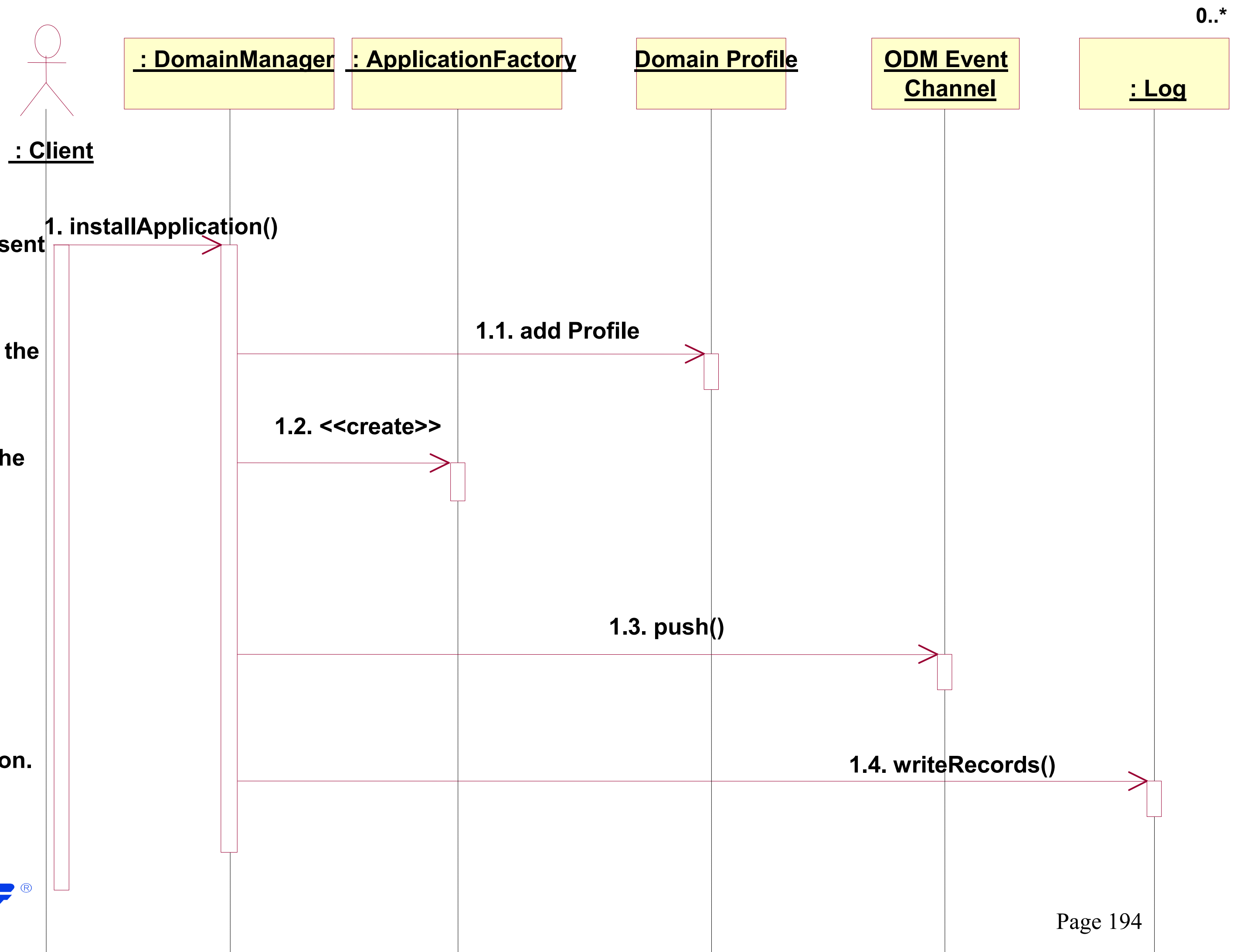
An application installation request is sent to the DomainManager.

Add the application's S/W Profile into the Domain Profile.

Construct an ApplicationFactory for the specified application software profile.

The DomainManager sends an event to the ODM Event Channel to indicate the installation of the application.

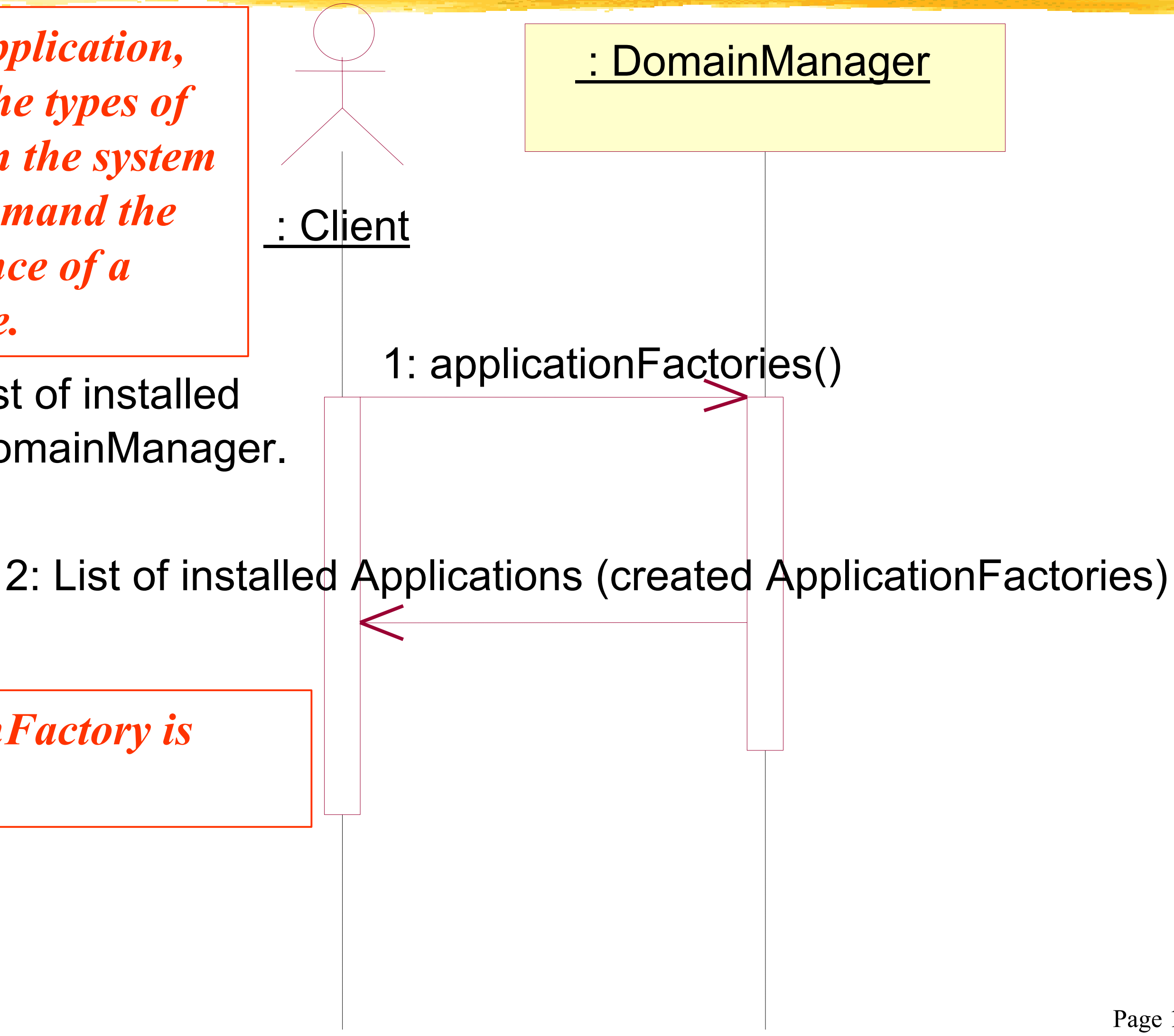
Log the result of application installation.



Listing Installed Applications

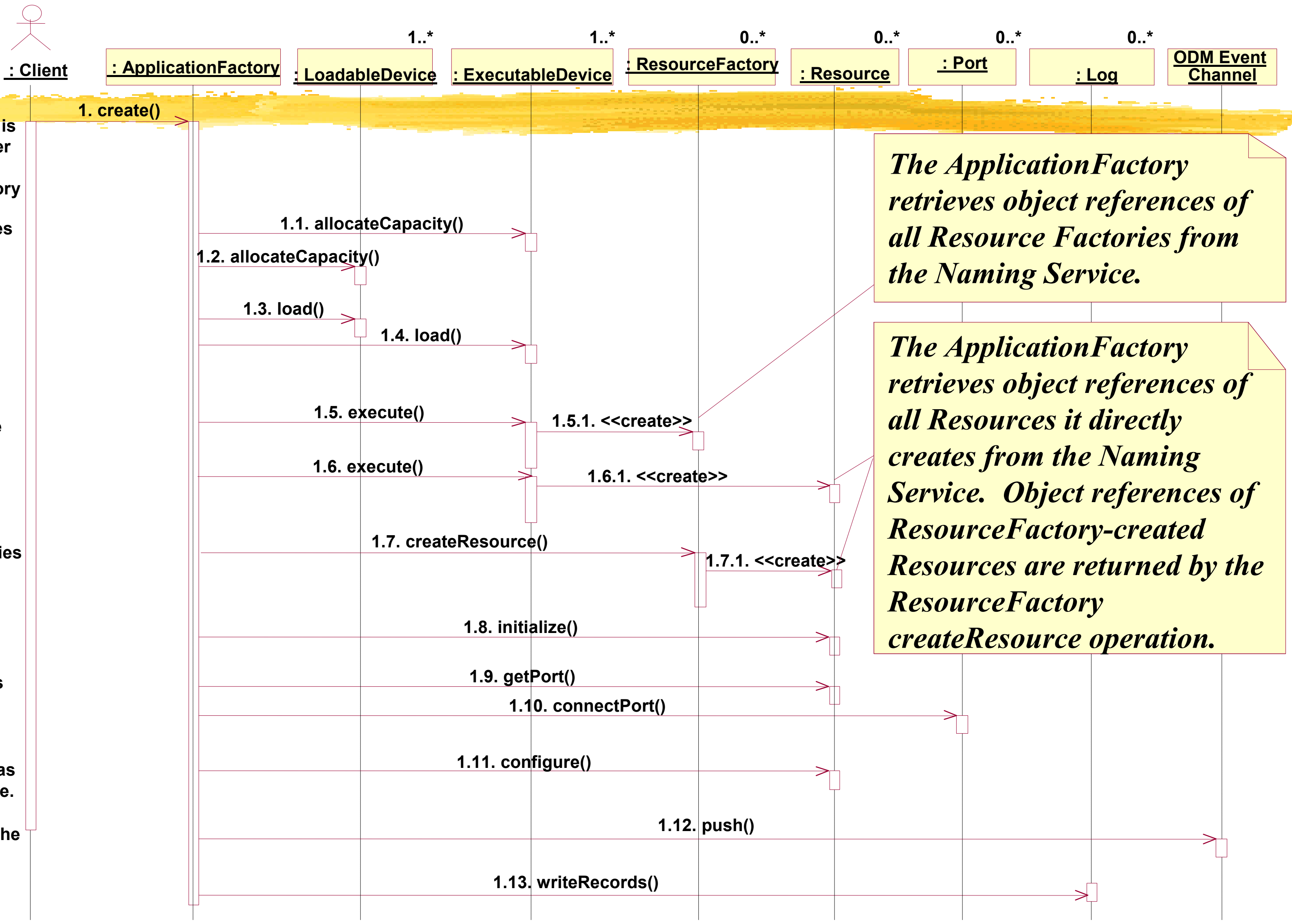
For a user to start an application, they must be aware of the types of applications available in the system and then be able to command the creation of a new instance of a selected application type.

A request for getting a list of installed applications is sent to DomainManager.


: DomainManager
: Client
1: applicationFactories()
2: List of installed Applications (created ApplicationFactories)

The desired ApplicationFactory is chosen.

Application Creation



An application creation request is sent to ApplicationFactory. After evaluating the application's SW profile, the ApplicationFactory

Allocates any required capacities on all applicable devices.

Loads all required SW on all applicable devices.

Creates all Resource Factories and Resources on all applicable devices.

Commands all Resource Factories to create their corresponding resources.

Initializes all app's resources.

Establishes all port connections for all app's resources as defined by the app's SW profile.

Configures all app's resources as specified by the app's SW profile.

Generates an event to indicate the application has been created.

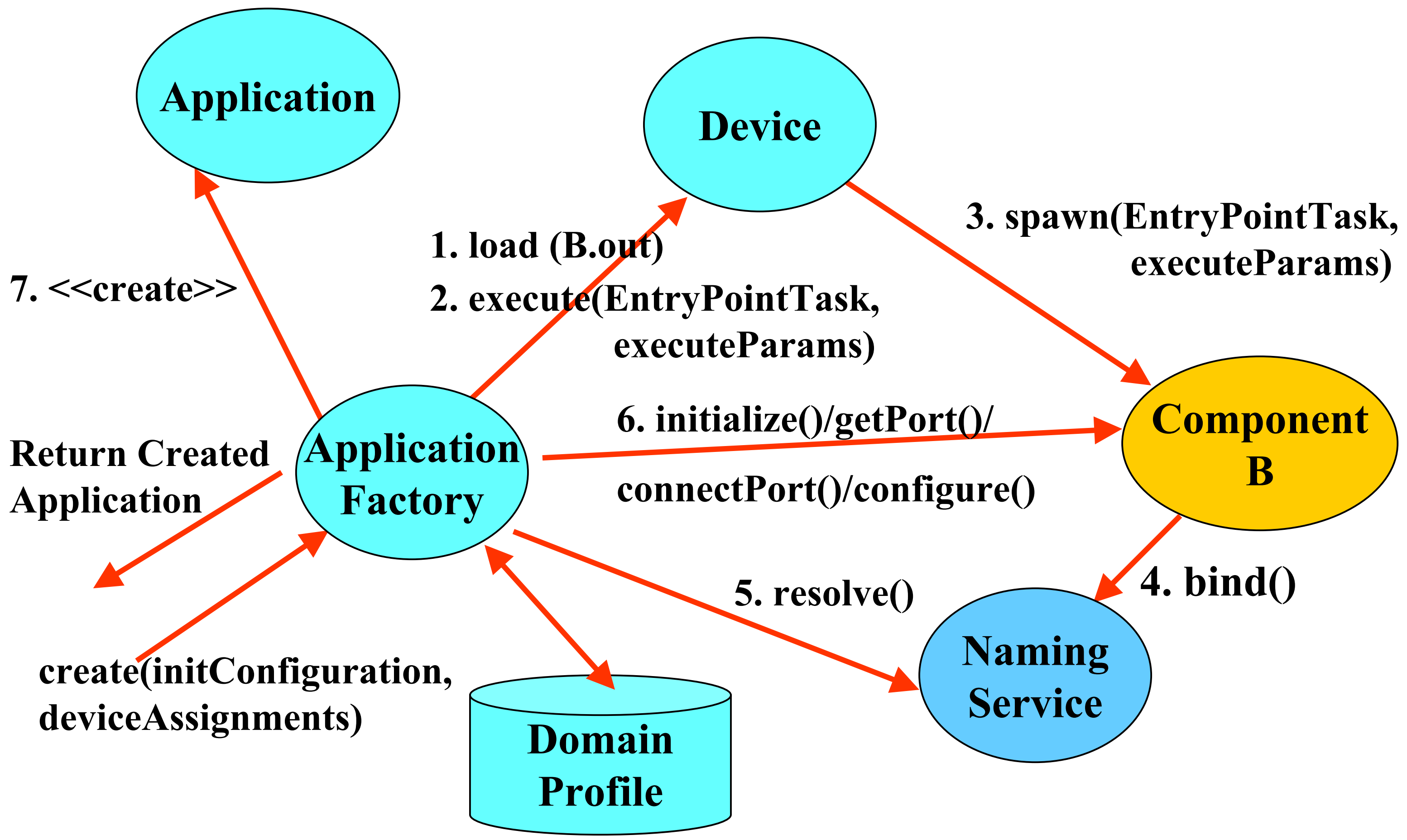
Logs the app creation result.

The ApplicationFactory retrieves object references of all Resource Factories from the Naming Service.

The ApplicationFactory retrieves object references of all Resources it directly creates from the Naming Service. Object references of ResourceFactory-created Resources are returned by the ResourceFactory createResource operation.

Finally, a CF Application is returned, providing proxy interface to Assembly Controller of created Resources.

Launching a Single Application Component



Application
Device
7. <<create>>
1. load (B.out)
2. execute(EntryPointTask, executeParams)
3. spawn(EntryPointTask, executeParams)
Return Created Application
Application Factory
6. initialize()/getPort()/ connectPort()/configure()
Component B
4. bind()
5. resolve()
Naming Service
create(initConfiguration, deviceAssignments)
Domain Profile

BOEING®

Use of Configuration and Execute Parameters during Application Component creation

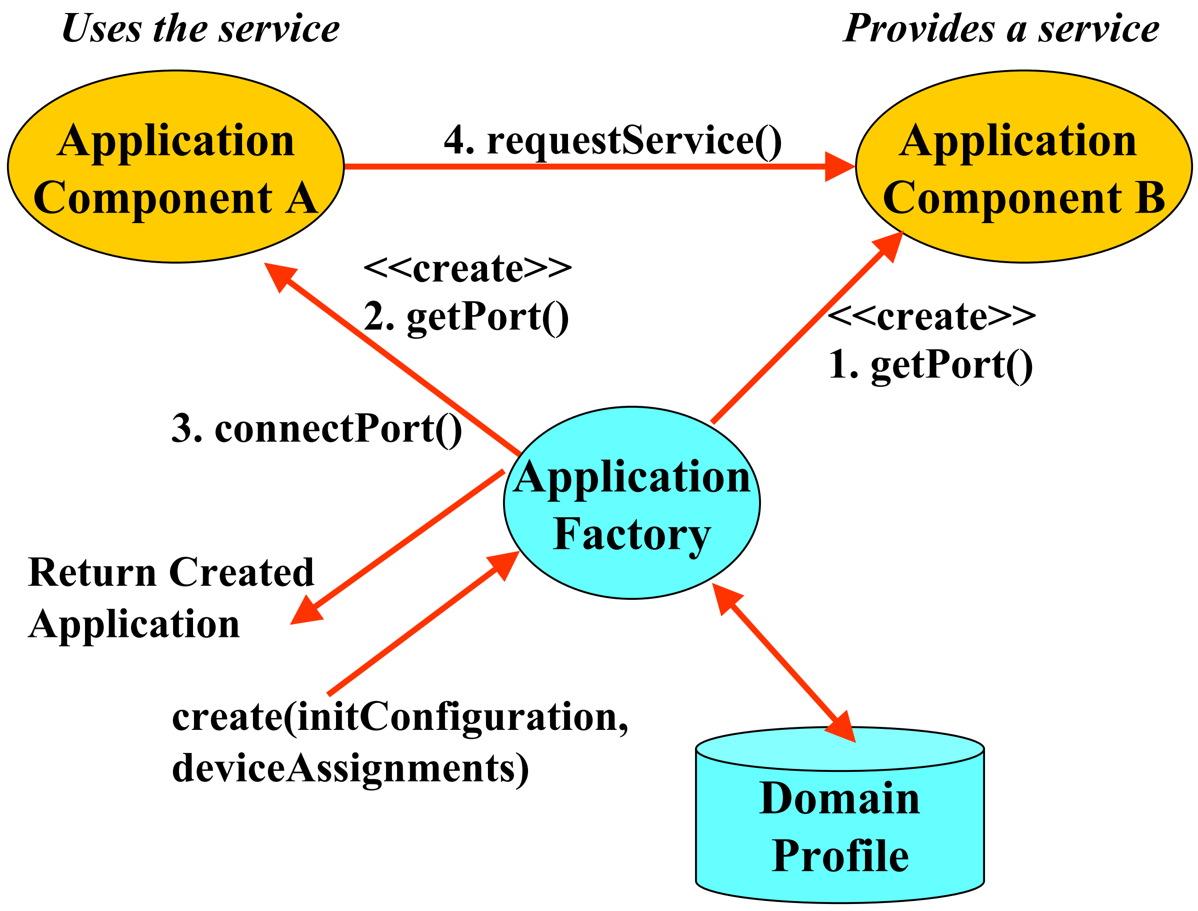
Configuration and Execute properties and their default values are defined in a Properties Descriptor, located in a Property File of the Application's Software Assembly Descriptor (SAD) in the Domain Profile.

CF will modify those properties based on the following precedence order:

- ApplicationFactory *create* method (platform specific overrides)
- SAD file (application specific overrides)
- SPD file (different implementation overrides)
- SPD Property file
- SCD Property file

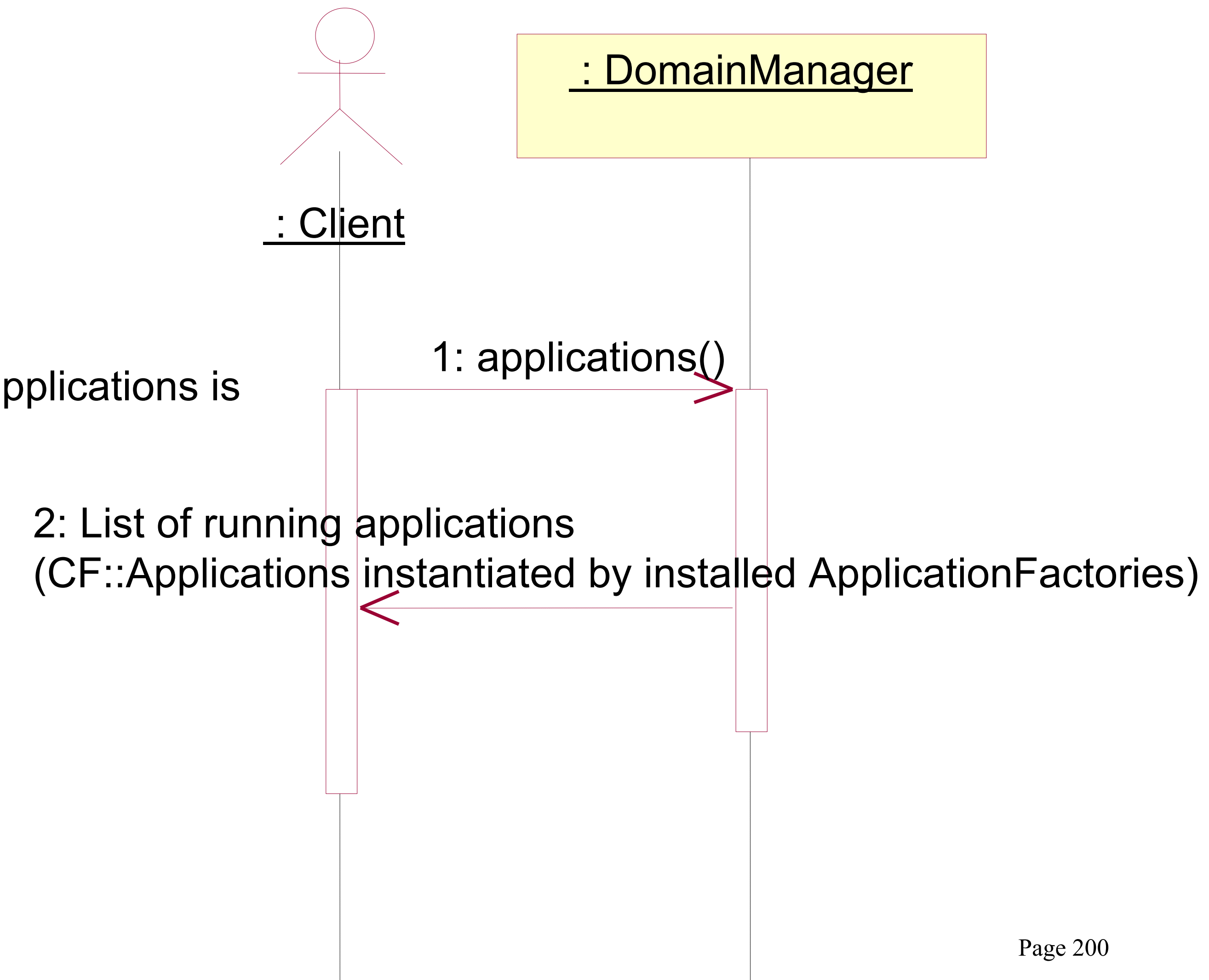
BOEING®

Connecting Application Components



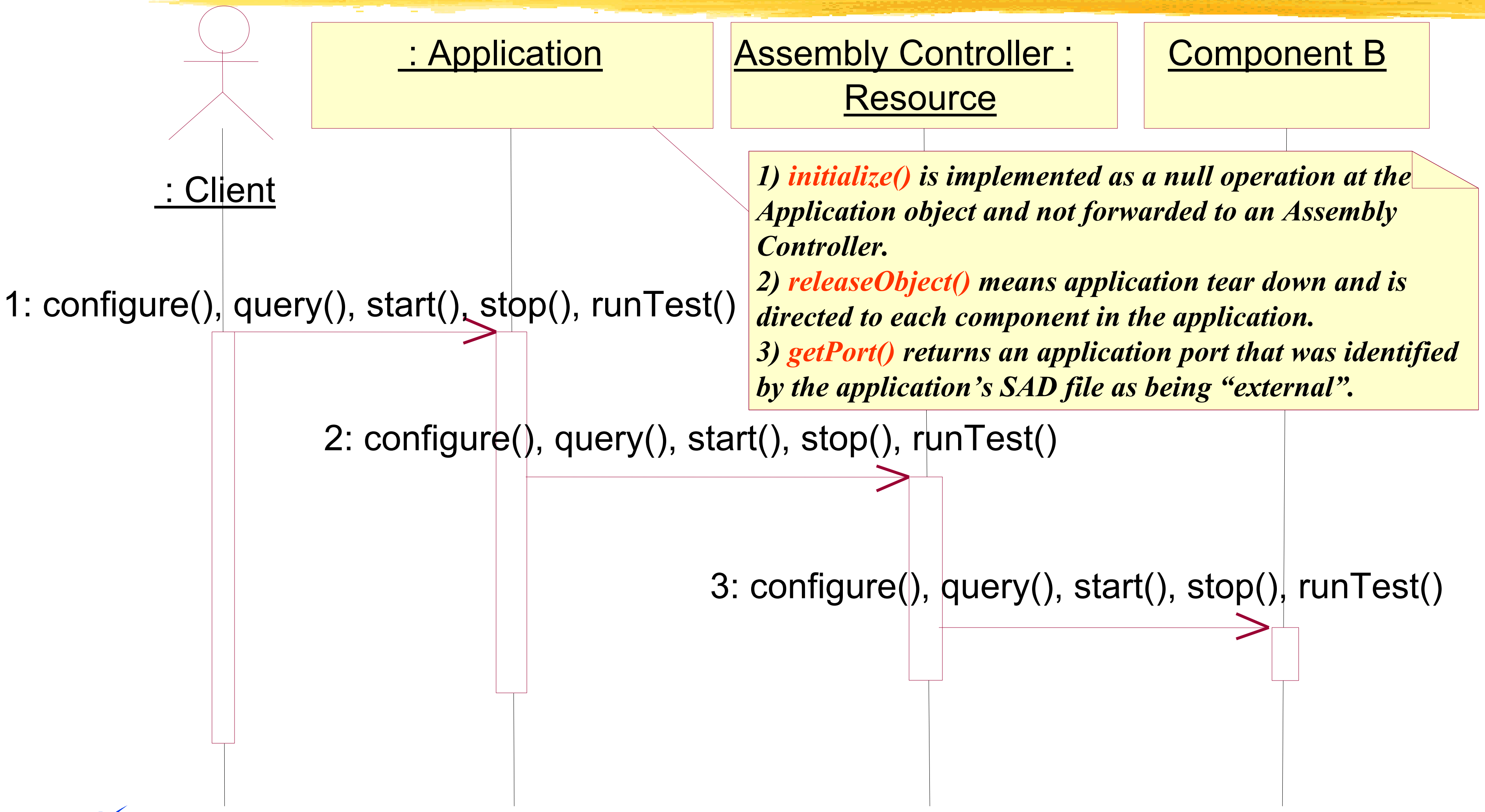
Listing Running Applications

A request for list of running applications is sent to the DomainManager.


: DomainManager
: Client
1: applications()
2: List of running applications
(CF::Applications instantiated by installed ApplicationFactories)

BOEING®

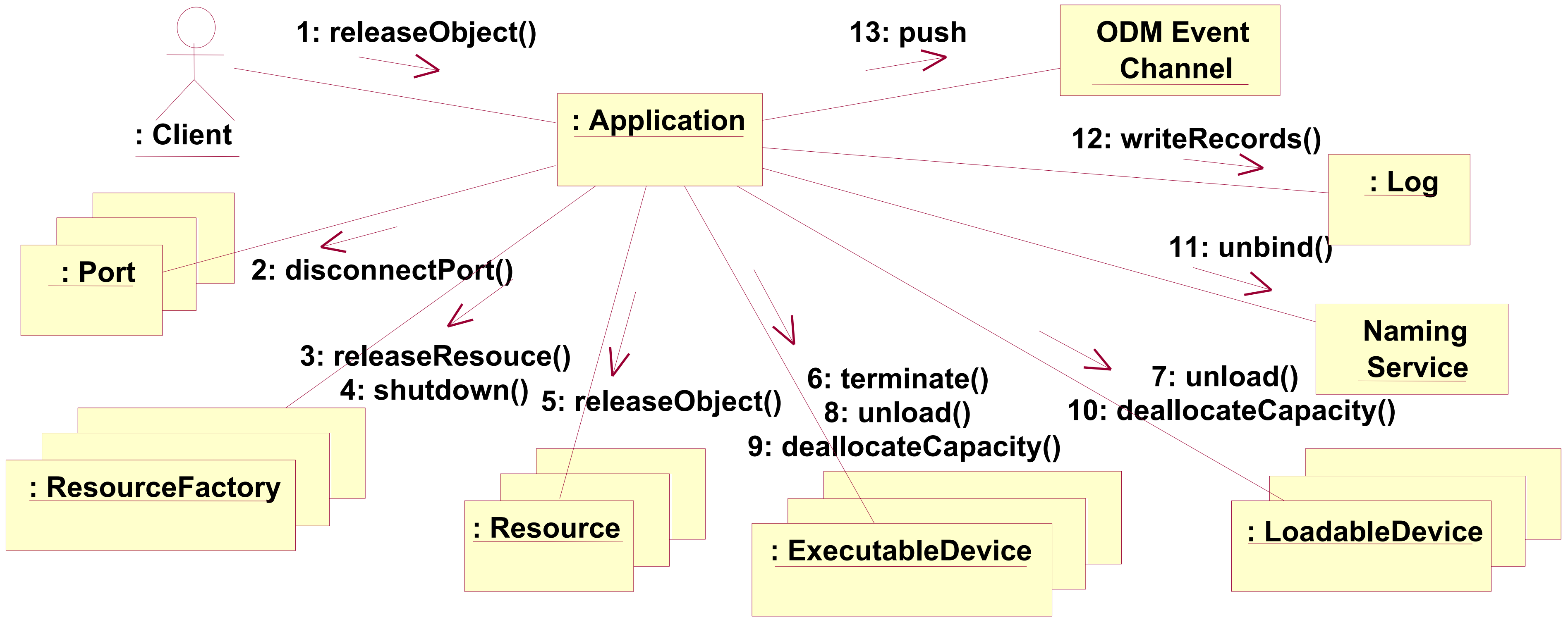
Invoking Operations on a Single Application Component



: Client
: Application
Assembly Controller : Resource
Component B
1: configure(), query(), start(), stop(), runTest()
2: configure(), query(), start(), stop(), runTest()
3: configure(), query(), start(), stop(), runTest()

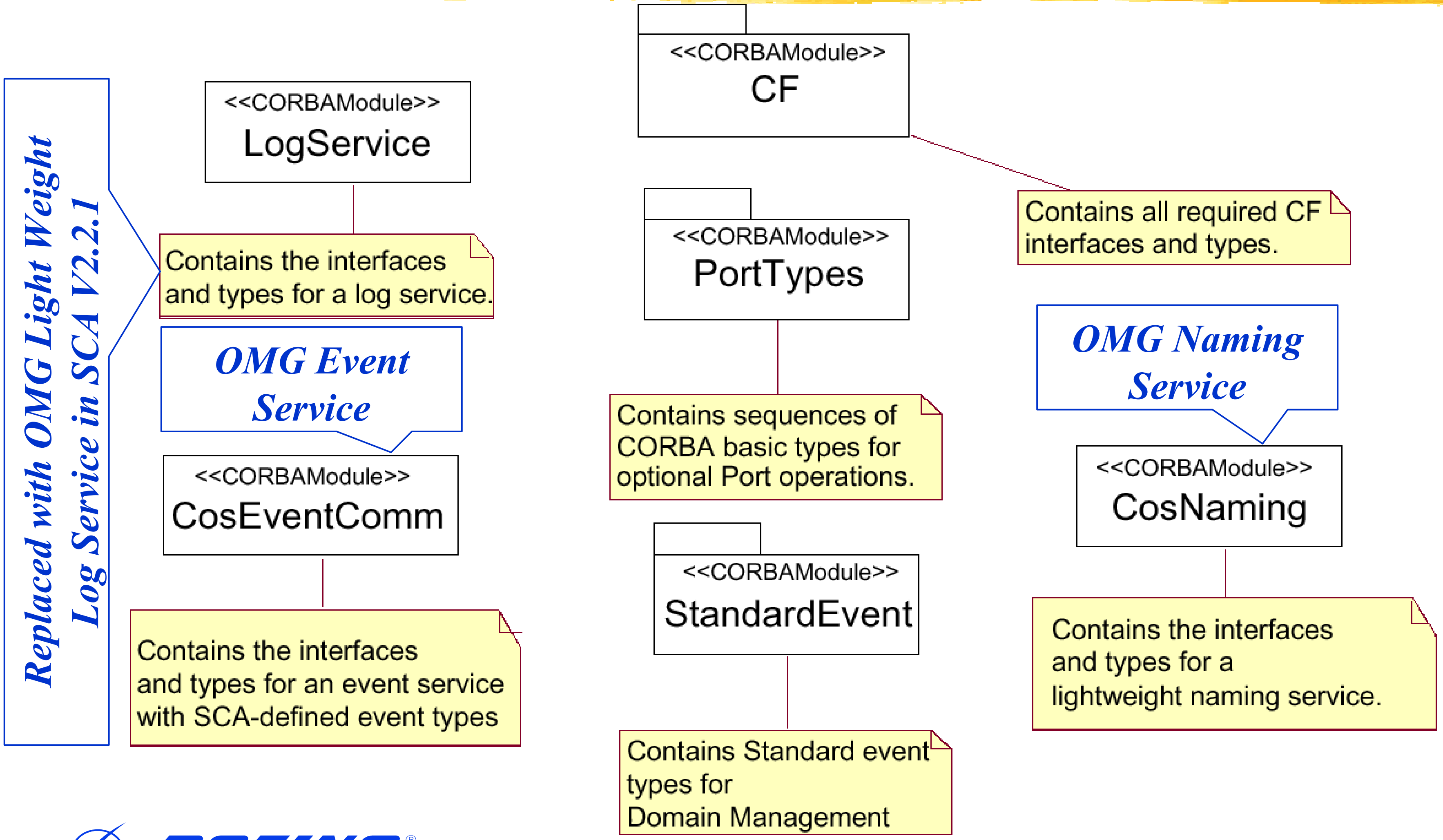
1) initialize() is implemented as a null operation at the Application object and not forwarded to an Assembly Controller.
2) releaseObject() means application tear down and is directed to each component in the application.
3) getPort() returns an application port that was identified by the application's SAD file as being "external".

Application Teardown



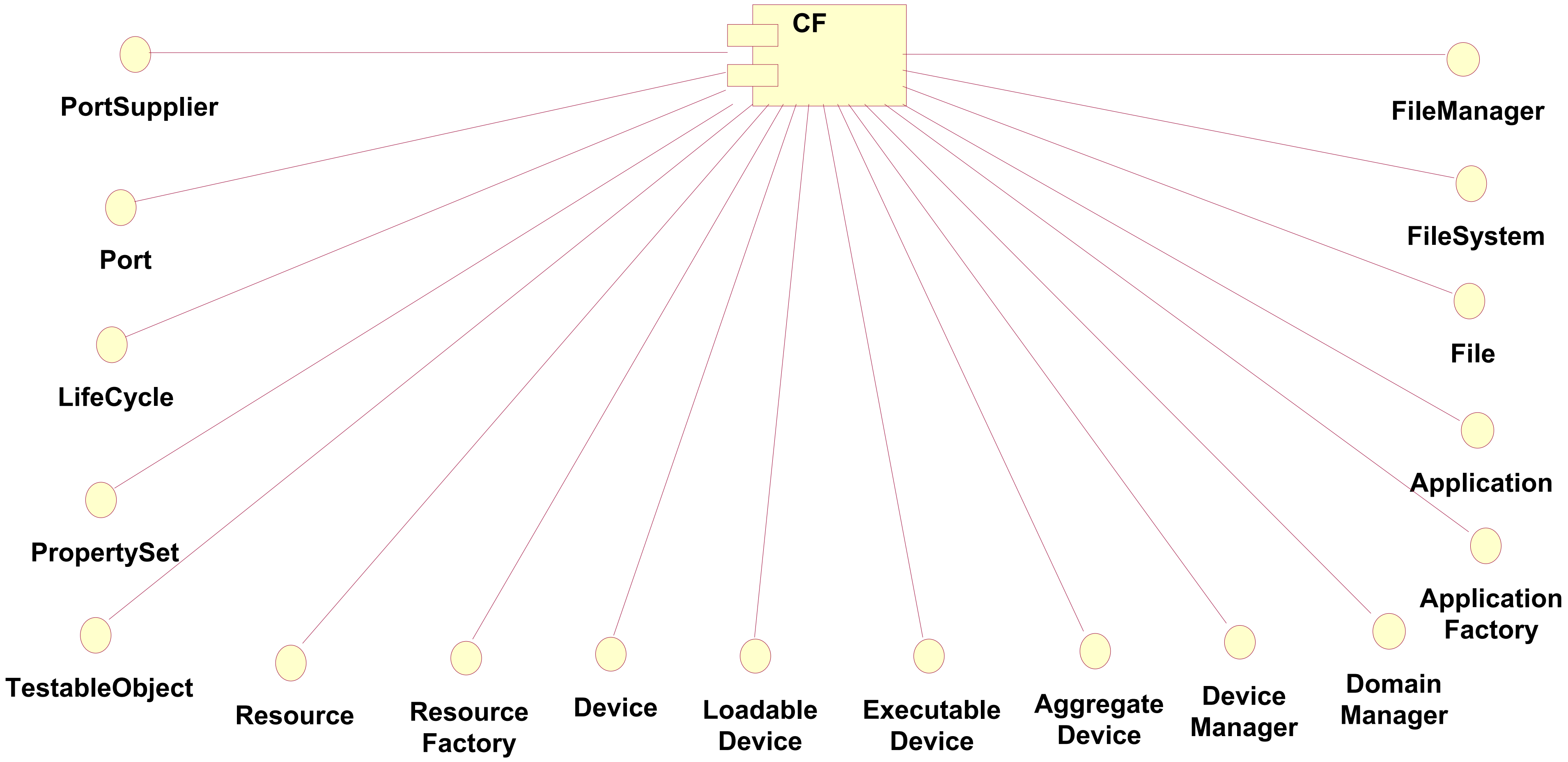
SCA IDL Modules

SCA IDL Modules



<<CORBAModule>>
CF
Contains all required CF interfaces and types.
<<CORBAModule>>
LogService
Contains the interfaces and types for a log service.
Replaced with OMG Light Weight Log Service in SCA V2.2.1
OMG Event Service
<<CORBAModule>>
CosEventComm
Contains the interfaces and types for an event service with SCA-defined event types
<<CORBAModule>>
PortTypes
Contains sequences of CORBA basic types for optional Port operations.
<<CORBAModule>>
StandardEvent
Contains Standard event types for Domain Management
OMG Naming Service
<<CORBAModule>>
CosNaming
Contains the interfaces and types for a lightweight naming service.

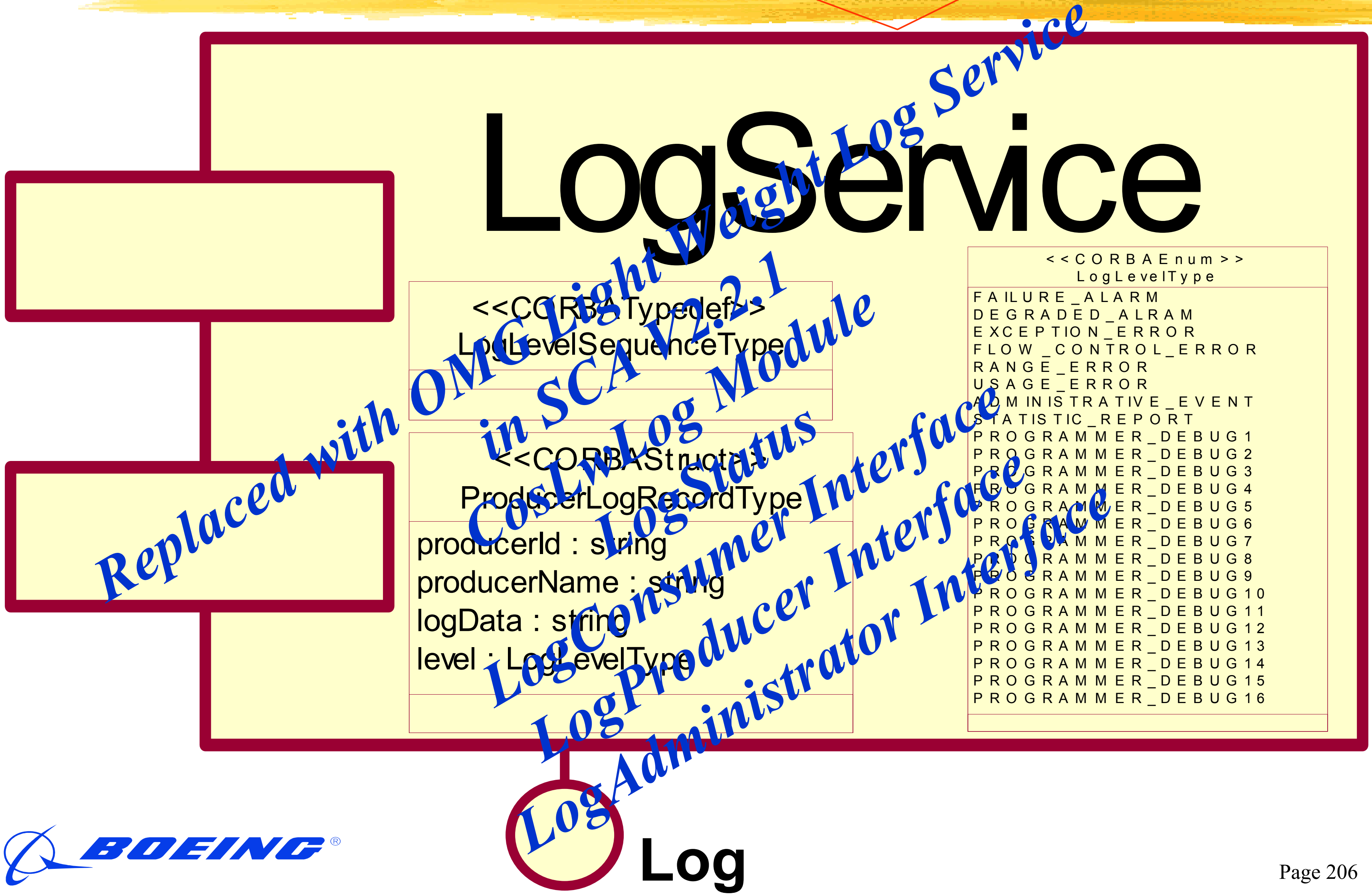
CF IDL Module



BOEING®

Log Service IDL Module

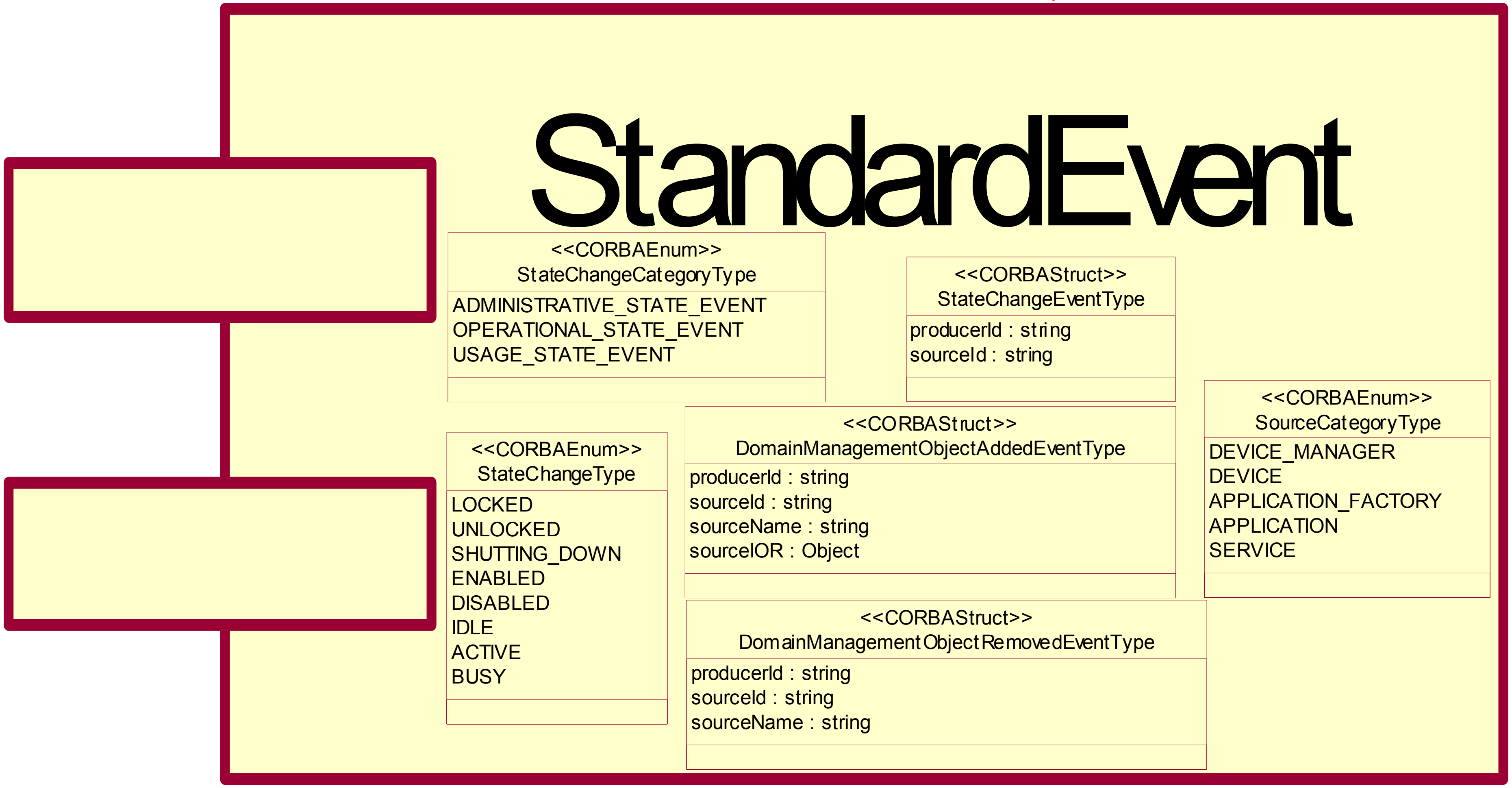
The Log interface, types for a log producer to generate standard SCA log records, and types necessary to control the logging output of a log producer.



Replaced with OMG Light Weight Log Service in SCA V2.2.1
CosLwLog Module
LogStatus
LogConsumer Interface
LogProducer Interface
LogAdministrator Interface

StandardEvent IDL Module

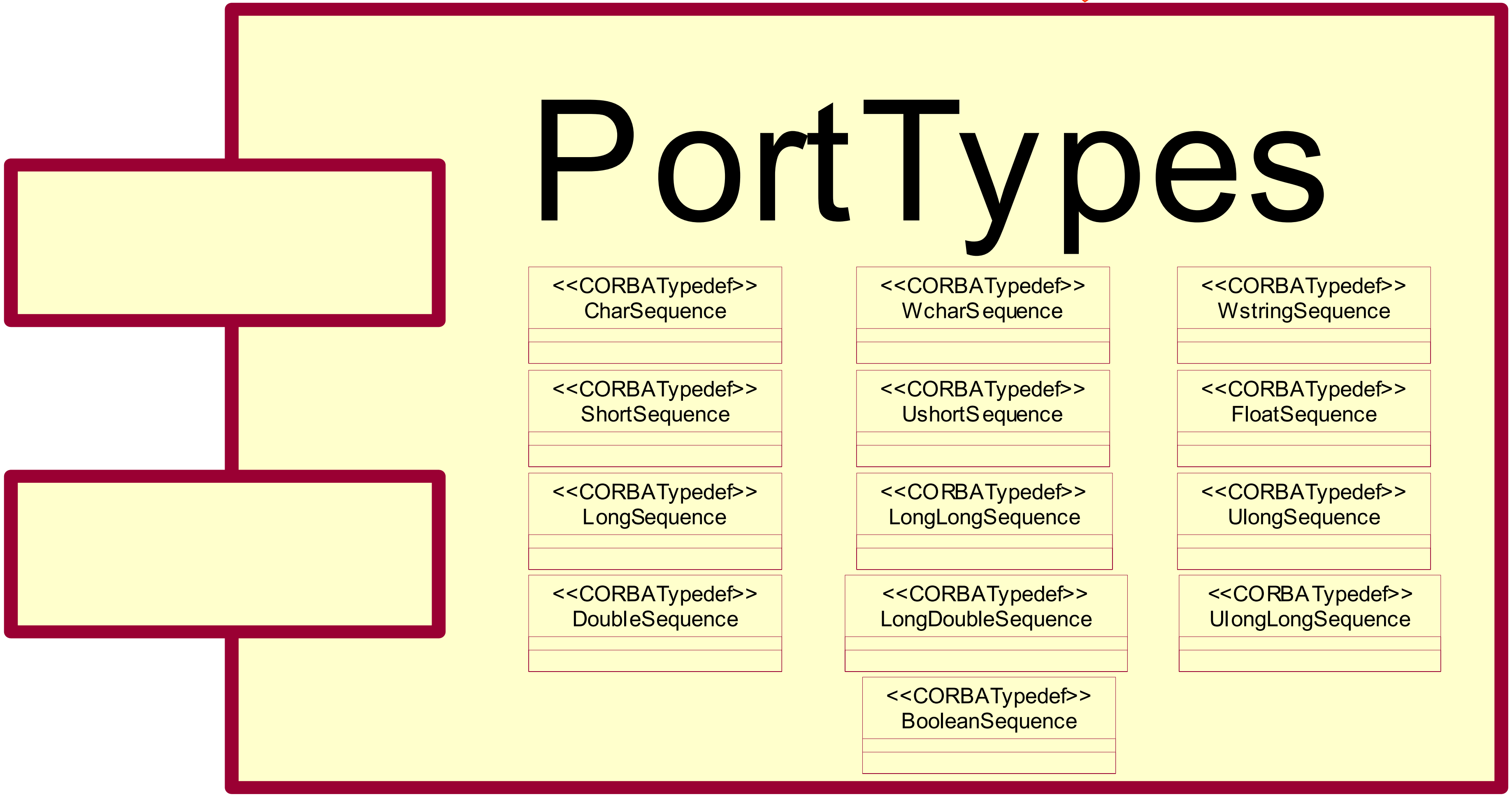
Types necessary for an event producer to generate standard SCA events.



BOEING®

PortTypes IDL Module

Sequences of base CORBA types for optional Port operations


PortTypes
<<CORBATypedef>> CharSequence
<<CORBATypedef>> WcharSequence
<<CORBATypedef>> WstringSequence
<<CORBATypedef>> ShortSequence
<<CORBATypedef>> UshortSequence
<<CORBATypedef>> FloatSequence
<<CORBATypedef>> LongSequence
<<CORBATypedef>> LongLongSequence
<<CORBATypedef>> UlongSequence
<<CORBATypedef>> DoubleSequence
<<CORBATypedef>> LongDoubleSequence
<<CORBATypedef>> UlongLongSequence
<<CORBATypedef>> BooleanSequence

References

References

http://jtrs.army.mil/

- Joint Tactical Radio System Program Background
- Software Communications Architecture Overview, Goals, and Standardization
- *Software Communication Architecture Specification*, V2.2
- *Software Communications Architecture Support and Rationale Document*, V2.2
- *Software Communications Architecture Developers Guide*, V1.1
- *Software Communications Architecture Application Program Interface Supplement*, V1.1
- *Software Communications Architecture Security Supplement*, V1.1
- Software Communications Architecture Training Course

http://www.omg.org

- *PIM and PSM for Software Radio Joint Revised Submission*
- *Experience Report from Developing and Fielding the First SCA 2.0 JTRS Radio*, BAE Systems, OMG Real-Time & Embedded Workshop July 2003

http://www.sdrforum.org

- Software Defined Radio Definition