Joint Program Executive Office
Joint Tactical Radio System

Deployment Optimizations –Push Model Registration




02 December 2010
JTRS SCA Working Group

JPEO JTRS

Task Overview

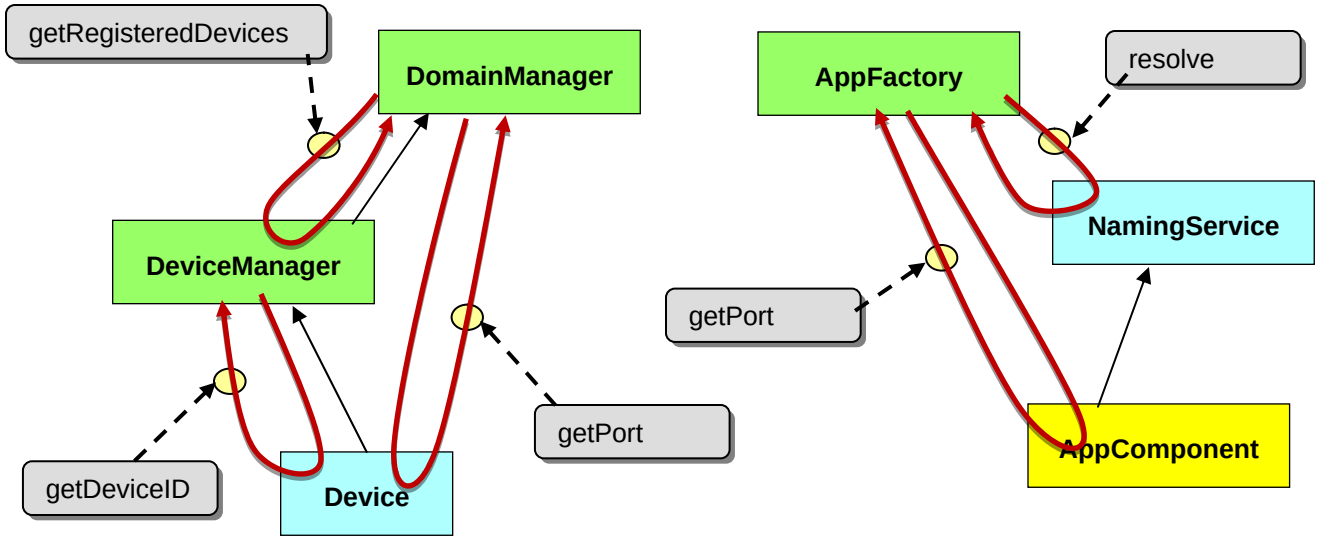
- **Objective**
  - To replace “pull model” registration behavior with “push model” behavior
  - To better support least privilege registration interfaces
  - To ensure all registration and port connection use cases are maintained or enhanced
- **Benefits**
  - Better Assurance
    - Opportunity for access can be limited to the push only
    - No Naming Service, App Components register directly with AppFactory
  - Better Performance
    - Less total number of calls involved
    - Reduces startup and instantiation time
    - Attributes can now become optional and when not used, can reduce the number of operations implemented
- **Impact**
  - Some interfaces refactored (see interface details)
  - Some registration behavior changes

Topics

- Overview
- Port Accessor
- Application Factory
- Device Manager
- Domain Manager
- Registered and Obtainable Provides Ports
- Interface Specifics
- Domain Profile Specifics

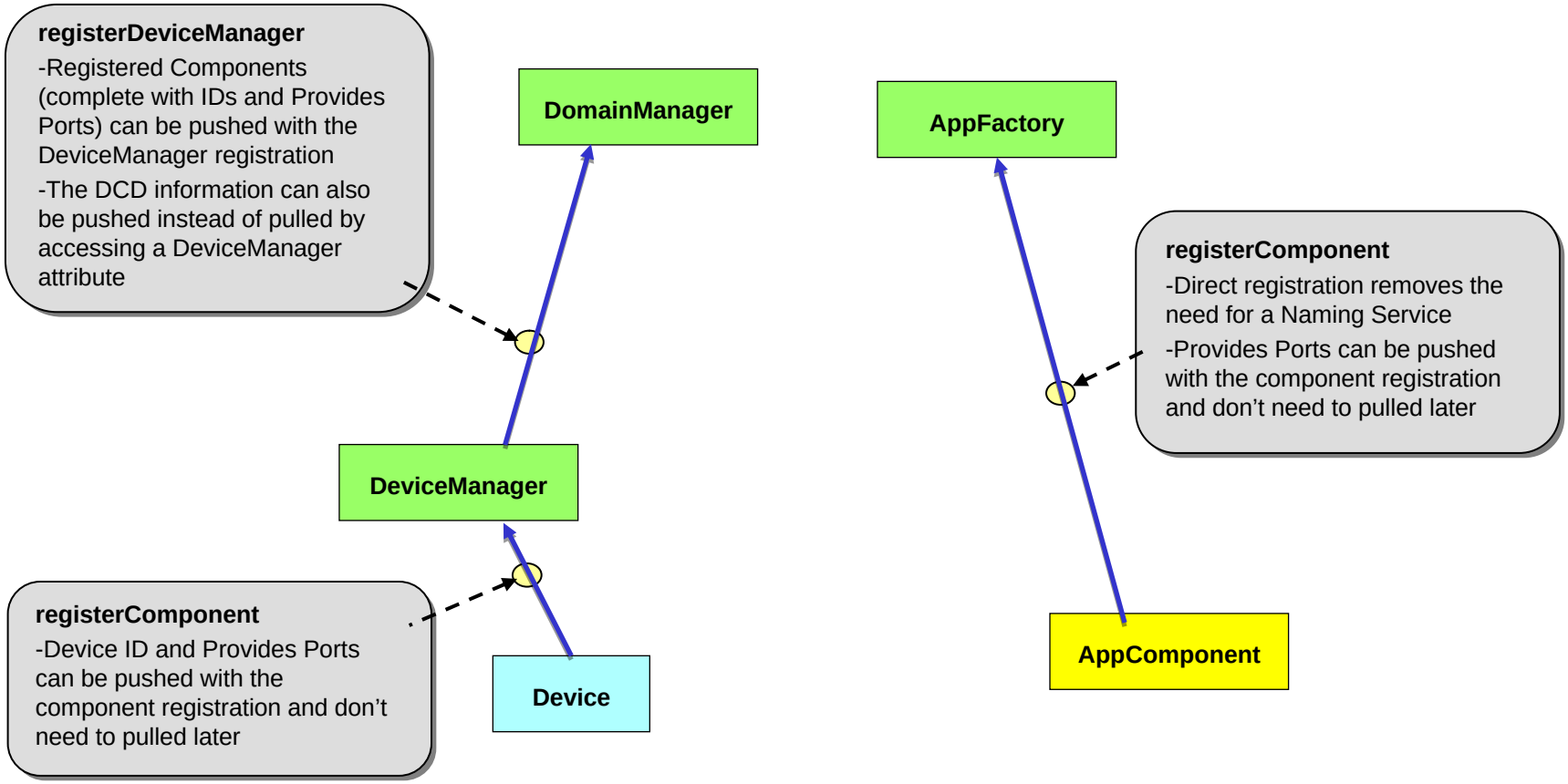
Overview

- **Pull Model**
  - SCA v2.2.2 and previous versions have relied heavily on a pull model
  - For example:
    - *getPort* for pulling uses and provides ports
    - Pulling attributes (e.g. deviceID, registeredDevices)
    - Pulling Application Components from a Naming Service



Overview

- **Push Model**

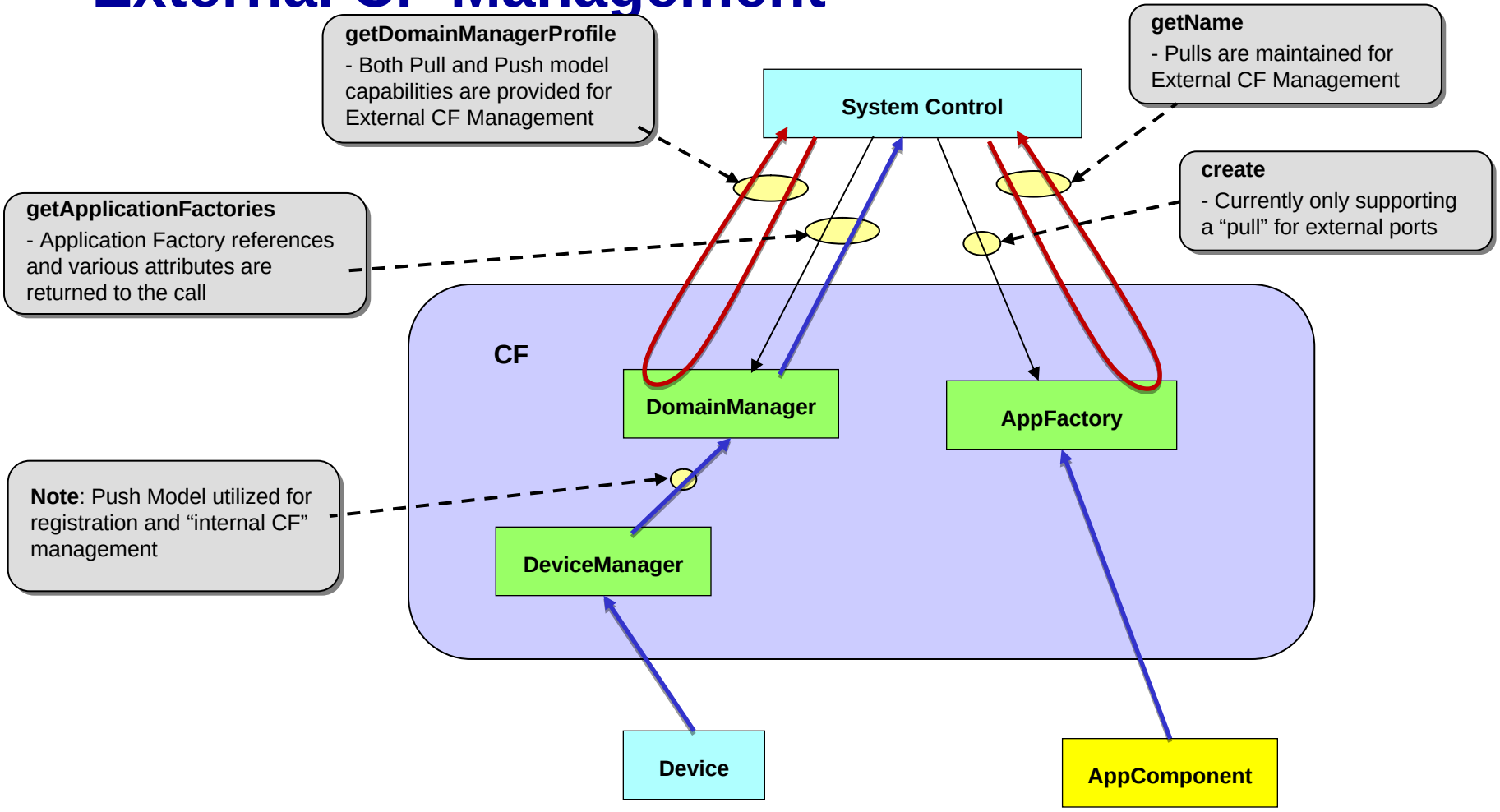


Overview

- **External CF Management**
  - Expand capability for a push model
    - "push" model currently supported in v2.2.2 through Events, but still requires some pulls
    - "push" information on various returns (e.g. *installApplication*, *create*) that would previously only been available via pulls
  - Continue to support pull model
    - Maintain "pull" type attributes (e.g. Domain Manager *applicationFactories* attribute)
  - Provides a good balance between performance and capability
    - Allows for greater performance when utilizing the push model for external management
    - Continues to support unique Use Cases where pulls may still be needed
    - Allows for backward compatibility
    - Doesn't violate the "least privilege" principle

Overview

- **External CF Management**



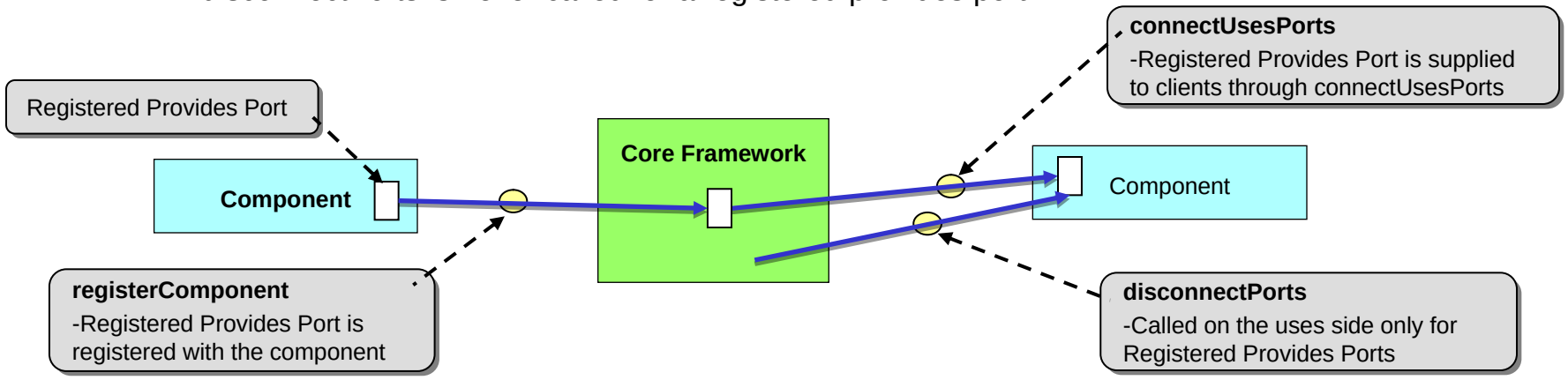
Overview

- **Refactored Registration Interfaces**
  - Refactored the registration interfaces into smaller, more concise interfaces that standalone
    - Ensures only the methods needed for registration are provided to registering components
  - Better Assurance
    - Follows “least privilege” principles
  - Better Performance
    - Opportunity to convert from a Pull to a Push Model
- **Refactored PortSupplier Interface**
  - Refactored PortSupplier to allow for direct sharing of connection information with a component
    - Eliminates the need for separate Uses Port servants
  - Better Performance
    - No need to obtain (whether push or pull) separate Uses Ports
    - Can make several connections with a single call
  - Better Functionality
    - Adds better support for “obtainable” provides ports
    - Adds a “release” on the provides side, which allows for provides ports to have a lifecycle fully tied to a “connection”

Registered and Obtainable Provides Ports

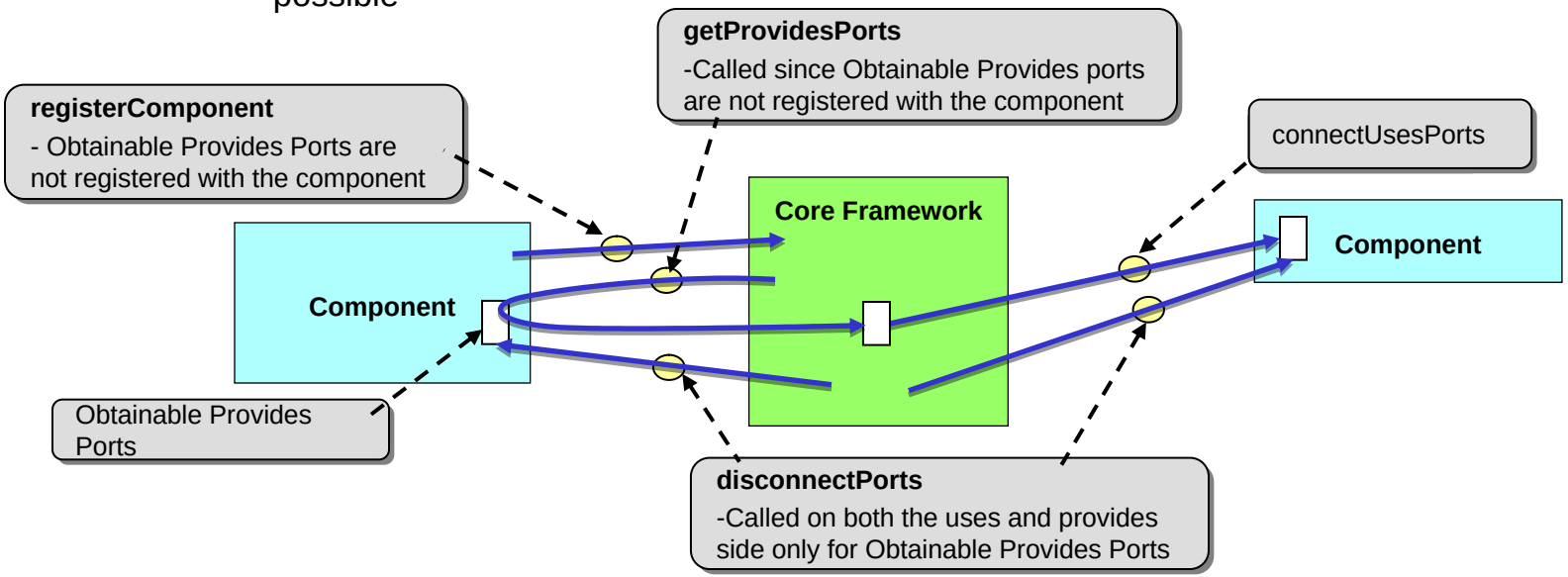
- **Two types of provides ports**
  - The new PortAccessor provides formal support for two types of provides ports, “Registered” and “Obtainable”

- **Registered Provides Ports**
  - Registered provides ports are provides ports which have a lifecycle tied to the lifecycle of the **component**
  - Registered provides ports are registered with the component and CF will not attempt to retrieve them when making connections
    - *getProvidesPorts* is typically not expected to be called for provides ports that are registered with the component
  - Registered provides ports are not explicitly released by CF except through the component’s *releaseObject* method
    - *disconnectPorts* is never called for a registered provides port



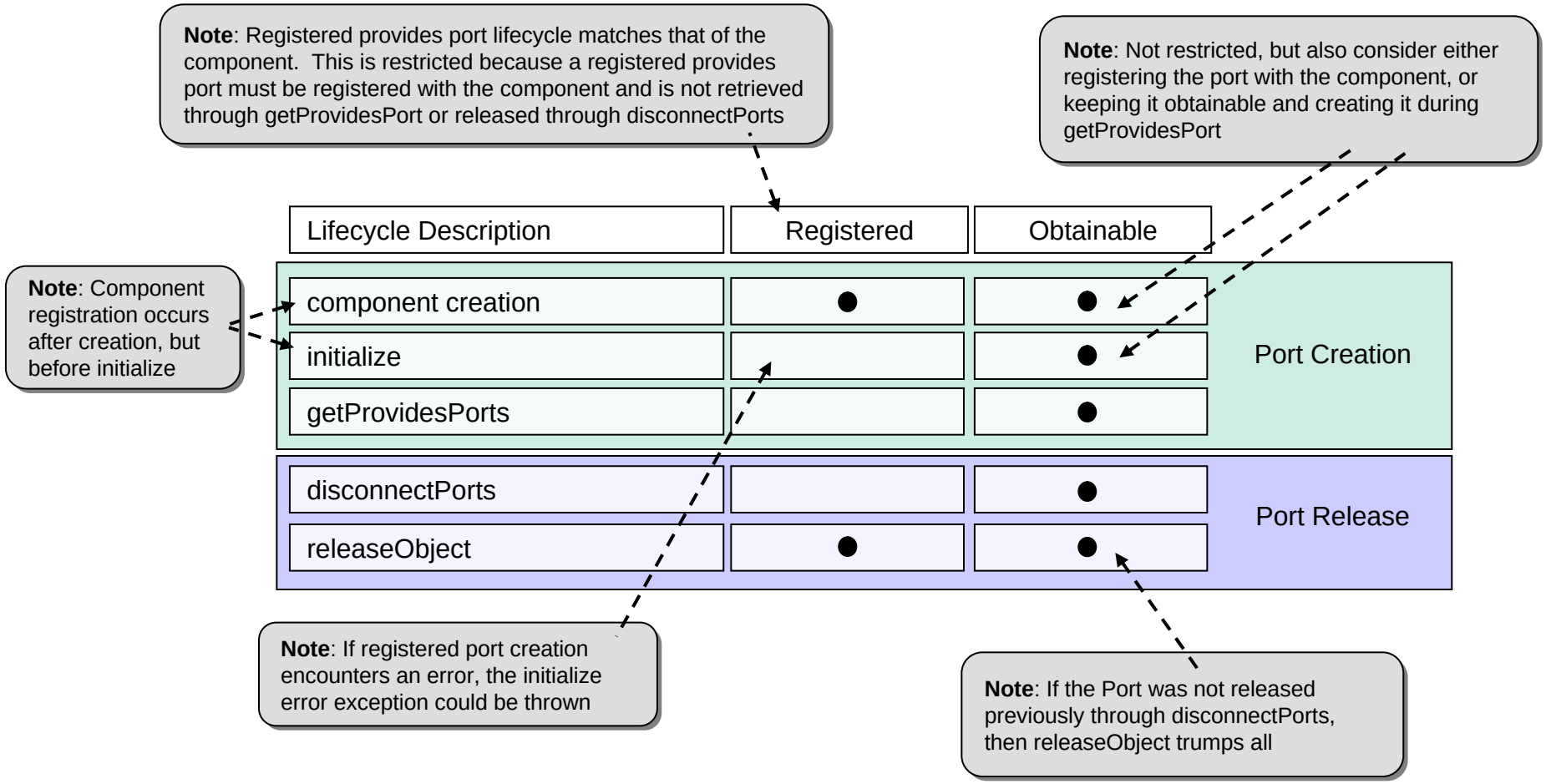
Registered and Obtainable Provides Ports

- **Obtainable Provides Ports**
  - Obtainable provides ports are ports which have a lifecycle tied lifecycle of the **connection**
  - Obtainable provides ports are *not* registered with the component and instead CF will attempt to retrieve the port when making the connection
    - *getProvidesPorts* is called for obtainable provides ports since they are not registered with the component
  - Obtainable provides ports *are* explicitly released by CF when the connection is torn down
    - *disconnectPorts* is always called for obtainable provides ports
  - Obtainable provides ports support added functionality not available with v2.2.2
    - *getProvidesPorts* with connectionIDs and *disconnectPorts* call on the provides side make this possible

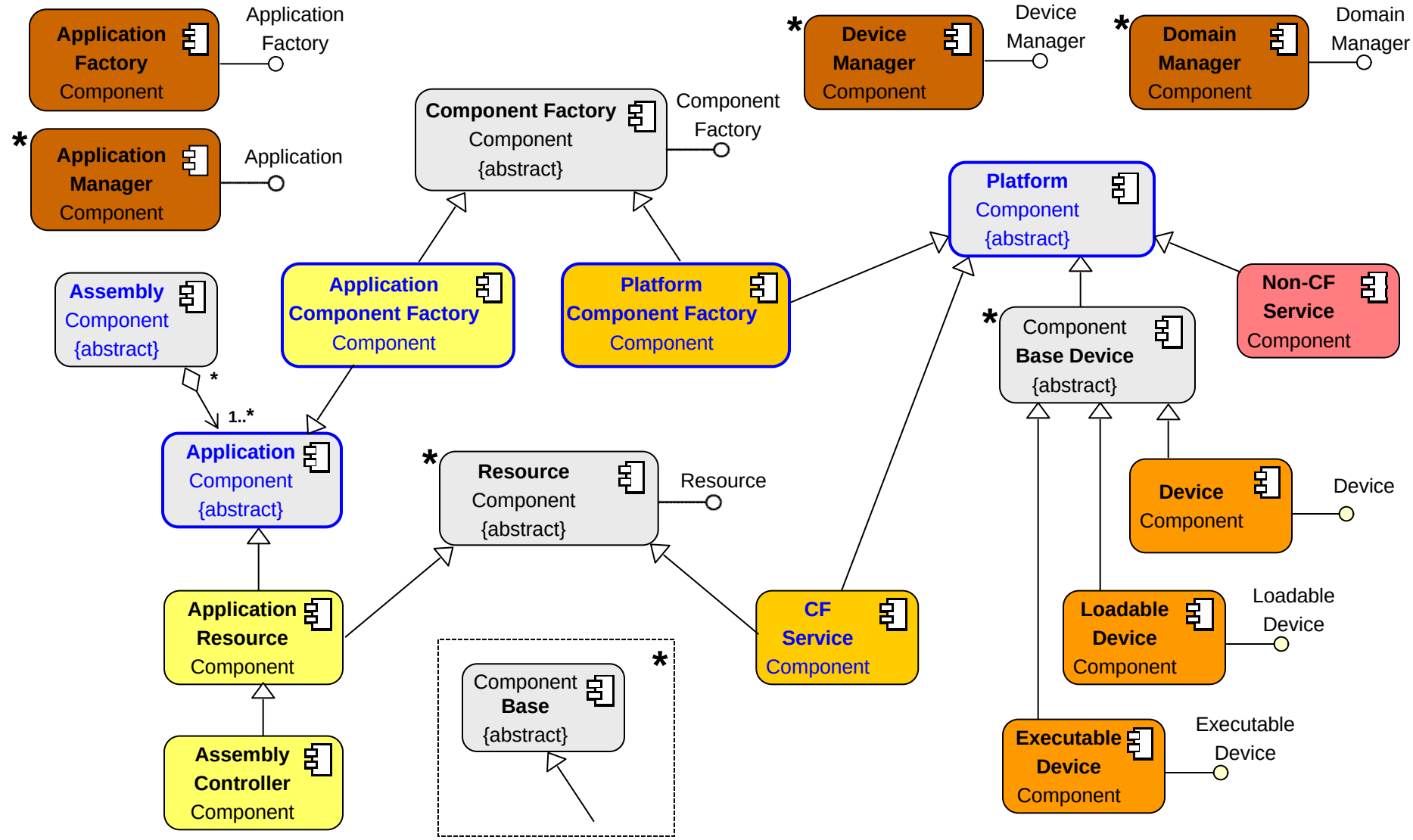


Registered and Obtainable Provides Ports

- **Provides Ports Lifecycle**



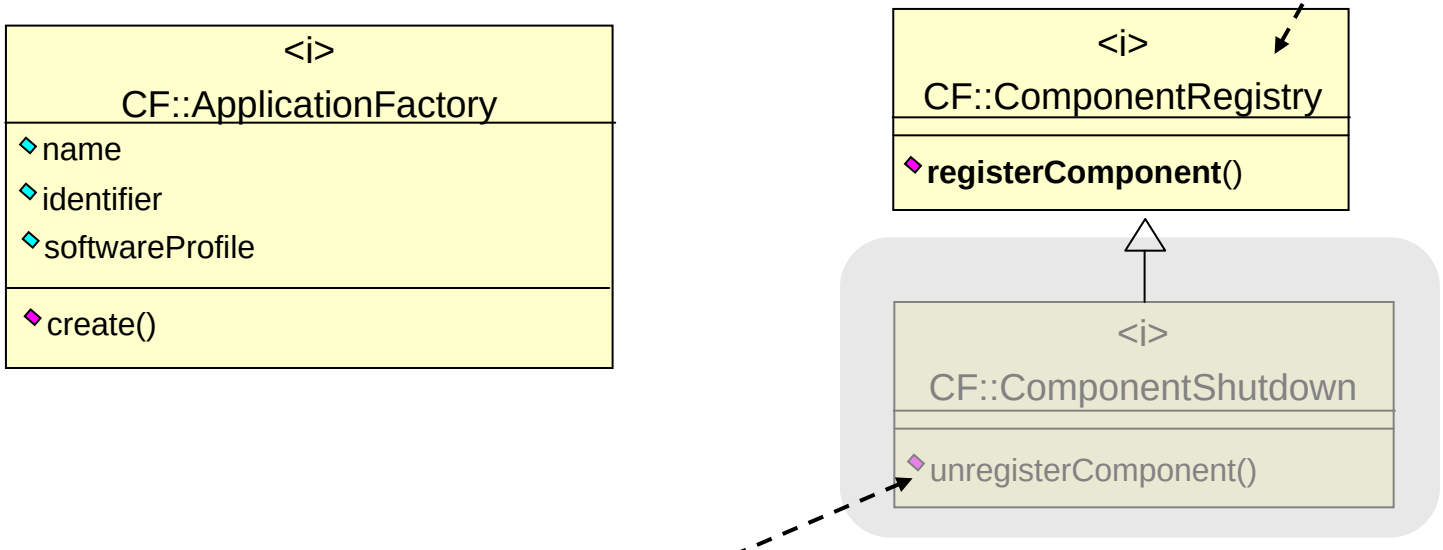
Component Hierarchy



ApplicationFactory Component Registration

Note: Instead of going through the Naming Service, Application (e.g. WF) Components should register directly with the ApplicationFactory.

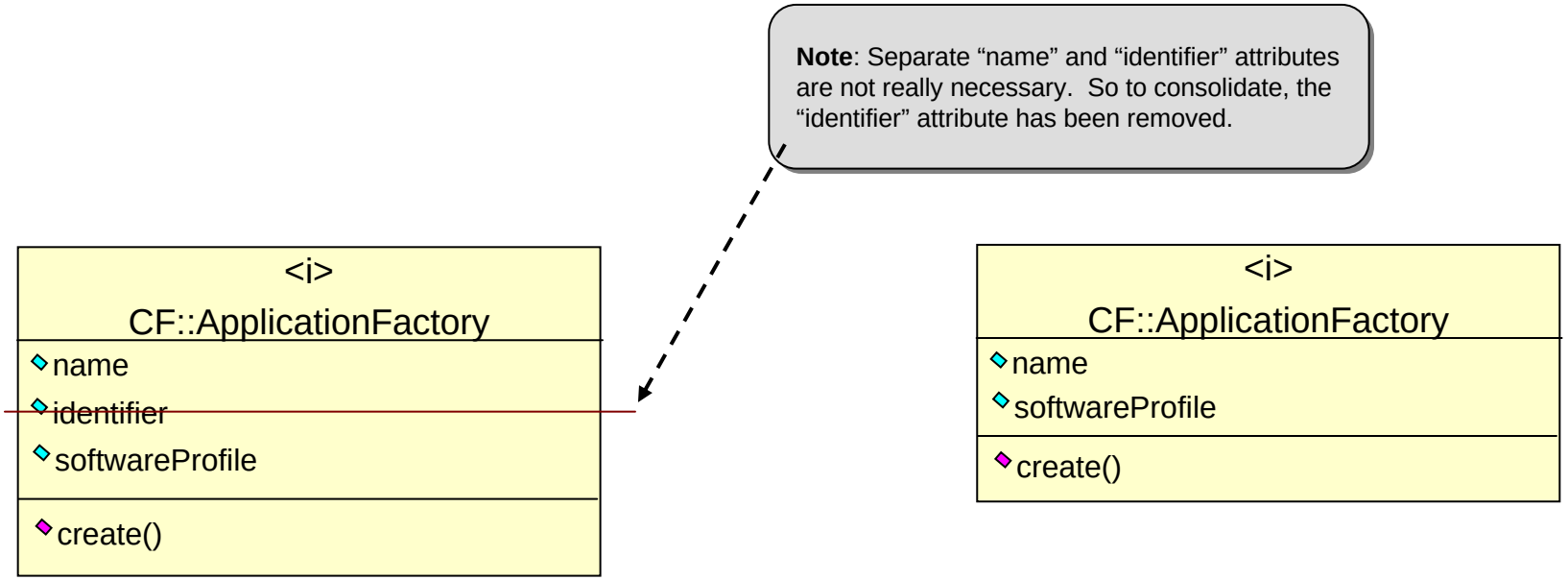
A new ComponentRegistry Interface is standalone from the ApplicationFactory interface in order to maintain least privilege.



Note: The unregisterComponent call is not needed by ApplicationFactory, so only the ComponentRegistry is used here. The ComponentShutdown is needed for OE components and will be used by Device and Domain Managers.

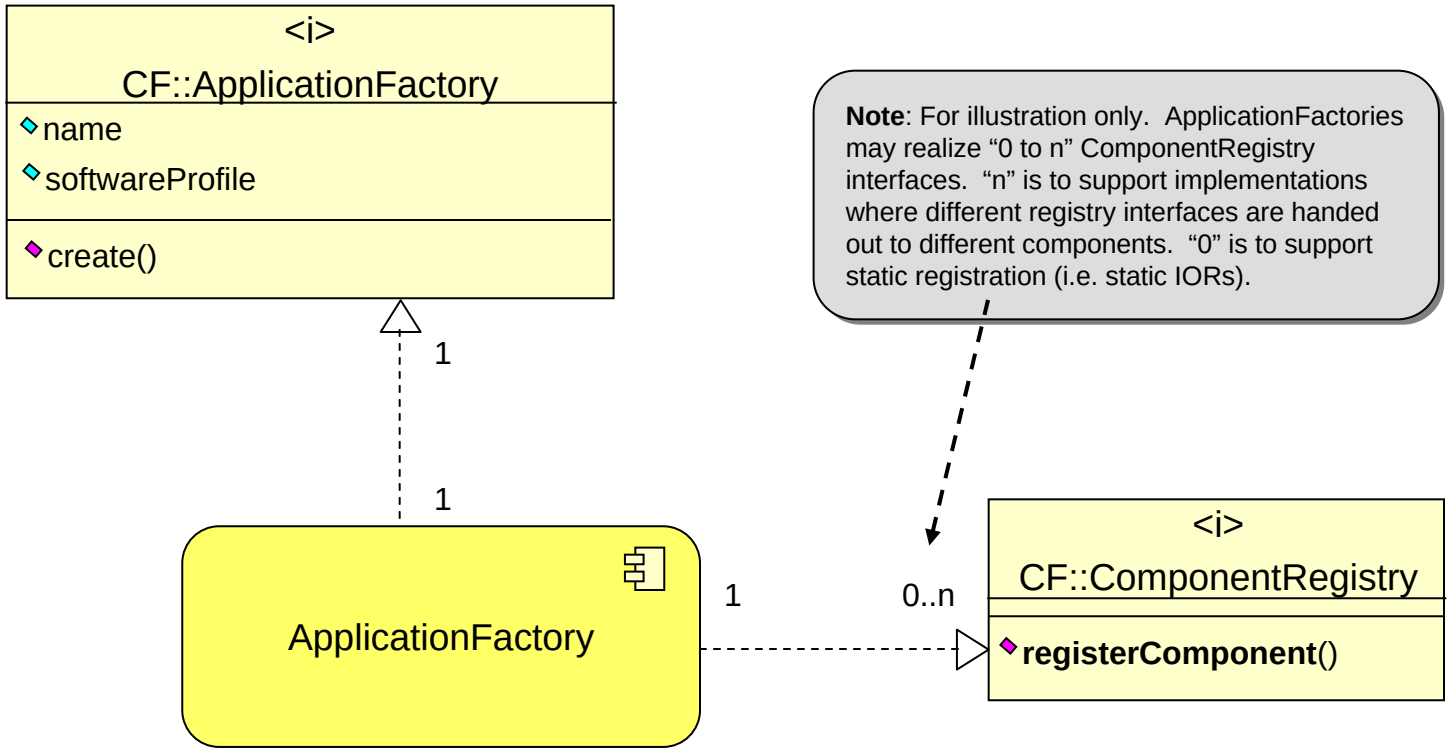
The use of unregisterComponent by WFs would be similar to WFs calling NS unbind in SCA v2.2.2 (this was not required in v2.2.2, but was sometimes implemented). In the SCA Next Primer, it should be made clear that these WFs don't need to replace that functionality with unregisterComponent (hence the reason unregisterComponent is not part of AppFactory).

ApplicationFactory Attribute Refactoring



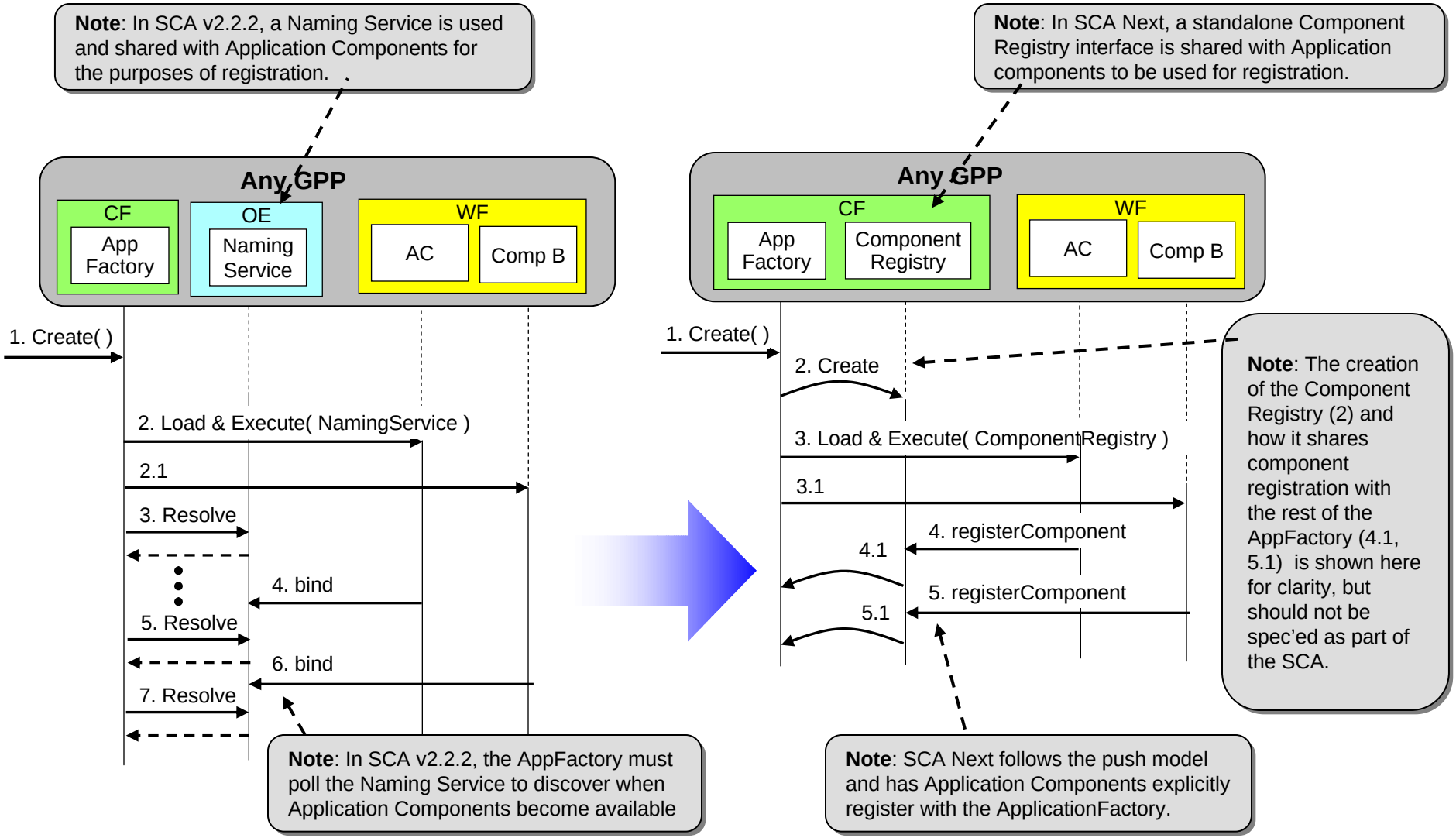
Proposed SCA Next

ApplicationFactory Component Model



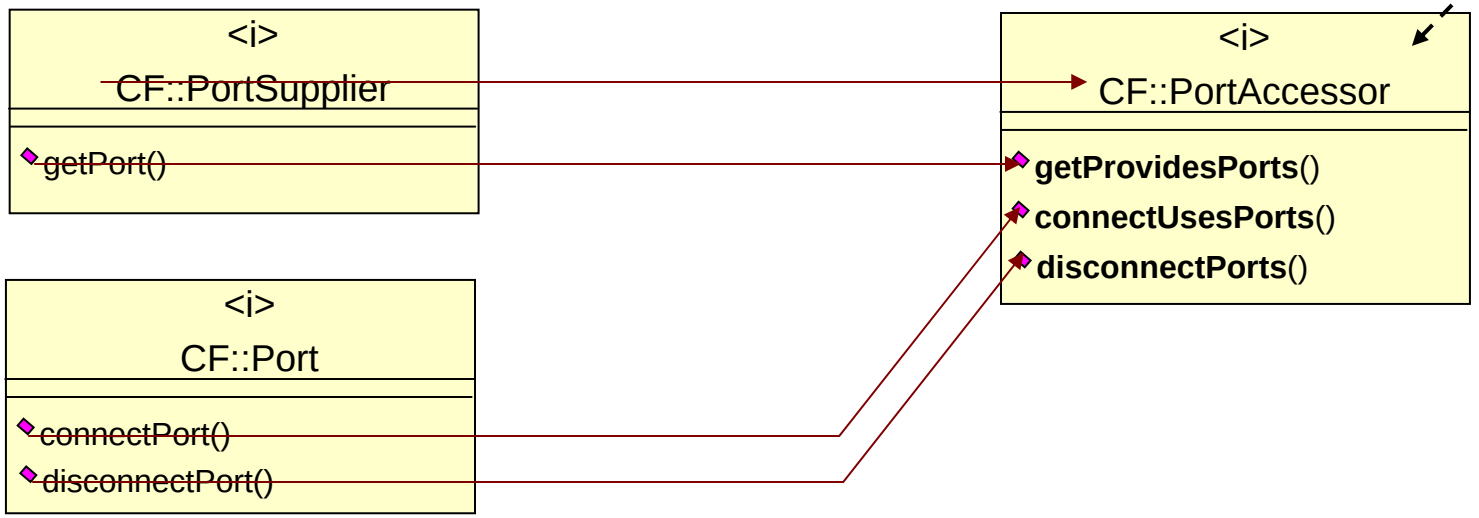
Proposed SCA Next

ApplicationFactory Component Registration



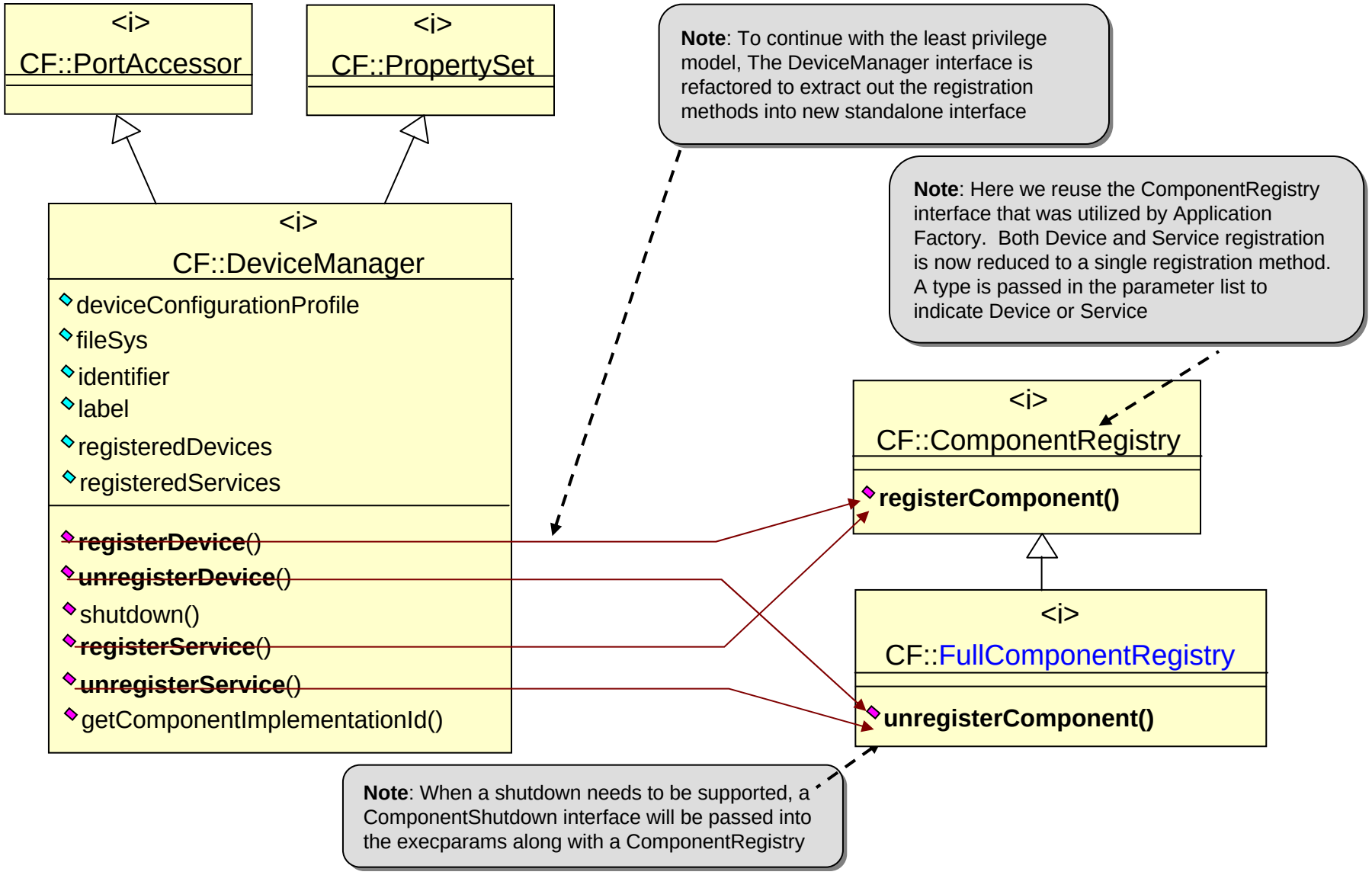
Port Accessor

Note: SCA Next builds upon the PortSupplier and Port concepts to create a PortAccessor, which provides a means of fully, directly accessing uses and obtainable provides ports to build connections.

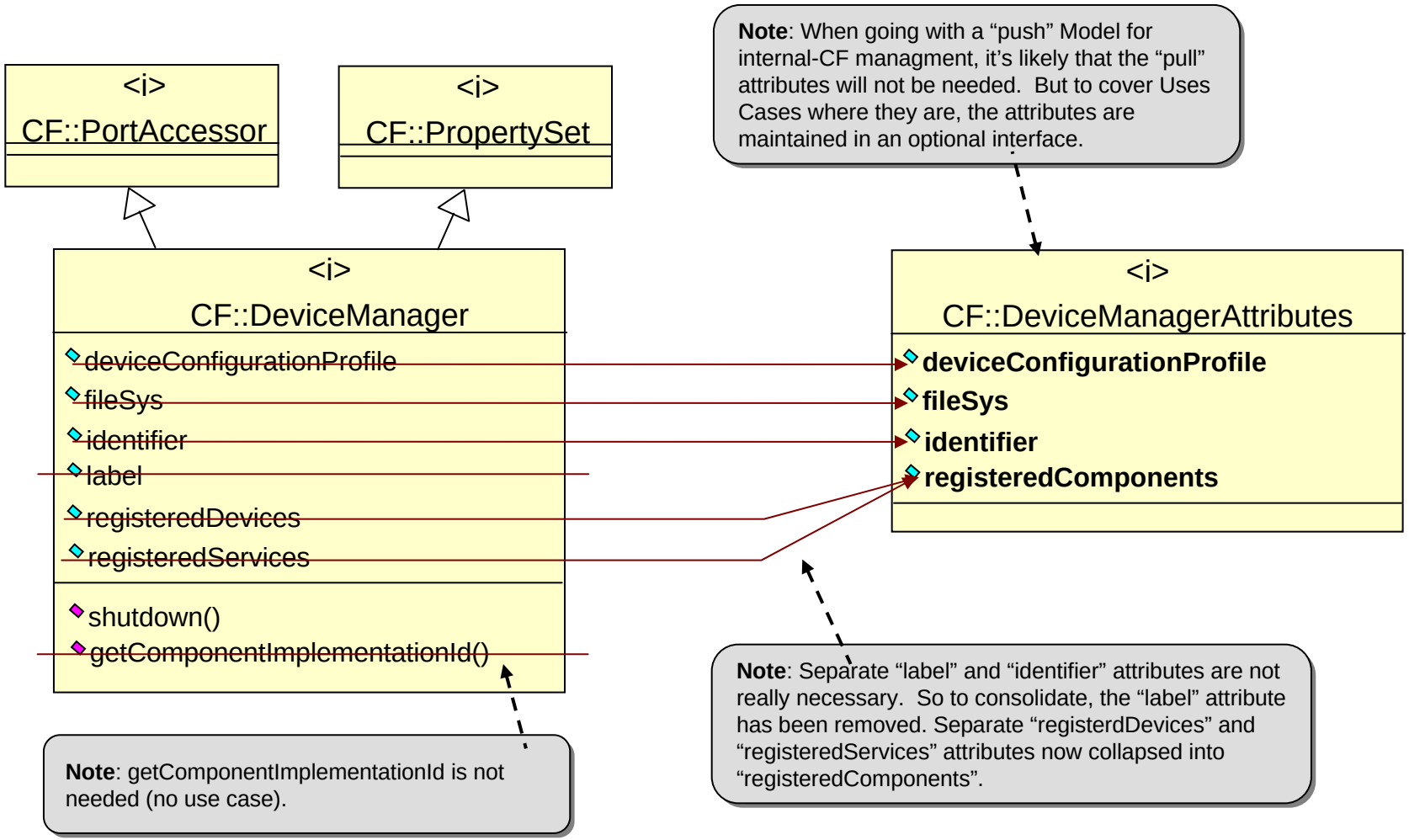


Proposed SCA Next

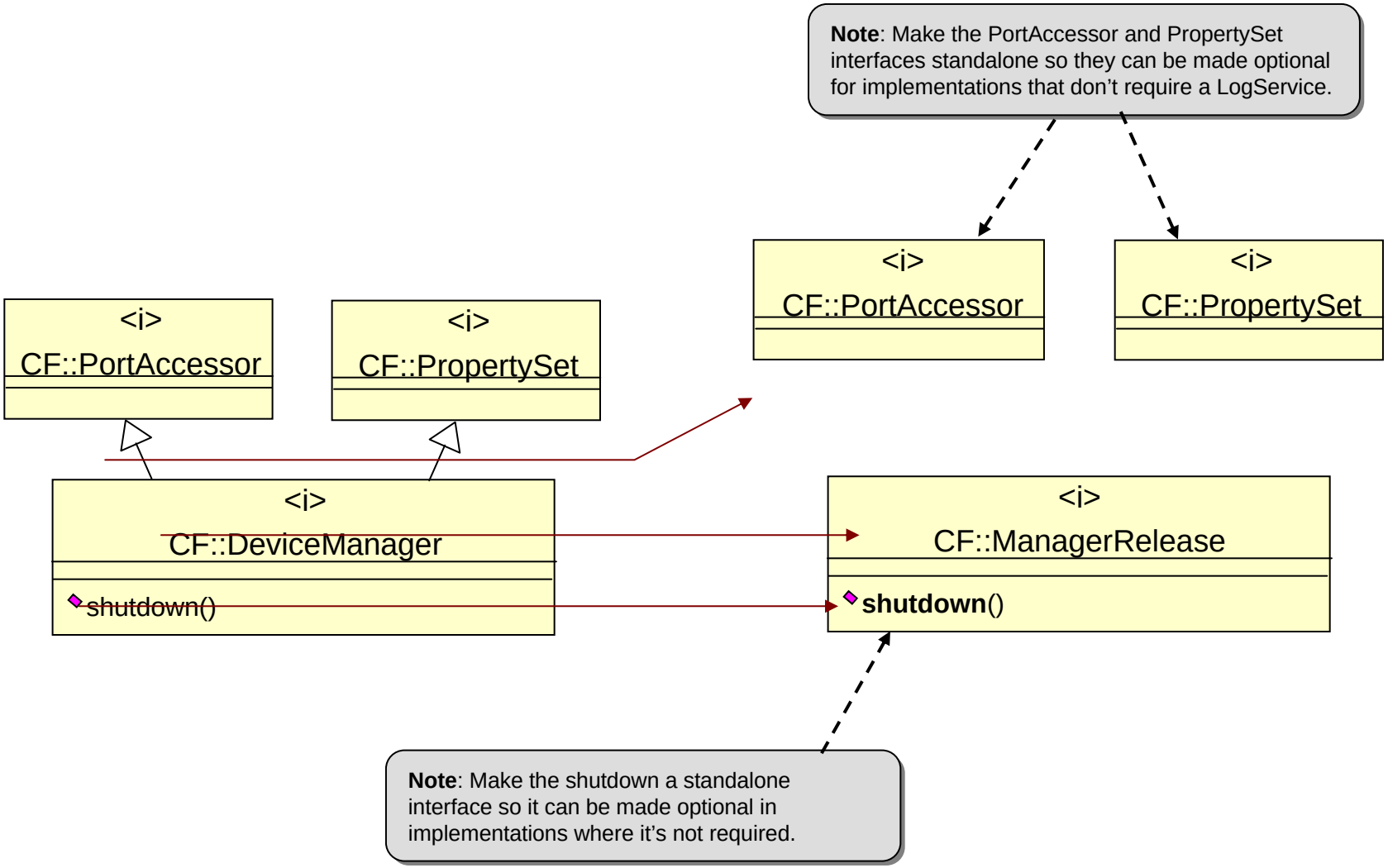
DeviceManager Component Registration



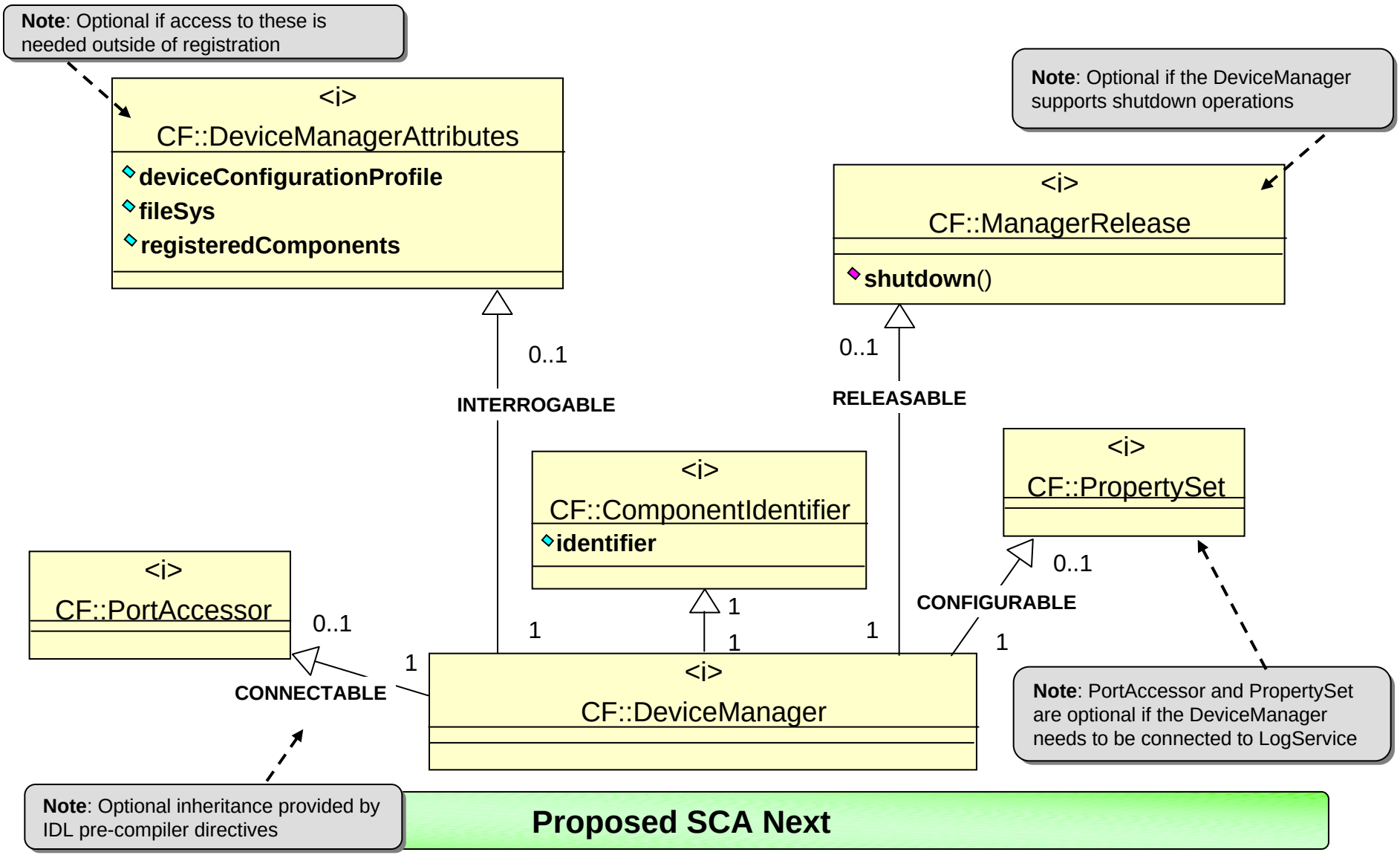
DeviceManager Attributes Refactoring



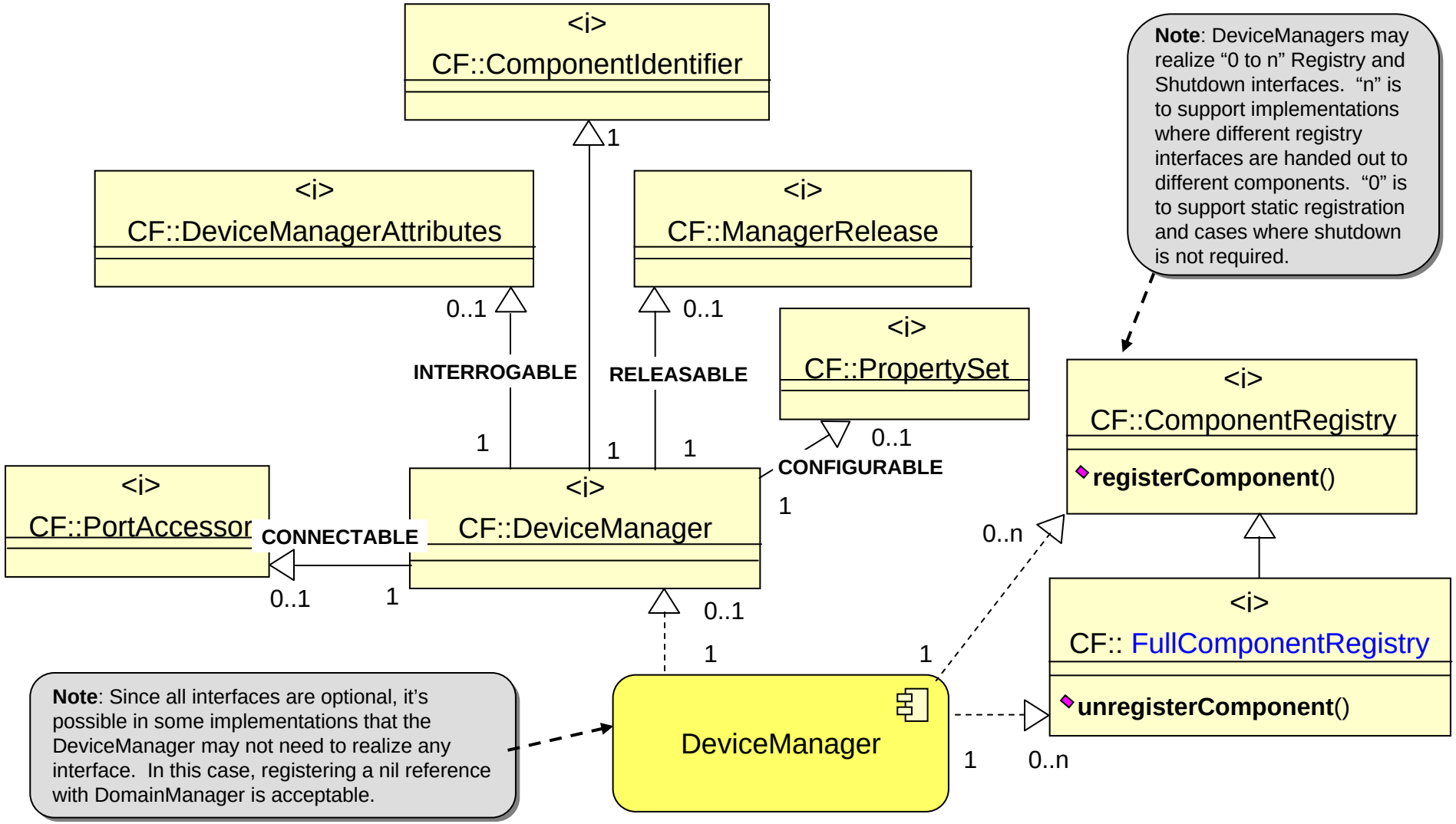
DeviceManager
Shutdown and LogService Interfaces



DeviceManager Optional Inheritance

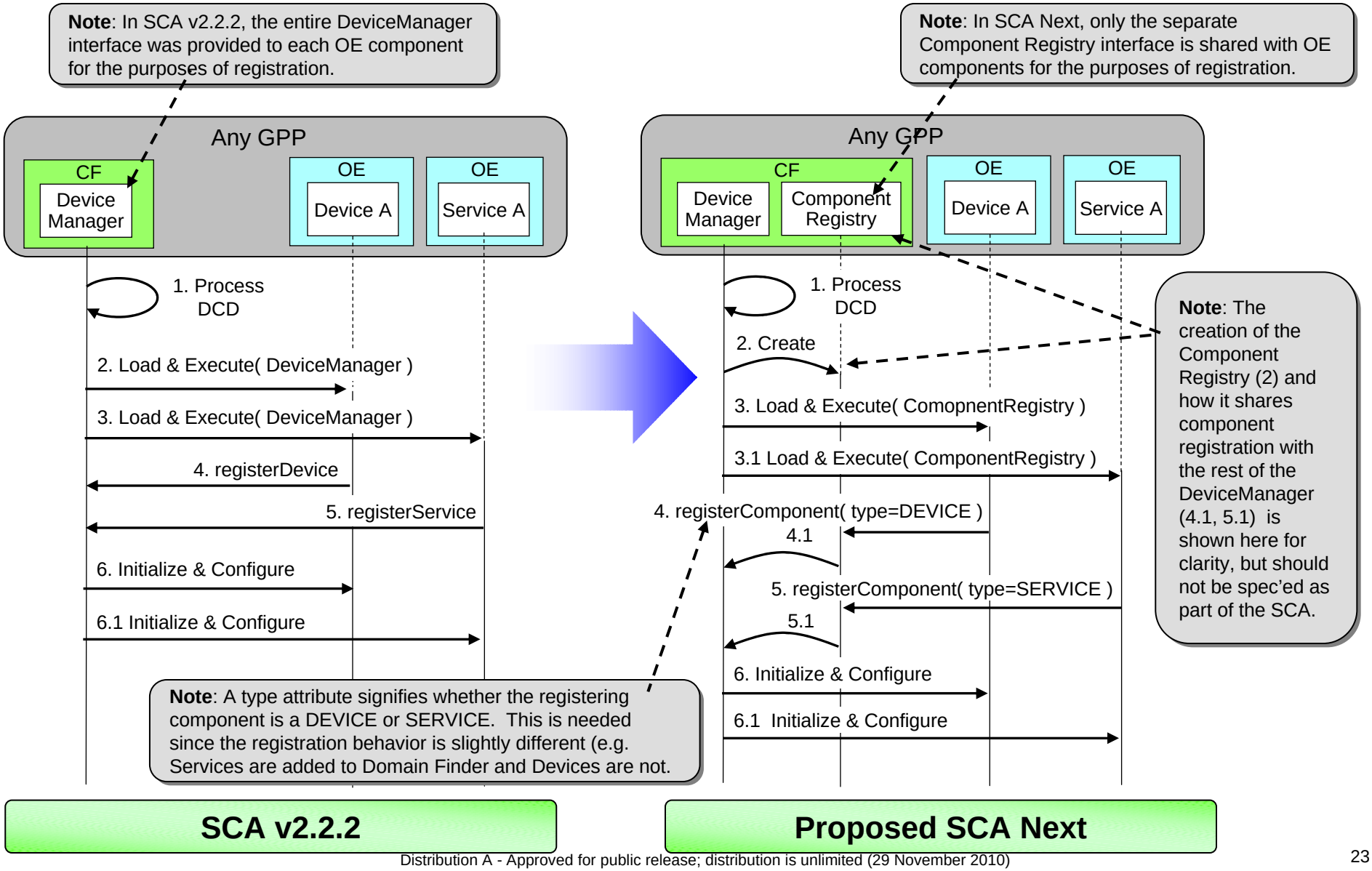


DeviceManager Component Model

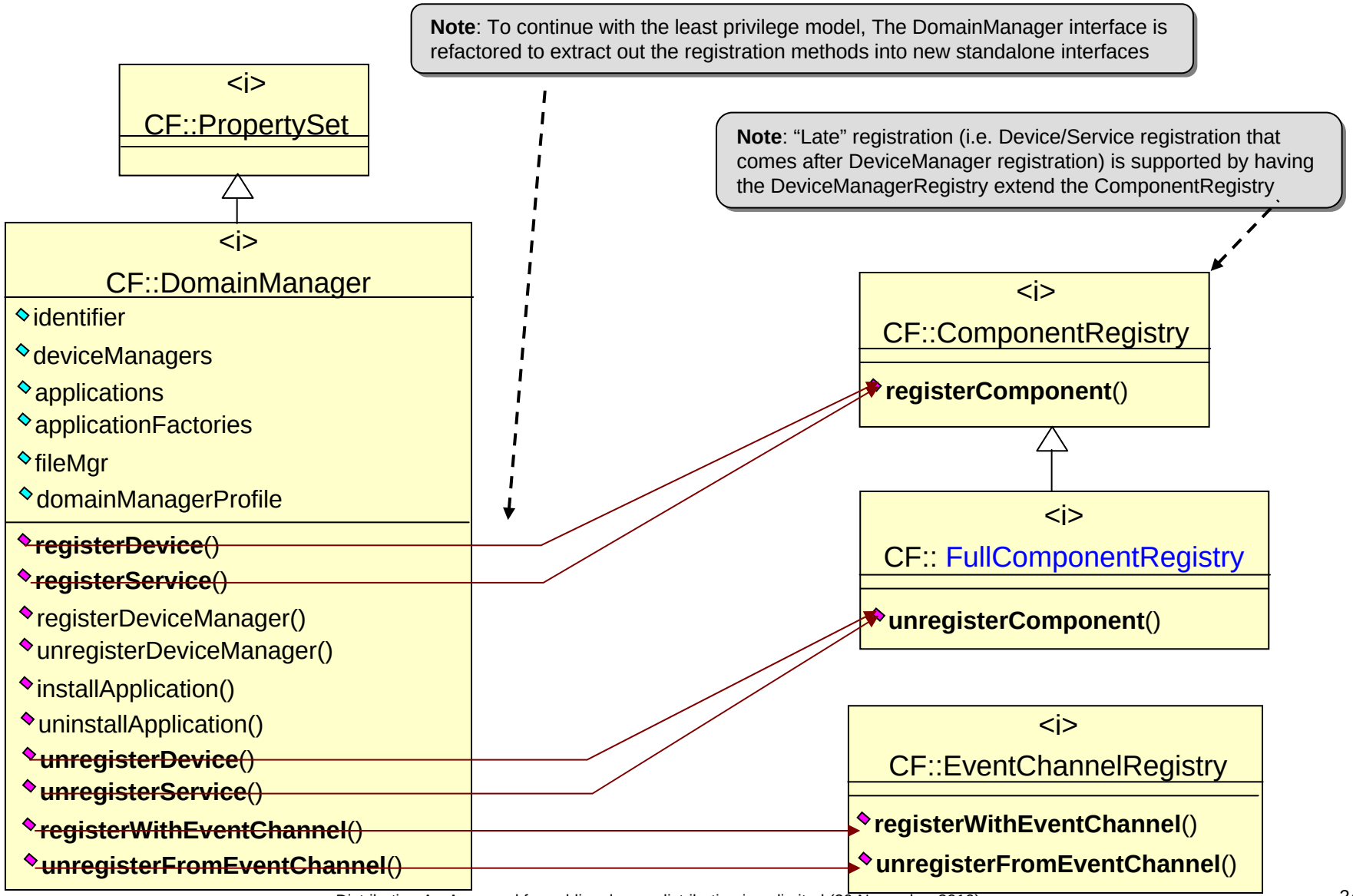


Proposed SCA Next

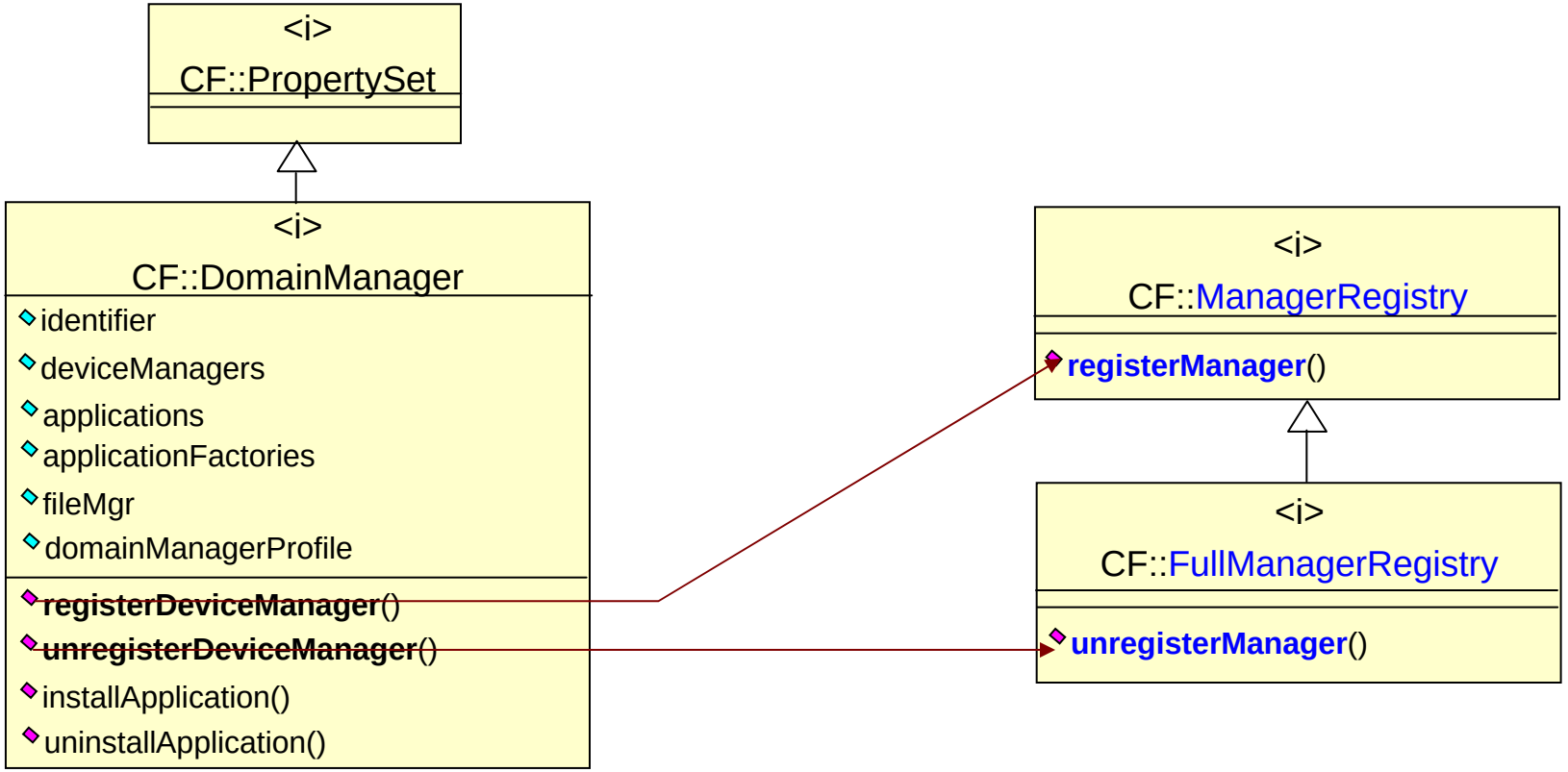
DeviceManager Component Registration



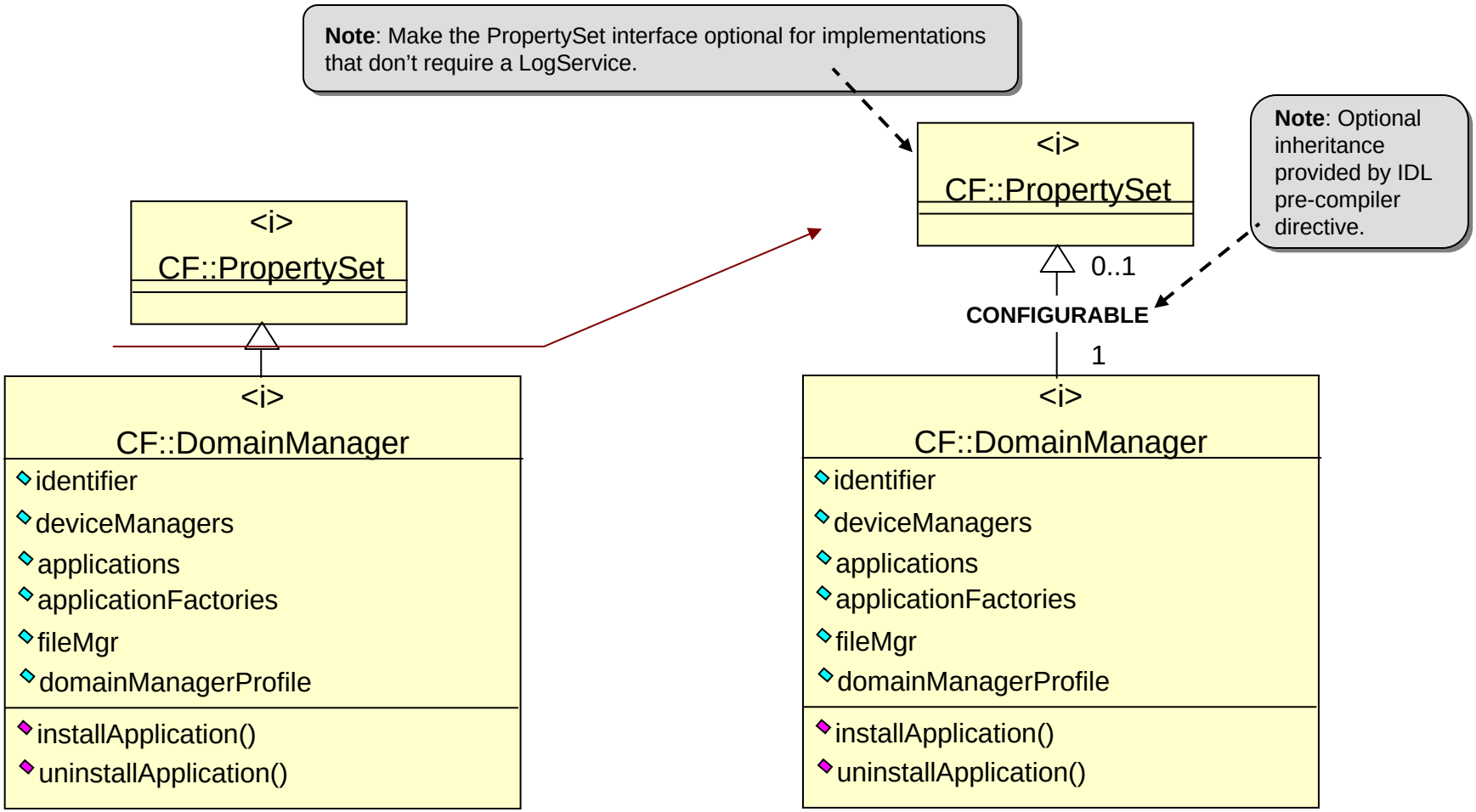
DomainManager
DeviceManager / Component Registration



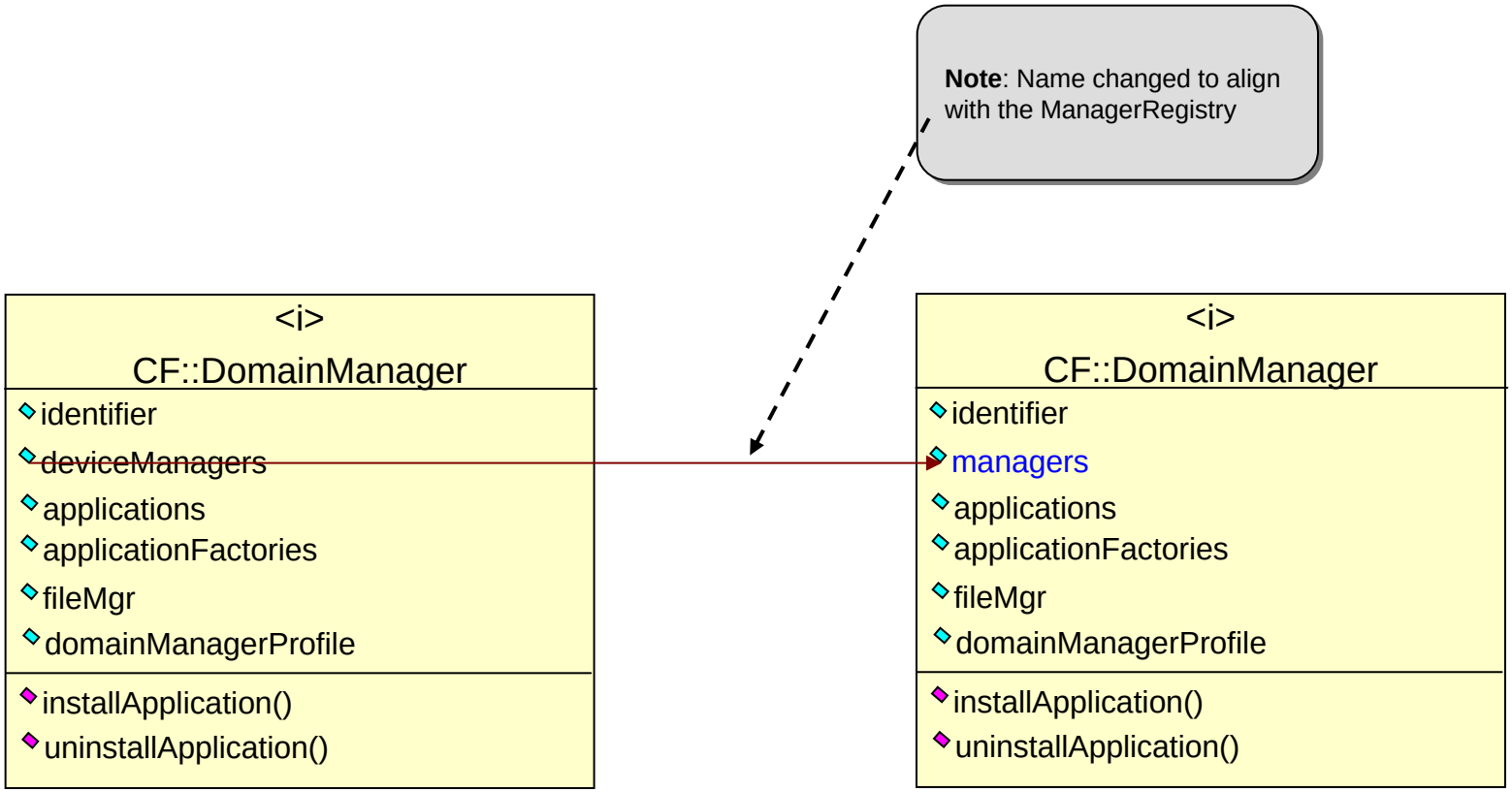
DomainManager
Manager / Component Registration



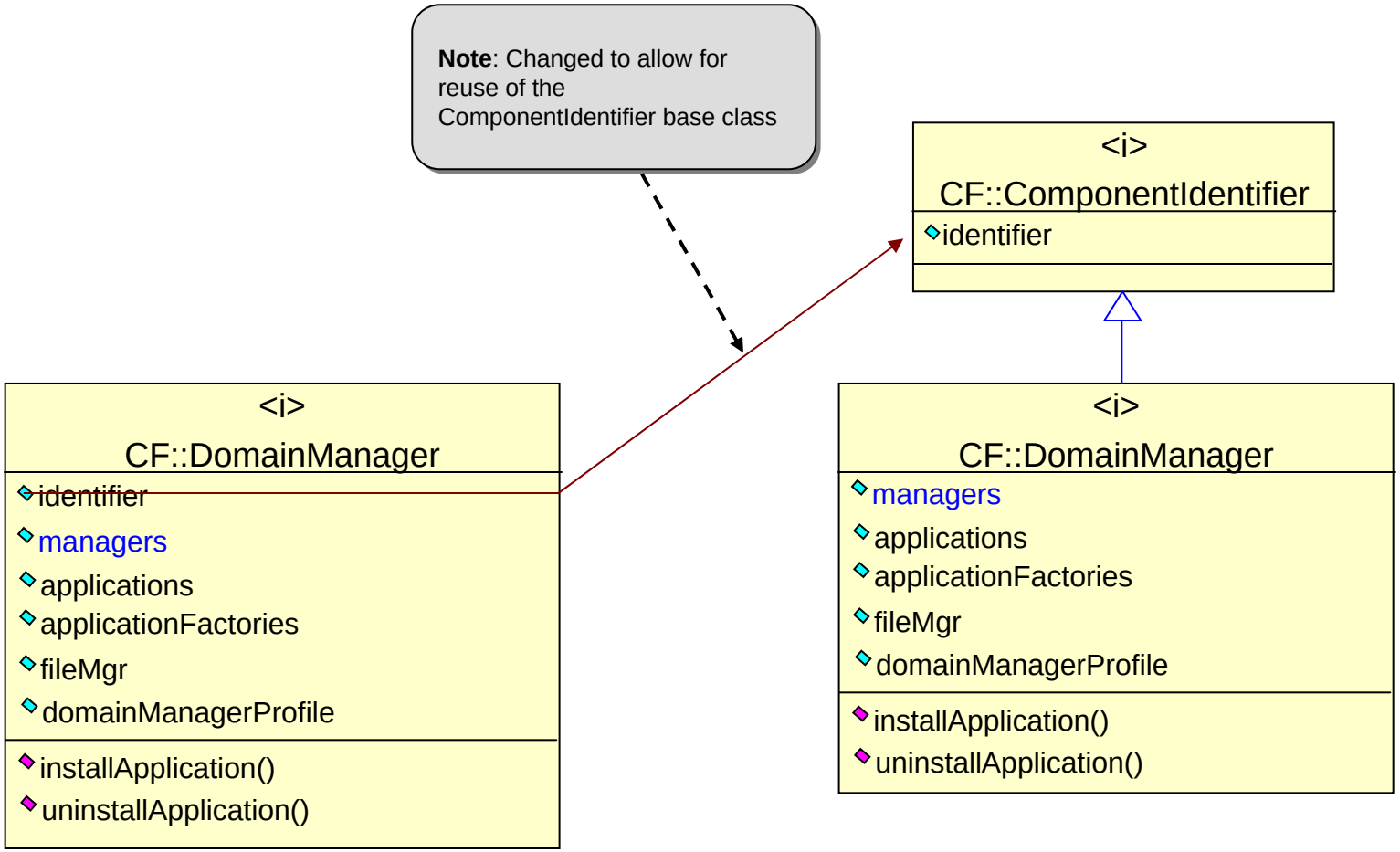
DomainManager
Manager / Component Registration



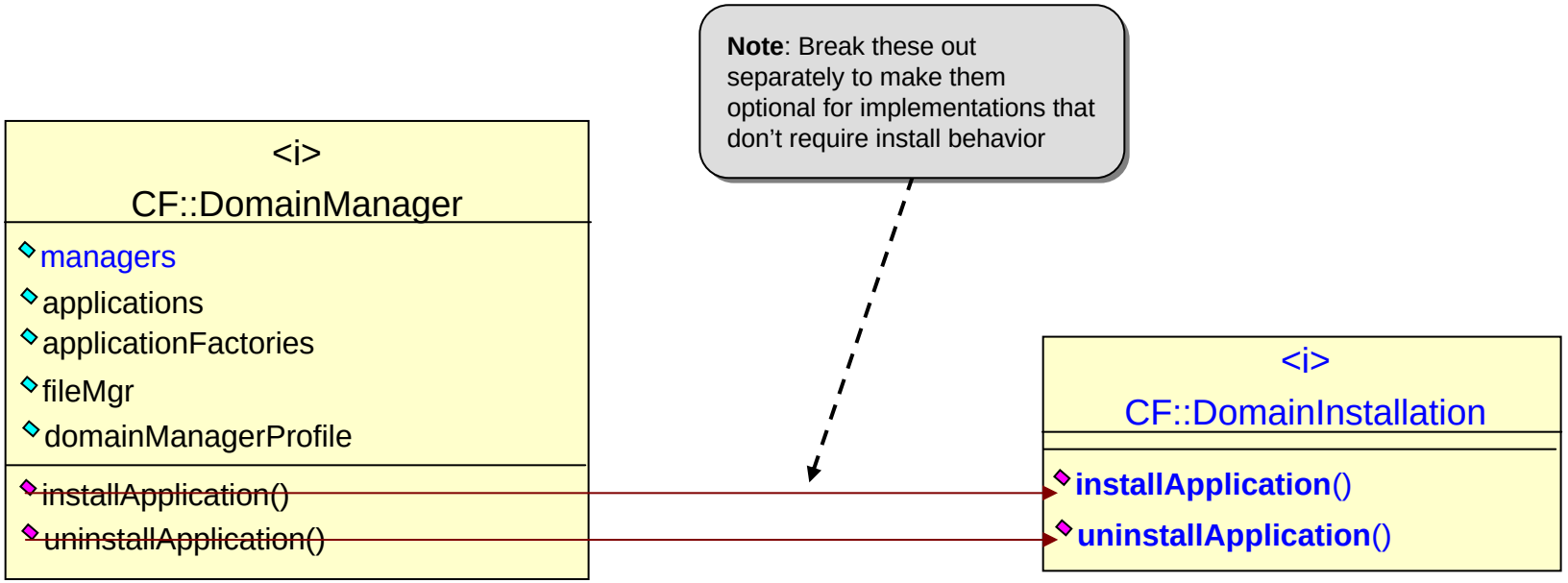
DomainManager
Manager / Component Registration



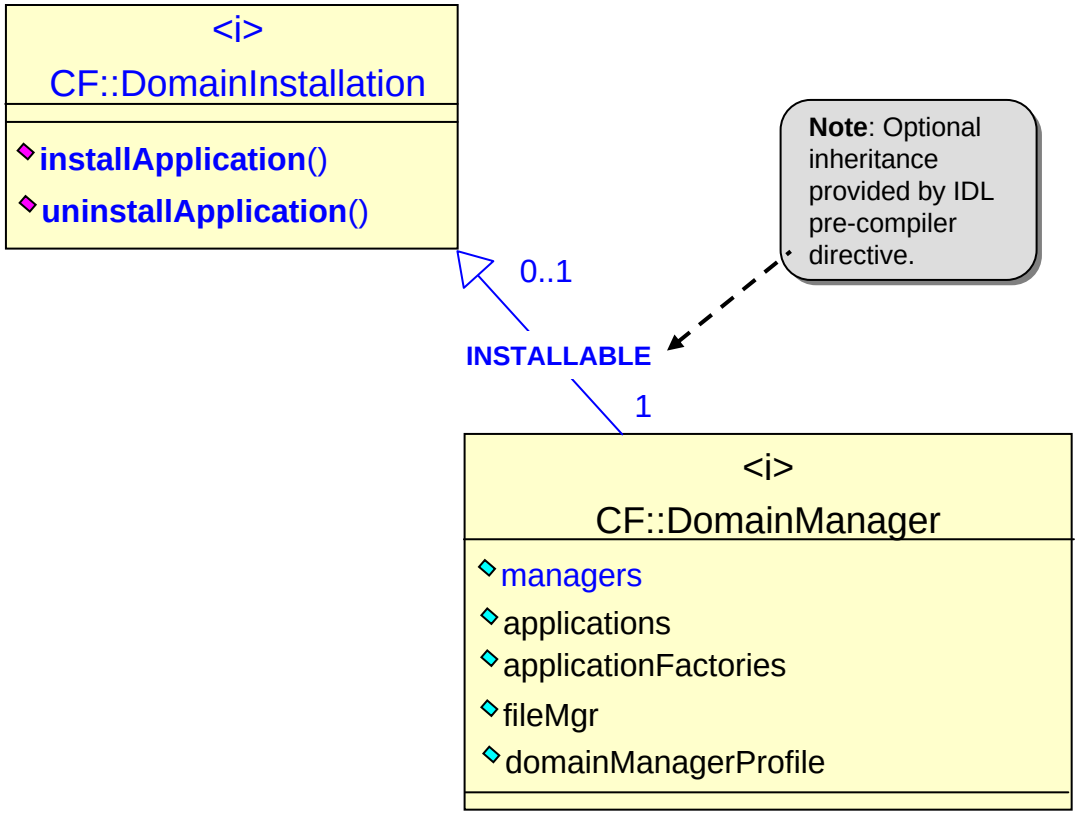
DomainManager
Manager / Component Registration



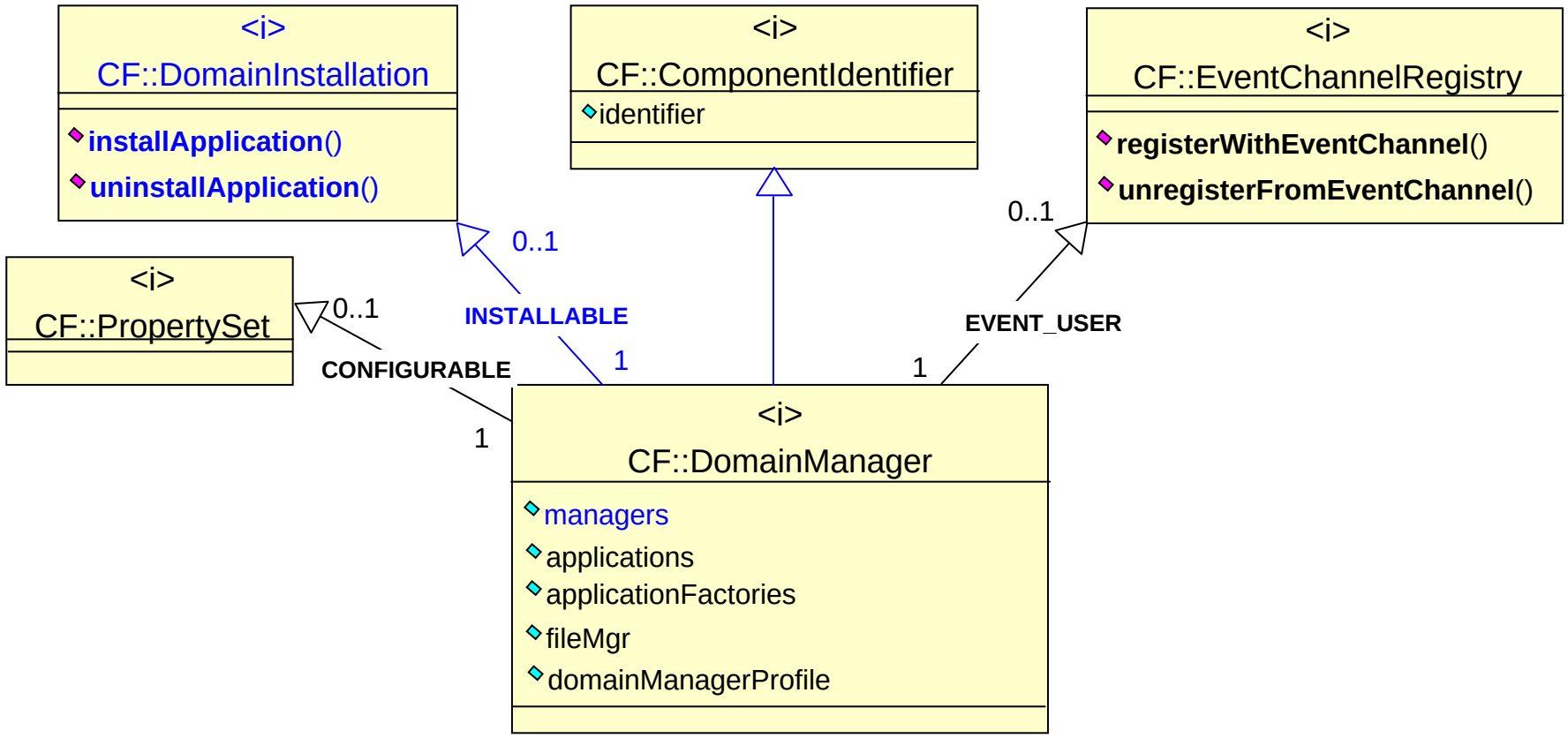
DomainManager
Manager / Component Registration



DomainManager
Manager / Component Registration

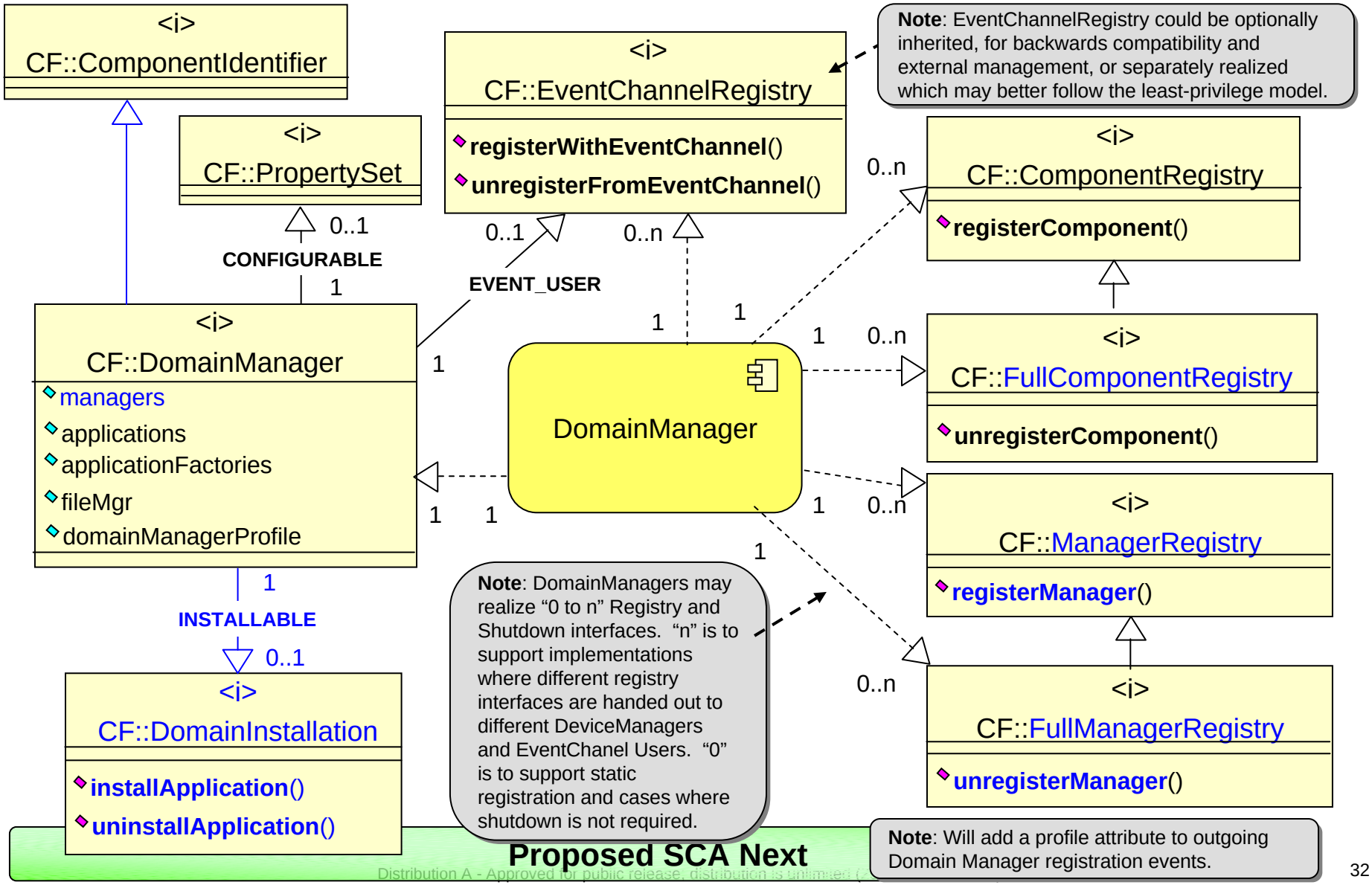


DomainManager
Manager / Component Registration

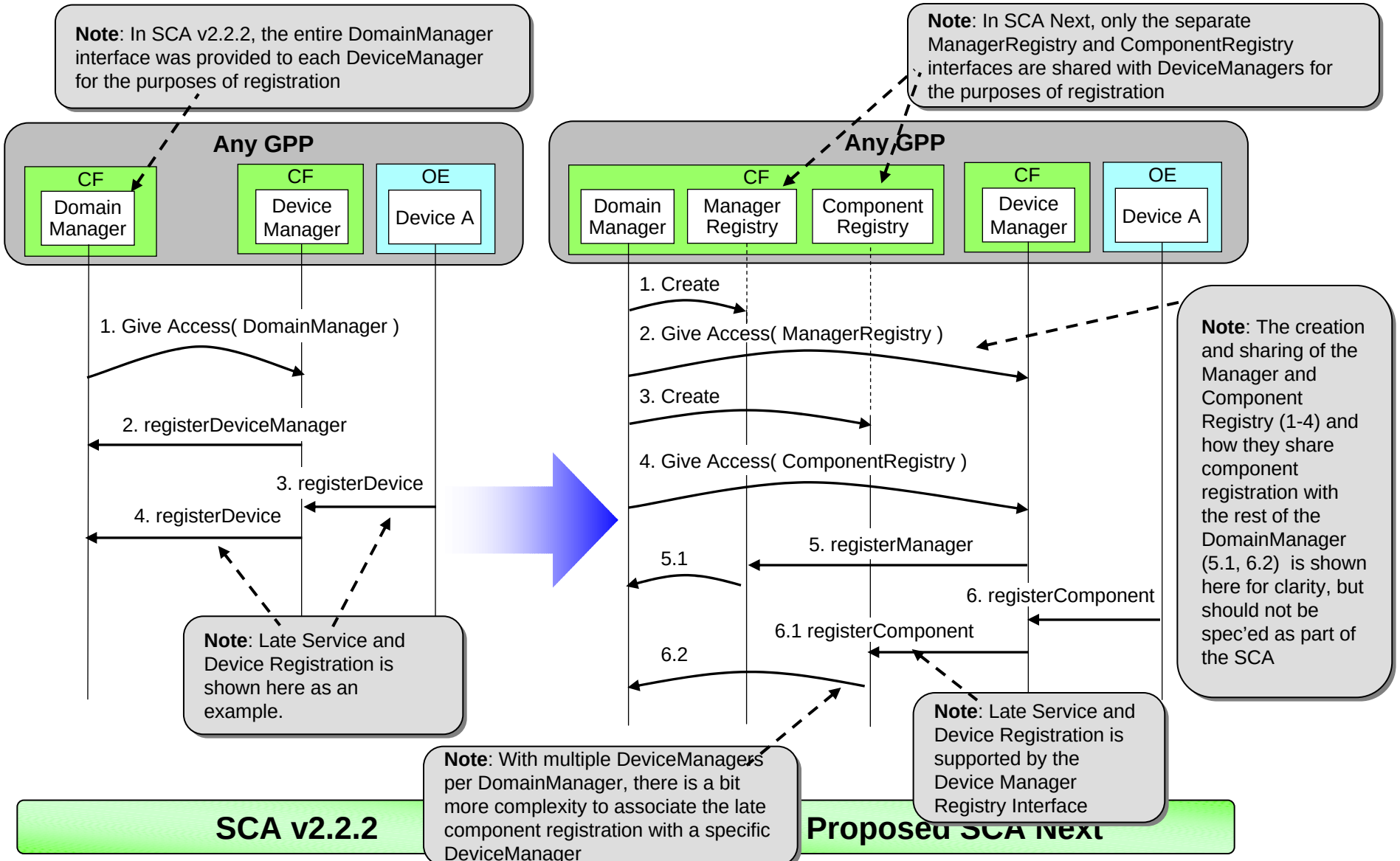


Proposed SCA Next

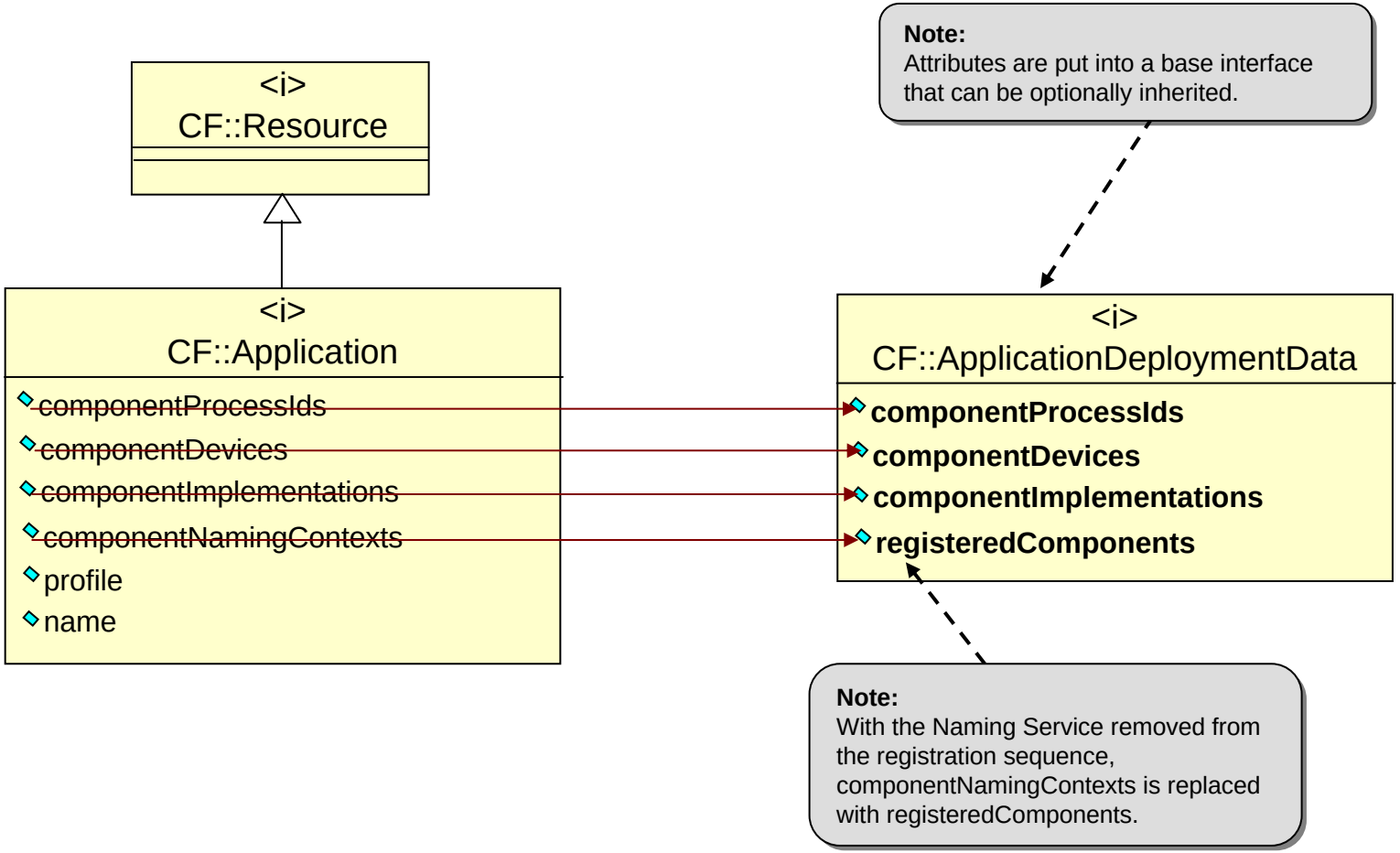
DomainManager Component Model



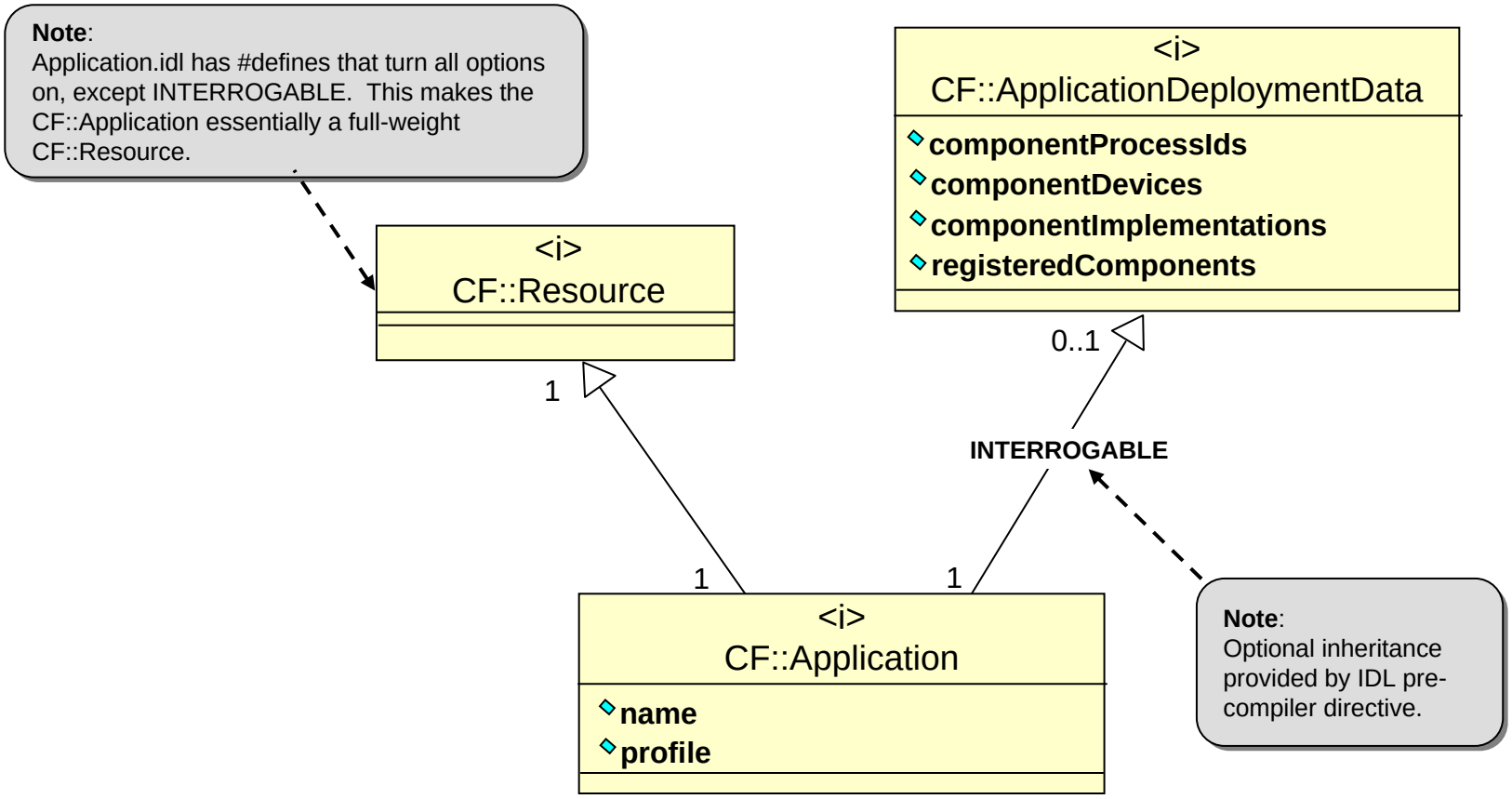
DomainManager
DeviceManager / Component Registration



Application Interface Refactoring

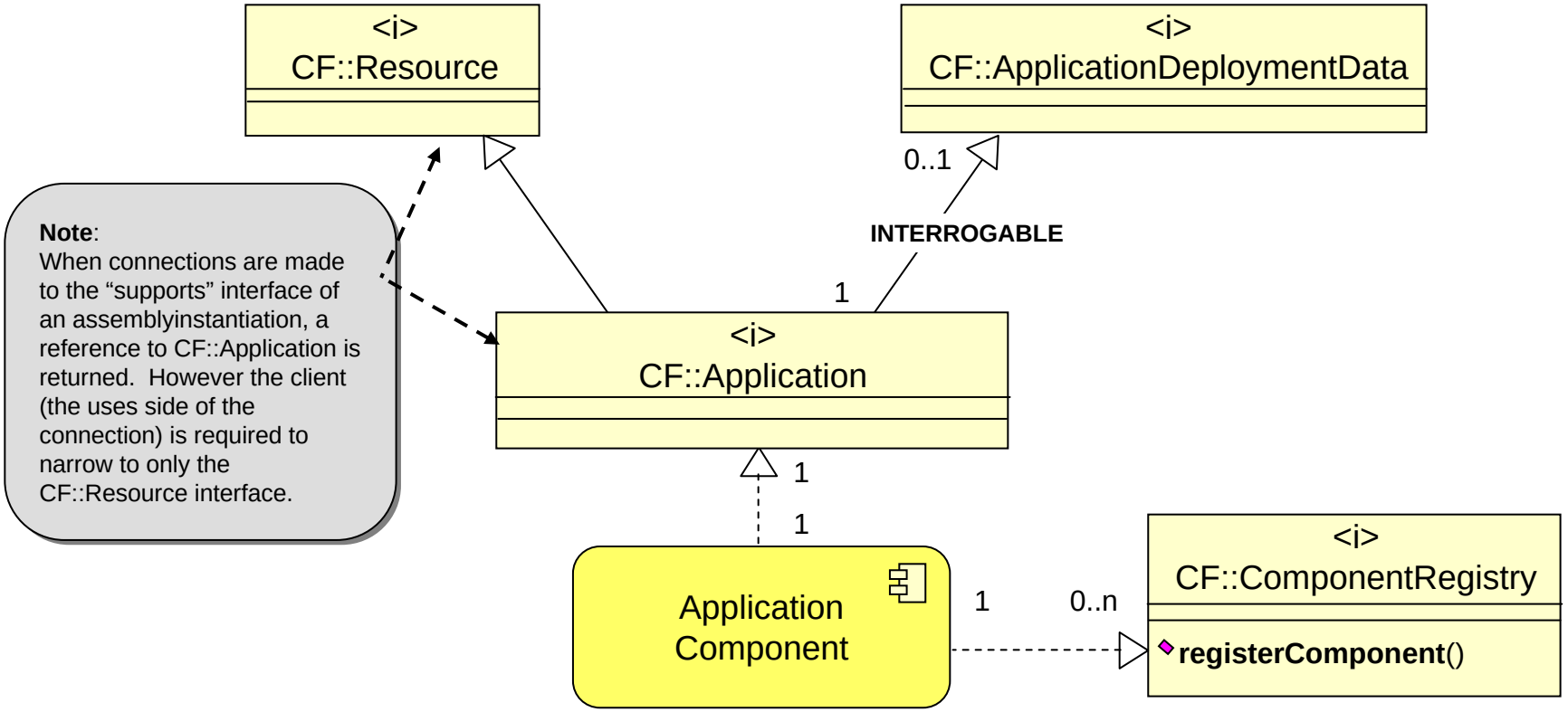


Application Interface Refactoring



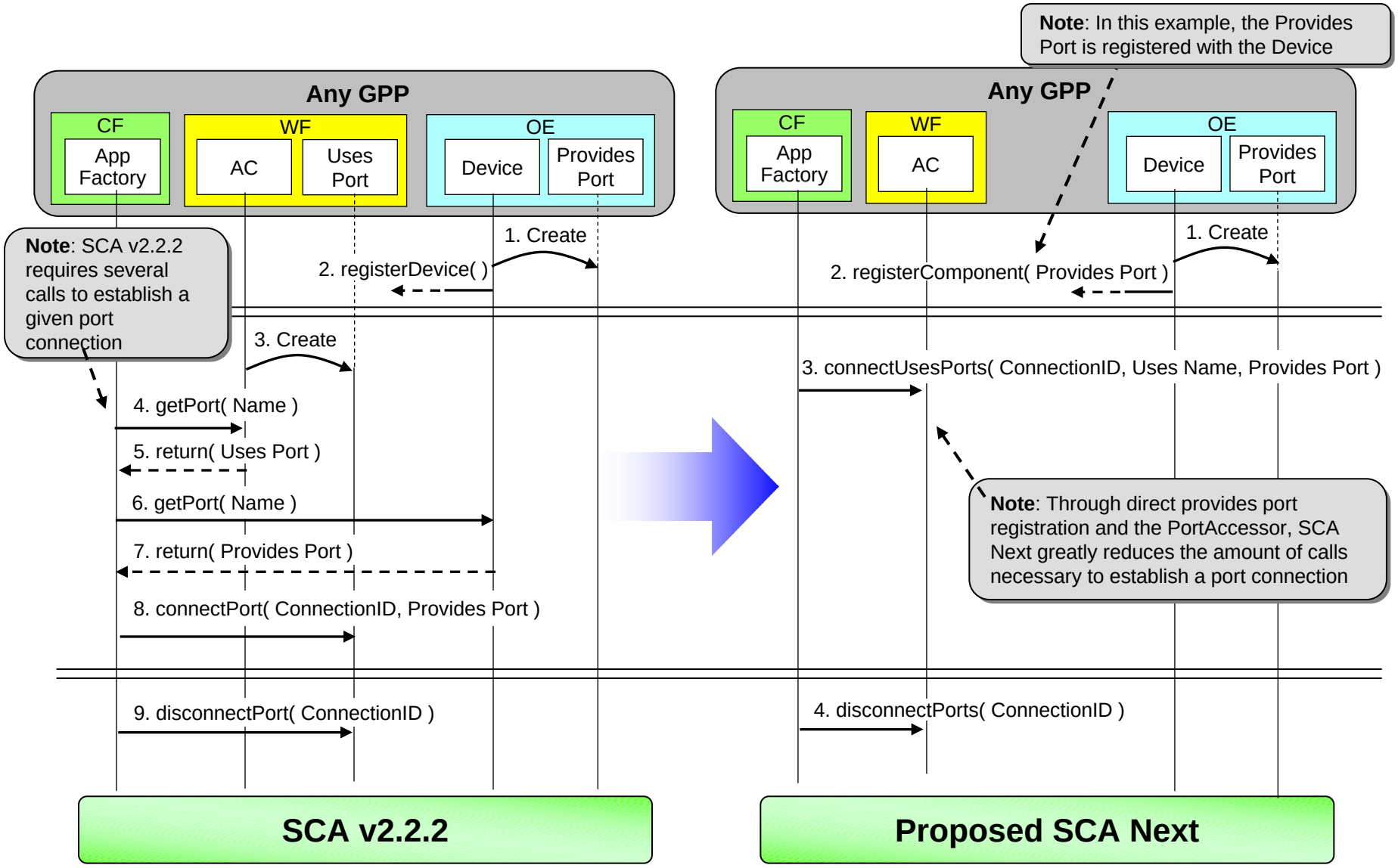
Proposed SCA Next

Application Component Model

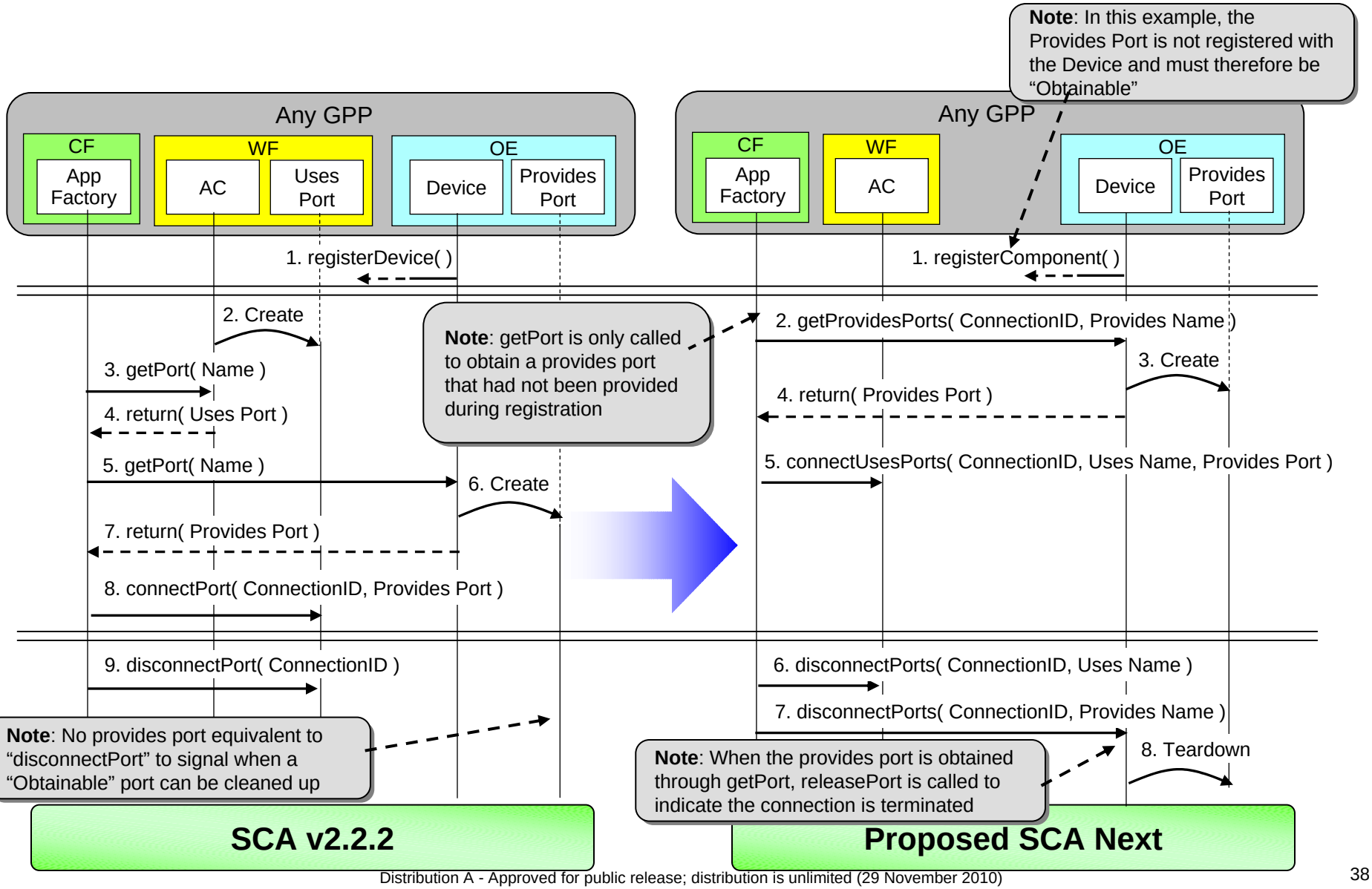


Proposed SCA Next

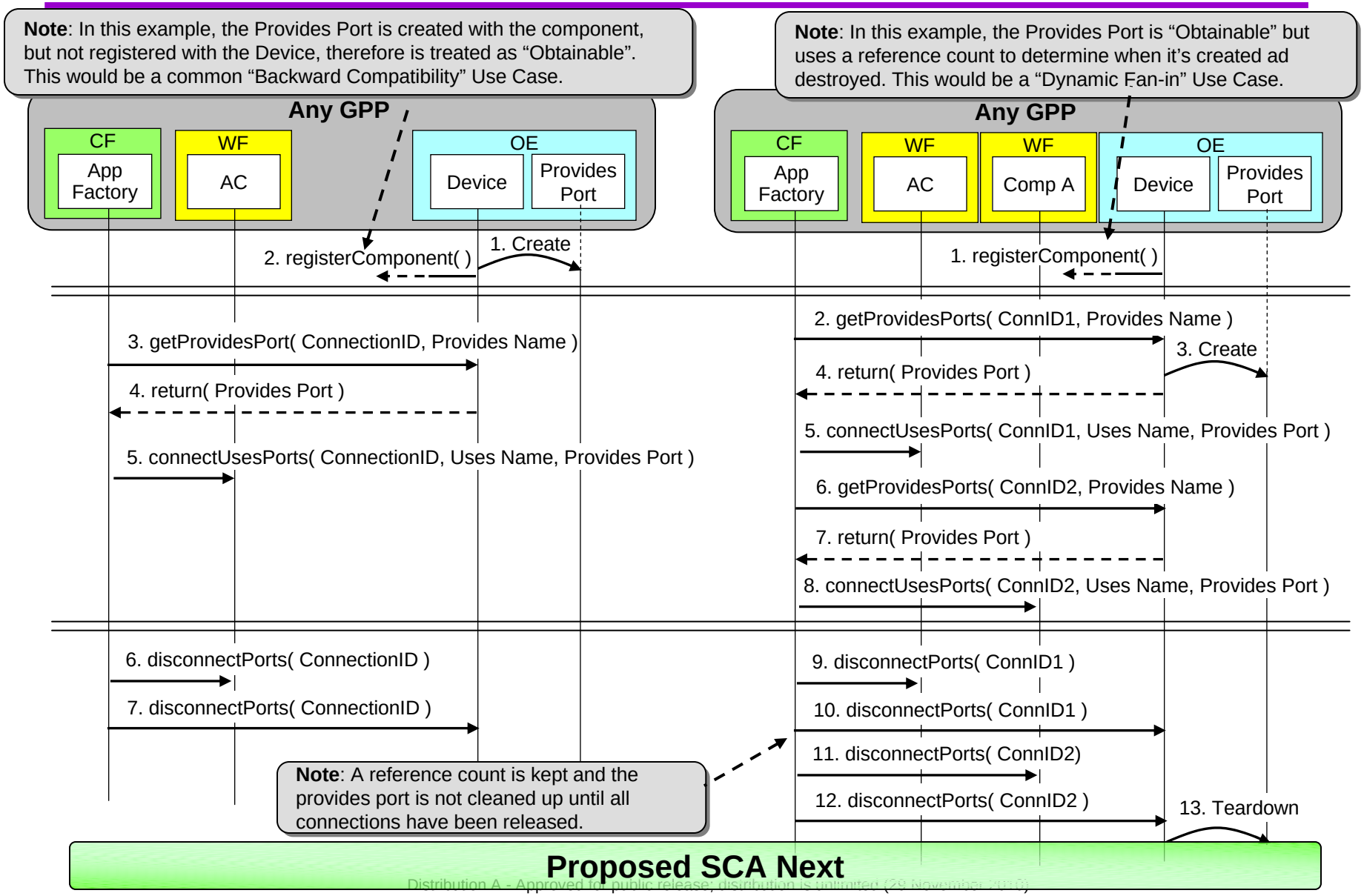
Registered Provides Port Connection



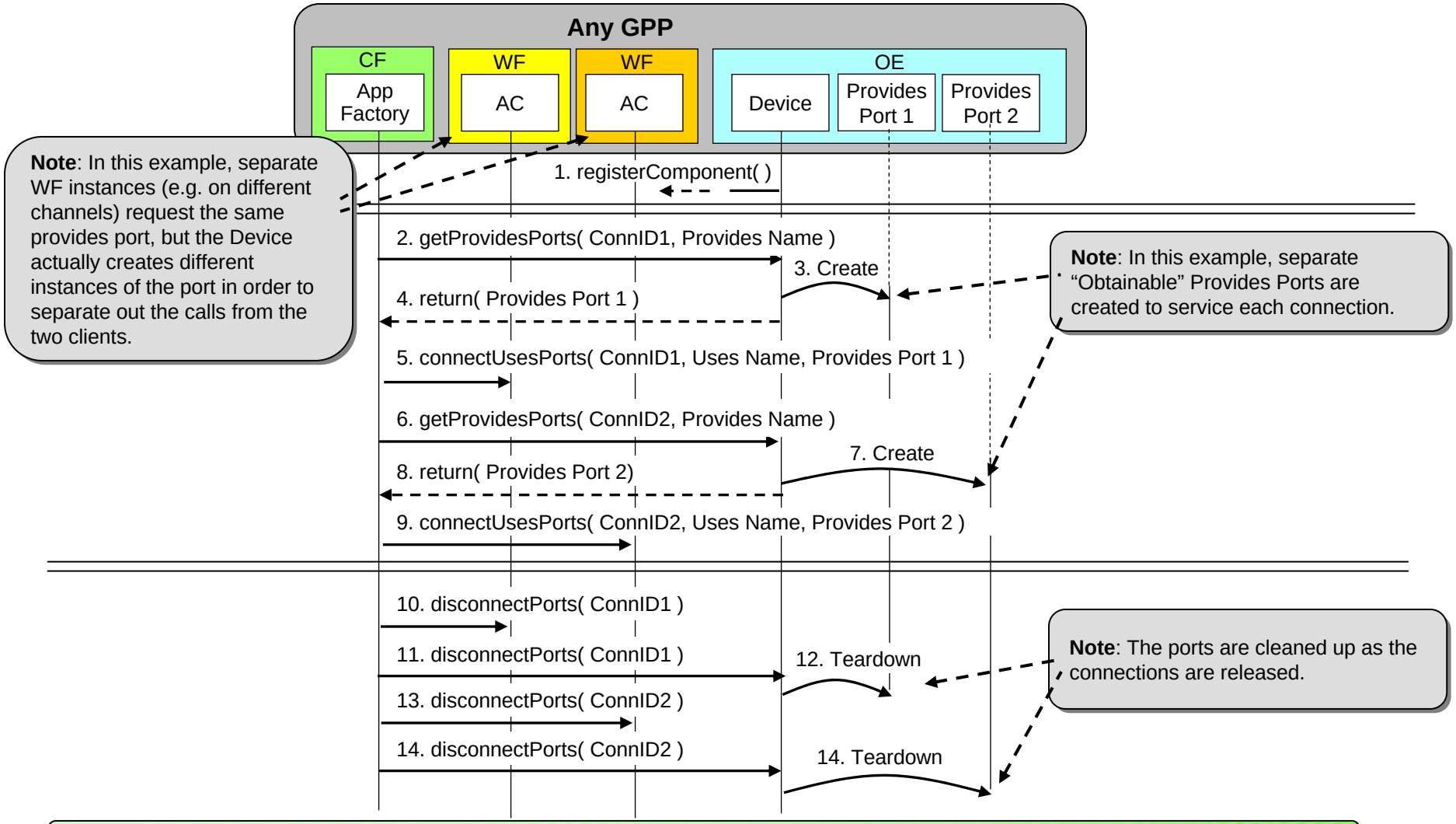
Obtainable Provides Port Connection



Obtainable Provides Port Connection



Obtainable Provides Port Connection



Proposed SCA Next

Interface Specifics

```
struct PortType {
   string portName;
   Object port;
};

typedef sequence < PortType > Ports;

enum ComponentEnumType {
   APPLICATION_COMPONENT,
   DEVICE_COMPONENT,
   CF_SERVICE_COMPONENT,
   NON_CF_SERVICE_COMPONENT,
   FRAMEWORK_COMPONENT
};

struct ComponentType {
   string identifier;
   string softwareProfile;
   ComponentEnumType type;
   Object componentObject;
   Ports providesPorts;
};

typedef sequence < ComponentType > Components;

struct ManagerType {
   ComponentType managerComponent;
   Components registeredComponents;
   FileSystem fileSys;
   string profile;
};

typedef sequence < ManagerType > ManagerSeq;
```

Note: Green text is to indentify areas where, even when using the `V222_COMPAT` flag, there are a few other minor changes that go against 100% backwards compatibility

Interface Specifics

```
interface ComponentRegistry {
   void registerComponent( in ComponentType registeringComponent );
}

interface FullComponentRegistry : ComponentRegistry {
   void unregisterComponent( in string identifier );
}

interface ManagerRegistry {
   void registerManager( in ManagerType registeringManager );
}

interface FullManagerRegistry: ManagerRegistry {
   void unregisterManager( in string identifier );
}

interface EventChannelRegistry {
   void registerWithEventChannel( in Object registeringObject,
                                  in string registeringId,
                                  in string eventChannelName );
   void unregisterFromEventChannel( in string unregisteringId,
                                    in string eventChannelName );
}
```

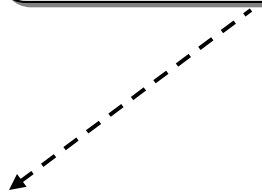
Interface Specifics

```
struct ConnectionType {
   string connectionId;
   string portName;
   Object port;
};

typedef sequence < ConnectionType > Connections;

struct DisconnectionType {
   string connectionId;
   string portName;
};

typedef sequence < DisconnectionType > Disconnections;

interface PortAccessor {
   void getProvidesPorts( inout Connections portConnections );
   void connectUsesPorts( in Connections portConnections );
   void disconnectPorts( in Disconnections portDisconnections );
};
```

Interface Specifics

```
interface PropertySet {
  void configure( in CF::Properties configProperties );
  void query( inout CF::Properties configProperties );
};

interface ControllableComponent {
  readonly attribute boolean started;
  void start();
  void stop();
};

interface ComponentIdentifier {
  readonly attribute string identifier;
};

interface ManagerRelease {
  void shutdown();
};

interface DeviceManagerAttributes {
  readonly attribute string deviceConfigurationProfile;
  readonly attribute CF::FileSystem fileSys;
  readonly attribute Components registeredComponents;
};
```

Note: **Device/Service registration interfaces** broken out and modified to follow the least privilege principle and implement the push model. These interfaces would be optionally provided by these components via “optional realization” instead of inheritance



```
interface DeviceManager : ComponentIdentifier
#if defined( CONNECTABLE   ) || defined( V222_COMPAT ) ,PortAccessor            #endif
#if defined( CONFIGURABLE  ) || defined( V222_COMPAT ) ,PropertySet             #endif
#if defined( RELEASABLE    ) || defined( V222_COMPAT ) ,ManagerRelease          #endif
#if defined( INTERROGABLE ) || defined( V222_COMPAT ) ,DeviceManagerAttributes #endif
{};
```

Interface Specifics

```
struct ApplicationFactoryType {
   string name;
   string profile;
   ApplicationFactory appFactory;
};
typedef sequence < ApplicationFactoryType > ApplicationFactorySeq;

struct ApplicationType {
   string name;
   string profile;
   Application app;
};
typedef sequence < ApplicationType > ApplicationSeq;

interface DomainInstallation {
   void installApplication( in string profileFileName );
   void uninstallApplication( in string applicationId );
};

interface DomainManager : ComponentIdentifier
#if defined( INSTALLABLE  ) || defined( V222_COMPAT ) ,DomainInstallation    #endif
#if defined( CONFIGURABLE ) || defined( V222_COMPAT ) ,PropertySet           #endif
#if defined( EVENT_USER   ) || defined( V222_COMPAT ) ,EventChannelRegistry  #endif
{
  readonly attribute ManagerSeq managers;
  readonly attribute ApplicationSeq applications;
  readonly attribute ApplicationFactorySeq applicationFactories;
  readonly attribute CF::FileManager fileMgr;
  readonly attribute string domainManagerProfile;
};
```

Note: **DeviceManger/Device/Service registration interfaces** broken out and modified to follow the least privilege principle and implement the push model. These interfaces would be optionally provided by these components via “optional realization” instead of inheritance

Interface Specifics

Note: **No componentNamingContexts.** Since the Naming Service is removed in favor of direct component registration, there's no need for these attributes. The are replaced by registeredComponents

```
#define CONNECTABLE
#define CONFIGURABLE
#define TESTABLE
#define CONTROLLABLE
#include "Resource.idl"

interface ApplicationDeploymentData {
   readonly attribute CF::Application::ComponentProcessIdSequence;
   readonly attribute CF::DeviceAssignmentSequence componentDevices;
   readonly attribute CF::Application::ComponentElementSequence componentImplementations;
   readonly attribute Components registeredComponents;
};

interface Application : Resource
#if defined( INTERROGABLE ) || defined( V222_COMPAT ) ,ApplicationDeploymentData #endif
{
   readonly attribute string name;
   readonly attribute string profile;
};


interface ApplicationFactory
{
   readonly attribute string name;
   readonly attribute string softwareProfile;
   CF::Application create( in string name,
                           in CF::Properties initConfiguration,
                           in CF::DeviceAssignmentSequence deviceAssignments );
};
```

Note: **No identifier attribute.** Redundant with "name".

Interface Specifics

```
struct PortType {
    string portName;
    Object port;
};
```

Description	A structure that defines a port

	Name	Type	Description
Attributes	portName	string	The name of the port
	port	Object	The object reference of the port

```
typedef sequence <PortType> Ports;
```

Description	A name/value sequence of PortType structures.

Interface Specifics

```
enum ComponentEnumType {
    APPLICATION_COMPONENT,
    DEVICE_COMPONENT,
    CF_SERVICE_COMPONENT,
    NON_CF_SERVICE_COMPONENT,
    FRAMEWORK_COMPONENT
};
```

Description	An enum that defines the basic type of a component

	Name	Description
Values	APPLICATION_COMPONENT	A component which is launched as part of a Software Assembly
	DEVICE_COMPONENT	A Device launched by a Device Manager
	CF_SERVICE_COMPONENT	A Service launched by a Device Manager that the Framework can manage through the CF based interfaces
	NON_CF_SERVICE_COMPONENT	A Service launched by a Device Manager that could implement possibly any interface (e.g. Log, FileSystem, etc.)
	FRAMEWORK_COMPONENT	A Device Manager, Domain Manager, Application, or Application Factory

Interface Specifics

```
struct ComponentType {
    string identifier;
    string softwareProfile;
    ComponentEnumType type;
    Object componentObject;
    Ports providesPorts;
};
```

Description	A structure that defines the basic elements of a component

	Name	Type	Description
Attributes	id	string	The id of the component as specified through execparams
	softwareProfile	string	Either the component's SPD filename or the SPD itself
	type	ComponentEnumType	The type of component
	componentObject	Object	The object reference of the component
	providesPorts	Ports	A sequence of registered ports provided by the Component

```
typedef sequence <ComponentType> Components;
```

Description	A sequence of ComponentType structures.

Interface Specifics

```
void registerComponent(
    in ComponentType registeringComponent )
    raises( InvalidObjectReference, RegisterError );
```

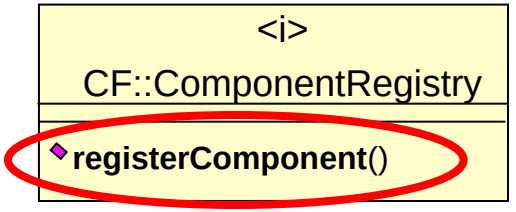
Note: For each operation, an attempt was made to maintain the exceptions from its predecessor (e.g. in this case, registerDevice). But these are not finalized. In the next pass, where changes to the specification are proposed, these will be updated.

Description	This operation registers the Component and its provides ports.

	Name	Type	Description
Parameters	registeringComponent	ComponentType	The id, type, object reference, and provides ports of the component being registered

	Type	Description
Return	None	
Exceptions	InvalidObjectReference	Raised when input parameter registeringComponent contains an invalid object reference
	RegisterError	Raised due to internal error or if the specified type is not accepted

Originator	Application Component, Device, Service, Device Manager



Interface Specifics

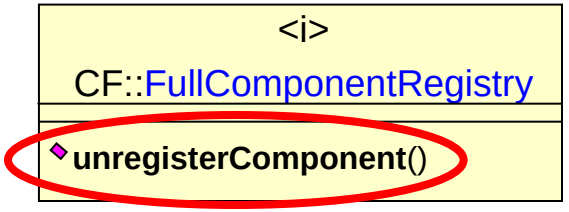
```
void unregisterComponent(
    in string identifier )
    raises( UnregisterError );
```

Description	This operation unregisters the component.

	Name	Type	Description
Parameters	identifier	string	Id of the registered component as specified through execparams

	Type	Description
Return	None	
Exceptions	UnregisterError	Raised due to internal error

Originator	Application Component, Device, Service, Device Manager



Interface Specifics

```
struct ManagerType {
    ComponentType managerComponent;
    Components registeredComponents;
    FileSystem fileSys;
    sting profile;
};
```

Description	A structure that defines the elements of a DeviceManager

	Name	Type	Description
Attributes	managerComponent	ComponentType	Component information including id, type, object reference, and registered provides ports of the Device Manager itself
	registeredComponents	Components	A sequence of components that have registered with this Device Manager
	fileSys	FileSystem	The file system used by this Device Manager
	profile	string	Either the DCD filename or the DCD itself which was utilized by the DeviceManager

```
typedef sequence <ManagerType> ManagerSeq;
```

Description	A sequence of ManagerType structures.

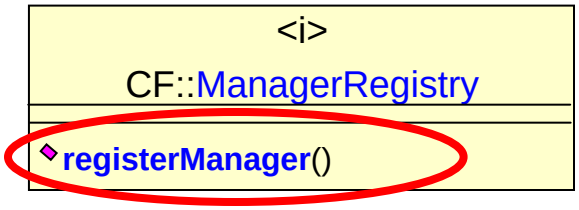
Interface Specifics

```
void registerManager(
    in ManagerType registeringManager)
    raises( InvalidObjectReference, InvalidProfile, RegisterError );
```

Description	This operation registers a Device Manager

	Name	Type	Description
Parameters	registeringManager	ManagerType	The id, type, object reference, registered provides ports, registered components, file system, and device configuration profile of the DeviceManager being registered

	Type	Description
Return	None	
Exceptions	InvalidObjectReference	Raised when input parameter registeringDeviceManager contains an invalid reference
	InvalidProfile	Raised when the device manager's DCD file and the DCD's referenced files do not exist.
	RegisterError	Raised due to internal error



Interface Specifics

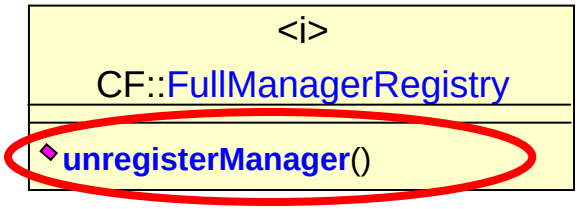
```
void unregisterManager(
    in string identifier )
    raises( UnregisterError );
```

Description	This operation unregisters the DeviceManager.

	Name	Type	Description
Parameters	id	string	Id of the registered DeviceManager

	Type	Description
Return	None	
Exceptions	UnregisterError	Raised due to internal error

Originator	Device Manager



Interface Specifics

```
struct ConnectionType {
    string connectionId;
    string portName;
    Object port;
};
```

Description	A structure that defines a information needed to make a connection

	Name	Type	Description
Attributes	connectionId	string	The id of the connection
	portName	string	The name of the (uses or provides) port
	port	Object	The object reference of the provides port

```
typedef sequence <ConnectionType> Connections;
```

Description	A sequence of ConnectionType structures.

Interface Specifics

```
void getProvidesPorts(
    inout Connections portConnections )
    raises( UnknownPort, OccupiedPort );
```

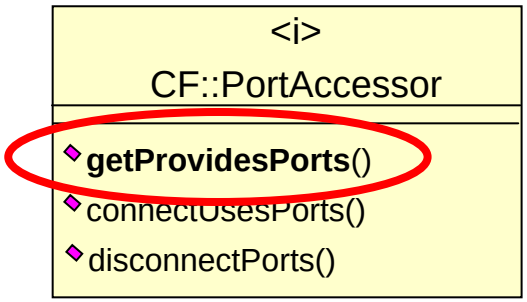
Note: For assurance reasons, it is acceptable to constrain this call to only return "obtainable" ports and not return "registered" provides ports. Also a "zero" length input sequence, which would indicate "get all ports", is not supported.

Description	This operation returns the specified provides ports.

	Name	Type	Description
Parameters	portConnections	Connections	Passed in is a sequence of connectionIds and names of the requested provides port. Returned is additionally the object references of the requested provides ports.

	Type	Description
Return	None	
Exceptions	UnknownPort	Raised if any of the names are unrecognized
	OccupiedPort	Raised if the port specified by name can not accept the connection

Originator	Application Factory, Domain Manager



Interface Specifics

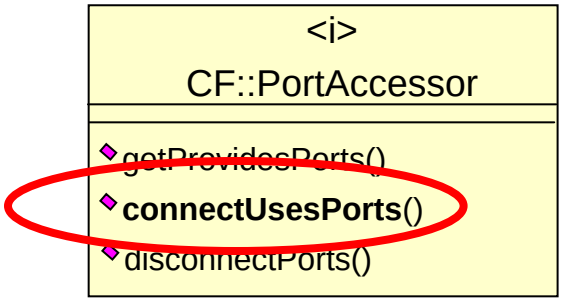
```
void connectUsesPorts(
    in Connections portConnections )
    raises( InvalidPort, OccupiedPorts );
```

Description	This operation supplies a component with a sequence of connection information

	Name	Type	Description
Parameters	portConnections	Connections	A sequence of connectionIds, uses port names, and provides port object references

	Type	Description
Return	None	
Exceptions	InvalidPort	Raised if the supplied provides port is invalid.
	OccupiedPort	Raised if the port specified by name can not accept the connection

Originator	Application Factory, Domain Manager



Interface Specifics

```
struct DisconnectionType {
    string connectionId;
    string portName;
};
```

Description	A structure that defines the information needed to disconnect a connection

	Name	Type	Description
Attributes	connectionId	string	The id of the connection
	portName	string	The name of the (uses or provides) port

```
typedef sequence <DisconnectionType> Disconnections;
```

Description	A sequence of ConnectionType structures.

Interface Specifics

```
void disconnectPorts(
    in Disconnections portDisconnections )
    raises( InvalidPort );
```

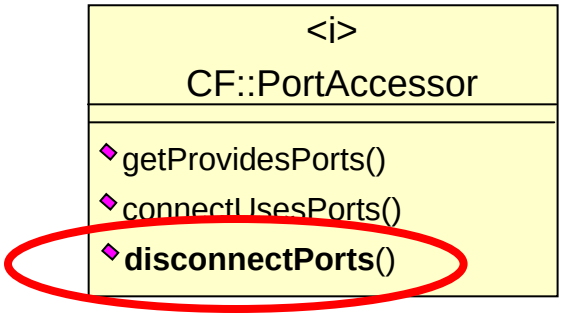
Note: A “zero” length input sequence, which means “disconnect all ports”, is supported on this call. Additionally, if “releaseObject” is called on a component, any ports not “disconnected” through this call should be released at that time.

Description	This operation releases a sequence of uses or provides ports from a given connection.

	Name	Type	Description
Parameters	portDisconnections	Disconnections	A sequence of connectionIds and (uses or provides) port names. A “zero” length sequence of disconnections indicates “release all ports”.

	Type	Description
Return	None	
Exceptions	InvalidPort	Raised if the connectionId specified is unknown.

Originator	Application Factory, Domain Manager



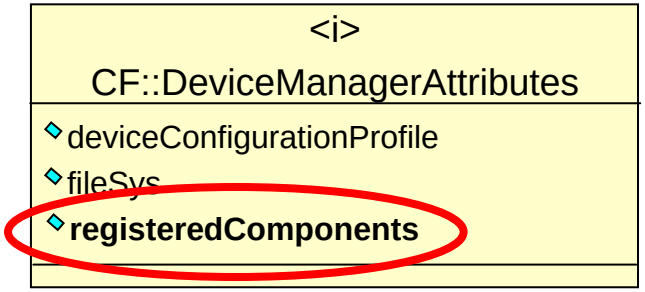
Interface Specifics

```
readonly attribute Components registeredComponents;
```

Description	This attribute returns a sequence of registered Components.

	Name	Type	Description
Attributes	registeredComponents	Components	A sequence of Components that have registered with this Device Manager.

Originator	External Management



Interface Specifics

```
struct ApplicationType{
   string name;
   string profile;
   Application app;
};
```

Description	A structure that defines the elements of an application.

	Name	Type	Description
Attributes	name	string	The user-friendly name of this application or the application created by this application factory. Equal to the *name* passed into the AppFactory *create* call.
	profile	string	Either the SAD filename or the SAD itself which represents this application or the application created by this application factory
	app	Object	The reference to the CF::Application

```
typedef sequence <ApplicationType> ApplicationSeq;
```

Description	A sequence of deployed applications

Interface Specifics

```
struct ApplicationFactoryType{
   string name;
   string profile;
   ApplicationFactory appFactory;
};
```

Description	A structure that defines the elements of an application factory

	Name	Type	Description
Attributes	name	string	The user-friendly name of this application or the application created by this application factory. Equal to the *softwareassembly* element *name* attribute of the SAD.
	profile	string	Either the SAD filename or the SAD itself which represents this application or the application created by this application factory
	app	Object	The reference to the CF::ApplicationFactory

```
typedef sequence <ApplicationFactoryType> ApplicationFactorySeq;
```

Description	A sequence of installed application factories

Interface Specifics

```
readonly attribute ManagerSeq managers;
```

Description	This attribute returns the Device Managers that have registered with this Domain Manager.

	Name	Type	Description
Attributes	managers	ManagerSeq	The Device Managers that have registered with this Domain Manager.

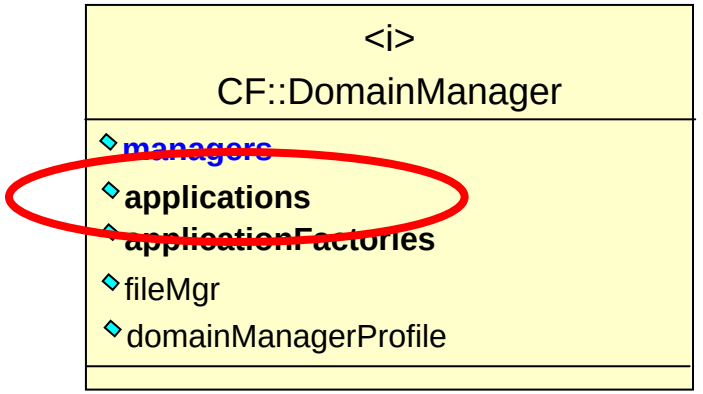
Originator	External Management

Interface Specifics

```
readonly attribute ApplicationSeq applications;
```

Description	This attribute returns the applications that have been deployed in this Domain

	Name	Type	Description
Attributes	applications	ApplicationSeq	The applications that have been created using AppFactories that are installed on this Domain Manager.



Originator	External Management

Interface Specifics

```
readonly attribute ApplicationFactorySeq applicationFactories;
```

Description	This attribute returns the Application Factories that are installed on this Domain Manager.

	Name	Type	Description
Attributes	applicationFactories	ApplicationFactorySeq	The Application Factories that are installed on this Domain Manager.

Originator	External Management

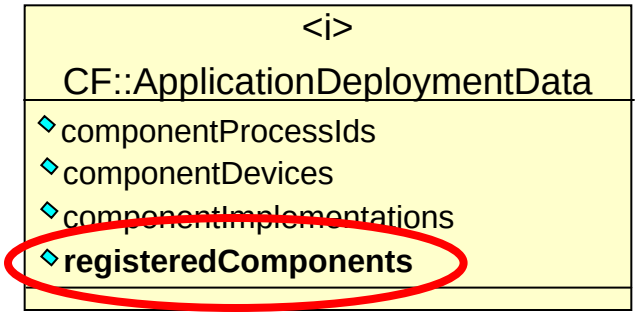
Interface Specifics

```
readonly attribute Components registeredComponents;
```

Description	This attribute returns a sequence of registered application components.

	Name	Type	Description
Attributes	registeredComponents	Components	A sequence of Components that have registered with this Application or AppFactory during instantiation.

Originator	External Management



Domain Profile Specifics

- **Remove Naming Service**
  - Remove *namingservice* from *findby* and *findcomponent*

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

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

- **Make *findcomponent* optional**
  - Only needed for ResourceFactory behavior
- **DeviceManagers making port connections**
  - Need to have some words that allow (but not “require”) DeviceManagers to perform intra-DCD connections
    - E.g. DomainManager may only have to connect components that weren’t already connected via its DeviceManagers
    - Maybe put this in a User Guide
- **Passing more information on registration to avoid distributed parsing**
  - E.g. DeviceManagers could pass parsed DCD information to DomainManagers upon registration to alleviate DomainManagers from parsing DCDs
  - Decided to leave this as something for the SCA Extensions where an alternate set of registration interfaces could be defined