Tools & Techniques for Deployment & Configuration of QoS-enabled Component Applications

Jaiganesh Balasubramanian
jai@dre.vanderbilt.edu
www.dre.vanderbilt.edu/~jai

Gan Deng
dengg@dre.vanderbilt.edu
www.dre.vanderbilt.edu/~dengg

Dr. Aniruddha Gokhale
gokhale@dre.vanderbilt.edu
www.dre.vanderbilt.edu/~gokhale

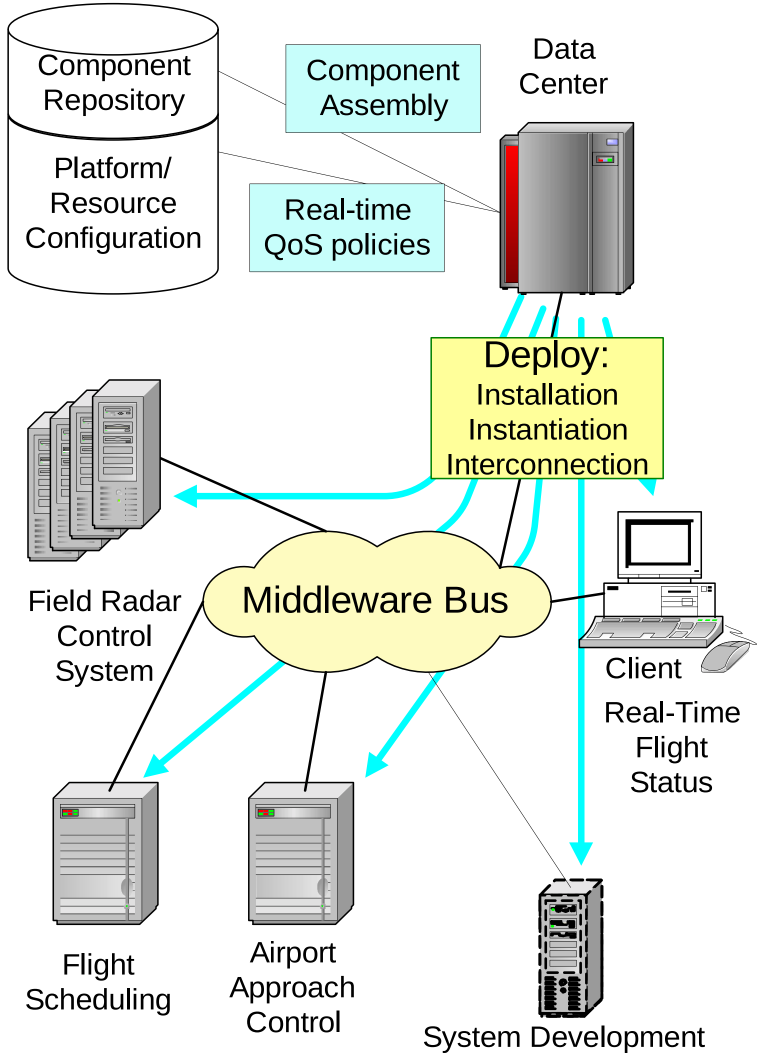
Dr. Douglas C. Schmidt
schmidt@dre.vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee

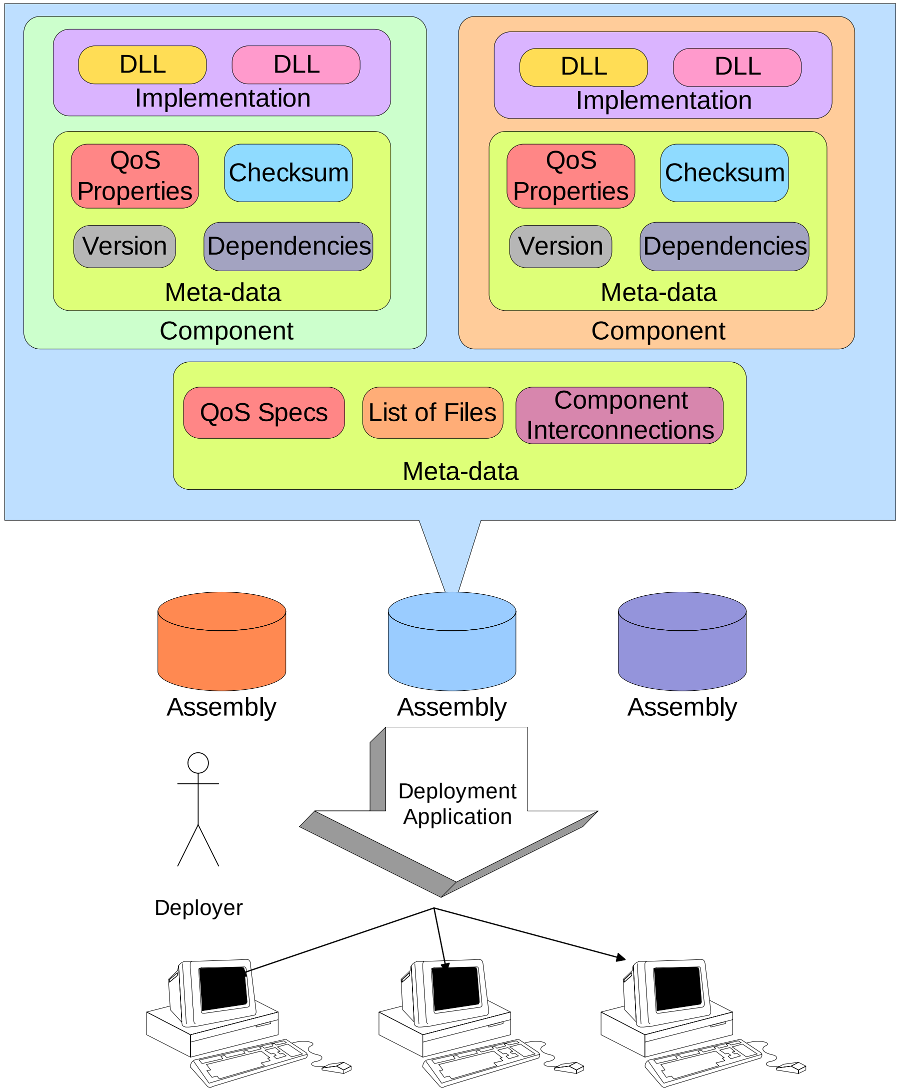
Motivation for Deployment & Configuration

- Goals
  - Promote component reuse
  - Build complex applications by assembling existing components
  - Automate middleware services configuration
  - Inject “real-time” QoS policies into applications declaratively
  - Dynamically deploy components to target heterogeneous domains
  - Defer system optimization later based on particular component configuration & deployment settings



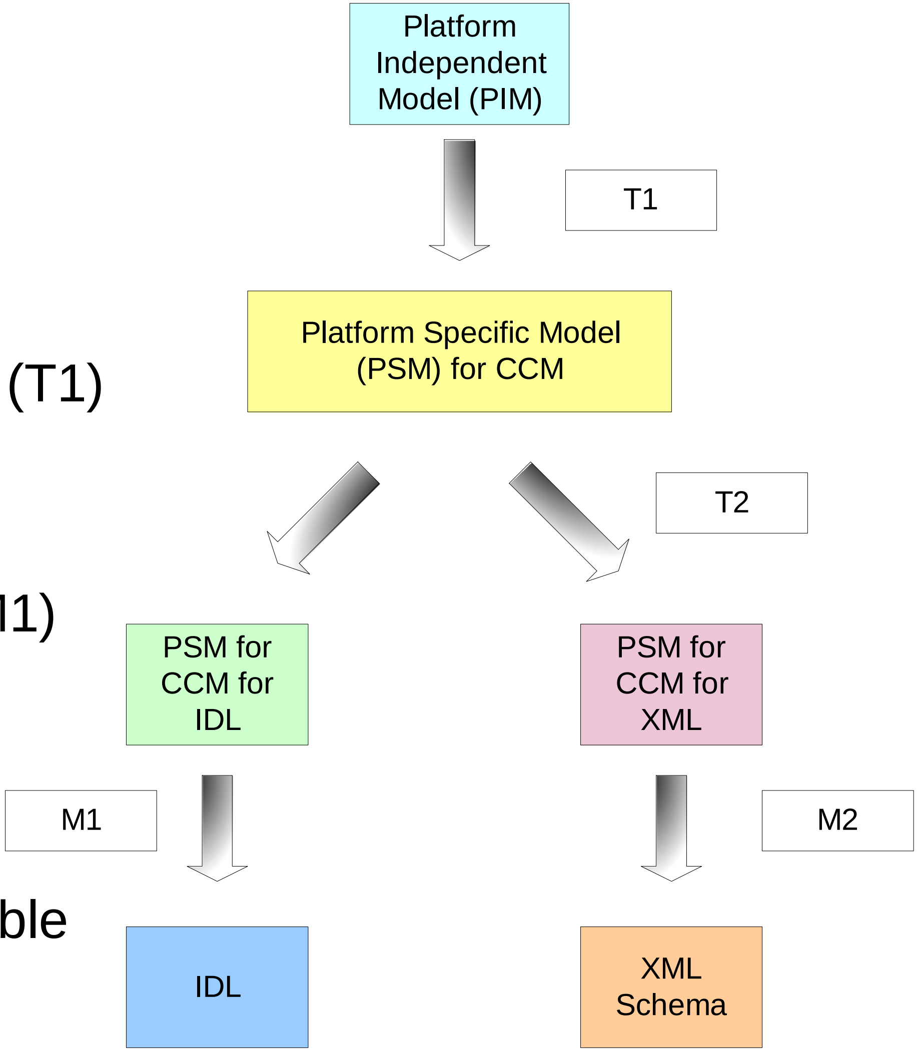
CCM Deployment & Configuration (D&C) Spec

- "D&C" spec was adopted by OMG in 2003
- Intended to replace *Packaging & Deployment* chapter of CCM (CORBA 3.0) specification
- Supports …
  - Hierarchical assemblies
  - Resource management
  - QoS characteristics
  - Automated deployment
  - Vendor-independent deployment infrastructure



D&C & Model-Driven Architecture

- D&C is specified using a platform-independent model
  - Defines “deployment” model
  - Independent of CORBA & CCM (specified in UML)
- Can be refined into CCM-specific model (T1)
- Uses standard mappings to generate
  - IDL (for “on-line” data)
    - using UML Profile for CORBA (M1)
  - XML Schema (for “off-line” data)
    - using XMI (M2)
- Intermediate transformation T2
  - Transforms PSM for CCM into suitable input for M1 & M2


Platform Independent Model (PIM)
T1
Platform Specific Model (PSM) for CCM
T2
PSM for CCM for IDL
PSM for CCM for XML
M1
M2
IDL
XML Schema

Deployment & Configuration “Segments”

PIM	Data Model	Run-time Model
Component Software	Metadata to describe component-based applications & their requirements	Repository Manager interfaces to browse, store, & retrieve such metadata
Target	Metadata to describe heterogeneous distributed systems & their capabilities	Target Manager interfaces to collect & retrieve such metadata & commit resources
Execution	Metadata to describe a specific deployment plan for an application into a distributed system	Execution Manager interfaces to prepare environment, execute deployment plan on target, manage lifecycle

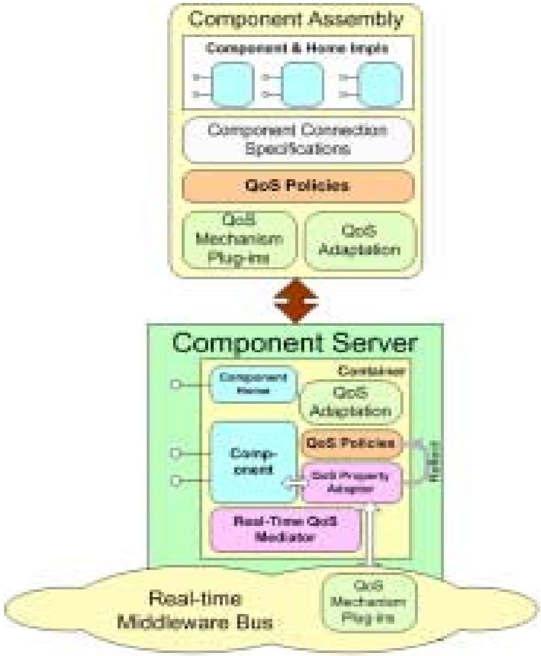
Data model

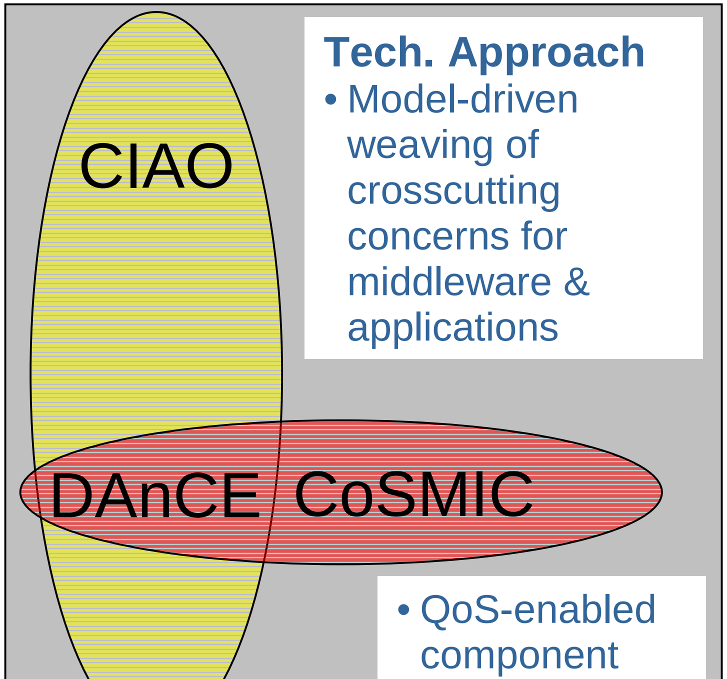
- – Metadata, usually in XML format

Run-time model

- – Deployment interfaces (similar to CORBA services)

Overview

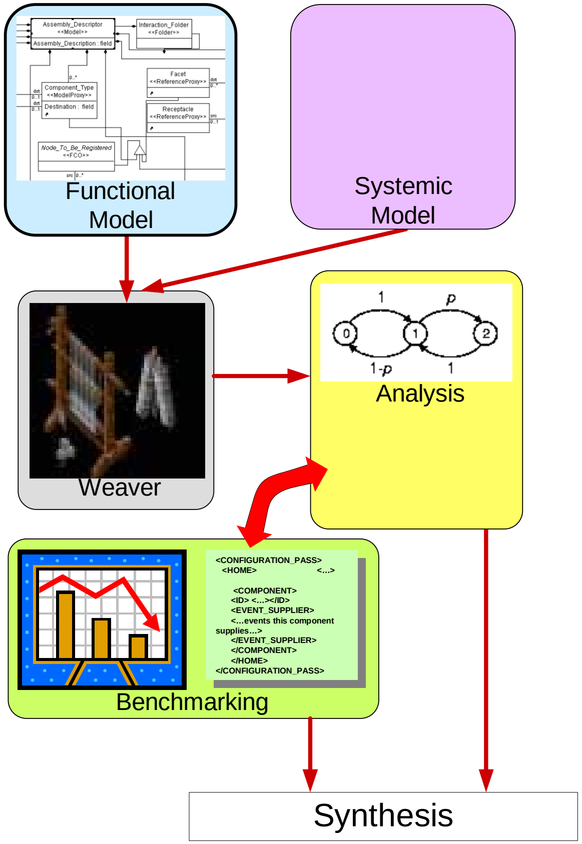


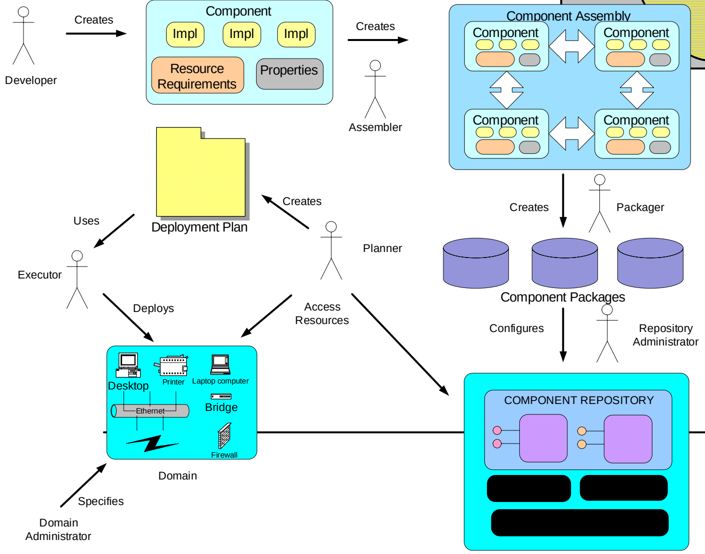


Tech. Approach

- Model-driven weaving of crosscutting concerns for middleware & applications

- QoS-enabled component middleware



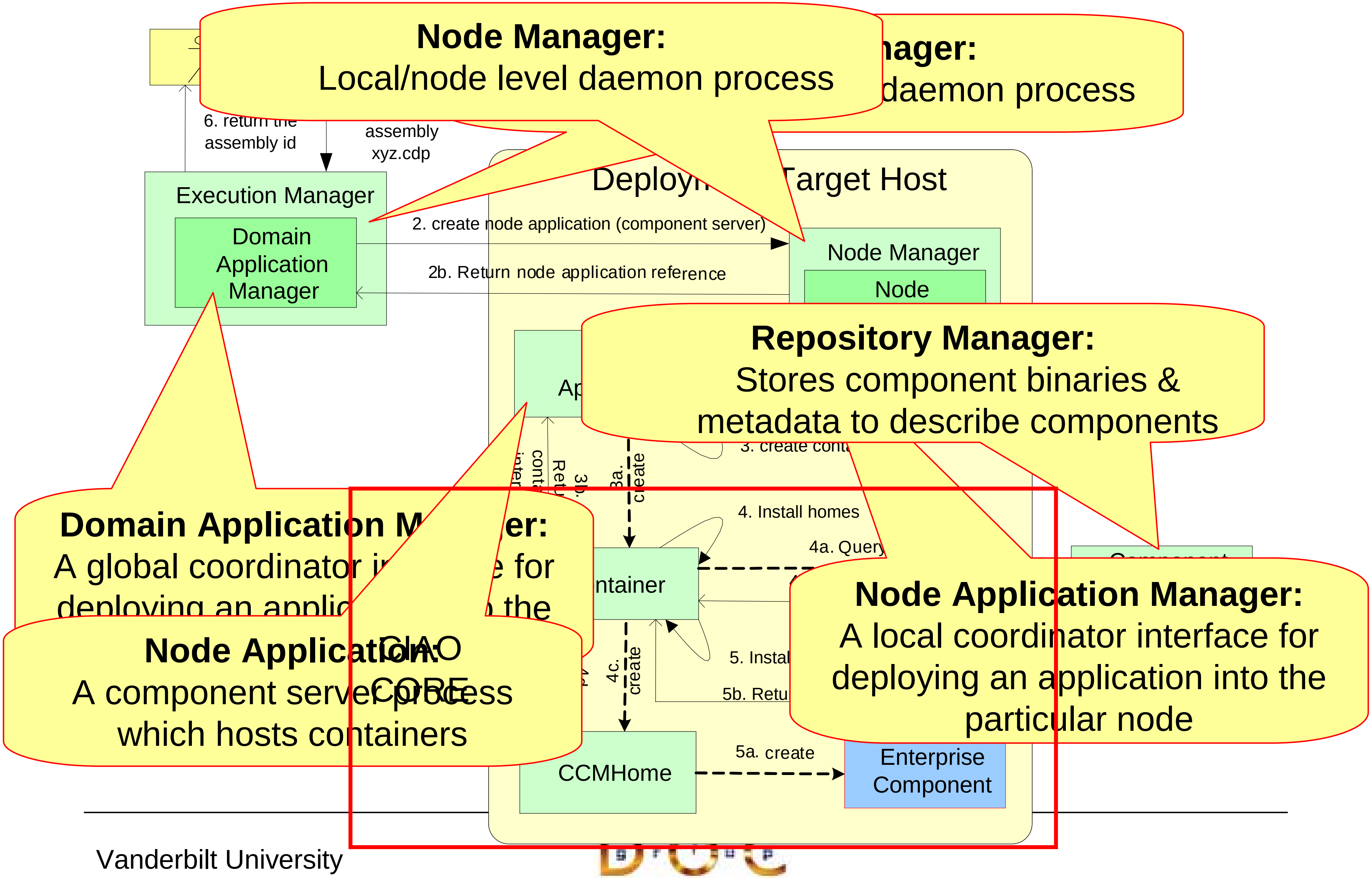


CIAO – QoS-enabled component middleware

CoSMIC – Modeling development, configuration, & deployment concerns

DAnCE – Deployment And Configuration Engine

DAnCE Infrastructure Overview



Node Manager:
Local/node level daemon process

Repository Manager:
Stores component binaries & metadata to describe components

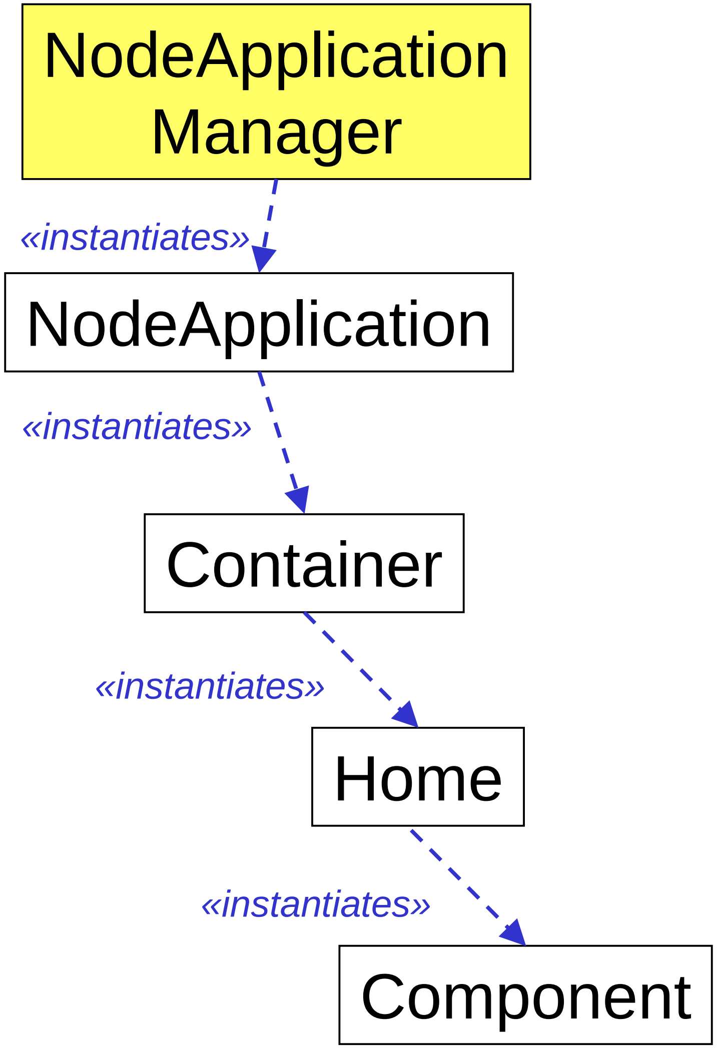
Domain Application M er:
A global coordinator i e for
deploying an applic the

Node Application:
A component server process which hosts containers

Node Application Manager:
A local coordinator interface for deploying an application into the particular node

CIAO
CORE

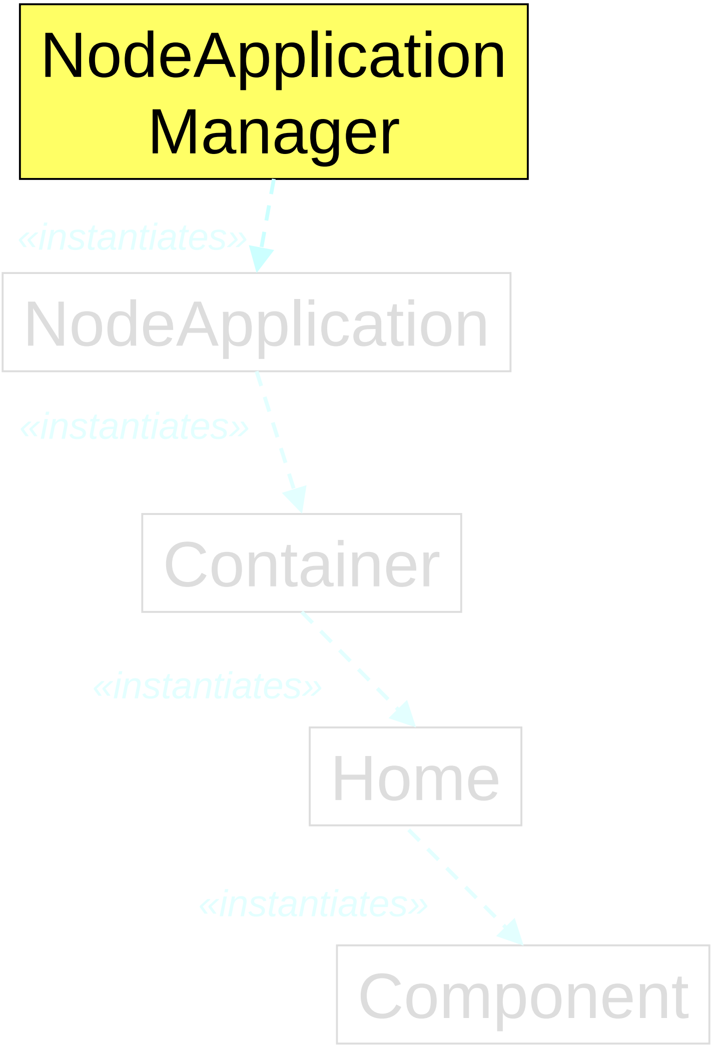
Deployment & Configuration Process



Canonical steps in the application deployment & configuration process (performed by CCM Deployment & Configuration engine):

- Create the *NodeApplication* environment within which containers reside
- Create *containers* for the components
- Create & register *homes* for components
- Create & register the *components* themselves
- Establish *connections* between components

Deployment & Configuration Process – Step 1



NodeApplication Manager

«instantiates»

NodeApplication

«instantiates»

Container

«instantiates»

Home

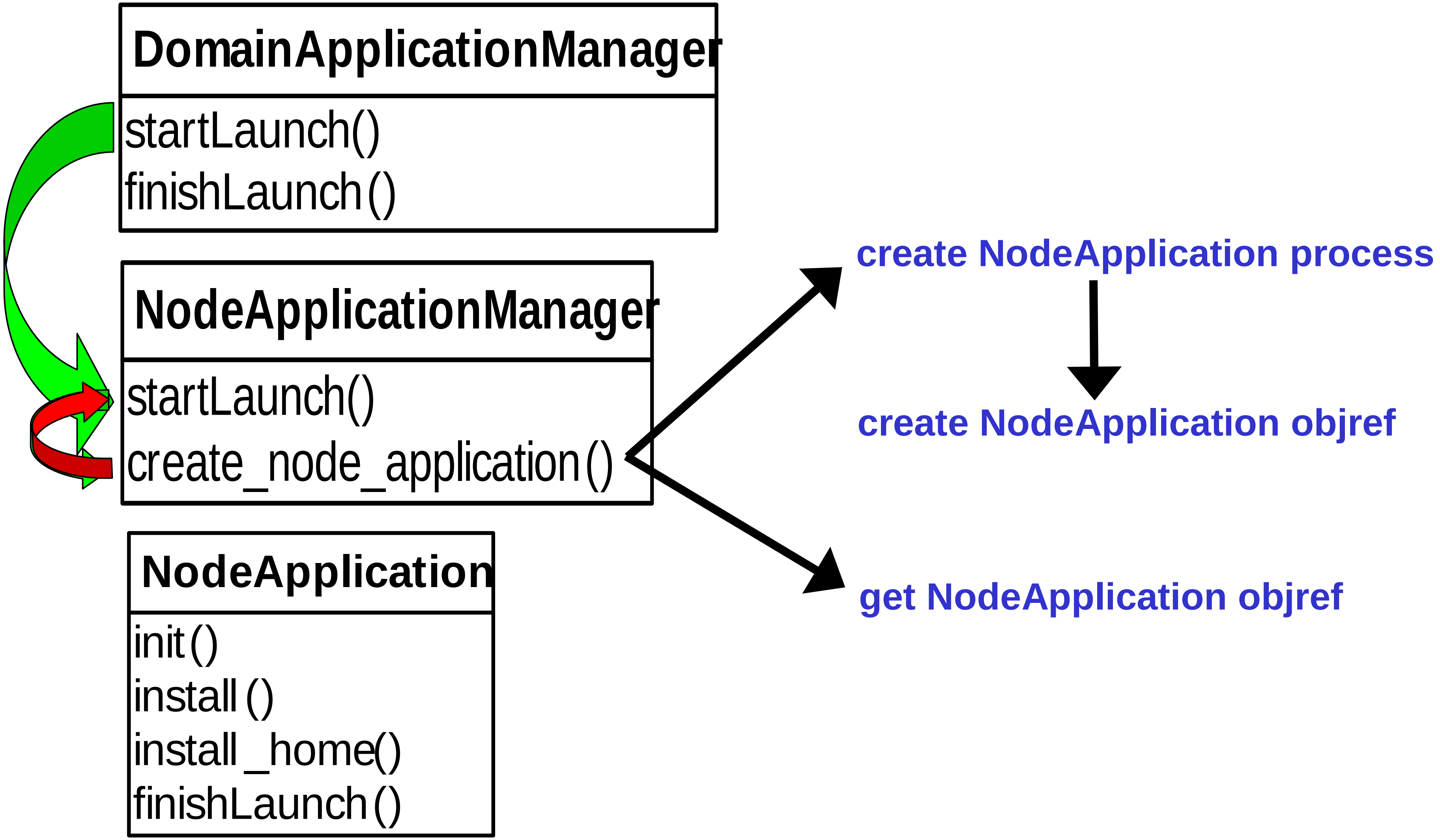
«instantiates»

Component

Canonical steps in the application deployment & configuration process (performed by CCM Deployment & Configuration engine):

- Create the *NodeApplication* environment within which containers reside
- Create *containers* for the components
- Create & register *homes* for components
- Create & register the *components* themselves
- Establish *connections* between components

Creating a NodeApplication



DomainApplicationManager

startLaunch()
finishLaunch()

NodeApplicationManager

startLaunch()
create_node_application()

NodeApplication

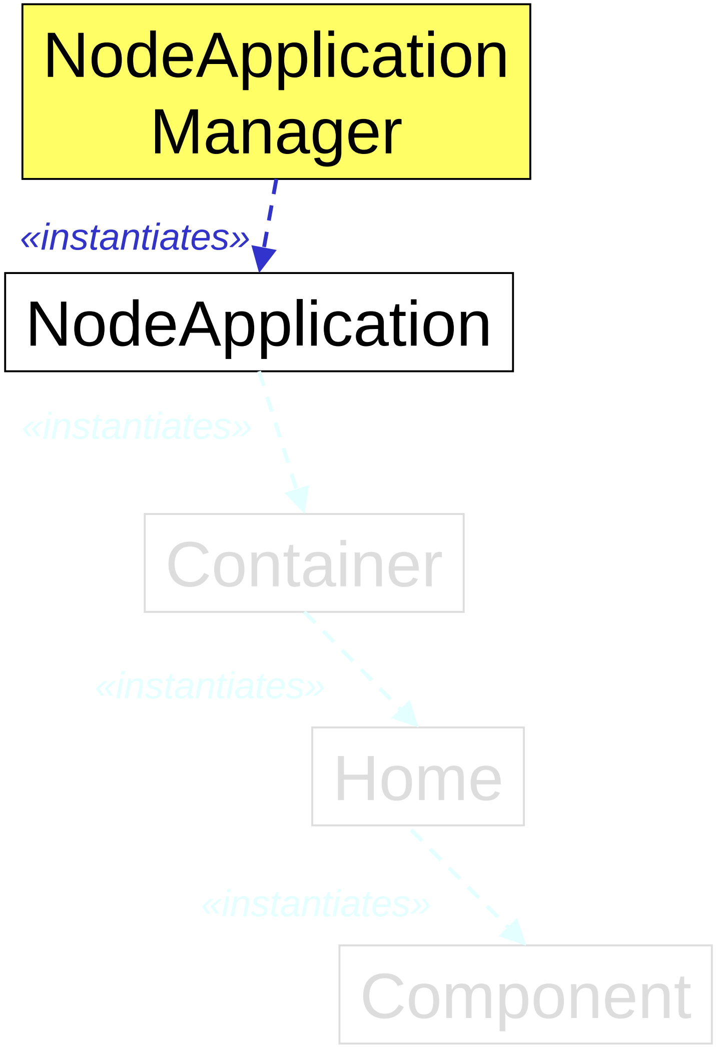
init()
install()
install_home()
finishLaunch()

create NodeApplication process

create NodeApplication objref

get NodeApplication objref

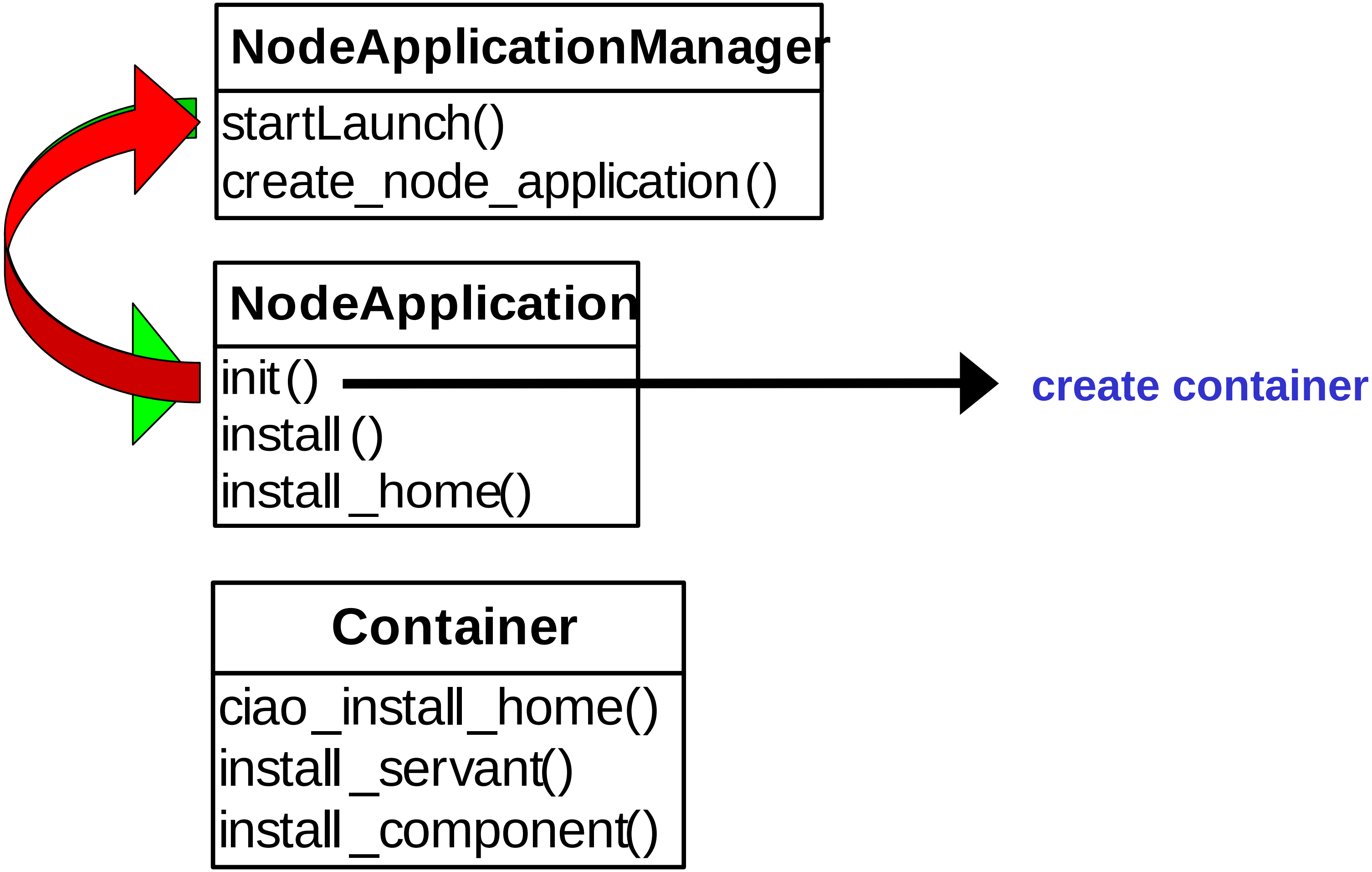
Deployment & Configuration Process – Step 2



Canonical steps in the application deployment & configuration process (performed by CCM Deployment & Configuration engine):

- Create the *NodeApplication* environment within which containers reside
- Create *containers* for the components
- Create & register *homes* for components
- Create & register the *components* themselves
- Establish *connections* between components

Creating a Container



NodeApplicationManager

startLaunch()
create_node_application()

NodeApplication

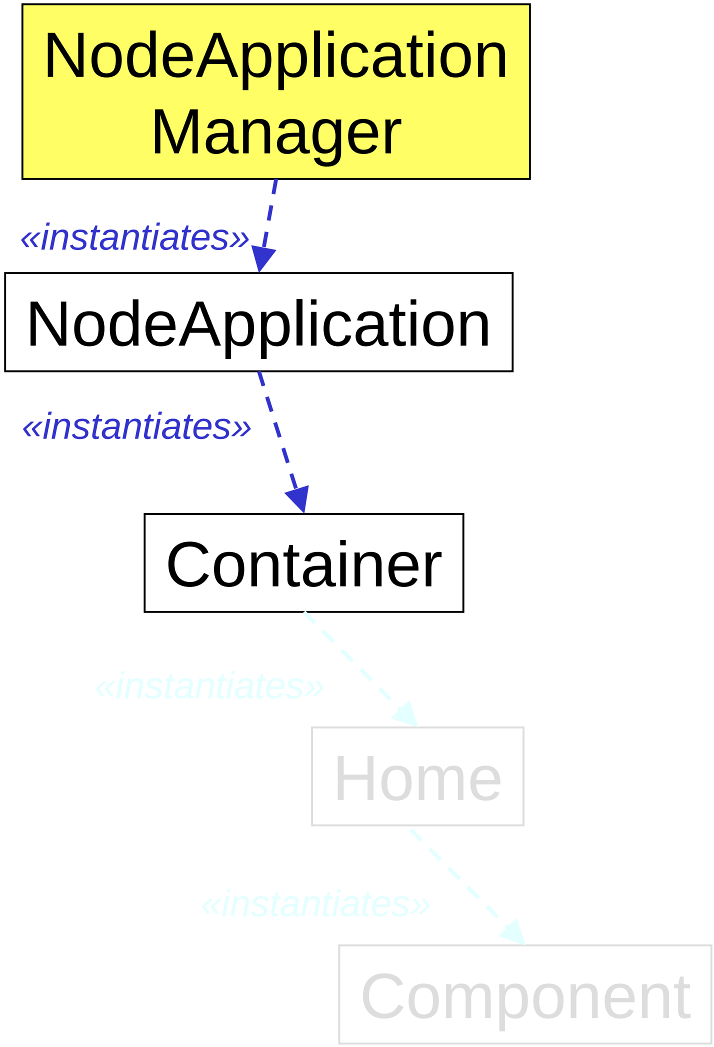
init()
install()
install_home()

create container

Container

ciao_install_home()
install_servant()
install_component()

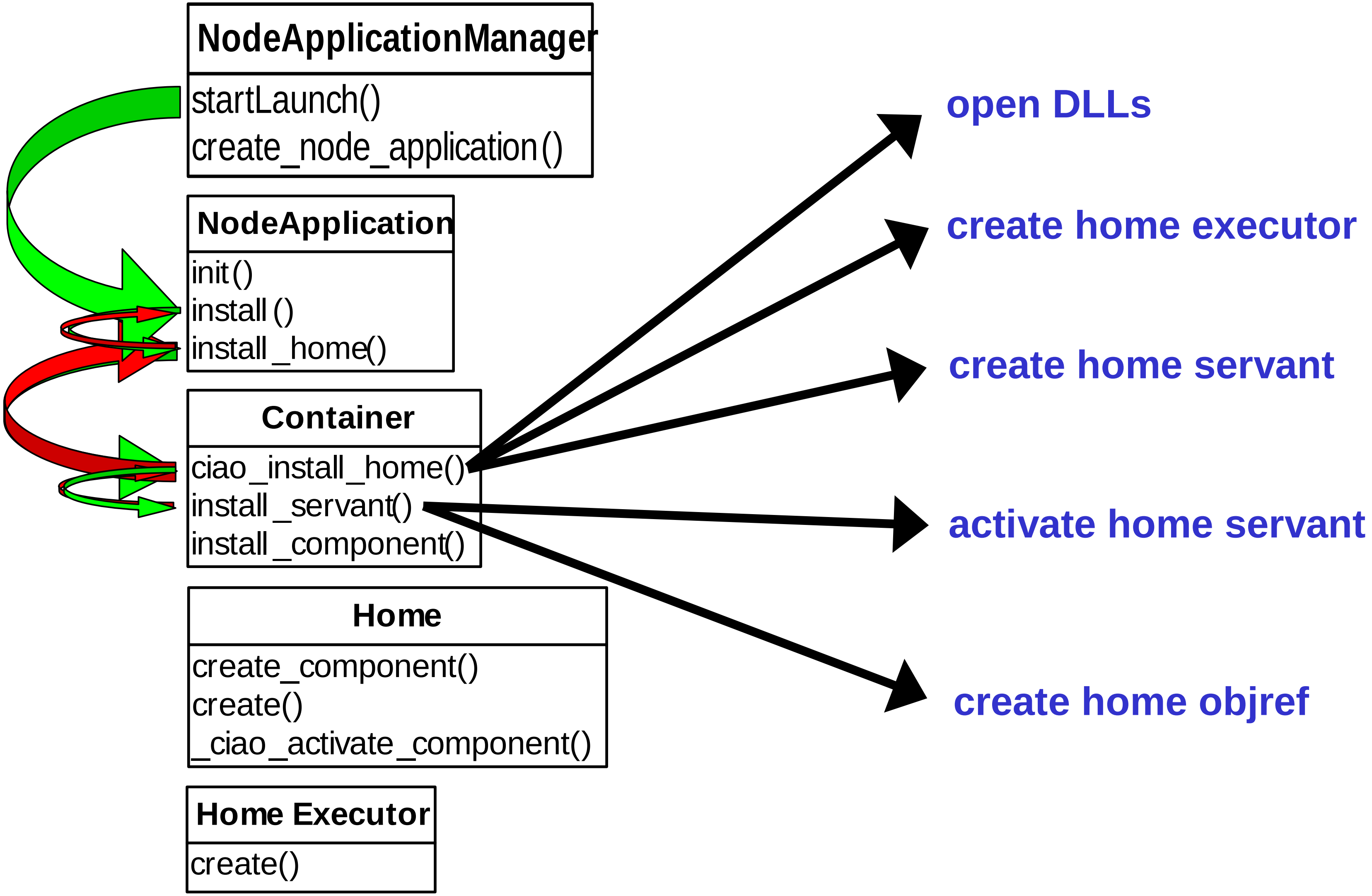
Deployment & Configuration Process – Step 3



Canonical steps in the application deployment & configuration process (performed by CCM Deployment & Configuration engine):

- Create the *NodeApplication* environment within which containers reside
- Create *containers* for the components
- Create & register *homes* for components
- Create & register the *components* themselves
- Establish *connections* between components

Creating a Home Executor & Home Servant



NodeApplicationManager

startLaunch()
create_node_application()

NodeApplication

init()
install()
install_home()

Container

ciao_install_home()
install_servant()
install_component()

Home

create_component()
create()
_ciao_activate_component()

Home Executor

create()

open DLLs

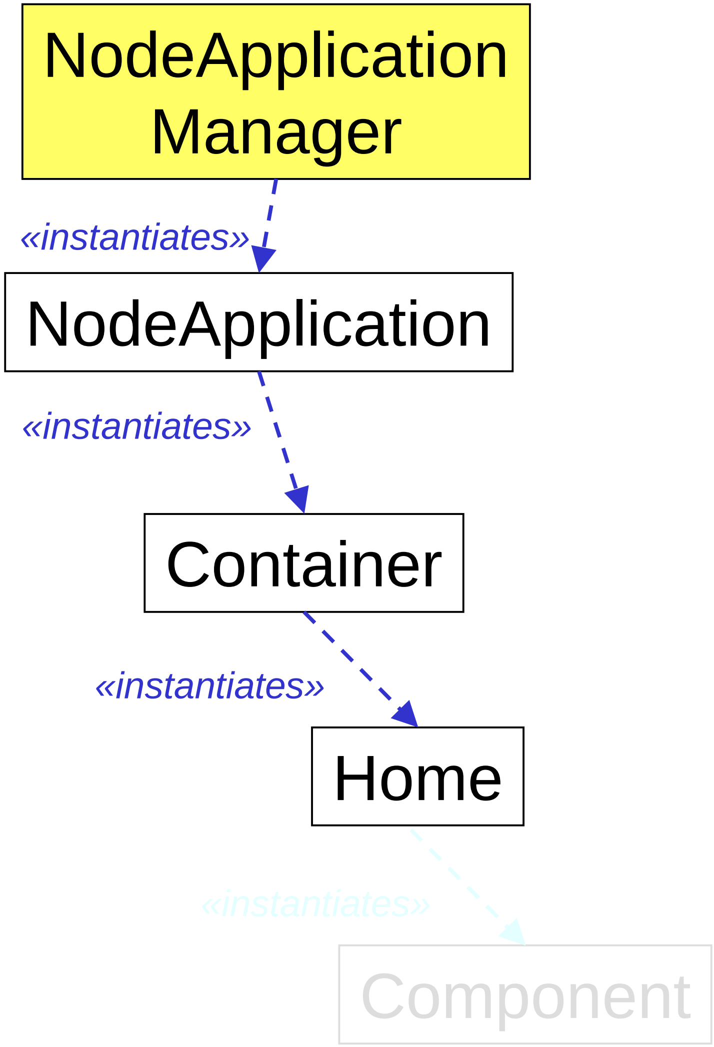
create home executor

create home servant

activate home servant

create home objref

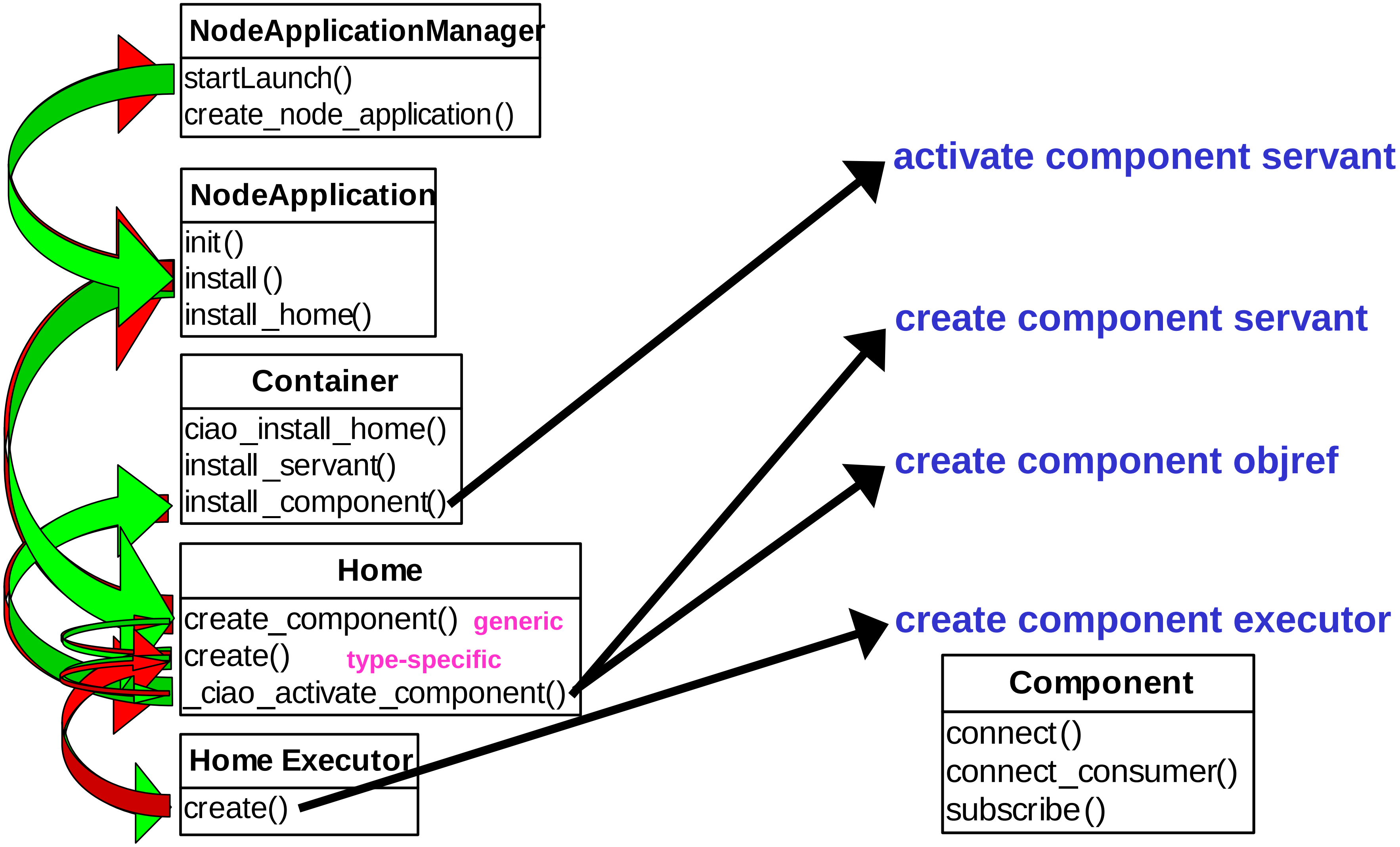
Deployment & Configuration Process – Step 4



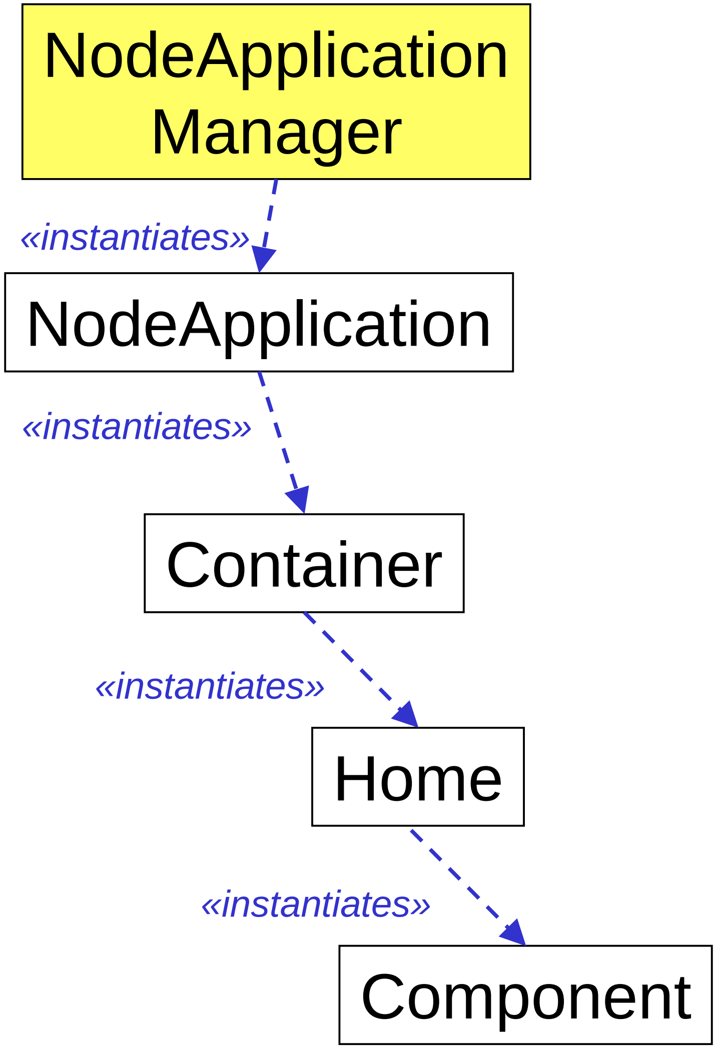
Canonical steps in the application deployment & configuration process (performed by CCM Deployment & Configuration engine):

- Create the *NodeApplication* environment within which containers reside
- Create *containers* for the components
- Create & register *homes* for components
- Create & register the *components* themselves
- Establish *connections* between components

Creating a Component



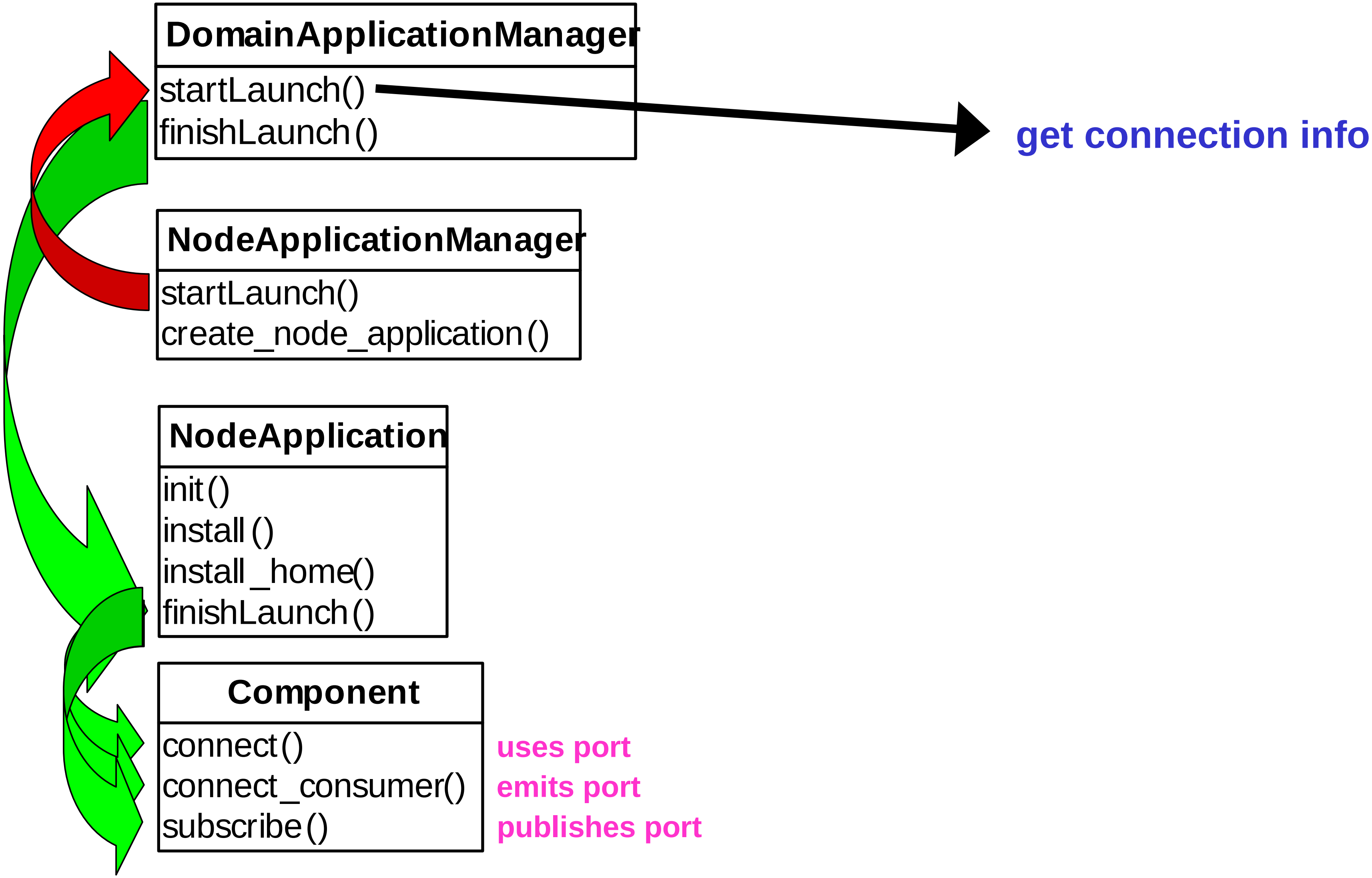
Deployment & Configuration Process – Step 5



Canonical steps in the application deployment & configuration process (performed by CCM Deployment & Configuration engine):

- Create the *NodeApplication* environment within which containers reside
- Create *containers* for the components
- Create & register *homes* for components
- Create & register the *components* themselves
- Establish *connections* between components

Establishing Connections



DomainApplicationManager

startLaunch()

finishLaunch()

get connection info

NodeApplicationManager

startLaunch()

create_node_application()

NodeApplication

init()

install()

install_home()

finishLaunch()

Component

connect() **uses port**

connect_consumer() **emits port**

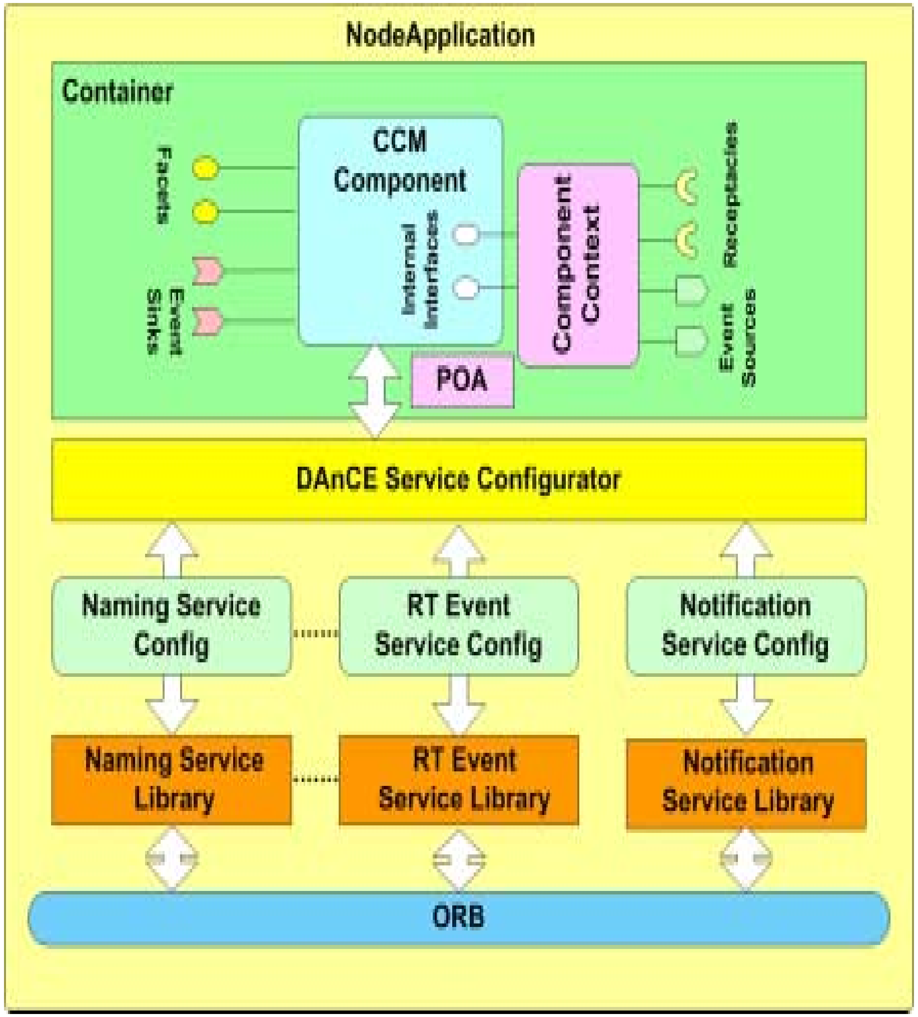
subscribe() **publishes port**

Configuring Middleware Services (1/4)

- Traditional middleware like CORBA 2.x provide applications with access to common middleware services like naming and event service using the underlying ORB middleware.
- In contrast, in component middleware, the focus is on
  - Providing reusability of components by implementing just the application business logic.
  - Enabling easier integration into different application run-time environments and contexts.
- Hence the component deployers have to provide mechanisms to integrate common middleware services into component-based applications.

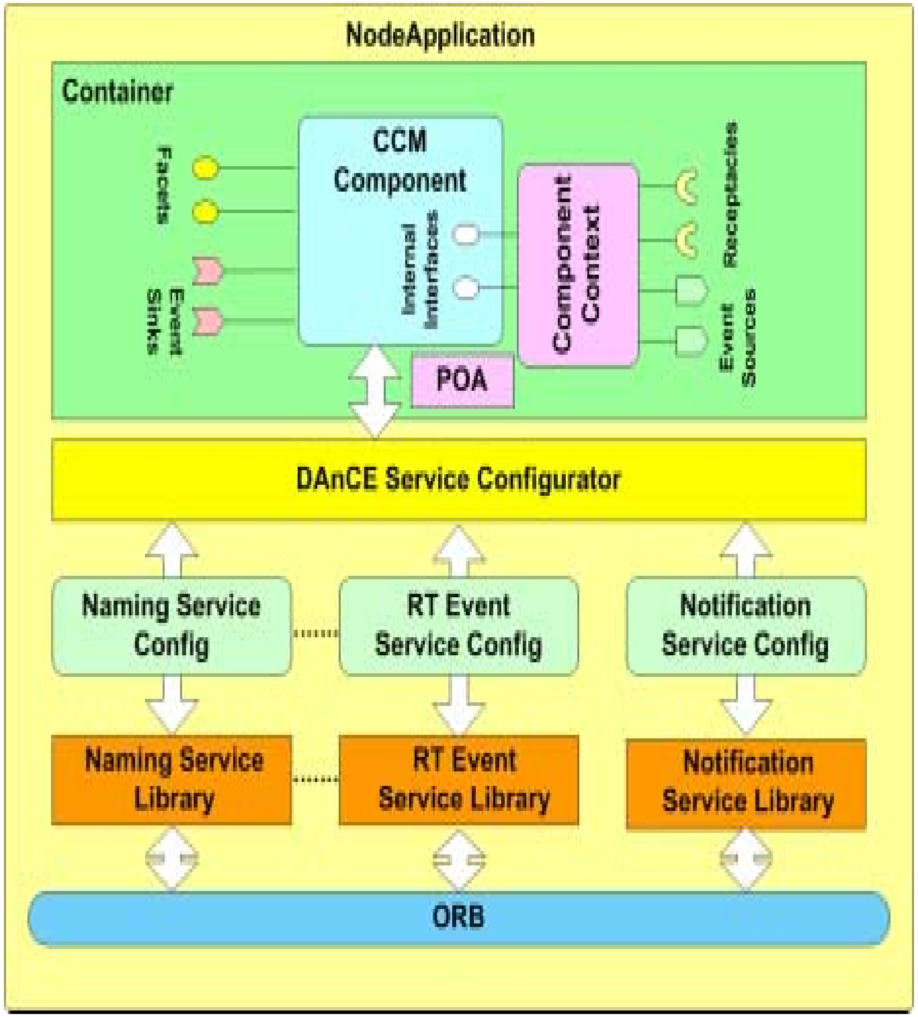
Configuring Middleware Services (2/4)

- Capture usage of common middleware services and formulate as patterns.
  - Example, need to create, initialize and configure the event channel properties before using the Real-time Event service.
- Encapsulate all the usage patterns to provide reusable service libraries.
  - Example, the RT Event Service library encapsulates the usage patterns for the TAO RT Event Service



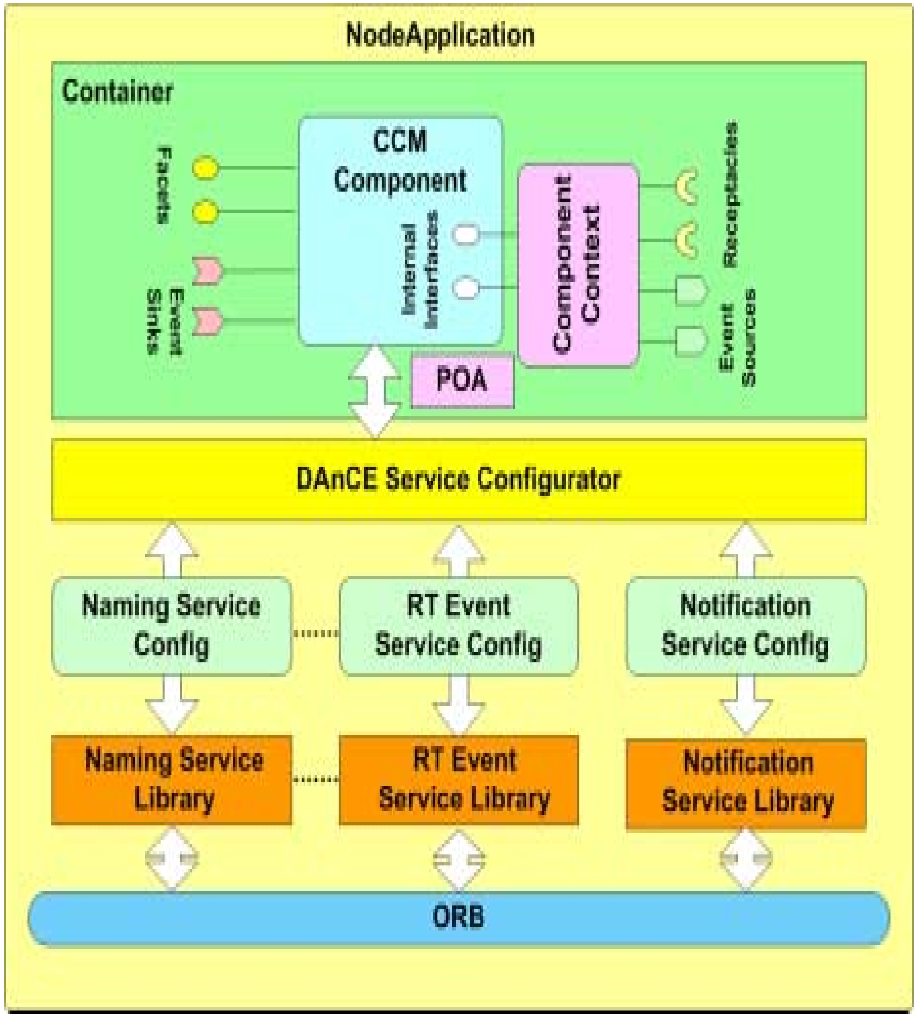
Configuring Middleware Services (3/4)

- Service libraries
  - Wrapper facades for the underlying traditional middleware services.
  - Shields component developers from the tedious programming tasks for configuring and using the services.
- Need a service integration framework that can
  - Automatically start these services.
  - Configure the services based on the QoS properties specified by the component developers using the modeling tools



Configuring Middleware Services (4/4)

- DAnCE Service Configurator
  - Manages the configuration of services, which are exposed as service configuration files.
    - Example, RT Event Service Config captures the various usage options for the RT Event Service Library.
  - Gets instructed by NodeApplicationManager and NodeApplication to automatically load the service
  - Loads the service and configures the service based on the usage options.



Conclusion

- Conventional middleware lacks mechanisms to handle deployment concerns in DRE systems.
- DAnCE leverages modeling languages and component middleware to support the
  - Efficient storage and retrieval of component implementations.
  - Automatic deployment of components.
  - Integrating common middleware services into applications.
- Future work concentrate on:
  - Applying reliable multicast mechanisms to the various managers to communicate.
  - Enhance DAnCE to support dynamic component assembly reconfigurations, redeployments and migrations.

Conclusion