CORBA Component Model

By:
Arnstein Rosvold Andreassen
Espen Abrahamsen
Thomas Kvalvåg

CORBA History

- **Drawbacks with CORBA 2.x**
  - No standard way to deploy object implementation
  - Limited support for standard CORBA server programming patterns
  - Limited extension of object functionality
  - No standard object life cycle management

- As of CORBA 3.0 CCM is introduced
Introduction: What is CCM?

- The CORBA Component Model (CCM) is a specification for creating server-side scalable, language-neutral, transactional, multi-user and secure enterprise-level applications. It provides a consistent component architecture framework for creating distributed n-tier middleware.

Introduction Cont.

- A typical CCM Architecture contains
  - CCM Containers
  - CORBA components that run in these containers
  - Portable Object Adapter (POA),
  - Object Request Broker (ORB)
  - Other standard services
    - CORBA Transactions
    - CORBA Security
    - CORBA Persistence
    - CORBA Events
    - etc.
CCM Architecture

Introduction Cont.

- CCM is build much like the J2EE architecture, in addition CCM enables a language independent component architecture.
CCM

- CCM extends Corba Object Model
- Enforce composition rather than inheritance
- Two forms: Basic and Extended
  - Basic used to “componentize” old corba implementations
  - Extended components support the richer set of functionality – that we will speak about.

IDL Extensions

- Extension to IDL to define component interfaces
- Components will be represented in the Interface Repository
- Components have ports that
  - Define possible connections
  - Define requirements and offerings
  - Allow configuration
- Ports:
  - Facets
  - Receptacles
  - Event sources/sinks
  - Attributes
CCM – facets in IDL

- Specify name of component
  - **Supports** – the supported interface
    - Used directly by component unaware clients
    - Same as interface inheritance in trad. IDL
  - **Facets** – keyword **provides**
    - Allowing multiple interfaces
    - Obtained by Navigation interface
    - All component inherit CCMObject which in turn inherit Navigation – more later.

---

CCM – facets in IDL cont.

Component Button: GraphicControl supports Embeddable{
  provides Embeddable embed;
  provides Printable print;
  provides Externalizable extern;
}

- **Button** inherits GraphicControl
- Supports Embeddable (Can be narrowed to it)
- Provides three interfaces
CCM - receptacles

- Tells about the components connection references
  Component myComp ..{
    uses Embeddable dependant;
  }

- Can have multiple receptacles connected
  - Using Cookies to identify them
  - CCMObject inherits Receptacles interface, defining connect/disconnect operations

- Connects to other components facets.

CCM –facets and receptacles in IDL

[Diagram showing component, facet references, and receptacles]
CCM - Events

- Decoupled communication
- Container provides a Notification service.
- Components can declare that they
  - publishes or emits (only one consumer) events
  - consumes events (event sinks)

- CCM object inherits generic Event interface

Component configuration - Attributes

- Similar to interface attributes, but may throw exceptions
- Allow configuration – per instance – as the factory creates it.
- Can define configurators
  - Mechanisms exists to avoid or allow changes to component after deployment.
- Mapped in Component Assembly descriptor
Navigation Interface

- Clients can check if a component provides a specific interface by using the *Navigation Interface*
- Enumerates all facets provided by a component
- CCM allows binding to unrelated interfaces – clients does not need to know all implementation details to access interfaces.

Component Home

- Home
  - Standard way to access components
  - Each home interface is specific to the a component type – A type manager
  - Created at deployment, defined by idl extensions
  - Provides Factory and Finder operations
    - Some are always available - other can be defined through extended IDL.
  - Two types : keyed and keyless
Keyless home

- **Keyless**
  - Used to create a stateless instance of a component.
  - Only has create operation
  - Extended IDL declaration:
    Home ButtonHome manages Button { #operations#}
  - Compiler generates factory operation
    Button create();

Keyed

- **Keyed**
  - Used to resolve specific instances of a component
  - Identified by value-type pair
  - Compiler generates finder operations
    valuetype ButtonName : Components::PrimaryKeyBase{
      public string name;}
    Home ButtonHome manages Button primarykey ButtonName{..}
  - Compiler generates:
    Button create(in ButtonName key) raises Duplicatekey;
    Button find_by_primary_key (in ButtonName key)...
    ButtonName get_primary_key (in Button comp) ...
HomeFinder

- How to locate Home interfaces?
  - Bootstrap process - centralized database
  - Homefinder – similar to The CORBA Interoperable Naming Service.
  - `resolve_initial_references` to find HomeFinder
  - Then resolve wanted home
  - Invoke appropriate factory operation to create or find target component reference.

Component-unaware Clients

- CCM also allow component unaware clients:
  - Can only interact with supported interface
  - Must use Naming or Trader service or stringified object reference to locate objects
  - Allows soft migration
Component Implementation Framework

- Framework for component implementations
- Purpose: shield component developers from middleware details
  - Lifecycle, (de)activation, etc.
- Includes Component Implementation Definition Language (CIDL), a language for describing the implementation
- IDL and CIDL specifications are used to generate component skeletons

Executors

- Programming artifacts that implement the behaviour of a component or component home
- Analogous with servant in CORBA 2.x
- Monolithic or segmented
- Component homes are always monolithic
- Hosted by a container
- Component executors interact with a container context interface
CIDL

- Component Implementation Definition Language
- Declarative language
- Language- and platform-independent
- Describes the composition of a component
  - Must describe what is required to implement the component
- Persistent State Definition Language (PSDL) is a subset of CIDL for managing components with persistent state.
  - Specifies which attributes that make up the persistent state

CIDL Properties

- Component type (lifecycle types)
  - Service, session, process, entity
- Name of home executor skeleton
- Name of executor skeleton
- Executor segments (if not monolithic)
  - Different segments may implement different facets
CIDL Example

```idl
import dbg.idl;

composition session GameComposition {
    home executor GameHomeSessionImpl {
        implements dbg::GameHome;
        manages GameSessionImpl;

        segment Seg {
            provides facet play;
        };
    };
};
```
CIDL/IDL Compiler

- Input from IDL and CIDL
- Perform IDL mapping
- Generates executor skeletons
  - Callback implementation
  - Executor segmentation
  - Persistent state

CORBA Containers

- A component lives inside a **container** on a CCM component server
- The outside world speak to the component *through* its container (forwards client requests)
- Responsible for activation and deactivation
- Contains a Portable Object Adapter (POA)
- Manages one kind of component, or has a user defined policy ("empty" container)
- Provides simplified interfaces to CORBA services
  - Security, transactions, persistence, ...
- Provides callback interface
Component Assembly and Packaging

- **Component package**
  - Contains one or more implementation of a component
  - XML Descriptor for the component
    - Component implementation descriptor
    - CCM characteristics descriptor

- **Component assembly**
  - Contains a set of Component packages
  - XML Descriptor for the assembly
    - Describes how the components interacts
    - Assembly description

Component Deployment

- The deployment application installs the component assembly defined by the assembly descriptor
- The deployment tool will install and create initial instances if required.
Related Technology

- As mentioned is CCM a Component architecture. There are several other competing technologies:
  - J2EE
  - Com+
  - .Net
  - etc.

J2EE – Java 2 Enterprise Edition

- Sun Technology
- Enterprise Java Beans (EJB)
- A clean Java platform only
- Platform independent
- De Facto standard for component technology
- Supports communication with CORBA objects through IIOP
Com, Com+ and DCOM

- Microsoft component technology
- Windows based only
- COM – The original component model
- DCOM distributed
- COM+ - Added business services (MTS)
- CORBA defines a bridging mechanism between CORBA objects and DCOM components

.Net Framework

- Microsoft based component technology
- Derived from original component based development CORBA,J2EE,COM
- Platform independent ?
  - Mono
  - Rotor
Questions

?????