Komponenttiväliohjelmistot

- Komponenttijärjestelmät vs objektijärjestelmät
- Komponenttijärjestelmien käyttö
- CORBA-komponenttimalli
- Muita komponenttijärjestelmiä

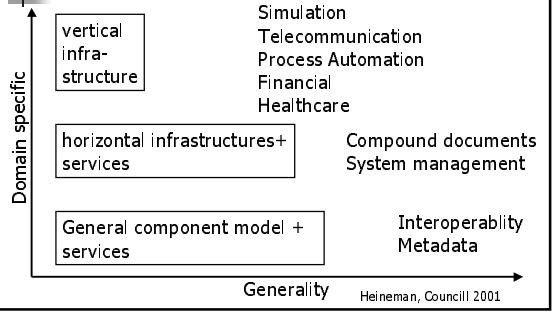
Objekti- ja komponenttimallit

- Eroihin johtaneita seikkoja
  - Kommunikointi -- siirrettävyys
  - Toiminnallisuus -- hallittavuus
  - Palvelevuus -- uudelleenkäytettävyys
- Objektimallit keskittyvät operaatioiden käynnistämiseen
  "Objects are discrete software units – they contain data, and can manipulate it. There are four keys: encapsulation, inheritance, polymorphism, and instantiation."
- Komponenttimallit kuvaavat komponentin toimintaympäristöä

Components

- Components are entities running in a standardized environment that provides
  - transactions, security, event-handling, persistence, and
  - component model services (horizontal and vertical infrastructure services)
    - COTS: Operating systems, compilers, network managers, database systems, CASE tools; aircraft navigation algorithms, banking transaction handlers
- Examples in CCM, EJB, DCOM

Komponenttijärjestelmät



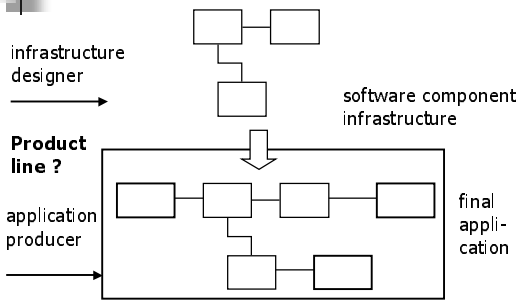
Components

- Independently deployed; Component transfer and deployment is standardized.
- Subject to third-party composition
- In the global software component markets; packaging and licencing
  - Downloadable packages
  - Deployment descriptor provides information about the contents of the package; used for installing and configuring the component properly
- Components are produced using declarative languages.

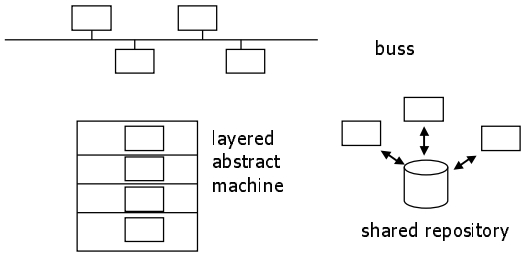
Component system production

- Tranditionally coordination among components by a skeletal software infrastructure
  - Invokes each component
  - Handles communication
  - Handles coordination -> architectural styles
    - Set of component types
    - Topological layout of components indicating their interealtionships
    - Set of interaction mechanims

Component system production



Mismatch of architectural styles



CORBA-komponenttimalli CCM

- abstrakti komponenttimalli
- komponenttien suoritusaikainen ympäristö
  - Component container programming model
- komponenttien toteutus
  - Component Implementation Framework CIF + Component implementation language CIDL
- Komponenttien pakkaaminen ja käyttöönsaatto

The Abstract Component Model

- Allows component designers to capture how CORBA components are viewed by other components and clients
  - What a component **offers** to other components
  - What a component **requires** from other components
  - What collaboration modes are used between components
    - Synchronous via operation invocation
    - Asynchronous via event notification
  - Which component **properties** are configurable
  - What the business life cycle operations are (i.e. **home**)
- Expressed via OMG IDL 3.0 extensions
  - Syntactic construction for well known design patterns
  - Mapped to OMG IDL interfaces for clients and implementers

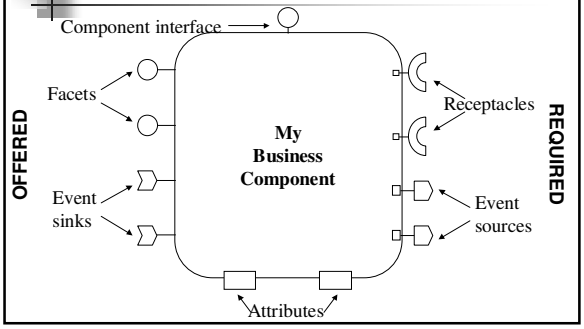
What is a CORBA Component?

- component is a new CORBA meta-type
  - Extension of Object (with some constraints)
  - Has an interface, and an object reference
  - Also, a stylized use of CORBA interfaces/objects
- Provides component features (also named *ports)*
- Could inherit from a single component type
- Could `supports` multiple interfaces
- Each component instance is created and managed by a unique component home

CORBA components

- extends and specializes object type (server side!)
- ports
  - Facets – interfaces provided to clients
  - Receptacles – named connection points, required operation interfaces
  - Event sources – named points that send events
  - Event sinks – named points to which events are pushed
  - Attributes – named values with set/get, for configuration; can raise exceptions
- Equivalent interface
  - Navigation
  - Modifying port states

A CORBA Component



Component runtime support

- Component home
  - Controls a set of components using the component equivalent interface
  - Provides component factory facilities
    - Specializations
- Container
  - Defines the environment for supporting dynamic collections of components
  - The core of the CCM is the container: a running piece of code that you buy from a vendor and install on your server machine. The container includes an ORB with a POA. Corba components are server-side objects; your system administrator installs components into the container, which takes charge of them when they run.

Component Facets

- Distinct named interfaces that provide the component's application functionality to clients
- Each facet embodies a view of the component, corresponds to a role in which a client may act relatively to the component
- A facet represents the component itself, not a separate thing contained by the component
- Facets have independent object references

Component Receptacles

- Distinct named connection points for potential connectivity
  - Ability to specialize by delegation, compose functions
  - The bottom of the Lego, if you will
- Store a simple reference or multiple references
  - But not intended as a relationship service
- Configuration
  - Statically during initialization stage or assembly stage
  - Dynamically managed at runtime to offer interactions with clients or other components (e.g. callback)

Component Events

- Simple publish / subscribe event model
  - "push" mode only
  - Sources (2 kinds) and sinks
- Events are value types
  - Defined with the new **`eventtype`** meta-type
  - `valuetype` specialization for component events

Component Event Sources

- Named connection points for event production
  - Push a specified eventtype
- Two kinds: *Publisher* & *Emitter*
  - `publishes` = multiple client subscribers
  - `emits` = only one client connected
- Client subscribes or connects to directly component event source
- Container mediates access to CosNotification channels
  - scalability, quality of service, transactional, etc.

Component Event Sinks

- Named connection points into which events of a specific type may be pushed
- Subscription to event sources
  - Potentially multiple (n to 1)
- No distinction between emitter and publisher
  - Both push in event sinks

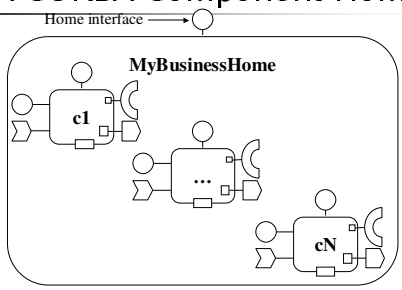
Component Attributes

- Named configurable properties
  - Vital key for successful re-usability
  - Intended for component configuration
    - e.g., optional behaviors, modality, resource hints, etc.
  - Could raise exceptions
  - Exposed through accessors and mutators
- Could be configured
  - By visual property sheet mechanisms in assembly or deployment environments
  - By homes or during implementation initialization
  - Potentially readonly thereafter

Navigation and Introspection

- Navigation from any facet to component base reference with `CORBA::Object::get_component()`
  - Returns nil if target isn't a component facet
  - Returns component reference otherwise
- Navigation from component base reference to any facet via generated facet-specific operations
- Navigation and introspection capabilities provided by `CCMObject`
  - Via the `Navigation` interface for facets
  - Via the `Receptacles` interface for receptacles
  - Via the `Events` interface for event ports

A CORBA Component Home



Component Home

- Is instantiated at deployment time
- Manages a unique component type
  - More than one home type can manage the same component type
  - But a component instance is managed by a single home instance
- Allows life cycle characteristics or key type to vary/evolve without changing component definition
- Optional use of *primarykey* for business component identity and persistency primary key
- Standard *factory* and *finder* business logic operations
- Extensible with arbitrary user-defined business logic operations

Primary Keys

- Values exposed to clients to create, find, and destroy component instances
  - Uniquely identifies a component instance within a home
  - Assigned at creation time, or in pre-existing database
  - Must be a value type derived from Components::PrimaryKeyBase (empty, abstract)
- Association between a primary key and a component is defined and maintained by its home
  - Different home types may define different key types (or no key) for the same component type
  - Primary key is not necessarily a part of the component's state

CCM development project stages

1. Analysis/design
   - UML + business object profiles
2. Component declaration
   - Define component's methods and home with extended IDL and compile to produce
     - for clients: Operations, navigation operations, stubs
     - for servers: skeletons, IR entries, some code, packaging and deployment descriptors in XML

Esimerkki

```
component ShoppingCart {
    provides ShoppingCartIntf Cart1;
    uses CheckoutIntf CheckOut1;
}
home ShoppingCartHome manages
  ShoppingCart{};
```

CCM development project stages

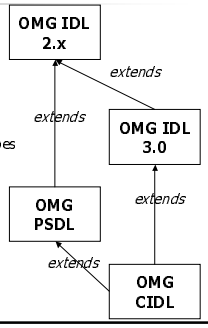
3. component implementation
   - declare component's persisent state in PSDL and some behaviour aspects in CIDL
   - compile to get skeletons; fill in with business logic; compile to joint result to get compiled libraries

Esimerkki

```
abstract storagetype CustomerState{
   state long AcctNum;
   state ...
}
storagetype PortableCustomerState implements CustomerState{};
typedef sequence <Customer> CustomerList;
abstract storagehome CustomerStorageHome of Customer {
      primary key AcctNum(AcctNum);
      factory create(AcctNum, Name, ...);
}
storagehome PortableCustomerStorageHome implements
   CustomerStateHome{};
```

Relations between OMG Definition Languages

- OMG IDL 2.x
  - Object-oriented collaboration
  - i.e. data types, interfaces, and value types
- OMG IDL 3.0
  - Component-oriented collaboration
  - i.e. component types, homes, and event types
- OMG PSDL
  - Persistent state definition
  - i.e. [abstract] storage types and homes
- OMG CIDL
  - Component implementation description
  - i.e. compositions and segments



Component usage patterns

- 7 component categories: 4 CCM, 2 EJB, 1 clustomizable
- servant lifetime policies: method, transaction, component, container

category	usage model	container API type	primary key
service	stateless	session	no
session	conversational	session	no
process	durable	entity	no
entity	durable	entity	yes

Composition

- Defined in CIDL
  - Category
  - Home executor
  - Executor
  - Bindings
  - State management (storage)

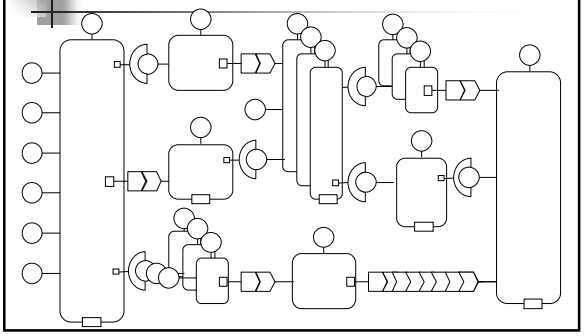
Esimerkki

```
composition entity CustomerImpl {
    implements Customer;
    home executor CustomerHomeImpl
      delegatesTo abstractstoragehome
      CustomerStateStorageHome;
}
```

CCM development project stages

4. component packaging
   - (use interactive tools to produce)/write component descriptor (in XML) to tell CCM runtime how to connect up and manage the implementation
   - package up implementation and component descriptor into a component archive file CAR
5. component assembly
   - An application or part composed of some components with predefined interaction pathways in an assembly archive file AAR
   - Customizes configuration, connections, partitioning to different computers

Building CCM Applications = Assembling CORBA Component Instances



CCM development project stages

6. component deployment and installation
   - Prior to this: system admin has installed and configured runtime environement
     - Installer program for CARs
   - Deploy component factories and managers
7. Runtime: component instance activation
   - Part of the application logic
   - Components are activated by the container POA

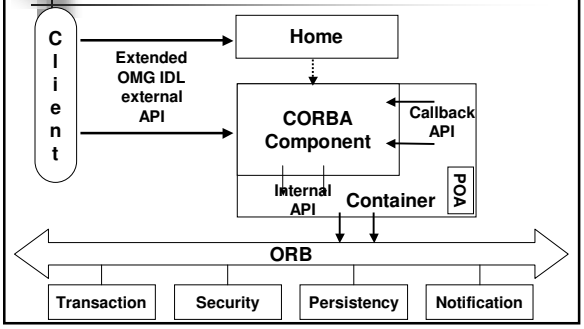
Esimerkki

```
<componentassembly id="374...2304">
<description>Assembly descr for example
</description>
<componentfiles>
<componentfile id="ShoppingChartFile">
<fileinarchive name="shoppingchar.car">
</componentfiles>
<partitionhg>
<homeplacement id ="ShoppingCartHome">
<componentfileref idref="ShoppingCartFile">
</homeplacement>
...
</partitioning>
<connections>
<connectinterface>
<usesport>
<usesidentifier> Check1 </usesidentifier>
<homeplacementref idref="ShoppingChartHome">
...
```

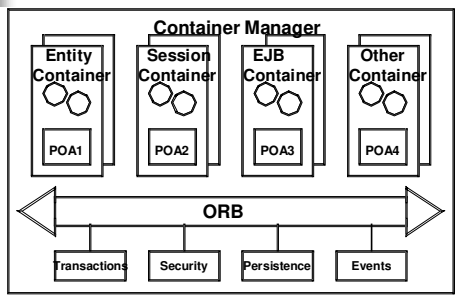
Komponenttien suoritusaikainen ympstö

- component container model
  - Framework for component application servers
  - Mostly built on the POAs
    - Automatic activation/deactivation
    - Resource usage optimization
    - Provides simplified interfaces for corba services
    - Uses callbacks for instance management
  - Container encapsulates 1-N POAs
  - References exported thourhg component home finder, naming or trading

The Container Architecture



The Container Server Architecture



The Client Programming Model

- Component-aware and -unaware clients
- Clients see two design patterns
  - Factory – Client finds a home and uses it to create a new component instance
  - Finder - Client searches an existing component instance through Name Service, Trader Service, or home finder operations
- Optionally demarcation of transactions
- Could establish initial security credentials
- Invokes operations on component instances
  - Those defined by the client-side mapping

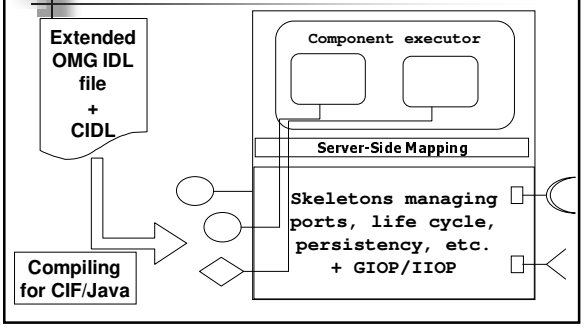
Cmponent Implementation Framework

- CIF defines a programming model for constructing component implementations
  - How components should be implemented
- Facilitates component implementation
  - "only" business logic should be implemented
    - Not activation, identify, port management and introspection

  => Local server-side OMG IDL mapping
    - Interactions between implementations and containers
- Manages segmentation and persistency

  => Component Implementation Definition Language

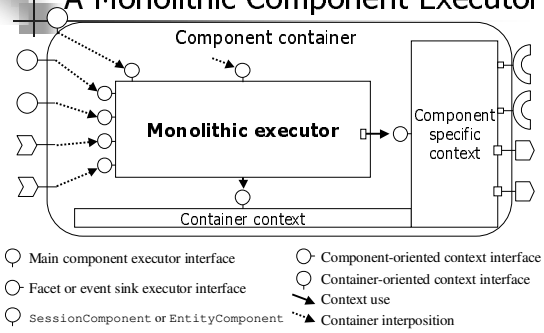
Component Implementation Framework to Component Skeleton Generation



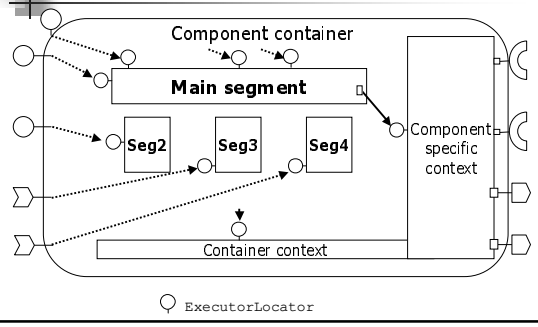
Component Implementation Definition Language CIDL

- describes component composition
  - Aggregate artifacts required to implement a component and its home
- Manages component persistent state
  - Persistent state definition language PSDL
  - Links storage types to segmented executors
- Generates executor skeletos providing
  - Segmentation of component executors
  - Default implementations of callback operations
  - Component's state persistency

A Monolithic Component Executor



A Segmented Component Executor



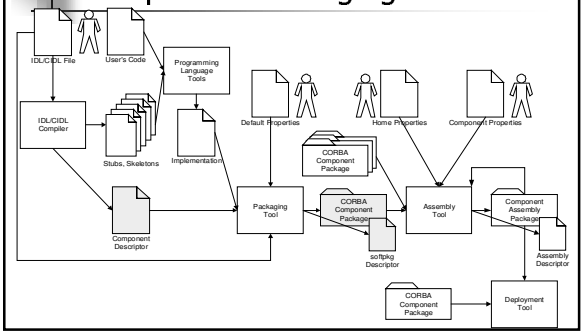
Packaging and Deployment

- Packaging and Deployment of Components
  - Components are packaged into a self-descriptive package
  - Packages can be assembled
  - Assemblies can be deployed
- Helped by XML descriptors
- Packaging and Deployment model Allows interoperability between deployment tools and containers

CCM Applications Deployment

- It is necessary for an application to
  - List component instances
  - Define logical location and partitioning
  - Specify connections between components
- It is necessary for a component to
  - Specify its elements
    - interfaces, implementations
  - Describe system requirements
    - OS, ORB, JVM, library releases, ...
  - Specify its initial configuration
- It is necessary for a connection to
  - Associate related component ports

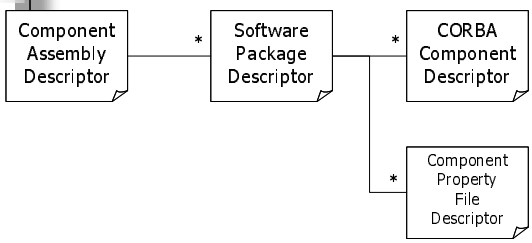
Component Packaging Artifacts



XML Descriptors Overview

- Software Package Descriptor (.csd)
  - Describes contents of a component software package
  - Lists one or more implementation(s)
- CORBA Component Descriptor (.ccd)
  - Technical information mainly generated from CIDL
  - Some container managed policies filled by user
- Component Assembly Descriptor (.cad)
  - Describes initial virtual configuration
    - homes, component instances, and connections
- Component Property File Descriptor (.cpf)
  - name/value pairs to configure attributes

Relationship Between CCM XML Descriptors



Component Assembly Package

- Archive (ZIP file) containing
  - One or more component packages, either
    - Including a package's contents
    - Including the original package
    - Referencing the package by URL
  - Property File Descriptors defining initial attribute values
  - Component Assembly Descriptor (.cad)
    - Defines home instances to be created
    - Defines component instances to be created
    - Defines connections between ports to be made
- Self-contained and self-descriptive unit
- For automatic and easy "one step" deployment
- No programming language experience necessary

Component Assembly Descriptor (.cad)

- References one or more Component Software Descriptors
- Defines home instances and their collocation and cardinality constraints
- Defines components to be instantiated
- Defines that homes, components or ports are to be registered in the ComponentHomeFinder, Naming or Trading Service
- Defines connections to be made between component ports, e.g. receptacles to facets and event sinks to event sources

Software Package Descriptor (.csd)

- Descriptive general elements
  - title, description, author, company, webpage, license
- Link to OMG IDL file
- Link to default property file
- Implementation(s)
  - Information about Implementation
    - Operating System, processor, language, compiler, ORB
    - Dependencies on other libraries and deployment requirements
    - Customized property and CORBA component descriptor
  - Link to implementation file
    - Shared library, Java class, executable
  - Entry point

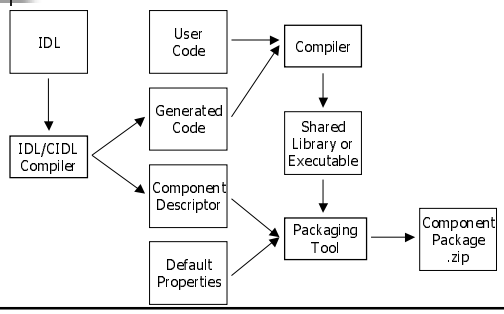
CORBA Component Descriptor (.ccd)

- Structural information generated by CIDL
  - Component / home types and features
  - Ports and supported interfaces
  - Component category and segments
- Container policies filled by the packager
  - Threading
  - Servant lifetime
  - Transactions
  - Security
  - Events
  - Persistence
  - Extended POA policies
- Link to component and home property files

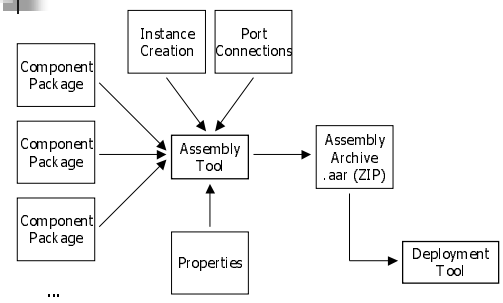
Property File Descriptor (.cpf)

- Used to set home and component properties
  - However, it could be used for anything
- Contains zero or more name/value pairs to configure attributes
- Referenced by…
  - Software Package Descriptors to define default values for component attributes
  - CORBA Component Descriptors to define default values for component or home attributes
  - Assembly Descriptors to configure initial values for home or component instances

Component Packaging



Component Assembly



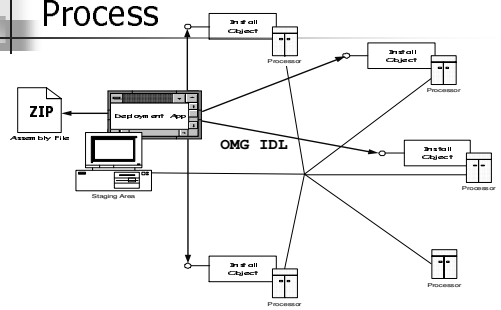
Deployment

- An Assembly Archive is deployed by a deployment tool
- The deployment tool might interact with the user to assign homes and components to hosts and processes
- The deployment application interacts with installation objects on each host

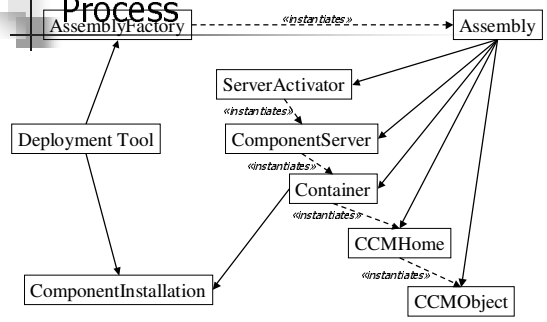
Deployment Objects

- `ComponentInstallation`
  - Singleton, installs component implementations
- `AssemblyFactory`
  - Singleton, creates `Assembly` objects
- `Assembly`
  - Represents an assembly instantiation
  - Coordinates the creation and destruction of component assemblies and components
- `ServerActivator`
  - Singleton by host, creates `ComponentServer` objects
- `ComponentServer`
  - Creates `Container` objects
- `Container`
  - Installs `CCMHome` objects

The Component Deployment Process



The Component Deployment Process



More Information

- CORBA 3.0: New Components Chapters
  - OMG TC Document ptc/2001-11-03
- CORBA 3 Fundamentals and Programming
  - Dr. John Siegel, published at John Wiley and Sons
- "The CCM Page", Diego Sevilla Ruiz
  - *http://www.ditec.um.es/~dsevilla/ccm/*

Kurssilukemisto

- Vinoski, S., New features for CORBA 3.
- Wang, Schmidt, O'Ryan: Overview of the CORBA Component Model

Muuta materiaalia

- Ohjelmointitutoriaaleja
  - http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/docs/tutorials/Quoter/
  - useimmilla toimittajilla omansa (Sun+Java+CORBA, etc)
- CORBA-sivustoja
  - http://www.cs.wustl.edu/~schmidt/corba.html
  - http://www.puder.org/corba/

Open Source CCM Implementations

- OpenCCM from LIFL & ObjectWeb
  - Java on ORBacus 4.1 & OpenORB 1.2.1
  - http://www.objectweb.org/OpenCCM/
- MicoCCM from FPX & Alcatel
  - C++ on MICO
  - http://www.fpx.de/MicoCCM/
- FreeCCM from Humboldt University
  - C++ on ORBacus 4.1
  - http://sourceforge.net/projects/cif

Commercial CCM Implementations

- EnagoCCM from IKV++/Fraunhofer FOKUS
  - C++ on MICO & ORBacus 4.1
  - ritter@fokus.gmd.de
- EJCCM from CPI Inc.
  - Java on OpenORB 1.3.x
  - http://www.ejccm.org
- K2 from ICMP
  - C++ on various ORBs
  - http://www.icmgworld.com