Technical White Paper

A Framework for COBA Server

19.11.2001

COBA
Connected Open Building Automation

Index

COBA TECHNICAL WHITE PAPER

Introduction

The control network meets the data network – COBA offers a standard for access to all building functionality

Scope

This document defines the COBA control system and specifies the role of the control server within COBA control system. The architecture and the framework for the control server, i.e. COBA Server, are also specified. The document describes the instrumentation mechanisms of controllable resources and outlines a set of basic services offered by the framework. Detailed specifications of these services and related Java classes are released in separate documents.

The other main purpose of this document is to define the concepts and terms used in connection with COBA Server. This provides a common reference point and guidelines for various interface specifications to be released at later stage.

Readership

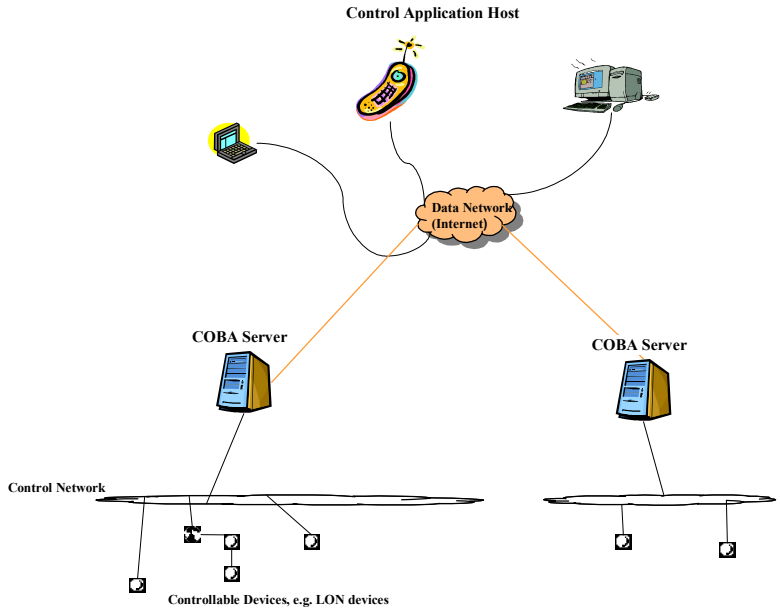
This document is meant to give a high-level understanding of COBA Server, especially for people working in the area of control systems (e.g. solution providers, system integrators, software designers and implementers). The document links to various detailed specifications. Readers such as software developers can follow the links to appropriate documents for in-depth information[1] of different aspects of COBA Server.

COBA Control System

As shown in Figure 1, COBA control system is a tiered system. It consists of control applications, data network, COBA Server and control network. COBA Server connects to control network and to data network. COBA Server acts as an agent for control applications that reside in the data network, through controlling the devices connected to the control network. Control of resources involves tasks such as monitoring, configuration, handling of alarms, etc.

[1] Since most of the interface specifications are yet to be written, some of the requirements imposed upon the interfaces are highlighted by the **Interface Requirements** heading. The developers shall pay attention to those requirements.

Figure 1 COBA Control System



Control Network

A control network includes the operative building automation functionality. Control network is not considered as part of the data network. Building automation functionality influences e.g. heaters, lights, air-handling units, etc.

Control application

A control application runs in the data network and controls devices connected to the control network. Control applications react to value or status changes of the controlled resources. For example, when an external event triggers an alarm in the control network, COBA Server provides the alarm to control applications.

External Application

From COBA Server point of view, a control application is an external application. The terms control application and external application are used interchangeably in this document.

COBA Server

COBA Server integrates the control network and the data network together, in order to increase the accessibility of the control network. Control applications running on the data network allow users to interact with the devices on the control network.

COBA Server communicates with controllable devices on the control network using a control network protocol. The first reference implementation of COBA Server supports LonWorks technology using LonTalk protocol. LonWorks is commonly used in existing control networks. COBA Server can also provide support for other control protocols, such as EHS, X10, etc. The control network may also include devices using proprietary or industry standard protocols.

The control server communicates with external applications over a TCP/IP network using messaging technology such as HTTP, RMI, CORBA, etc.

Hardware

Any hardware with the following qualities is a potential candidate for COBA Server:

- Sufficient CPU power, memory and storage capacity
- Capability to accommodate appropriate networking peripherals that connect to the control network and to the data network.

The selected hardware for the first reference implementation is Nokia Home Server.

Software

Since COBA Server is *NOT* intended to be used by a single control application alone, COBA Server software consists of, by design, COBA framework and service components. With this approach COBA Server can support a wide range of applications with less effort than otherwise.

COBA framework provides building blocks for service components. It is implemented on top of Java virtual machine using a set of Java packages including but not limited to OSGi. The first reference implementation will be based on Linux operating system.

A service component provides specific functionality that is either common to most of the control applications or specific to a particular application. These service components are referred to as COBA services. Some of the service components can be accessed directly by control applications (using HTTP or RMI mechanism); some can only be accessed indirectly. COBA services that can be accessed directly are referred to as *agent services*. COBA services that can only be accessed indirectly are referred to as *system services*.

Release Kit

COBA framework release kit consists of COBA framework and a set of agents. The release kit also includes interface and API specifications as well as user guide documentation. The framework is bundled into a number of packages.

COBA control system solution providers can add additional system services and agent services to the server if the services in the release kit do not have the functionality required by the application in question.

Design Requirements for COBA Framework

Requirements and goals for COBA framework design are listed and explained below.

Support for distributed applications

COBA framework is designed so that it can support control applications running in different locations. The framework will support multiple control applications running simultaneously.

The components in COBA Server act as mediator between the application and the control network. COBA framework shall have as little business logic as possible. This way, it can embrace wider range of control applications developed by third party solution providers.

Support for dynamic component update

COBA framework shall provide a set of common COBA services. Additional services will be created by developers or integrators of COBA control system solutions. All services and controllable resources can be dynamically loaded, unloaded or updated.

Scalable control system

COBA executable components shall be independent components that can be plugged into the control server when needed. COBA based control solutions, therefore, can scale from small footprint to large control servers, controlling from small to large number of devices.

Low investment for control application development

COBA framework shall include a set of COBA services and reusable objects such as beans. These will provide functionality that is required by a wide range of control applications. This way, required development effort and cost of control applications are minimized.

Short time to market

This goal dictates the use of existing and commonly deployed computing technologies whenever possible.

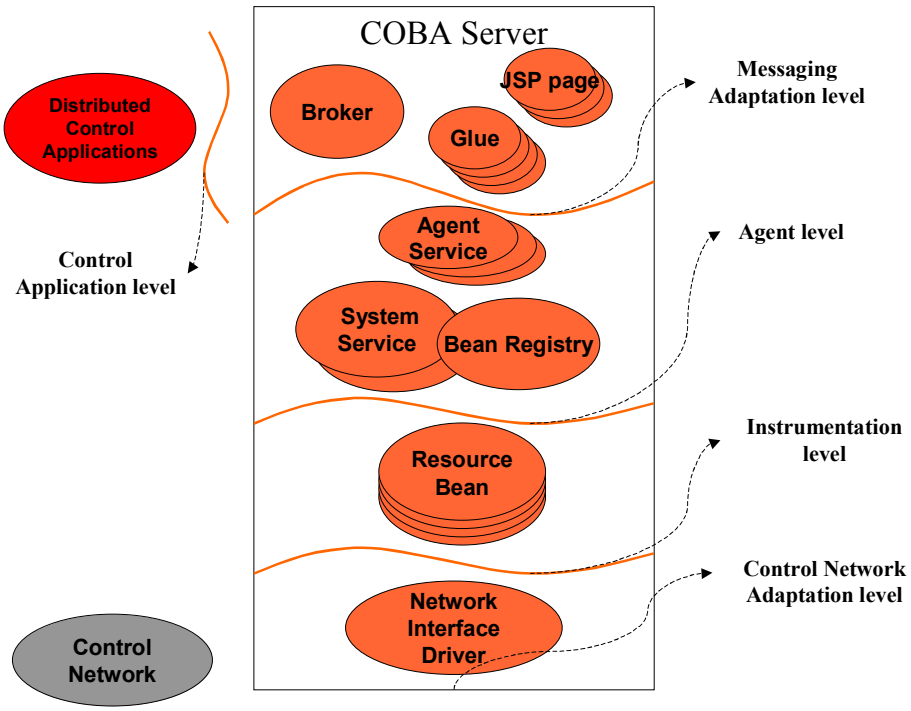
System Architecture

The architecture of COBA system is based on Java Management Extension (JMX) architecture. The architecture adopts many of JMX's key concepts and approaches, like the concept of agents, the use of JavaBeans, etc.

From specification point of view, COBA system architecture is divided into the following levels as shown in Figure 2:

- External distributed control application level.
- Messaging adaptation level
- Agent level
- Instrumentation level
- Control network adaptation level

Figure 2 COBA System Architecture



Levels in JMX and COBA architectures correspond to each other as follows:

- connector and protocol adaptation level in JMX ~ messaging adaptation level in COBA

– COBA includes an additional control network adaptation level at the bottom.

Control Network Adaptation Level

The control network adaptation level provides a specification for implementing network interface drivers. Network interface drivers allow physical control devices to interact with the rest of the COBA Server.

Network interface drivers for the control network must conform to this interface specification. The first reference implementation of control network device drivers is LonTalk protocol.

Instrumentation Level

The instrumentation level provides a specification for implementing logical representations of controllable resources.

A controllable resource can be a physical device, a group of physical devices, a logical device, a service, etc. The instrumentation of a controllable resource is provided by one or several Resource Beans, i.e. RBeans. Control applications can configure and monitor the instrumented resource through RBeans.

Messaging Adaptation Level

The messaging adaptation level includes the existing specifications and / or provides new specifications for messaging transportation and communication mechanisms between control applications and COBA Server.

Control applications communicate with COBA Server through a messaging adaptation component, which is also referred to as the 'broker' in the rest of this document.

The default mechanism for exchanging information about controllable resources and their attributes is XML. The default mechanisms for messaging communication are RMI and HTTP. RMI broker and HTTP broker are the default brokers included in the COBA framework. RMI broker uses Java RMI communication mechanism, while HTTP broker uses JSP to relay requests and replies from and to WEB based control applications.

COBA Server can also have other types of brokers without having to re-implement COBA services. For example, if the control applications are based on CORBA technology, there will be a CORBA broker for linking the applications to the existing services.

Agent Level

The agent level provides a specification for implementing agents, and interface specifications for implementing services provided by COBA Server. Services provided by COBA Server are referred to as COBA services. The services that are exposed directly to remote control applications are called agent services, while those accessible only by components inside the server are called system services.

An agent is an execution unit, which acts as the liaison between RBeans and the control application. An agent composes of a broker, one or more agent services, a collection of system services, a set of RBeans and the RBeans acting as device drivers.

Agents make RBeans available to control applications. Agents are built in a standard way without having to understand the semantics of the controllable resources. Controllable resources are hidden behind the RBeans and the network interface drivers. Furthermore, agents can be built so that they do not need to know the control applications that use them.

Agents are implemented or integrated by developers of the COBA control system solutions. The framework kit includes several common agents, which can be used as such by a wide range of control applications.

Distributed Control Application Level

The distributed control application level implements control applications operating on RBeans over agents. The specifications for distributed applications are outside the scope of this document.

The design of COBA framework is not concerned of the functionality of the control applications. The framework provides only the mechanism and services to allow appropriate RBeans to be plugged into the server, hence enabling control of instrumented devices.

The combination of distributed control applications and agents form a complete control solution. Control applications may also cooperate with one another across the network to provide distributed and scalable control solutions.

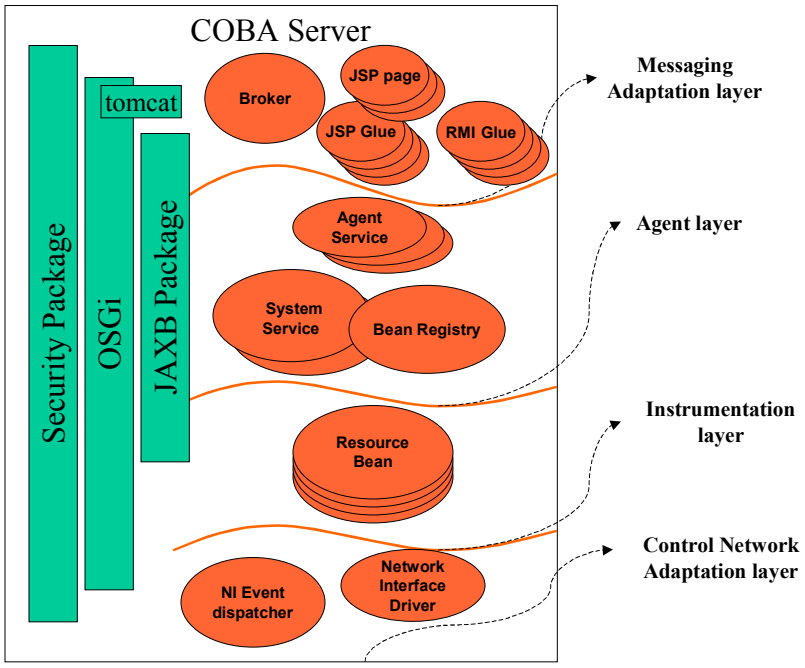
XML and Configuration File

XML is used to describe the characteristics of the controllable resources in text. A configuration file in XML format is the default way for COBA Server to configure the system at system initialization stage.

COBA Server Components

In component view, COBA Server is divided into layers as shown in Figure 3. Each layer consists of components offering a specific set of functionality. Figure 3 also shows the key packages that are part of the COBA Server infrastructure, but that are not part of the J2SE.

Figure 3 COBA Server Components



Control Network Adaptation Layer

Network Interface Driver (NID) and NI Event Dispatcher are the main components in the control network adaptation layer. Each of these components includes a set of Java interface classes defining the operations available to the clients, as specified in the COBA Network Interface Driver Specification.

NID

The Network Interface Driver communicates with physical resources over a network interface port, using the protocol understood by the resources. Each network interface port is served by only one Network Interface Driver.

The communication protocol is normally a data link layer protocol. The first implemented NID supports LonTalk Protocol. The server can have multiple instances of NID, each

attached to a different port. The server can also have multiple types of NID, each implementing a different protocol.

Requirements for the Network Interface Protocol

The protocol used by physical resources must be able to support setting and getting attribute values of the resources, and to enable the server to determine whether the physical device is still alive.

Interfaces

The key interface methods of NID are 'write' and 'read' operations for setting and getting attribute values of the devices. These operations are by default blocking kind of operations; in other words, the client will wait until the required value is set or returned. NID may also provide non-blocking kind of operations for those clients who do not care about the result of the operation.

The main parameters of these operations are address and payload or address, attribute type and attribute value. In Figure 4, the main interface class of NID is called NIDIF, other interface classes for entities such as address, message header etc are specified in the interface specification.

> **Interface Requirement: Operation Signatures**
>
> The signatures for read and write operations are dictated in part by how attributes are represented as outlined in the ***Attributes*** section in the instrumentation layer. The address signature is a Java class and is specified in the COBA Network Interface Driver Specification.

Relationships

The key client of the NID is the RBean that represents the physical device.

NI Event Dispatcher

The NI Event Dispatcher is responsible for distributing events received from the physical devices. NI Event Dispatcher uses Java event model to distribute events to subscribers.

Interfaces

The EventRegisterIF interface class as show in Figure 4 provides the subscribers a way to subscribe to events based on message types, source address and destination address of event messages. The subscriber of the NI Event Dispatcher needs to implement the EventSubscriber interface class in order to subscribe and receive events.

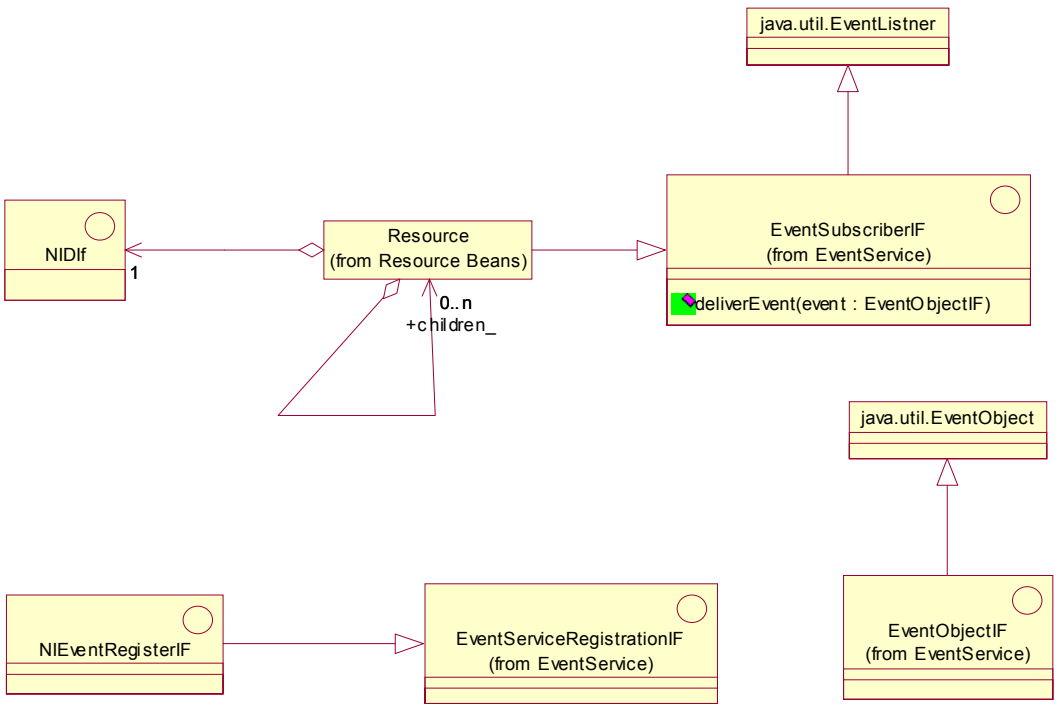
> **Interface Requirement: Be consistent with Event Notification System Service.**
>
> The event distribution related interfaces must be based on the interfaces defined in the Event Notification system service. This provides a uniform way for the clients to handle events, irrespective of the package of the implementation classes.

Relationships

An RBean of a physical device and an RBean of a logical resource can be clients of the NI Event Dispatcher. However, for performance reasons it is recommended to keep the number of subscribers at a minimum. As such, the clients of the NI Event Dispatcher component are mostly management or administration related entities. Such clients normally subscribe to a group of events and know how to deal with received events.

The event notification system service may also be a client of the NI Event Dispatcher, subscribing to certain categories of events. In this case, the event notification system service acts as an event adaptor and allows entities interested in specific events to subscribe to the event notification system service instead of the NI Event Dispatcher. One of the advantages of this approach is that potential event subscribers become less dependent on the NID component. COBA Event Notification Service Interface Specification defines the events that the event notification system service can subscribe to.

Figure 4 Network Interface Driver Relationships



Note: This Figure and the following few Figures are to demonstrate relationships described in the text, they do not represent a complete view of the relationships. Refer to the appropriate interface specifications for detailed descriptions. In addition, the class names appeared in these figures may change. The interface specifications are the ultimate authorities in the choice of the class names.

Instrumentation Layer

The instrumentation layer consists of RBeans, which are JavaBeans. An RBean represents a controllable resource. A controllable resource can be a single physical device, a group of devices, a piece of equipment, a facility, etc. Examples of controllable resources are sensors, pumps, rooms, floors, buildings, etc. A COBA service that possesses Java Bean characteristics can also be an RBean and be controllable. For example, a configuration agent service can be implemented so that the users allowed to change device attributes can be added or removed at run time.

Since RBeans are JavaBeans, they can be manipulated visually with a builder tool for building applications. The builder tool, however, is not part of the COBA framework kit.

RBeans

XML Binding

The JAXB package is used to bind resources expressed as XML elements to Java classes. RBeans must be derived from appropriate classes generated from JAXB.

Threading Model

To allow accessing RBeans simultaneously by multiple clients, the RBean assumes that it is running in a multi-threaded programming model. Therefore access to shared data needs to be synchronized. It is the responsibility of the RBean, not the client, to make sure that data is protected against simultaneous access. If an RBean cannot do this it shall be well documented in the appropriate specification.

Text Representation

Attributes or properties of the RBean will be represented in XML format. All RBeans include methods to marshal to or un-marshal from a valid XML document. The schema for RBeans is defined in COBA XML Specification. Marshaling and un-marshalling methods are provided by classes generated by JAXB package.

Common Attributes

Common attributes of Resource Beans are gathered in one class, the ***Attributes Information*** class. The key characteristics are described below. Refer to the COBA Resource Bean Interface Specification for details.

- ***Type***

  There are several kinds of resources. Resources are categorized into types. The type is represented by a ***Type*** class. New types can be created by creating a sub-class of the ***Type*** class.

- ***Identifier***

  Each controllable resource has a unique ID within the scope of the system controlled by a COBA Server. Each RBean within the COBA Server has a unique ID. The ID of a resource is specified in the XML document. It is the responsibility of the user who configures the control system to ensure that all identifiers are unique.

If COBA Server discovers that a controllable resource has a duplicate name, the corresponding RBean will not be created and an exception will be generated.

- ***Address***

  The address of an RBean is the network address of the corresponding physical resource. Different types of resources may have different ways to address themselves. The address interface class defines the interface for retrieving or setting the address. Bean developers may need to supply a specialized address class along with each new type of resources.

Interface

A resource is exposed to control and management through the interface of its RBean representative. The interface of an RBean consists of the methods for reading and writing its attributes and for invoking its operations. Because each type of resources may include different attributes that are not necessarily shared by all resources, the RBeans may not always have the same interface methods. However, there is a set of methods that are provided by all RBeans. These methods are captured in the ResourceIF class as shown in Figure 5. Note, the methods shown in the figure are for demonstration purpose, the complete methods and up to date details are defined in the COBA Resource Bean Interface Specification.

The control interface of a standard RBean consists of

- Constructors
- Attributes, i.e. the properties that are exposed through getter and setter methods.
- Operations, i.e. the remaining methods exposed in the interface.

Naming Convention

When defining the interface for RBean classes, a set of naming rules should follow the pattern of JavaBean component architecture. The naming rules for COBA are a subset of JMX rules and are outlined below. See Resource Bean Interface Specification for details.

- ***Class***

  The name of an RBean's Java interface is formed by adding the ***IF*** suffix to the RBean's Java class name. For example, the Java class *Pump* would implement the Java *PumpIF* interface class. The interface of a *Pump* is referred to as its *Pump interface(PumpIF)*.

- ***Attributes***

  An attribute is represented by an attribute name and is associated with a specific type. The name of an attribute follows the same rule of naming a Java class; the type of an attribute is a valid Java class or a primitive Java type.

Interface Requirement: Attribute Identity

XML uses the attribute name to denote an attribute of a resource. If the name string is not efficient enough to identify an attribute in NID, numbering scheme can also be used to identify an attribute. NID interface designer should consider this and decide whether an attribute shall also be represented by a unique number or not.

The following rule is used to identify attributes over Resource Bean interfaces:

```
public AttributeType get AttributeName ();

public void set AttributeName (AttributeType value);
```

If a class definition contains a matching pair of getAttributeName and setAttributeName methods that take and return the same type, these methods define a read-write attribute called attributeName. If a class definition contains only one of these methods, the method defines either a read-only or write-only attribute.

The AttributeName cannot be overloaded. There cannot be two setters, getters or a getter and setter pair for the same name that operates on different types.

The AttributeType may be of any Java class, or an array of any Java class.

When the type of an attribute is an array type, the getter and setter methods operate on the whole array. Indexed method is used for accessing individual array elements.

- ***Operations***

  In RBean, an operation is a Java method specified in its interface and implemented in the class. Any method in the RBean interface that doesn't fit an attribute design pattern is considered to define an operation.

- ***Case Sensitivity***

  All attribute and operation names are case sensitive.

Relationships

As shown in Figure 5, the Resource Bean interacts with or relates to the following components:

- Network Interface Component
  - NID

    The driver that communicates with the "real" resources. In the case where the RBean does not represent a physical device, the driver is a dummy driver. This is optional for RBeans that do not represent physical devices.

  - NI Event Dispatcher

    The event register for Event Dispatcher in the Network Interface Component.

- Data Store System Service
  - Serializable

    The interface to make the RBean persistent and to retrieve the persistent data of the RBean.
- Bean Keeper System Service

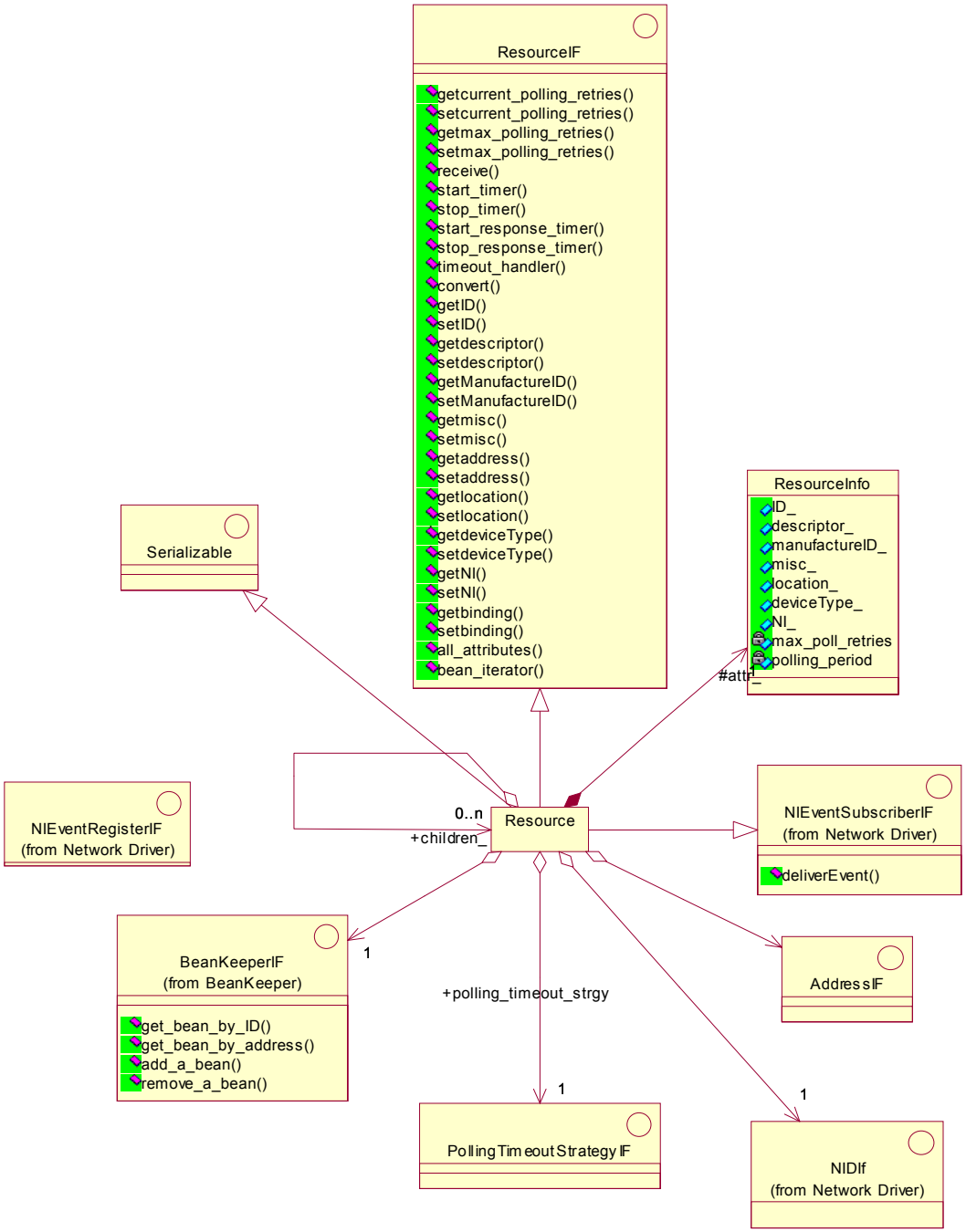
  The administrator that manages the life cycle of RBeans.
- Event Notification System Service

  The event register for the Event Notification system service in the agent layer. This is optional.
- Polling Timeout Strategy

  The interface for implementation of actions when there is no reply to a number of consecutive polls. This is a utility class for the RBean.

An RBean can interact with multiple instances of event dispatcher, as long as each of these dispatcher implements the same set of event dispatching interfaces.

Figure 5 RBean Relationships



Agent Layer

The components in the agent layer are COBA services.

A COBA service is a system service if it is not meant to be accessed directly by control applications. COBA services provide internal services to other parts of COBA Server.

A COBA service is an agent service if it can be used directly by control applications. Since control applications are only allowed to operate on Bean interfaces, an agent service is also a Java Bean.

System Service

The following system services are included in the COBA framework kit:

- Bean Keeper, a central registry of RBeans. Bean keeper is a lightweight Java naming service. The interface of the bean keeper can be divided into two groups: registration of RBeans and object references resolution of RBeans, from bean's ID or address attributes. See COBA Resource Bean Keeper Interface Specification for details.
- Event Notification, a publisher-subscriber kind and event type based notification service.
- User Access Control, for controlling users' access rights. User Access Control is built upon Java security model. See COBA System Security System.
- Dynamic Loader, allows the RBeans to be instantiated using Java classes and native libraries to be dynamically downloaded from network. See COBA Dynamic Loader Specification and OSGi Specification.
- Data Store Access, provides a uniform interface to store and retrieve information about an object such as an RBean, etc. The data store can be a shared memory, files, relational database management system, etc. See COBA Data Store Access Specification for details.

Agent Services

The following agent services are included in the COBA framework kit:

- Event Notification, a publisher-subscriber kind event type based notification service. Event Notification is built on top of Event Notification system service. See COBA Event Notification Interface Specification.
- Logger, provides the applications a mechanism for selectively logging to certain activities or events. See COBA Logger Interface Specification.
- Timer, provides time based notification services. See COBA Timer Interface Specification.
- Monitor, provides polling mechanism to monitor registered resources periodically. See COBA Monitor Interface Specification.
- Access Control, allows selected clients to set up user profiles for authorization and authentication purposes. This agent service is built on User Access Control system service.

Messaging Adaptation Layer

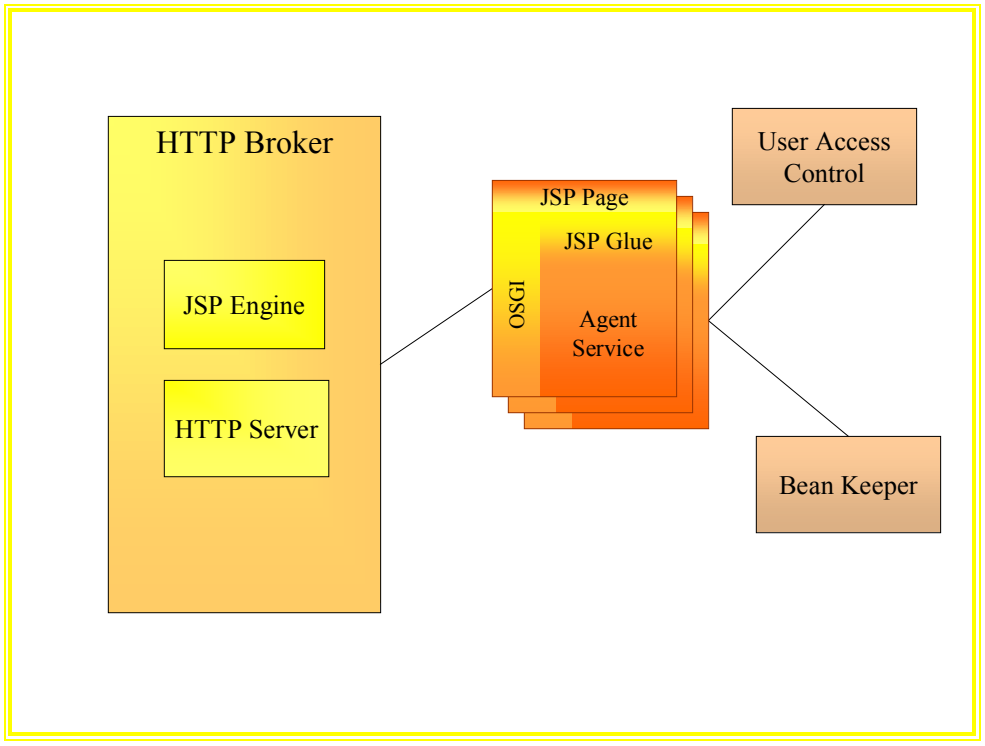
The messaging adaptation layer handles message transportation. The layer ensures that a message or a request from the control application is passed to the right RBean with the appropriate context; similarly a response from the RBean is returned to the right client. This process normally involves message marshalling and un-marshalling as well as handling of protocol operations.

Several messaging mechanisms can be used by a control application to access the control server. Examples of such mechanisms include JSP, RMI, CORBA, DCOM, SOAP, etc. COBA Server supports JSP and RMI mechanisms in the first reference implementation. XML is used to represent web page content.

The major components based on JSP technology are HTTP broker, JSP glue and JSP pages. The major components based on RMI technology are RMI broker and RMI glue.

JSP Technology

Figure 6 HTTP Broker and COBA Services



HTTP Broker

The HTTP Broker relays JSP requests from the control application to the appropriate agent service and relays responses from the agent service back to the control application.

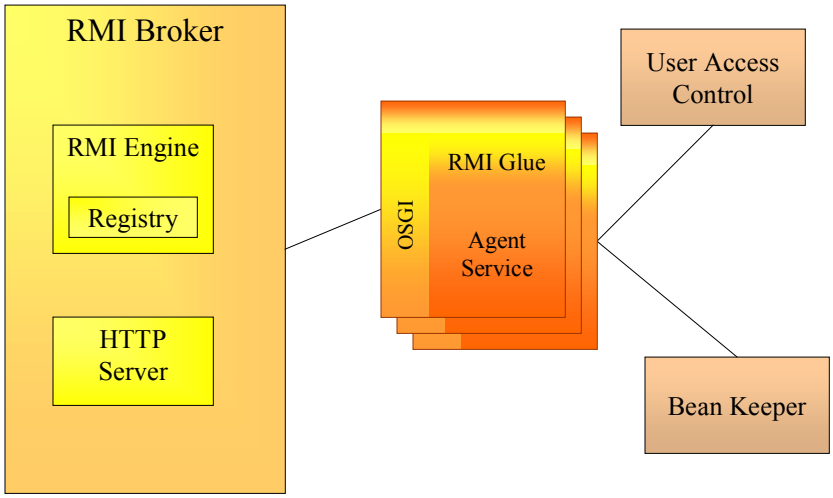
As shown in Figure 6, the major components in HTTP Broker are JSP Engine and HTTP Server. The components implement HTTP protocol and relay messages. At the time of writing, tomcat from Apache is chosen as the JSP engine (and HTTP server).

JSP Glue & JSP page

The JSP Glue is one or more Java classes making the agent service available to all the control applications (under the control of the HTTP broker). A JSP page embeds the application logic within standard HTML (or XML) templates and invokes appropriate methods implemented by the JSP Glue. The JSP pages are not likely to be shared among different control applications.

RMI Technology

Figure 7 RMI Broker and COBA services



RMI Broker

RMI Engine and HTTP server are the main components in the RMI broker.

RMI Engine consists of a set of classes provided by Java RMI packages that enable RMI. One of the key elements in the engine is the registry. The RMI glue calls the registry to associate a name with an object (an agent service or an RBean), the control application looks up the object by its name in the registry and then invokes a method on it. The registry implemented by COBA is based on the implementation provided by Java RMI registry.

The object references of RBeans in COBA Server are HTTP style of URL names. The RMI broker uses HTTP server to load class bytecodes.

RMI Glue

For example, the RMI glue of a Pump RBean involves implementing the necessary RMI interfaces such as java.rmi.Remote and extending the Pump implementation to expose the entity Pump to the control application.

A control system may include hundreds or thousands of controllable resources. To avoid straining the resources, not all the RBeans shall be active and remain active. In addition, RBeans need the ability to store persistent references to objects, so that communication among objects can still be established after a system restart. For this reason, the object activation mechanism in RMI is used for providing persistent references to RBeans representing physical resources. In general, RBeans representing agent services are active and remain active. Other RBeans will be brought to active when they are referred to.

COBA Agents

A COBA agent is an entity that runs in the control server and acts as the liaison between the RBeans and the control application. COBA agents export agent services to control applications.

A COBA agent involves the following components:

- Bean Keeper
- A set of RBeans representing the controlled resources
- One or more agent services
- JSP glue for web based control application, or RMI glue for RMI based control application.

COBA agent uses the User Access and Security system services to control the access rights of the agent services.

There may be more than one COBA agent running on COBA Server simultaneously. The agents can either share the same Bean Keeper or have their own one, depending on the characteristics of the agents.

Control applications access the agents through the interfaces of the agent services. The document How to Build a COBA Agent and a COBA Control Solution describes how to create an agent by integrating RBeans, Bean Keeper, system services, agent services, etc.

The following common agents are included in the framework:

- Configuration Agent
- Query Agent
- Event Notification Agent

The solution provider can also build a monitor agent based on the monitor agent service.

Common Agents

Configuration Agent

The Configuration Agent is responsible for configuring the control network. In essence, it exports the Bean Keeper interface methods to control applications. See COBA Agents Specification for details.

Query Agent

The Query Agent allows control applications to retrieve attributes of the controlled resources available to the agent in question. See COBA Agents Specification for details.

Event Notification Agent

The Event Notification Agent allows the control applications to receive events based on either event types or timers. The agent exports the interfaces of the Event Notification services to the appropriate clients.

See COBA Agent Specification for details.

Application Specific Agents

The control solution providers can implement their own agents as counterparts of the control applications running in the data network. The framework of COBA Server provides several building blocks for solution providers for implementing and integrating the agents. See How to Build a COBA Agent and a COBA Control Solution for details.

Terminology and Abbreviations

Agent

An agent is the execution CPU and memory resource that performs specified services.

Agent Service

A COBA service that is also an RBean. In other words, an agent service is a COBA service that can be accessed directly by the control application.

Bean Keeper

A registry of RBeans.

Broker

A Java component in the server that handles messaging adaptation over data network.

COBA Server

A mediator between the control application and the control network, allowing the control application to control and manage the network of controllable devices.

Component

A physical unit of implementation with well-defined interfaces, intended to be used as a replaceable part of a system. The definition comes from the UML Reference Manual.

Control application

A program that runs in the data network to manipulate or monitor the devices in the control network. A control application often reacts to changes in the states of these devices.

Control Network

A network consisting of controllable devices, connected to each other according to some protocol.

Control Server

See COBA Server.

CORBA

Common Object Request Broker Architecture from OMG

COBA Service

A component in the COBA Server that offers a specific set of functionality. A COBA service is either an agent service or a system service.

Core Service

A COBA service that is not an RBean. In other words, a core service is a COBA service that CANNOT be accessed directly by a control application.

Device Bean

See RBean.

External Application

An External Application is a control application outside the COBA Server.

HTTP

Hypertext Transfer Protocol.

ID

A unique identifier for an RBean within the scope of a system. ID is composed of a sequence of local IDs, starting from the root parent to the specific RBean.

Local ID

A unique identifier for an RBean within the scope of its enclosing parent.

LonWorks

A communication technology used in the Control Network. LonWorks is a trademark of Echelon Corporation.

LonTalk

The control network protocol in the LonWorks technology. LonTalk is a trademark of Echelon Corporation.

NID

Network Interface Driver

RBean

Resource Bean. A representation of a controllable entity, such as a device, a service, etc. An RBean implements the Bean interface. If a Resource Bean represents a physical device, it may also be called as a Device Bean.

Resource Bean

See RBean.

RMI

Remote Method Invocation. A Java technology for developing networked applications without having to know the details of low level networking.

XML

Extensible Markup Language.

References and Related Documents

JAVA Management Extensions Instrumentation and Agent Specification, V1.0

User Guide: How to Build a COBA Agent and a COBA Solution
Yet to be written

User Guide: How to Write a Configuration File in XML
Yet to be written

COBA Resource Bean Interface Specification
Yet to be written.

COBA Bean Keeper Interface Specification
Yet to be written.

COBA Event Notification Interface Specification
Yet to be written.

COBA Data Store Access Specification
Yet to be written.

COBA Device Discovery
Will be done after Release 0.5

COBA Dynamic Loader Specification
Yet to be written.

COBA Logger Interface Specification
Will be done after Release 0.5

COBA Monitor Interface Specification
Yet to be written.

COBA Network Interface Driver Specification
Yet to be written.

COBA System Security Specification
Will be done after Release 0.5

COBA Timer Interface Specification
Yet to be written.

COBA User Access Control Specification
Will be done after Release 0.5

COBA XML Specification
Yet to be written.

COBA Agent Specification
Yet to be written.

Nokia Home Server Hardware Specification
Yet to be written

OSGi Service Gateway Specification Release 1.0