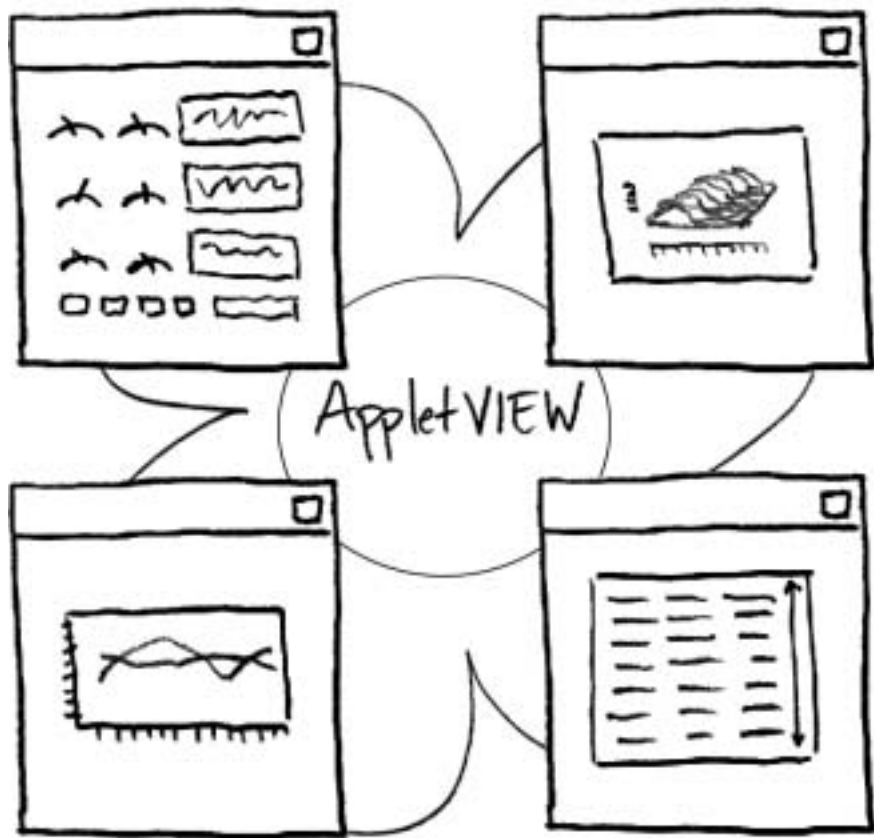
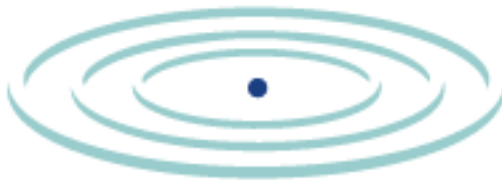


## Conceptual Overview

### What Is AppletVIEW?

The AppletVIEW technology implements chat rooms for computers with windows to see into what the computers are doing. We will refer to this as visualization windows. The chat rooms allow software distributed on a network ability to talk in real-time about mutually held objects and properties. The visualization windows allow people to interact with, control, and graphically visualize data flowing in the system using web browsers or other remote interfaces.





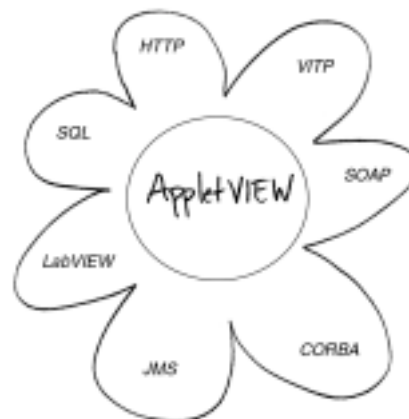
*we connect all sorts of things to the web*

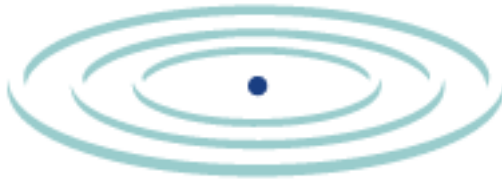
1-2

Whereas normal chat tools are operated by people to send text messages to each other, the AppletVIEW system is operated by software systems to communicate about abstract objects and their properties. The software systems which communicate using AppletVIEW can represent hardware (such as digital instruments, sensors, etc.), software systems (such as databases, legacy or enterprise systems, etc.), or interactive graphical user interfaces (such as web browsers, applications, mobile devices, etc.).



AppletVIEW is an underlying framework and enabling technology which can be used to build distributed, event-driven software applications. It is a very flexible technology which can be either built upon as a core technology or as an addition to existing software infrastructures.





## **Conceptual Summary**

As described below, AppletVIEW consists of three intertwining ideas.

### **1. A system of communicating over a network in real-time about groups of objects and their property values.**

AppletVIEW uses a message passing system to synchronize objects across a network. Arbitrary property values are sent and received on an event-driven basis. Messages are partitioned according to groups called channels - every object is a member of a particular channel, and a client subscribes or connects to a specific channel in order to communicate only about objects in that channel, or to communicate only with other clients using that same channel. This grouping mechanism is a type of name space - a way to organize clients and objects.

### **2. A system of security to control access to and/or authenticate connections to the real-time communication streams.**

In order to connect to an AppletVIEW system, a client must obtain a "token". This token is some binary data that gives location and access information needed by a client to connect to an AppletVIEW server.

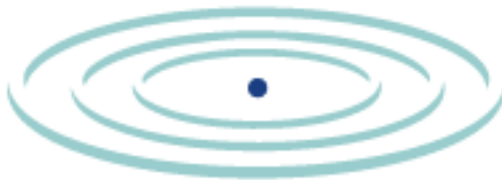
The token is typically delivered as XML having two types of information. The first type is a map of objects and classes represented within a channel. The second type is security information telling who the client is and what they can access. The tokens are generated either statically or dynamically, and can be integrated with normal website security.

### **3. An object-oriented framework for realizing objects associated with a data stream independent of hardware or operating systems.**

The description of objects within AppletVIEW consists of their membership groups, their types, and their properties. The specific client instantiation of an object type is not necessarily known. This allows the platform, environment, operating system, graphical look, user preferences, user permissions, etc. to be unique for any given client of an AppletVIEW system.

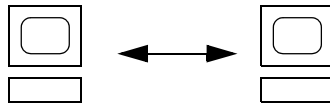
## **Implementation Details of Chat Rooms**

Previously, we described the three concepts that make up AppletVIEW: the message architecture, the security architecture, and the object architecture. The following section describes how these three concepts are implemented.

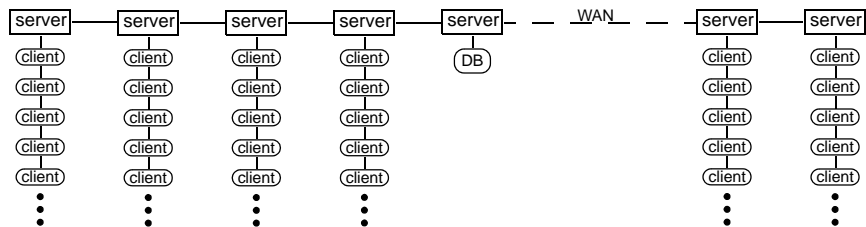


### Message Architecture

AppletVIEW makes use of a modified hub-and-spoke model for client interconnection, where the “hub” is either a single server:



or an interconnected set of servers:

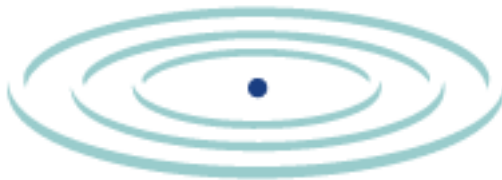


Any message submitted by a client to the system must be submitted through a server, and any message received by a client from the system is received from a server. The message transmission path is an acyclic graph - a message travels from a client through one or more servers to a set of clients, and no loops are allowed.

Within the message passing system, all clients are considered peers. Once submitted to the system, a message is merely delivered to the proper destination(s). The Security Architecture, however, can prevent clients from either submitting or receiving certain types of messages. It can also allow a client to receive permission to be the answering authority for certain types of messages, so that the client is the “owner” of a particular object and the one that maintains its current properties and state.

The types of messages currently used within AppletVIEW are: property values, property requests, method calls, method results, component events, and system messages.

Referring to the OSI Network Model, the AppletVIEW message system sits above the Application Layer protocol. It is a higher level abstraction than the TCP/IP protocols, meaning the messages can run across HTTP, FTP, UDP and a variety of other protocols.



A publish and subscribe model is supported by allowing clients to subscribe to receive certain types of messages, or messages about an individual object or group of objects. When a message is submitted by a client to the system, the message is routed to all other clients subscribed to the type of message it represents.

A service model is supported by allowing clients to send or receive remote methods calls. The object servicing a method call can either be a global object, or an individual object spawned for each client connection. For example, a service to send an email message would typically be a global object service, while a service allowing database queries would typically use a different object for each query.

A global data store model is implemented by the optional use of caches. When a cache is in place for an object, the most recent message about each of the object's properties is kept by the server in the cache. Subsequent requests for a property value are fulfilled from the cache. This allows clients to set and/or get a globally accessible "current" value of an object property.

Messages are platform independent and use the same simple data types defined for properties (described further [below](#)).

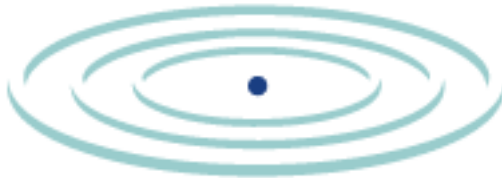
## **Security Architecture**

In order to connect to an AppletVIEW server, a client must first obtain an XML "token" to use as a form of passcard.

Various degrees of security and encryption are supported by the XML passcards. In scenarios of low security requirements, any client which connects to a server and performs the correct message passing handshake is assumed to have a valid passcard for the channel it requests and is allowed access.

In scenarios of high security, as part of the connection handshake the client must present its identity and an array of bytes out of its XML passcard. These two items, along with the client's IP address, connection time, etc. is then verified by the server by any appropriate internal methodology. In this scenario, usually an authentication service creates the array of bytes in the passcard for the specific identity, and the same service subsequently verifies its valid use.

The XML token additionally gives information useful to the client, such as definitions of object classes, objects defined in the channel, default values of properties, etc. If the XML is for a human client, it will also contain abstract information about the visual representation of objects, so that a graphical user interface can be created.



An XML language called VIML is used to express the passcard/token information.

A further feature of VIML is its ability to store an archive of messages in addition to the channel and components it describes. In this way a VIML can be used as a complete history of a user interaction - a description of the interface which was used as well as every action the user performed.

A playback type of component is defined to allow continuous playback or incrementally stepping through a data store.

The message archive can also be used to implement batch processes, such as test recipes or other automated procedures. When the data store is played back, each command is broadcast in the server as if generated by a client. This allows the data store to be treated as a batch script which controls remote clients. Each command can have an optional time stamp associated with it, so that playback can reproduce the correct timing of the commands.

## **Object Architecture**

AppletVIEW uses a very simple object model of channels, components and properties.

### ***Channels***

A channel is a logical grouping of objects, and corresponds to the analogy of a “chat room” - a client connects to a given channel in order to communicate with the components and clients using that channel.

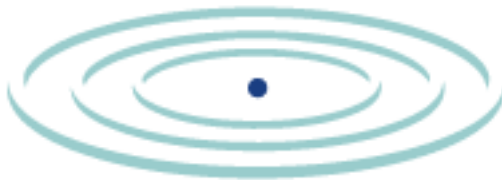
A channel is primarily known by its numerical ID, which is guaranteed to be unique within an AppletVIEW system. A directory service, which normally runs as a client in an AppletVIEW system, can be used to lookup a channel ID based on its name.

A channel can have an arbitrary set of properties associated with it, similar to a component (see below).

### ***Components***

A component is an object which resides in a specific channel. Clients connected to the same channel exchange messages about the components in the channel.

A component is an abstract formulation used to coordinate communication between clients - it may or may not correspond to an actual software object existing somewhere. For example, in the case where all clients are considered peers, the peers could communicate messages about a component, but not actually have a software object representing it anywhere; or in fact each client might have its own internal software representation of the component, so that the component “exists” in multiple



places. On the other hand, in the case where a client is considered the “owner” of a component, the component usually exists in a single place as a software object within that client.

Every component has a class and an arbitrary list of properties associated with it. If the component is of a “well known” class, that class defines the minimum list of properties the component can be assumed to have.

In addition to properties, a component can also service methods and generate events (see [below](#)).

A component is referenced primarily by its numerical ID, which is guaranteed to be unique within its channel. The directory service can be used to look up a component ID based on its channel and name.

### **Properties**

A property on a component is a value of simple type which can be accessed through get/set methods on the component. Currently the list of simple types includes byte, short, integer, float, double, boolean, string, point, dimension, rectangle, color, font, file path, error message and object reference; arrays of those types; and structures of those types. Properties can be read only, write only, or read/write.

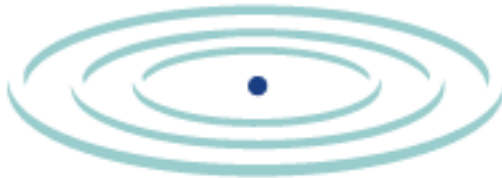
A component usually has a “default” property which is its primary value. For example, a component representing a number may have properties such as its minimum, maximum, units, etc.; but the actual numerical value would be its default property.

Properties have a numerical ID which is unique within its component’s class, a name, and the simple type of the property. Properties are referenced either by name or numerical ID.

### **Methods**

A component method differs from a component property in that it takes a certain number of simple values as parameters, and returns another simple value. A component can have an arbitrary number of methods.

In order for method calls to be serviced, a client must register to receive such method requests. Such a client is considered to be the “owner” of the methods, and every method request for the given channel and component is routed to that client. If multiple clients have registered to receive method requests for a specific channel/component, the first one that registered is given priority. If that client disconnects from the system, ownership is passed to the next client that has registered. If at some point no clients exist to answer a method request, the request returns with a failure code.



**Events** A component event represents a transient state which cannot easily be represented by get/set properties or methods. An example is a time-out. Clients can register to receive events generated by components.

## **Implementation Details of the Visualization Windows**

As indicated previously, AppletVIEW implements “windows” into the chat rooms which can be used by people to oversee or control activity in the rooms.

This is a mechanism for dynamic creation of graphical user interfaces for the components represented in a channel, so that a user can view and/or set properties on the components.

AppletVIEW uses the VIML notation to express information needed to generate interactive visual displays for a channel. If the recipient of the VIML is just a software client (such as a database, digital instrument, etc.), the visual information is usually ignored. If the recipient of the VIML file is a person, however, the information inside of it is used to build a graphical interface.

For “well-known” classes of components, a default list of possible visual properties can be assumed. Properties common to all well-known classes include size, location, foreground, background, enabled, visible and font. Specific classes define additional visual representation properties appropriate to their type. For example, a class representing a graph has visual properties for the scales, plot colors, etc.

The recipient of the VIML is responsible for using the information it contains to build a platform-specific graphical user interface. Example recipients could be Java applets, native applications, palm pilots, cell phones, etc.

Currently AppletVIEW implements Java applet and Java application clients which are able to realize interfaces from VIML content.

## **Platform Independence**

The information stored in a VIML file is platform independent, and a given VIML file can be used to generate an interface on a wide variety of platforms, such as web browsers, hand-held computers, cell phones, etc.