Understand UCCX Finesse Architecture Deep Dive

Contents

Introduction

This document describes Finesse architecture in a thorough way so that the underlying processes make sense while troubleshooting finesse issues.

Prerequisites

Requirements

Cisco recommends knowledge of these tools and features:

JTAPI - Java Telephony API

API - Application Programming Interface

UCCX - Unified Contact Center Express

CUCM - Cisco Unified Communications Manager

CTI - Computer Telephony Integration

Components Used

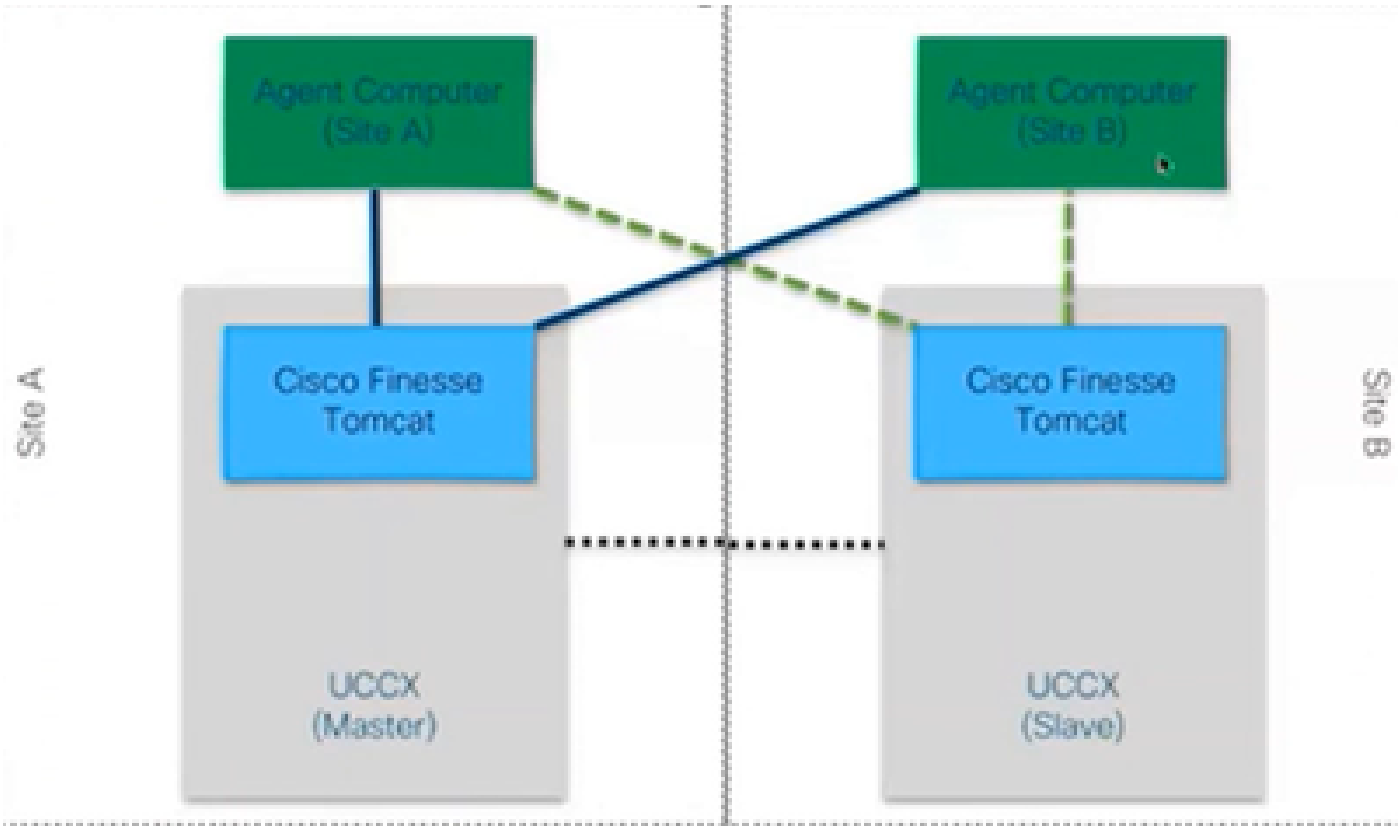
- Cisco Unified Contact Center Express (UCCX)

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

This document describes the Finesse architecture starting from high level overview and then the in depth signal flow along with examples and diagrams.

50,000 ft view



50000 view

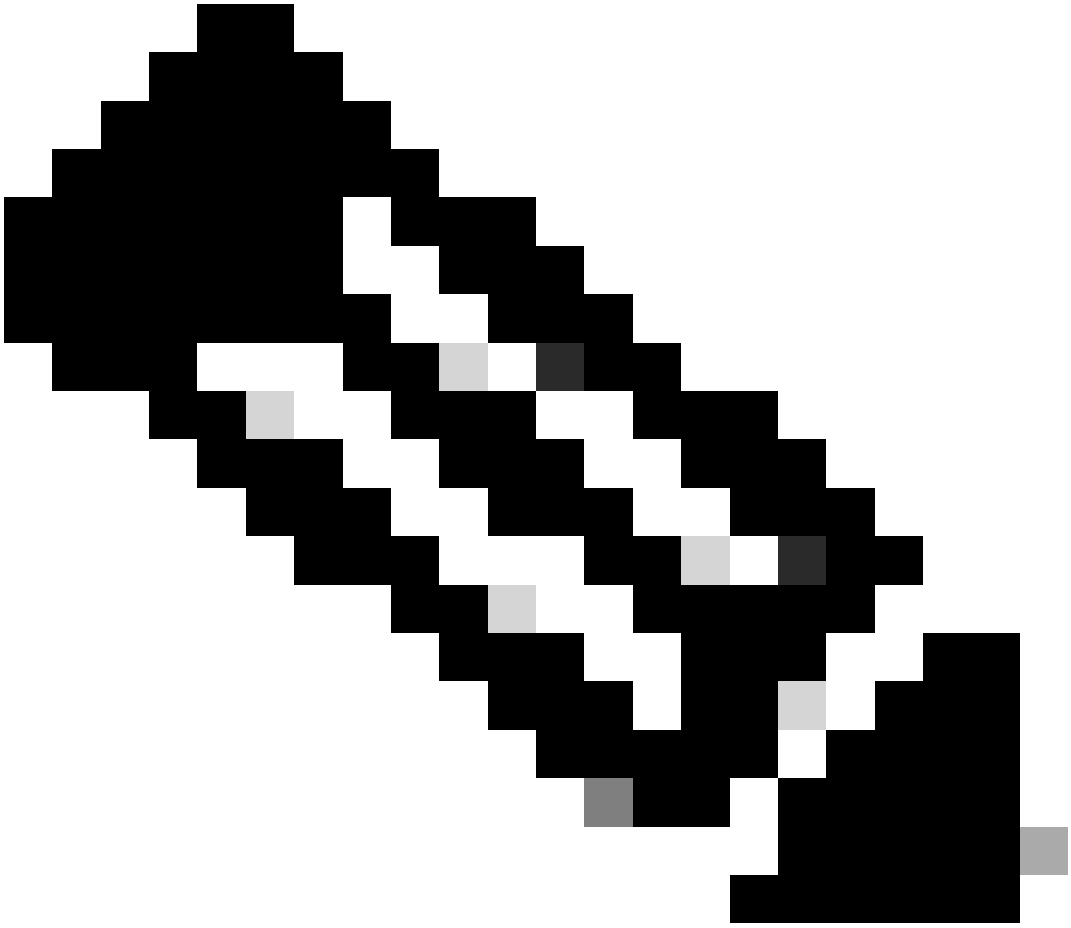
Finesse Tomcat

Finesse Tomcat is similar to Cisco Tomcat in CUCM as the functionality is same to load the web pages but for a different service called Finesse. Tomcat was designed for Finesse because it is a separate web application. You can only log into finesse using CCX master node as per versions prior 11.5. From 11.6 onwards, you can log into any node (but not recommended), only during failover. So, if you consider a WAN cluster, where both the agents (on site A and site B) are connected to Finesse on master node, then at all times, there is an inactive connection to the other node from the Cisco Finesse to the engine , which is a CTI openconf request which is required for failover.

Openconf request is sent regularly to check if the node is active or standby. Encase there is a failover, the notification service uses an API called the systeminfo API that tells the client that this node is down and a failover is needed. Then a file runs that redirects the browser to the other node and then the openconf rejavascriptsponse is sent. This failover works only if the certificates are accepted. (to establish a secure connection to the server).

3 certificates are presented which are:

- Notification service certificate for that node.
- Notification service certificate for remote node.
- Finesse service certificate for remote node.



Note: The remote node certificates need to be accepted in order for failover to work.

HTTP(S)

It is an application layer protocol for sending hypermedia documents such as HTML. It is designed for communication between web browsers and web servers.It is a stateless protocol which means that the server does not keep any data between the two requests. This is a client server protocol. For UCCX, the Finesse client runs on the agent browser(PC). It makes a request to the server using HTTP. Here are some common HTTP methods:

GET – to get information from a server.
POST – to send information to a server.
PUT – to replace anything on a server.
DELETE – to remove information from a server.

Finesse uses systeminfo api requests inside the http request. For instance, if you wish to change the state of an agent, browser sends a PUT instead of POST because if POST is sent, then server gets confused as it has 2 states in hand, which one to select? So using PUT, it replaces the current state.

XMPP

eXtensible Messaging and Presence Protocol

It is a set of protocols used for instant messaging and presence. All XMPP entities are identified using their Jabber IDS or JIDs. One of the extensions of this XMPP protocol which is used for gadgets is known as PUBSUB.

PUBSUB

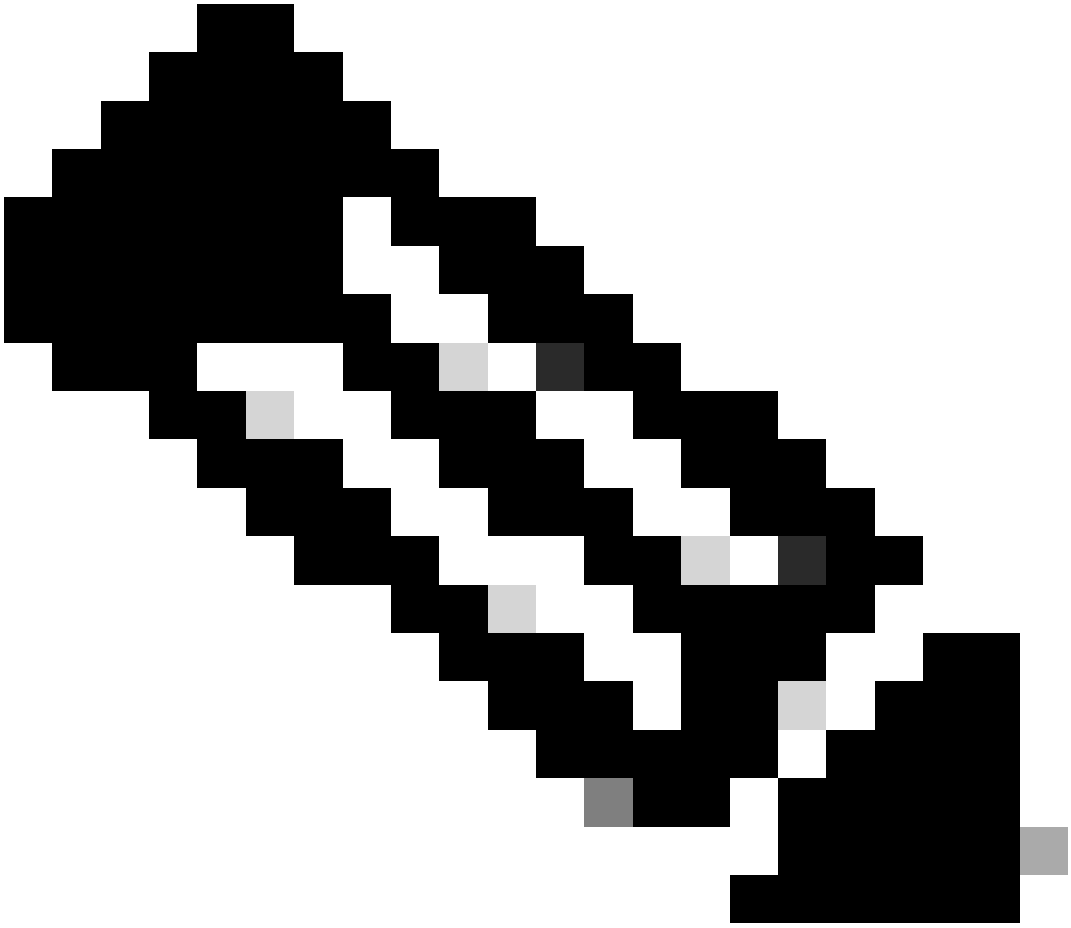
It is not the publisher subscriber you can think of. It still publishes and subscribes but it has nothing to do with the databases. XMPP uses a mechanism called nodes. Node is basically monitoring that entity which you care about. Anything which is important to you and wish to monitor it, you add a node to it. Then what PUBSUB does is, you subscribe for the updates or notifications on that node. You can have nodes for each type of entity like agent, dialog and so on. If you create a node for an agent, you get subscribed to the node on that agent and then whatever agent does, you get notified about it.

The purpose of this specification is to allow the XMPP server (Notification service) to get information published to XMPP nodes (topics) and then to send XMPP events to entities subscribed to that node.

In case of Finesse, the Computer Telephony Integration (CTI) server sends CTI messages to the Finesse web service to tell Finesse about configuration updates such as, but not limited to, agent or Contact Service Queue (CSQ) creation or information about a call. This information is then converted into an XMPP message that the Finesse web service publishes to the Finesse Notification service.The Finesse Notification service then sends XMPP over BOSH messages to agents that are subscribed to certain XMPP nodes.

BOSH – Bi-Directional Streams Over Synchronous HTTP

BOSH is a long lived HTTP connection where the server holds the request for a longer time till it has a response to it, otherwise sends an empty response. This works for XMPP clients and XMPP servers, but can be used for non XMPP applications as well.



Note: XMPP is stateful whereas HTTP is stateless (it does not store the information about the last request)

If a web application needs to work with XMPP, multiple issues arise.

Problem 1: Browsers do not support XMPP over Transmission Control Protocol (TCP) natively.

Solution 1: Web servers and browsers communicate via HyperText Transfer Protocol (HTTP) messages, so Finesse and other web applications wrap XMPP messages inside of HTTP messages.

Problem 2: HTTP is a stateless protocol.

Solution 2: You can use cookies/post data for this.

Problem 3: Third problem is the unidirectional behavior of HTTP which means only the client sends requests, and the server can only respond. The server inability to push data makes it unnatural to implement XMPP over HTTP.

Solution 3: To overcome this issue, you need to have a bridge between HTTP and XMPP.

The proposed solutions are:

1. Polling (legacy protocol): repeated HTTP requests asking for new data defined : HTTP Polling
2. Long polling also is known as BOSH: transport protocol that emulates the semantics of a long-lived, bidirectional TCP connection between two entities by efficiently using multiple synchronous HTTP request/response pairs without requiring the use of frequent polling. The reason to use BOSH is to cover up the fact that the server does not have to respond as soon there is a request. The response is delayed up to a specified time until the server has data for the client, and then it is sent as a response. As soon as the client gets the response, the client makes a new request and so forth.

The Finesse desktop client (web application) establishes a stale BOSH connection over TCP port 7443 every 30 seconds. After 30 seconds, if there are no updates from the Finesse Notification Service, the Notification service sends an HTTP reply with a 200 OK and a (nearly) empty response body. If the Notification Service has an update on the presence of an agent or a dialog (call) event, for example, the data is sent immediately to the Finesse web client.

To summarize:

The Finesse web client has a stale HTTP connection (http-bind) set up to the Finesse server via TCP port 7443. This is known as a BOSH long poll. The Finesse Notification Service is a presence service that posts updates regarding the state of an agent, call, and so on. If the Notification service has an update, it replies to the http-bind request with the state update as an XMPP message in the HTTP response body. If there are no state updates 30 seconds after receiving the http-bind request, the Notification Service replies without any state updates to allow the Finesse web client to send another http-bind request. This serves as a way for the Notification service to know that the Finesse web client is still able to connect to the Notification Service and that the agent did not close their browser or put their computer to sleep, and so on.

CTI

You can use Computer Telephony Integration (CTI) to take advantage of computer-processing functions while making, receiving, and managing telephone calls. CTI applications allow you to perform such tasks as retrieving user information from a database using a caller ID, or to work with the information gathered by an Interactive Voice Response (IVR) system to route a call coming from user along with their information, to the appropriate service representative. CTI Manager on CUCM responds to the JTAPI requests from UCCX. The CTI server TCP port is 12018. This is how Finesse Server and Engine (CTI Server) talk to each other.

Here is some of the Information exchanged via CTI :

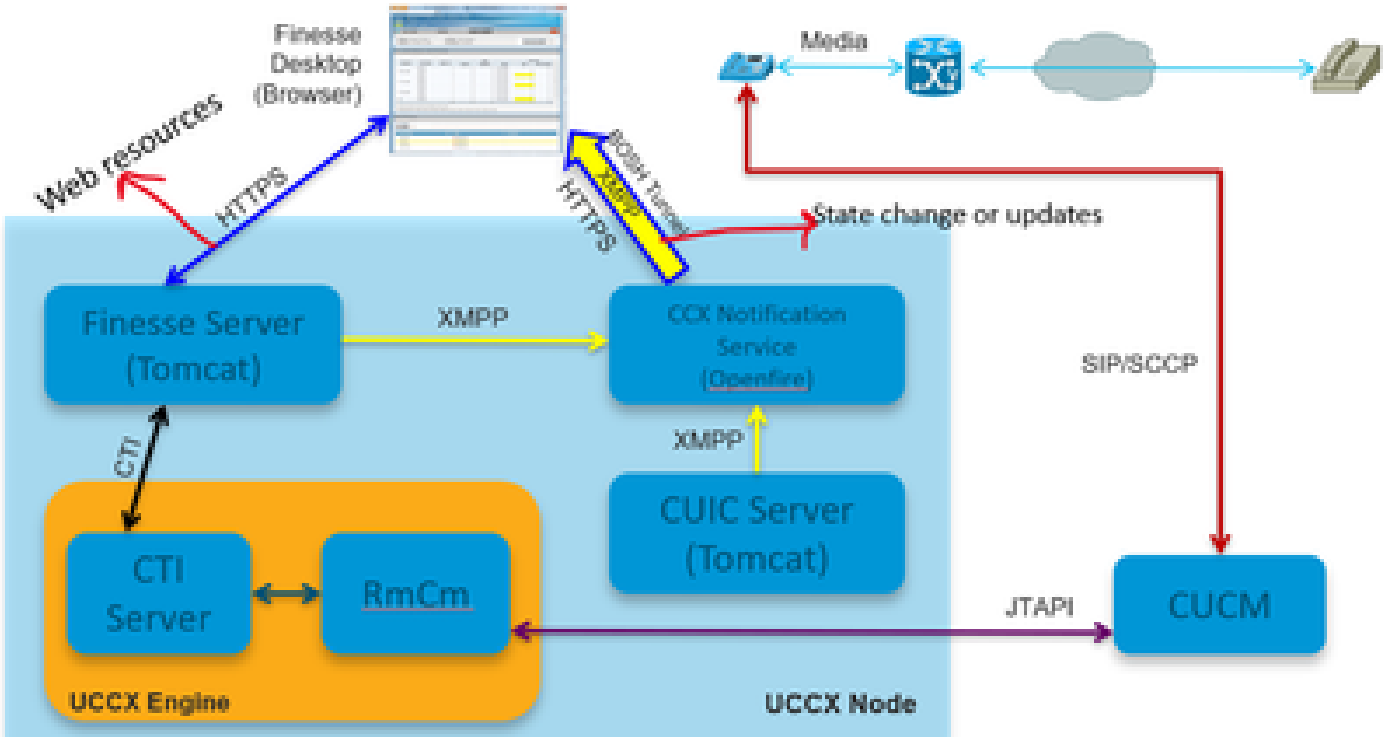
- Current system configuration and future updates.
- Agents and their states.
- Calls and their states.
- Stats for agents, calls and queues on real time.

JTAPI

Cisco Unified JTAPI serves as a programming interface standard developed by Sun Microsystems for use with Java-based, computer–telephony applications. Cisco JTAPI implements the Sun JTAPI 1.2 specification with additional Cisco extensions. Any communication between UCCX and CUCM resides on JTAPI. This is how CUCM and Engine(Telephony subsystem) talk to each other. JTAPI is used to control and monitor CUCM phones, route calls using CTI ports and Route points, start and stop recordings on CUCM and for any call routing functionality

30000 ft view

The next diagram describes how the UCCX Engine, Finesse, CUCM and Browser communicate with each other.

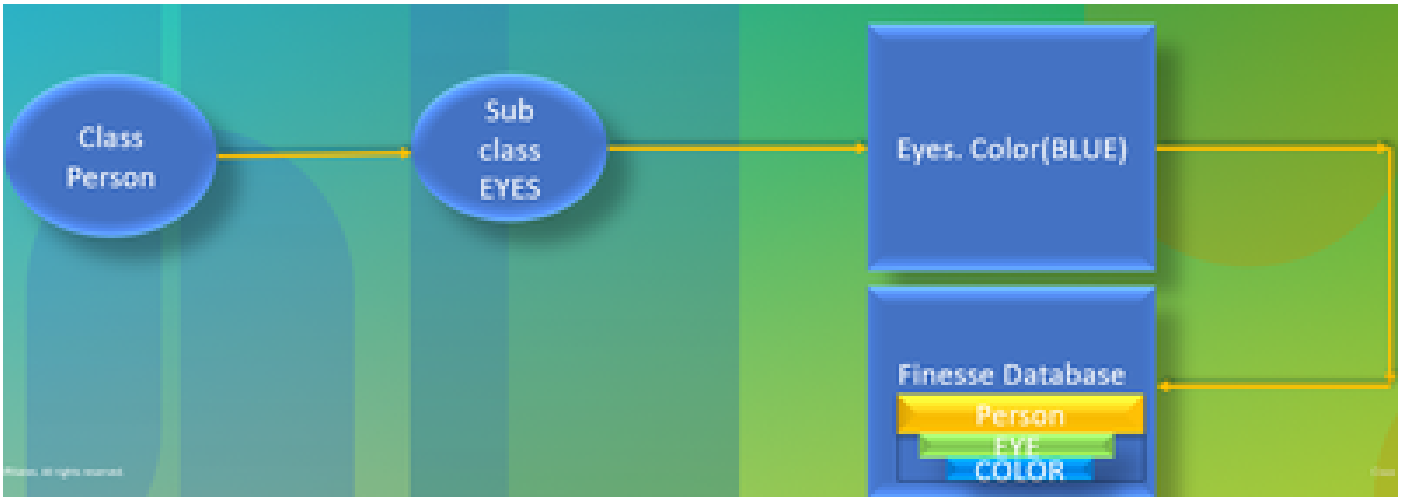


30000 view

Let us consider that the call is established with the agent. Now, RmCm who is monitoring the agent extension through JTAPI, tells the CTI server about the state change that the agent is talking. This information is sent from the CTI Server (inside CCX Engine) to the Finesse Server(Tomcat) using CTI. Finesse server sends this information to the CCX notification service using XMPP about the state change. Notification service (Openfire) opens a BOSH tunnel to the agent browser and updates the information about the state change and that is how you see the agent going to RESERVED state. Any type of web resources are requested to the finesse server using HTTPS like WAR files, gadgets and so on (if not in cache already).

HIBERNATE

The next diagram explains about Hibernate service.



hibernate

HIBERNATE is referred to as High-Performance Object/Relational Persistence and Query Service. Simply put, it maps JAVA Classes to Database tables. For instance, you have a JAVA object called Team and you have a database table in the finesse database called Team. JAVA class controls what information is inside the table and HIBERNATE is what makes that happen. Instead of using SQL queries, it uses java classes to update the information.

AXL

Administrative XML.

XML stands for eXtensible Markup Language and is a markup language that defines some relatively simple rules for encoding data. It was primarily designed to transmit and receive structured data in a well-defined format both systems can understand. In the most basic form, XML defines tags which are enclosed in angle brackets (<>) and these tags surround the data described by the tag. Tags can form a hierarchy with tags inside of other tags. For example, to define a basic phone device, you can say that a phone device needs three parameters, a name, a description, and a phone number.

It is a SOAP Based API that enables the remote provisioning on CUCM. It is used to add, update, remove or retrieve information from CUCM database. Retrieval capabilities include checking user authentication and running SQL queries.The AXL API provides you with access to the entire CUCM database. The AXL API is purely for provisioning and does not provide access to run-time or performance data.

AXL API utilizes HTTPS Basic Authentication. Any CUCM User (Application or EndUser) has read/write access via AXL if they are members of the **Standard CCM Super Users** access control group or any group with the **Standard AXL API Access** role assigned to it. This means that all super user accounts implicitly already have access to the AXL API. To create an account dedicated for AXL API use, you must first create an access control group and assign the **Standard AXL API Access** role to it, then associate the application user with the newly created group. To provide read-only AXL API access you can create a separate Access Control Group and assign only the **Standard AXL Read Only API Access** role to it.

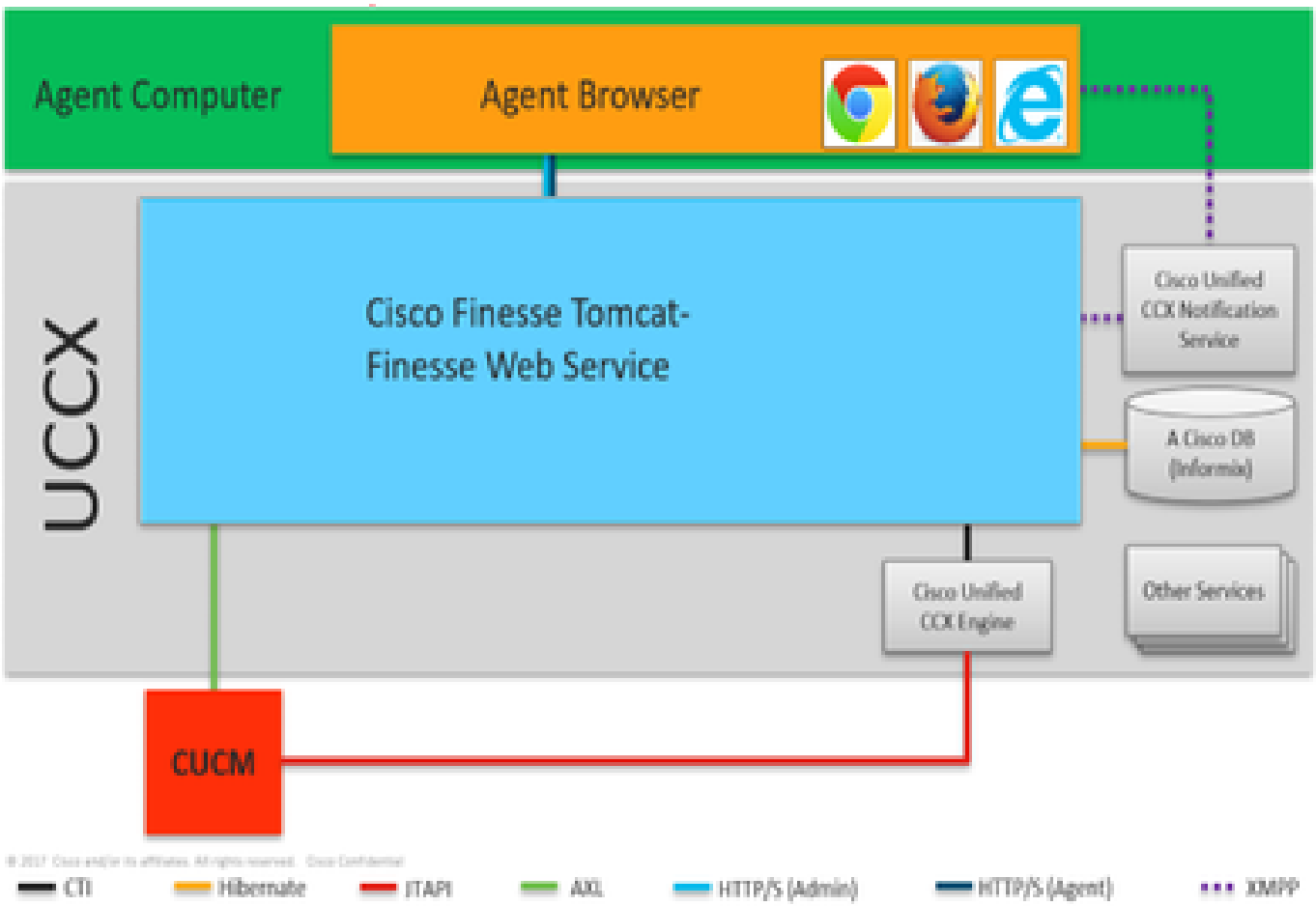
SOAP

The Simple Object Access Protocol (SOAP) is a way to pass information between applications in an XML format. SOAP messages are transmitted from the sending application to the receiving application, typically over an HTTP session. The actual SOAP message is made up of the Envelope element, which contains a Body element and an optional Header element.

- Envelope - This mandatory element is the root of the SOAP message, identifying the transmitted XML as being a SOAP packet. An envelope contains a body section and an optional header section.
- Header - This optional element provides an extension mechanism indicating processing information for the message. For example, if the operation using the message requires security credentials, those credentials must be part of the envelope header.
- Body - This element contains the message payload, the raw data being transmitted between the sending and receiving applications. The body itself can consist of multiple child elements, with an XML schema typically defining the structure of this data.

20000 ft view

The next diagram explains in a more detail way about the protocols involved in Finesse architecture.

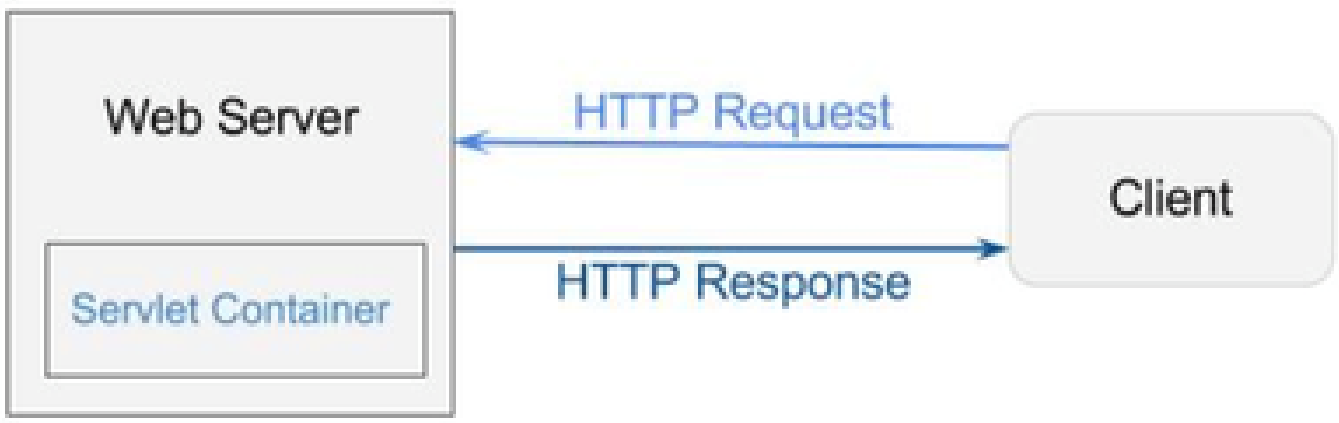


20000 view

These are the protocols responsible for the communication between different UCCX components.

- UCCX Engine and Finesse Server talk over CTI protocol.
- UCCX Engine and CUCM talk over JTAPI protocol.
- Finesse Tomcat and CUCM talk over AXL.
- Finesse Notification service and Agent browser talk on XMPP/BOSH.
- Finesse Tomcat and the database talk over Hibernate.
- Finesse Tomcat and Finesse Notification talk over XMPP.

APACHE SHINDIG



Shindig

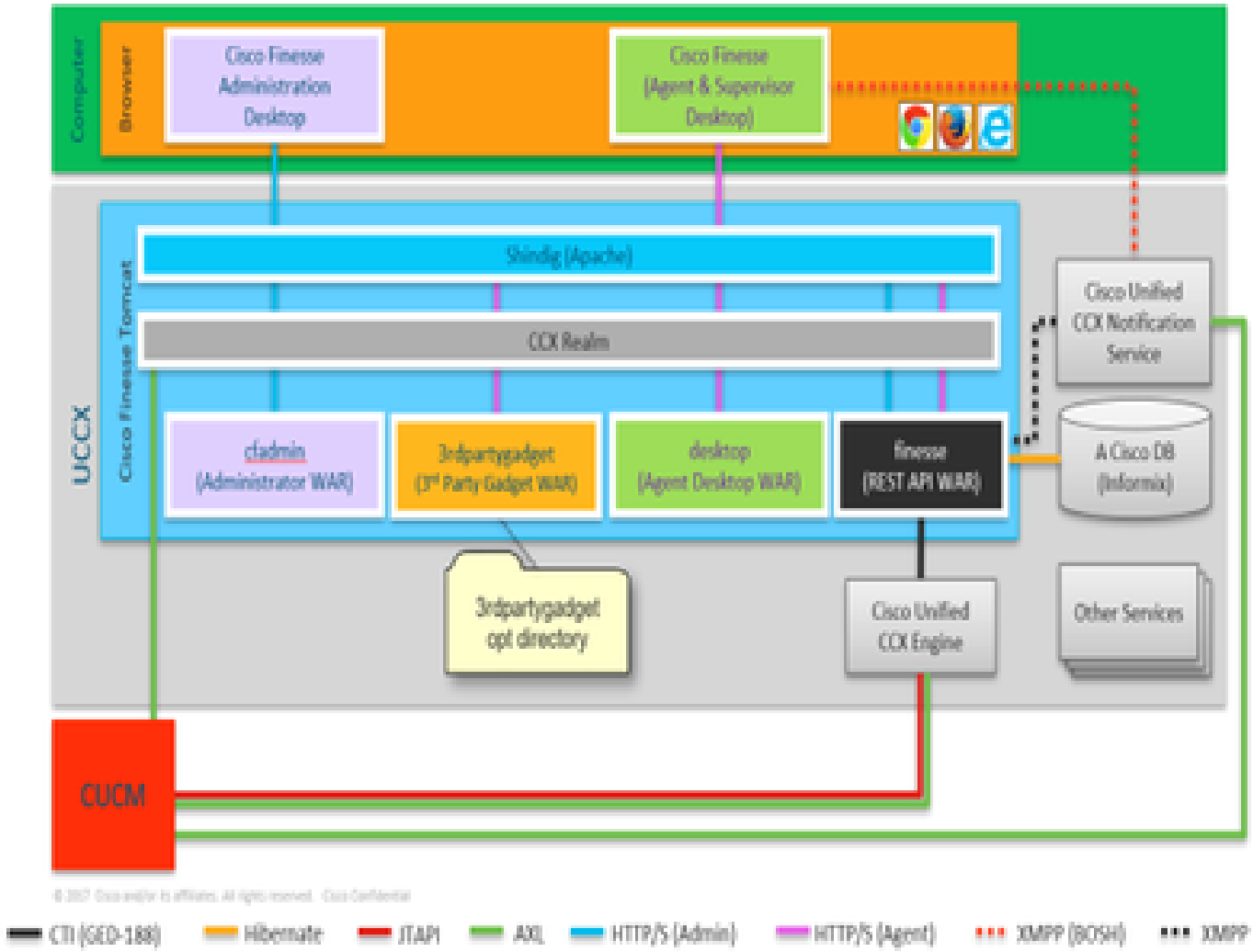
Apache Shindig is an OpenSocial container and helps you to start hosting OpenSocial apps quickly by providing the code to render gadgets, proxy requests, and handle REST and RPC requests. OpenSocial is a set of APIs for building social applications that run on the web. (Web/Servlet) Container is used by a web server to dynamically generate web pages.

WAR FILES

WAR stands for Web Archive. It contains files of a web project. It can have servlet, XML, JSP, image, HTML, CSS, JS, and so on. Catalina logs contain the information about WARs getting deployed.

10000 ft view

The next diagram explains in detail about the how the authentication flow works within the components of UCCX and Finesse.



10000 view

WAR files are required to display and create the page depending on how you log in. Browser asks Shindig that it needs to render a gadget, shindig then talks to CUIC to render the gadget. CCX Realm is used for authentication with CUCM using AXL. Notification service also authenticates with CUCM using AXL.

Finesse Rest API WAR is the main repository that actually communicates with notification service, CCX Engine and DB. Shindig talks only with Finesse Rest API (WebServices) because cfadmin and desktop WARs are just to display the page. Anything that comes to the Finesse Rest API WAR, you can see that in the Finesse WebServices logs which are the most important logs for finesse. You talk HTTP between

Shindig and Finesse web service (Rest API WAR). Finesse web service (Rest API WAR) and Engine talk to each other via CTI.

AJAX - The beauty of Finesse

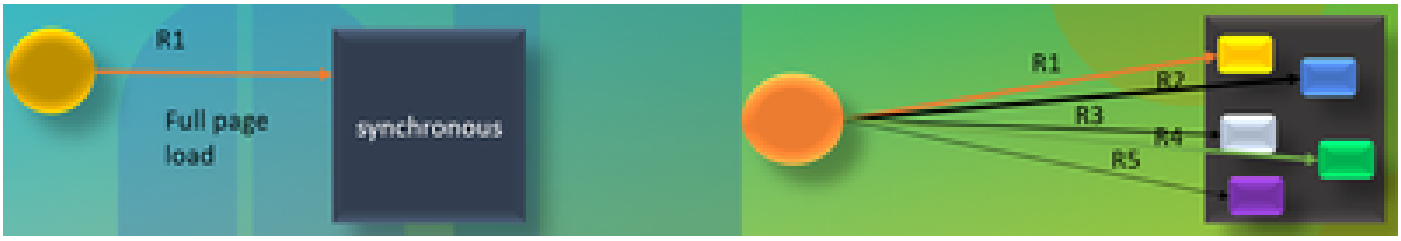
AJAX stands for Asynchronous Javascript and XML. It is not a programming language, but a method to access web servers from a web page. AJAX is a mechanism for making partial page updates. It enables you to update sections of a page with data that comes from a server without needing to refresh the whole page.

For instance, if you talk about Facebook messenger, when a new message comes in, you do not need to refresh the whole page to get the message, instead the message section of the page itself refreshes and gets the new messages in real time without needing to refresh the whole page.

Every browser has a built in object called XMLHTTPREQUEST (also called **XHR**). Every request to AJAX in the server goes through this XML request. This contains the specifics of what you need to update.

Advantages of using AJAX

The next diagram explains the difference between asynchronous and synchronous requests.



AJAX

In case of a synchronous request, you have to wait for the first request to get processed and then you can send second request. For instance, page refresh is required and you cannot do anything until the page is refreshed. On the other hand, in case of an asynchronous request, you do not have to wait for the first request to get completed to send the second request. You can send multiple requests simultaneously. For instance, weather app gadgets on websites. You can refresh the weather section of the page and meanwhile also work on the other sections of the website simultaneously without needing to refresh the whole page. This is the main advantage of Asynchronous request.

WORKING OF AJAX

AJAX is a combination of an XMLHTTPREQUEST (**XHR**) which is used to send and receive updates from webserver along with Javascript and HTML which are used to display or use the data.

SENDING REQUEST WITH AJAX TO SERVER

This is a 3 step process which is mentioned next:

1. Creating a variable and storing the **XHR** object in to it.

Var request = new XMLHttpRequest();

2. Accessing the request variable which has the payload inside the XHR object.

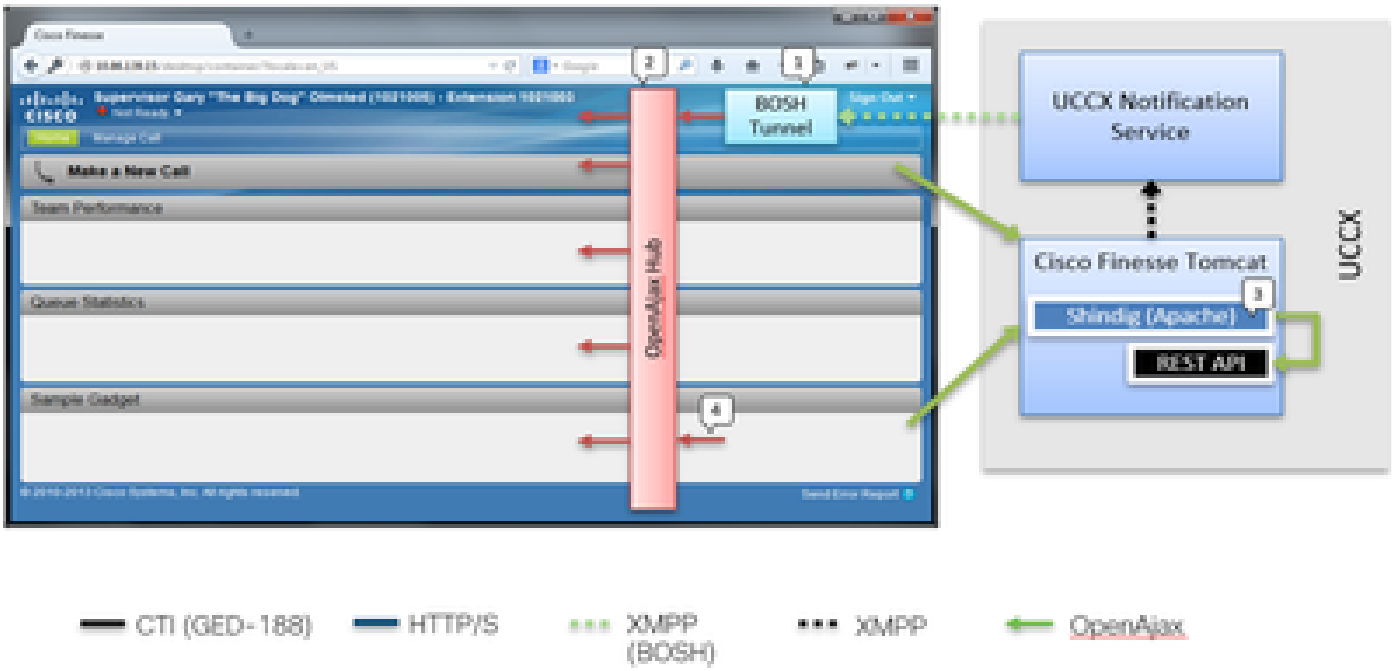
request.**open(GET, URL) ;**

3. Sending the request

Request.send() ;

Desktop architecture

The next diagram explains the flow of AJAX signals when gadget renders on the webpage.

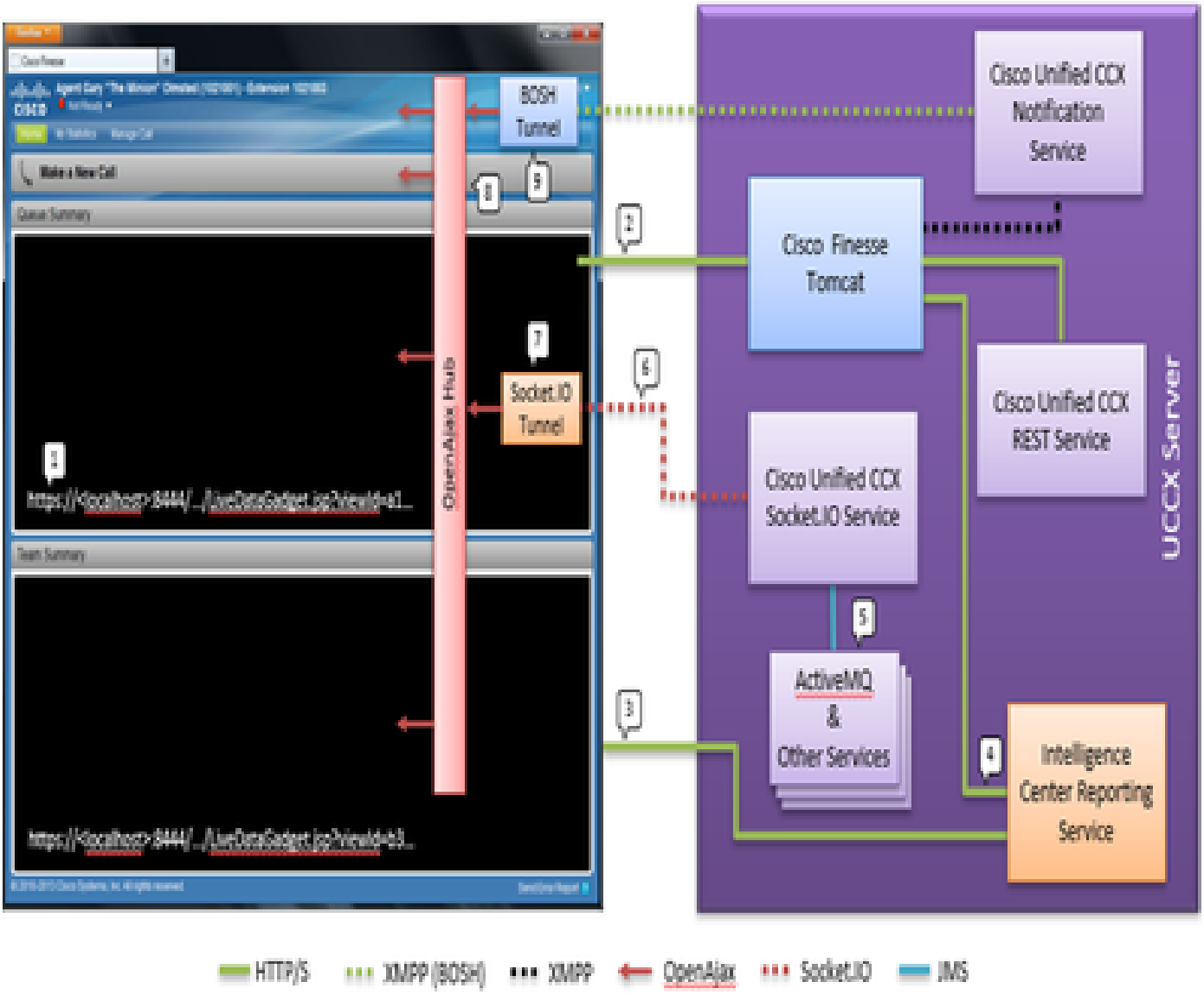


desktop architecture

IFrame resides int he container to host the BOSH Tunnel. OPENAJAX hub is provided to publish messages across the gadgets (using pubsub method) REST requests are proxied through Shindig to other servers as well. Gadgets can publish their own messages on AJAX hub.

Gadget Architecture

The next diagram explains the Finesse Gadget architecture in detail.



gadget architecture

Unlike typical Gadgets, CUIC Gadgets also receive a real-time XMPP feed from OpenFire as well. In the case of UCCX, where CUIC and Finesse are co-resident with UCCX, there a shared OpenFire instance. Most of the Gadget content and all the REST API's are proxied through Shindig in the Finesse Server. This goes for Finesse Gadgets and REST API as well as CUIC Gadget instances and REST API. CUIC Gadgets use a D-Grid for rendering their reports. There is a bootstrapping process that must occur and this is done in conjunction with CUIC directly. For this reason the CUIC Gadgets initially talk to CUIC Server directly during the loading process. For this reason, the CUIC Certificate must be accepted into the user browser (in addition to the Socket.IO Tunnel). Gadget contents and REST API's are proxied to the client between Finesse and CUIC. Rest API calls are made to both the Intelligence Center Reporting Service as well as the CCX Web Service. The CCX Live Data Socket.IO Service gets the messages from Live Data via JMS from ActiveMQ. The CCX Live Data Socket.IO Service publishes Real-Time Reporting JSON over the Socket.IO connection from the client. Similar to the way the Finesse Desktop has a BOSH Tunnel iFrame that maintains the BOSH connection with the Cisco Finesse Notification Service, the master Live Data Gadget has a Socket.IO Tunnel iFrame that maintains the Socket.IO (websocket) connection with the CCX Live Data Socket.IO Service.

The OpenAjax Hub distributes all the events to the subscribed listeners. This would be both Gadgets as well as parts of the Finesse Container itself. The Finesse Desktop has a BOSH Tunnel iFrame that maintains the BOSH connection with the Cisco Unified CCX Notification Service. This publishes events onto the OpenAjax Hub.

Reference Links

- Finesse Web Services Developer Guide