**IoT Resource Tree Representation(RTR) for Massively Large Data Sets Project Proposal**

**1. Vision and Goals of the project:**

IoT(Internet of things) is a promising new technology. It may link all the embedded computing devices using traditional internet infrastructure. The IoT RTR UI is a user-friendly platform to collect, manage and visualize mass quantities of data. Goals of IoT RTR:

* A webapp to view 1 million+ data points collected from the IoT. Visual aids must give the user a sense for extracting meaning from these mass quantities of data in an intuitive way.
* Optimize the client-side response time in the web browser to provide a fluid user experience.
* Implement management functions for adding or deleting nodes from the tree.

User/Personas Of the Project

The IoT is still in its infancy, and the industry is still trying to figure out how to best package and present this mass quantity of data. Our project, while utilizing industry standards for IoT systems, seeks to become a prototype that will be used as a model for further integration as the IoT evolves.

This tool is designed to be a webapp hosted on a website. The users will be companies and individuals with large sets of data who would like to extract meaning through visual aids as well as manage their data.

Team Roles

Lionel (Product Owner)

Chen (Scrum Master) = Database Optimization

Yingchao + Nik (Development Team) = Front-End Development

Qingqing (Development Team) = Documentation & All-Purpose Dev

**2. Scope and Feature of the Project:**

RTR-UI

* Present a simple and compelling front-end interface for end-users
* Client-Side Render - Use the standard D3(JQuery) library to render the data clientside in the web browser
  + Multiple Views - Different visualizations will be used depending on the amount of devices and data points in the database. As the user narrows down the area of focus on the data, new visualizations will be used to better inform the user of the data’s significance. (described more later)
    - Cluster Overview
    - Grouped Bar Chart
    - Zoomable Sunburst Model
    - Collapsible Tree
* Advanced Query – Have an advanced multi-parameter query engine that adjusts parameters based off information available in the data
* Scalability – Must scale to potentially 1 million+ devices/datapoints
  + Must optimize to keep client-side response time low
* Data Management – User should be able to modify the database
  + Add or Delete Devices
  + Add or Delete Data points
* Server-Side Processing – Processing of data must be done in the cloud rather than in the client in order to keep the response time low.
  + Data will need to be cached to not waste resources downloading the same data.
  + Sort the data and make it qualified for the D3(JQuery) API.

§ Extract only relevant information from the database to build a second more optimized database.

§ Use the ResouceType to extract data type, so that we can categorize and block data together in the UI by the type of data they represent.

§ If available attribute, allow filter by geographic region.

§ Allow user to filter data by certain year -> month -> day -> hour -> minute or date range.

**3. Solution Concept**

Global Architectural Structure Of the Project:

This project has been previously implemented with a single tree, the treewalker, displaying 1M+ nodes. However the treewalker shows great difficulty in rendering the data at scale. Additionally, the data is rather meaningless as one might have to scroll through 1M leaves in a web browser to find a specific point or device.

We envision a system based on multi-parameter queries as its focus point. The graphic models for navigating the data through a tree-like structure should all come secondary, as users are more likely to have an explicit area of focus when searching for data. However, while some parameters we are guaranteed to have such as time time\_created or resourceType, others suchs as OntologyRef and labels may or may not be included. Our query engine needs to be flexible enough to realize whether OntologyRef or label data is available and make those options visible to the user.

Our Models for Visualizing Big Data

Our concept is to implement several UI data models in different layers. Different size data sets are better accustomed to being displayed in different ways. Something that represents 50,000 data points very well may not be suitable for representing 10,000,000 data points.

Layer 1

The first layer can be implemented with a grouped bar chart (<http://bl.ocks.org/mbostock/3887051>)**.** Each group can represent all the data referring to a specific resourceType.

The 1M+ nodes thus can be scaled down to a certain degree depending on the container size of a certain range of data. For example, temperature can be used in one “group block” with each bar representing the quantity of nodes within a certain range of temperature. This gives a good view of the distribution of values for a certain date type, when dealing with mass amounts of data that are otherwise difficult to quantify as a whole.

However, this model does make some assumptions on the distribution of data that we need to talk to the customer about. If for example, we have 25 or less types of data then each data type can be allocated to one group. Then it doesn’t matter how many data points we have in those groups as they are covered by the bar graph for the corresponding data types group. If however there are 1,000+ types of data then there would need to be 1000 groups which would make this visualization very impractical as you would need to sidescroll to find the particular data type.

Layer 2

In addition (or as a replacement) for the above we can utilize a zoom-able sunburst model (Coffee Flavor Wheel, <http://www.jasondavies.com/coffee-wheel/> ) or a sequence sunburst model (<http://bl.ocks.org/kerryrodden/raw/7090426/>) as the second UI layer. This kind of structure can reduce the data size quickly and provide a clear view of the nodes. Several partition methods through meta data (ie. location, date/time, or data\_type) are needed for this model. Which can be problematic as we are not guaranteed these parameters in the meta data, and it will depend in large part on the particular devices and the kind of data they produce.

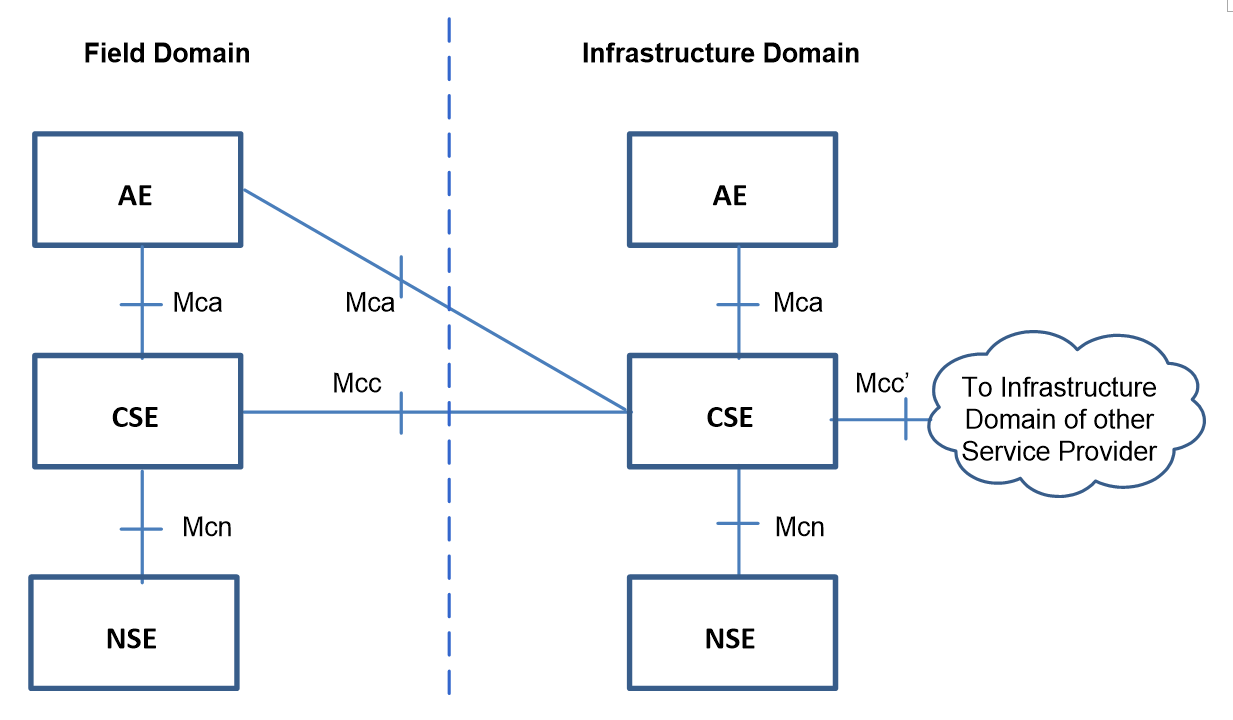
The way this UI works, is that all the data is categorized under labels (hence why we need the meta data). Each layer of the sunburst dial specifies different parameters. As you click on a parameter, that “slice” of the sunburst dial expands to cover the whole UI which gives a “tunneling” effect (see link above to play around with it if this is confusing). It allows you to quickly refine the dataset you’re working with to exactly what the user wants.

For example, we can start with year, and each layer farther outward apply month -> day -> AM/PM -> hour -> minute ect. With the location, we can group the data in layers from continent -> country -> state -> city. For data\_type we can have alphabetical ranges in each layer to narrow down the number of data types. For example to find temperature select S-Z in the first layer, and then if # of data\_types is still too large select letter T. Then select temperature from the data\_types starting with letter T.

Layer 3

When the nodes are reduced to a quantity of several hundred, a normal expandable tree structure can then be implemented for checking/modifying individual node information as the third layer of our model.

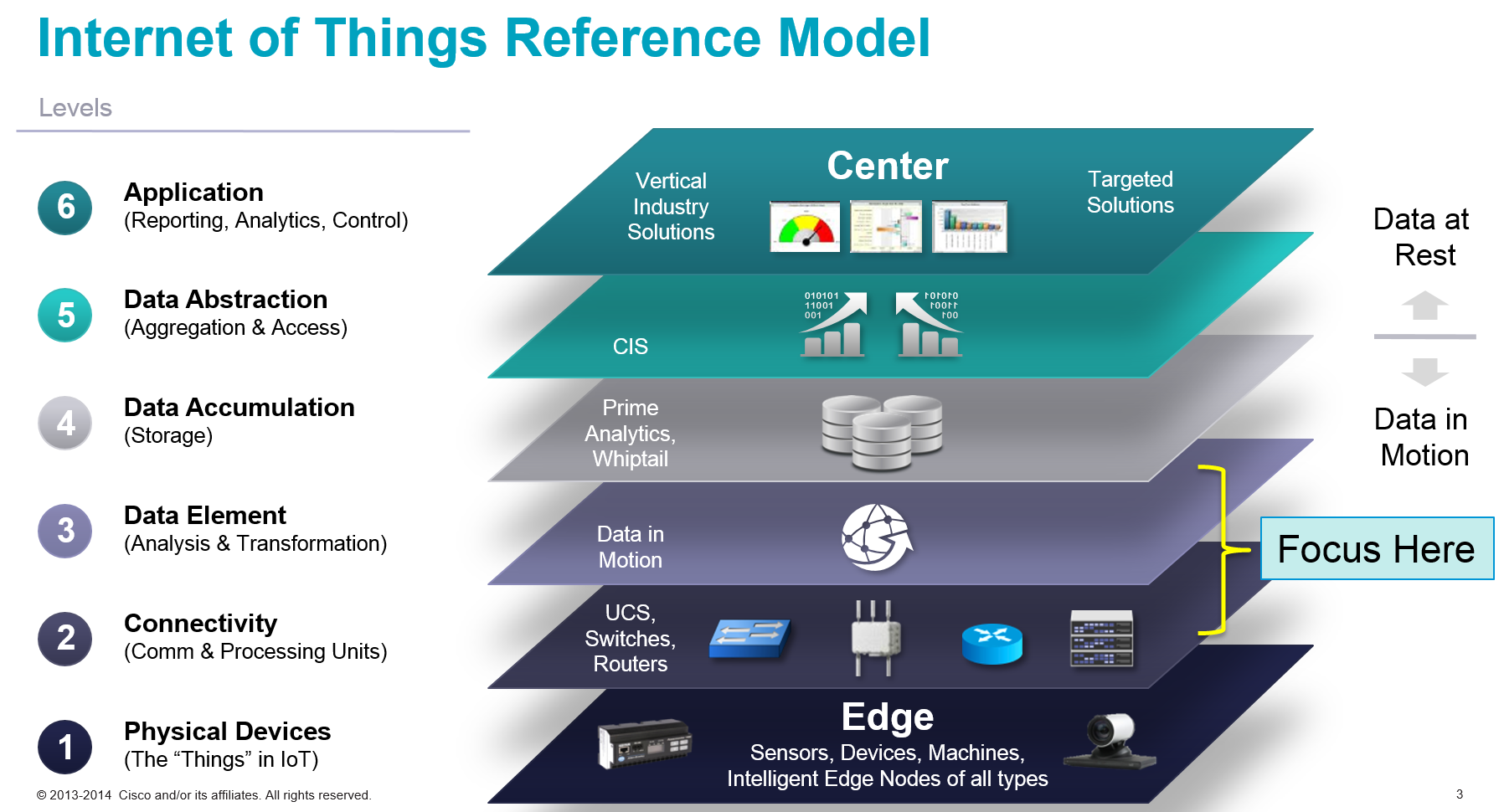
We are going to use the M2M structure in the project. Here is a top level introduction of M2M:( the structure introduction comes from TS-0001-V1.5.0--2015-January-23--OneM2M Technical Specification)



1）Application Entity (AE): Application Entity is an entity in the application layer that implements an M2M application service logic. Each application service logic can be resident in a number of M2M nodes and/or more than once on a single M2M node.

2）Common Services Entity (CSE): A Common Services Entity represents an instantiation of a set of "common service functions" of the M2M environments. Such service functions are exposed to other entities through the Mca and Mcc reference points.

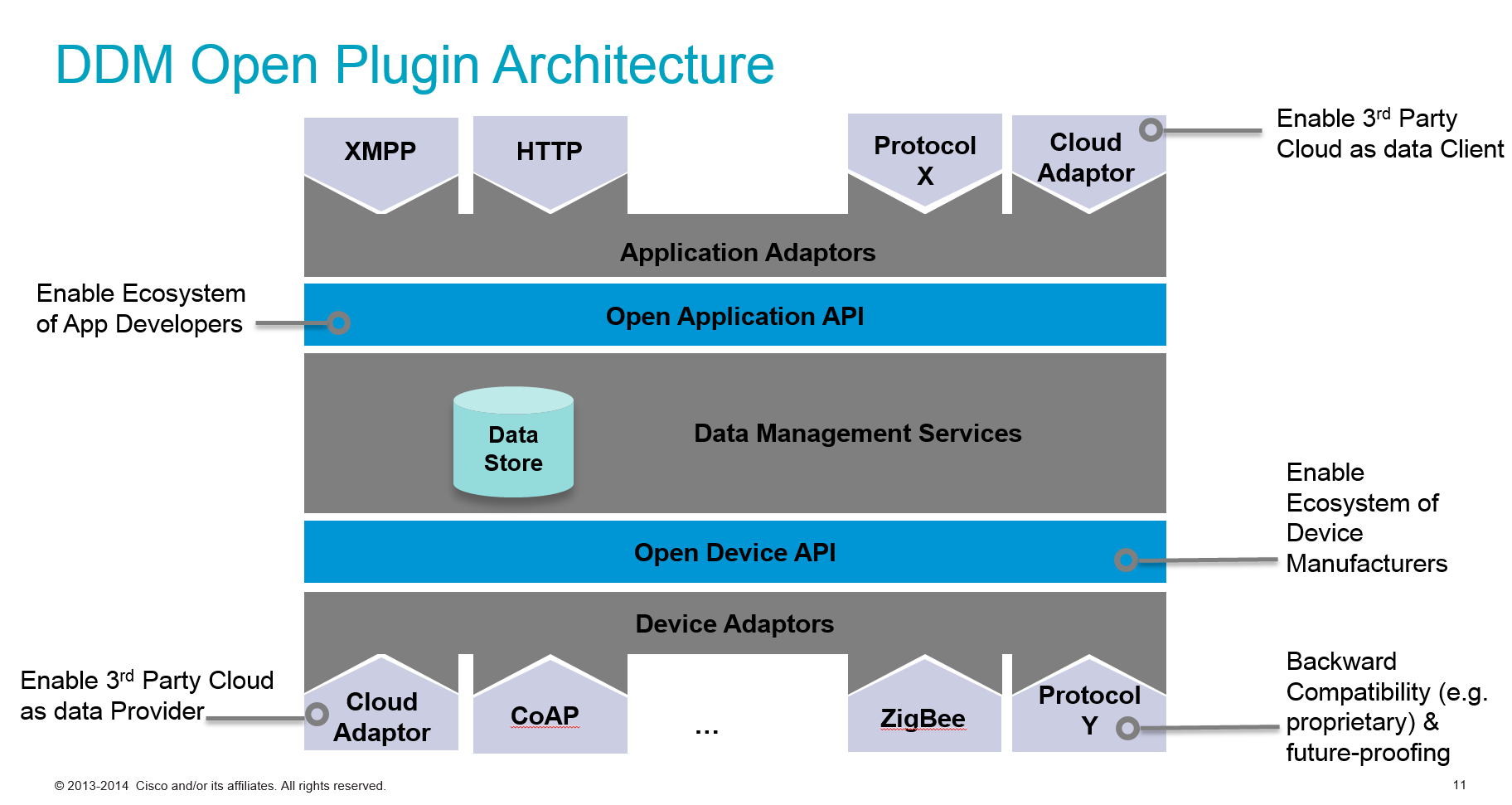
3）Underlying Network Services Entity (NSE): A Network Services Entity provides services from the underlying network to the CSEs.



In this project, we are not going to focus on a device-level API, instead we will develop a database-level API with the assumption that all the data comes from a database.

The core part of the project lies in the optimization of the visualization. We want to have a satisfying response time while maintaining a Show-What-The-User-Wants manner.

The SDN we use will be based on OpenDayLight(ODL). Here, we are not going to develop through the ODL API but instead use the ODL SDN function (e.g. create nodes, configure the topology, attach the interface etc.).



We are going to mainly focus on the southbound protocol in this part while getting familiar with the northbound protocol. Here we will use HTTP, XMPP and other existing application level protocols and we will try to integrate them into the same interface.

Cisco has documentation of both the concept and implementation of the northbound protocols we will be utilizing.

The data storage strategy is another critical topic we are going to focus on. In this topic, we will develop a fast and reasonable way to fetch and update data(e.g. we may use the node cache to lower response time)

The key challenge here is a good understanding of various protocols and how to apply them to the system. To begin with, we may use the basic HTTP protocol.

The tool we are going to use:

D3(JQuery): Using this library to render the tree structure of the layer and data retrieved from the database

MongoDB: The database used to store the IoT data

Model–View–Controller(MVC): A software design pattern that we are going to implement into the UI design.

**4. Acceptance Criteria**

Minimum acceptance criteria for this project are the visualization of 1M+ nodes as well as an advanced query system. The secondary goals are optimizing the client-side response time in the web browser and implementing management functions for adding or deleting nodes from the tree.

**5. Release Planning**

release #1 (Due by Week 6)

1. Show proof-of-concept resource Tree management by adding/deleting devices and/or data points by sending CRUD operations through Postman plugin in Chrome.
2. Have a D3 visualization up and running, displaying \*some\* data (not necessarily the resource tree). And be able to justify why these are the best visualization techniques to use when it comes to working with huge data set.
3. Have a skeleton database set up in MongoDB.

release #2

release #3

release #4