GIOTTO

Buildings form an essential part of our lives, and provide many services that make our indoor environment comfortable, productive and enjoyable. With advances in communication and computing, buildings are increasingly embedded with sensors and networked devices that enable a wide variety of services - contextual services to occupants, monitoring of building infrastructure, and energy efficient operation.

We have created Giotto as a successful IoT ecology. This includes a core building management middleware, an easy to use protocol agnostic API, robust security and privacy protectins, intergration of machine learning at multiple levels, to engage users with varying levels of experience. Our vision is to provide building users with a holistic view of the building across various different subsystems, such as lighting, water, air conditioning, and enable applications that work seamlessly across the underlying infrastructure. Giotto uses BuildingDepot, a prior building management system, as its core middleware and adds a number of other services.

The following are the *primary objectives of Giotto:*

- Integration of data from a wide variety of building systems and communication protocols
- Provide an easy to use API that is protocol agnostic
- Provide a search interface to query building data and metadata
- Provide an access control system that facilitates security and privacy
- Provide support for actuation of building control systems
- Provide a middleware that scales with big data across multiple buildings
- Support for standardization to enable portable building applications

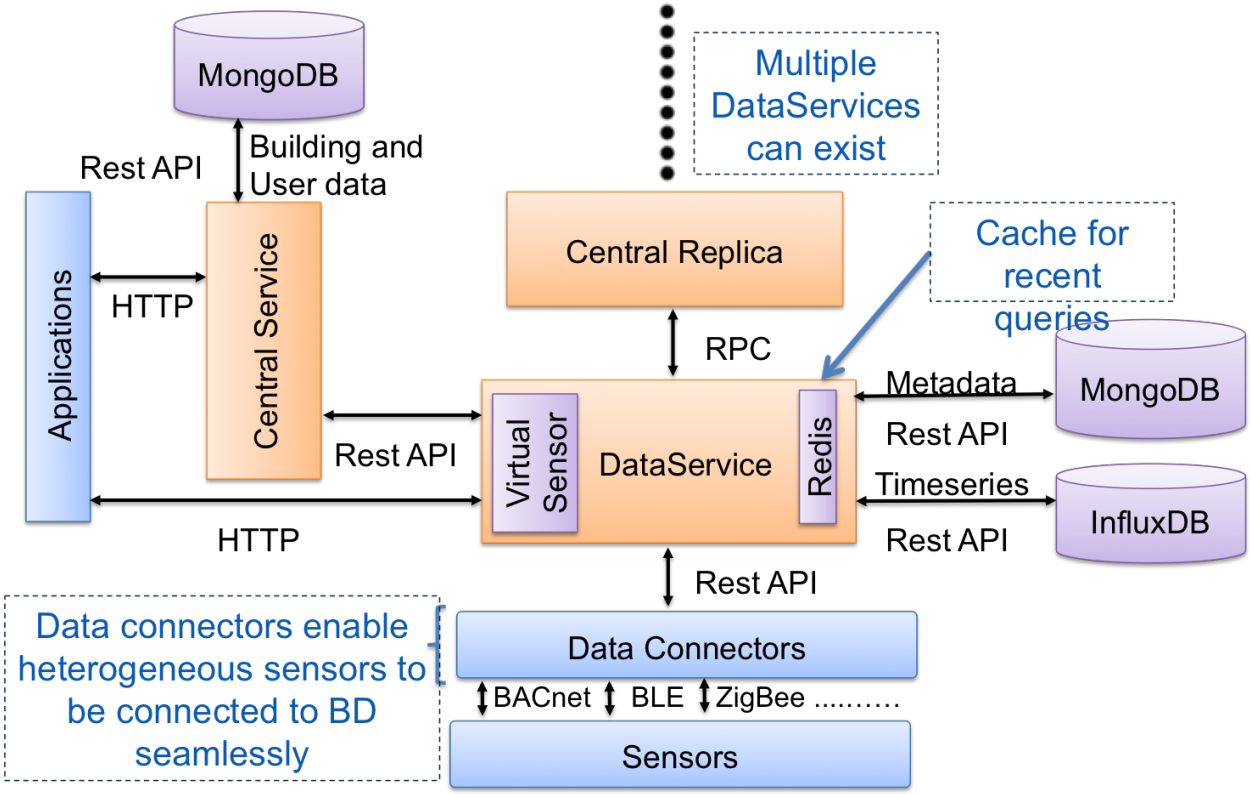
Design and Architecture

We have designed Giotto to incorporate a web service to support an easy to use RESTful API and leverage standard web service tools for scalability, modularity, maintainability and security.

Giotto contains two types of web services: Central Service and Data Service. The Central Service hosts the building metadata information while the Data Service hosts the timeseries data. As an example, measurements from a building power meter will be stored in Data Service, and associated metadata such as location of meter and unit of measurement is hosted in Central Service. The Central Service also hosts user information, access control lists and structural information such as rooms and floors.

A Giotto instance has only one Central Service that provides a uniform view of all the different systems across multiple buildings. The metadata in the Central Service provides the mapping between points and the corresponding Data Services where their timeseries is hosted. There

are no limits on the number of Data Services that can be associated with a Giotto instance. There could be one Data Service for each building, and even one Data Service per subsystem in a single building. Figure 1 shows the architecture of the core middleware of Giotto, BuildingDeport.



- *Figure 1: Giotto MiddleWare System Architecture*

Giotto also contains a Machine Learning layer which allows for supervised machine learning. The machine learning layer communicates with the CentralService and appropriate DataService to train a classifier then can classify new input sensor data. This is all done without any manual feature extraction, training, or model tuning, allowing a novice end-user to create example-driven machine learning classifiers.

Central Service

We define any entity to which a timeseries can be associated with as a *Point.* A *Point* can be measurements from sensors, or commands to turn a light on/off, or a configuration parameter such as cycle type in a washing machine or even a fault indicator. This includes both real-world sensors/actuators, as well as virtual sensors/actuators created in the Central Service as abstractions of RealWorld sensors/actuators (such as a 'building average temperature sensor'). Virtual and RealWorld points are treated identically within the CentralService, with differences occurring at the level of connectors (see Connectors document for interfacing between the

CentralService and sensors). In Giotto, each point is given a UUID (universally unique ID) and metadata is associated with it using *tags.*

Tags are key value pairs associated with a point. For example, an office temperature sensor would be associated with tags like “Room = 300”, “Type = Temperature Sensor”, “Unit = Fahrenheit” and so on. Tags themselves can refer to complex entities and be associated with other tags. For example, Room 300 can be associated with its metadata such as area and usage type[1]. Tags form the core of Giotto metadata, and are used for searching and defining permissions.

Giotto supports pre-defined *tag types* that acts as a template for a user to start tagging entities in a building. These templates are provided to support standard naming convention, such as the tags defined by Project Haystack[2]. These tags are also used as the key search mechanism within Giotto. Using REST APIs, users can query for individual entities based on single tag (such as a Room = 300) or based on more complex combinations of tags (Room = 300 and Type = Occupancy and Building = Example_Building). All entities which meet the requirements of the search are returned as JSON objects which contain their UUID, tags, and Metadata.

In addition to tags on entities, Giotto also utilizes context based tags for Users. Depending on how a user logs in (e.g. with or without admin privileges) they will have a context tag added to determine the privileges that they enjoy (user credentials). Users groups are created by assigning a user-group tag to each user which is part of the group. This is of particular importance later when determining permissions.

In addition to User Groups, Sensor groups are as the name suggests a set of points that have been grouped together on the basis of the tags that the user selected while creating the group. While creating a Sensor group each individual points that the user wants to put in the group do not have to be manually added. Simply selecting the tag will automatically add at the backend all the points containing that tag into this group.

Sensor groups and User groups come together to form the access control lists. Access control lists are a key element in GioTTO to facilitate both privacy and security. For pairs of User Groups and Sensor Groups, we choose a permission value with which we want to associate. There are four levels of permission defined in Giotto which are ‘d/r’ (deny read) ,’r’ (read) , ‘r/w’ (read write) and ‘r/w/p’ (read write permission). If there are multiple permission mappings between a user and a point then the one that is most restrictive is chosen. The deny read permission level, in particular, is important for maintaining privacy of data for various groups simultaneously using Giotto for a building.
Levels permission access are as given below :

[1] This feature is under development
[2] Project Haystack. http://project-haystack.org/

    - d/r
        - Deny-read - Will deny any access to the points
    - r
        - Read - Will give read only access to the points
    - r/w
        - Read-Write - Will give read and write access to the points (such as changing values of actuators)
    - r/w/p
        - Read-Write-Permission - Highest level of permission that can be assigned. Will give read and write access to the points. Permission additionally allows users to alter permissions for the points (see above).
- Ownership
    - Defined at point creation. Has all privileges of r/w/p. Additionally, cannot be revoked or changed under any circumstances. Super or permanent R/W/P.

When a r/w/p permission link is created between the UserGroup “Home_usergroup” and Sensor Group “Home”. All users in UserGroup get r/w/p access to points in SensorGroup. Any user can create any permission link that they want but the set of points that the users in the UserGroup get access to in the SensorGroup that they have been given permission to will depend on the user that has created the permission. Only the points that the creator of this permission has r/w/p access to will be the points that the users will gain access to based on this permission link.

Data Service

DataService stores point time series data points from the underlying point networks. The DataService manages the points and time series data points of points allocated to it. A DataService may belong to any single administrative group that requires sole control over who can access the underlying point data. Different buildings on a campus might desire their own DataService. Thus it is up to an institution to determine how many DataServices are needed depending on the specific groups that exist and their needs.

DataService needs to query CentralService for user accounts, building tags and permission information. The communication is immensely frequent as almost every request to DataService needs user authentication and permission check. Therefore, only keeping a single CentralService would be a performance bottleneck. To resolve this issue, we set up the CentralService in a master-slave mode. The master CentralService only accepts write requests and each of its replicas undertakes read requests from a single DataService. In this way, the request traffic load can be balanced on all replicas.

Machine-Learning

The Machine Learning layer allows for supervised machine learning. A GIoTTO administrator or end-user can provide a label to apply to all of the sensor time series available in a particular

location, for a particular segment in time. The machine learning layer communicates with the CentralService and appropriate DataService to train a classifier using these labels, and then can classify new input sensor data. This is all done without any manual feature extraction, training, or model tuning, allowing a novice end-user to create example-driven machine learning classifiers.

Security

Oauth:

In order to access the GioTTO all users will first have to generate an OAuth Token using the Client ID and secret Client Key that they obtain from their account at the CentralService on GioTTO. Once logged into the CentralService users can go to the OAuth tab and generate a Client ID and Key that should never be shared with any other users.
Once the Client ID and Key have been generated the user can request GioTTO for the access token. The access token will be valid for the next 24 hours after which another access token will have to be generated.The OAuth token should also never be shared publicly.

HTTPS:

HTTPS – Ensures that communication with all Giotto services happens over encrypted connections.We Suggest using LetsEncrypt to generate the SSL Certificates,although we do provide option for using Self-Signed Certificates.

Access Control Layer - Level of access granted to each user for a certain sensor is restricted through this layer based on permissions defined within Giotto middleware BuildingDepot. This is basically the permission levels that we discussed above.

Performance:

The performance of Giotto was benchmarked under the following conditions:

- Two Versions of Giotto using a uWSGI and nginx on a 1core/ 2GB VM

- 1000 clients were simulated to simultaneously send 1,000,000 requests to the service using apache benchmark

API Request	Latency (ms)	Throughput (# per sec)
Write a data point into a sensor	449.347	2225.45
Read the latest data point of a sensor (in cache)	406.837	2457.99
Read the recent ten data points of a sensor	458.018	2183.32

Example Applications

Building Genie, a software thermostat, directly integrated with a building's HVAC system, is a recent example of an application using Giotto as its backend. Genie displays essential information required from a modern thermostat in a web app and supports features such as (i) the ability for occupants to send thermal feedback to building managers, (ii) an expanded level of temperature control, (iii) the ability to turn On/Off HVAC as needed, and (iv) estimates and displays the energy used by each thermal zone to building occupants . We deployed Genie in a 5 floor university building to study its real world usage by 220 users for 21 months. A paper on this application "**Genie: A Longitudinal Study Comparing Physical and Software Thermostats in Office Buildings"** was recently accepted to UBICOMP 2016 for more details.

We also developed BuildingSherlock (BDSherlock), which is a web service based fault management framework that exposes building information to automatically detect faults using Giotto as its backend. Building faults can lead to large and unnecessary building operation costs (such as over-heating or over-cooling), and it is estimated that 5-20% of building energy costs are due to faults. We deployed BDSherlock in a 145000 sqft building at UC San Diego, successfully demonstrate faults using data driven analysis

Getting Started with Giotto:

We provide two additional documents with guidelines for how to set-up and use Giotto. As mentioned previously, Giotto is a superset of BuildingDepot. Giotto uses BuildingDepot at its core and adds a number of other services. Our getting started guide applies towards getting the core BuildingDepot functionality running, as well as writing connectors to Giotto.

For getting started with Giotto/BuildingDepot (Download/install/config):
https://docs.google.com/document/d/1XESPZSIt0lIMrCbVb-Uoopa-_t0yNx6uo-d2IBZGLyk/edit?

usp=sharing

For an example connector to Giotto/BuildingDepot See:
https://docs.google.com/document/d/1-Aqw5G361TXwSX-9gHA_bDsN9TV-1YIbctUJeFS2AjM/edit?usp=sharing