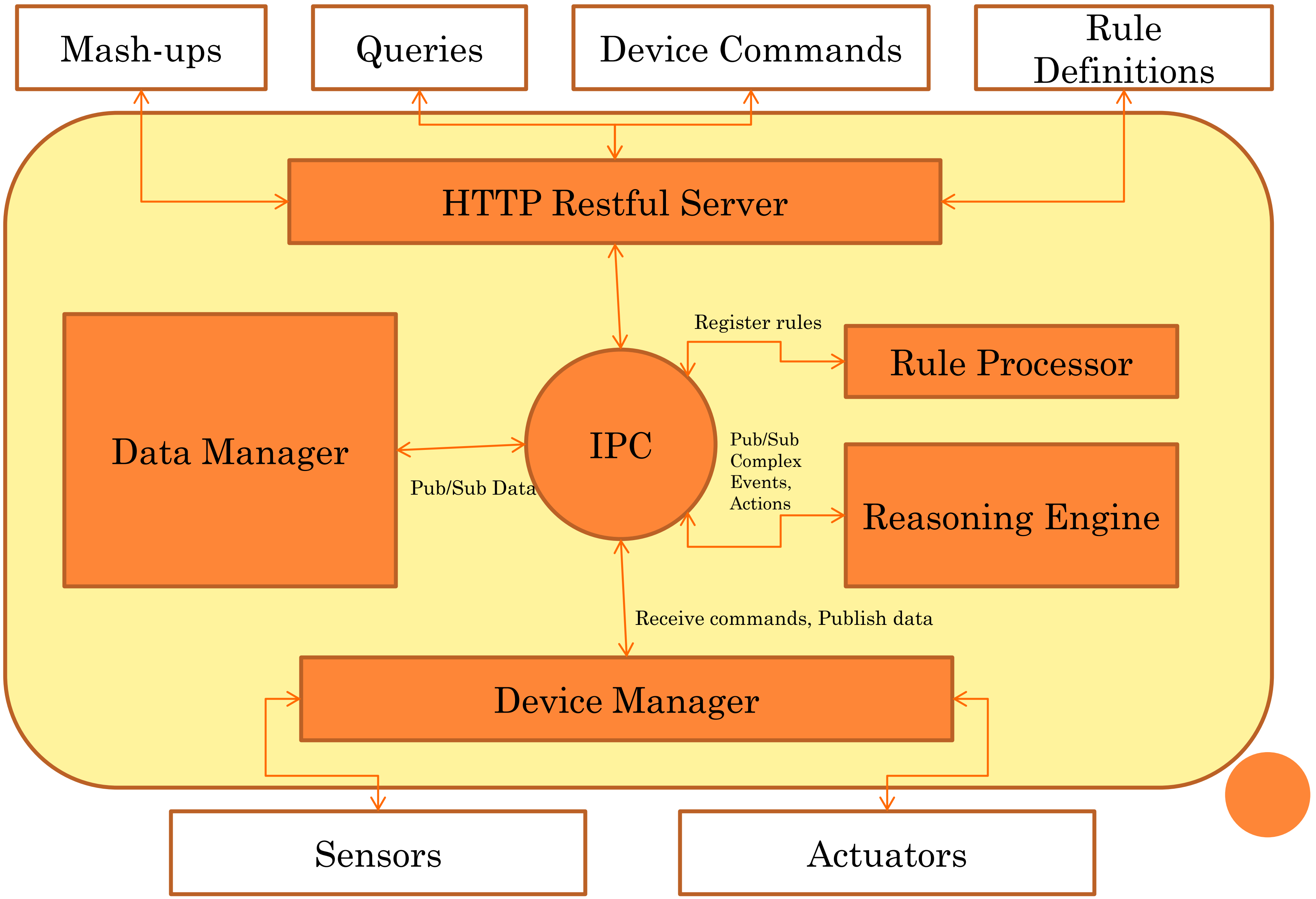
Smartgateway

- Dr. Yann-Hang Lee
- Shankar Nair

4 Basic Components

- Device Manager
  - Low level backend
- Http Restful Server
  - High level frontend
  - User explicit control
- Data Manager
  - Streaming data storage
  - Allows subscribing/publishing data
- Reasoning Engine
  - Detect temporal complex events
  - Execute actions based on defined rules

Smartgateway Architecture



Device Manager

- Main Purpose
  - Handle communication with physical devices.
  - Publish device-status changes to higher layers.
  - Receive commands that must be run on the physical device. These commands can come from:
    - HTTP Rest Server
    - Reasoning Engine
  - Exposes a device-instantiation server to handle dynamic device creations

Device Manager - Architecture

- DeviceBase
  - This is an abstract class that models the communication with a physical device such as its read/write frequency, and lays out a framework so that upper layers need only implement device specific functionality and do not care about the actual communication with the device.
  - It also takes care of multithreading and error handling.

Device Manager - Architecture

- Device
  - The template class inherits from the DeviceBase and adds the capability of adding a custom handler.
  - E.g. int for FDs or CvCapture for OpenCV programs

- DeviceDbus
  - Every physical device is represented by a Dbus object.
  - Implements interface as specified in the interface-xml file.

HTTP_REST_SERVER

- Main purpose
  - Expose RESTful web-interface.
  - Maintains catalog of devices.
  - Provides framework to access functionality of devices.
  - Provides basic query capabilities to lookup devices.
  - Handles actual instantiation of devices by contacting the DeviceManager.
  - Provides view of data and supports mashups (TBD)

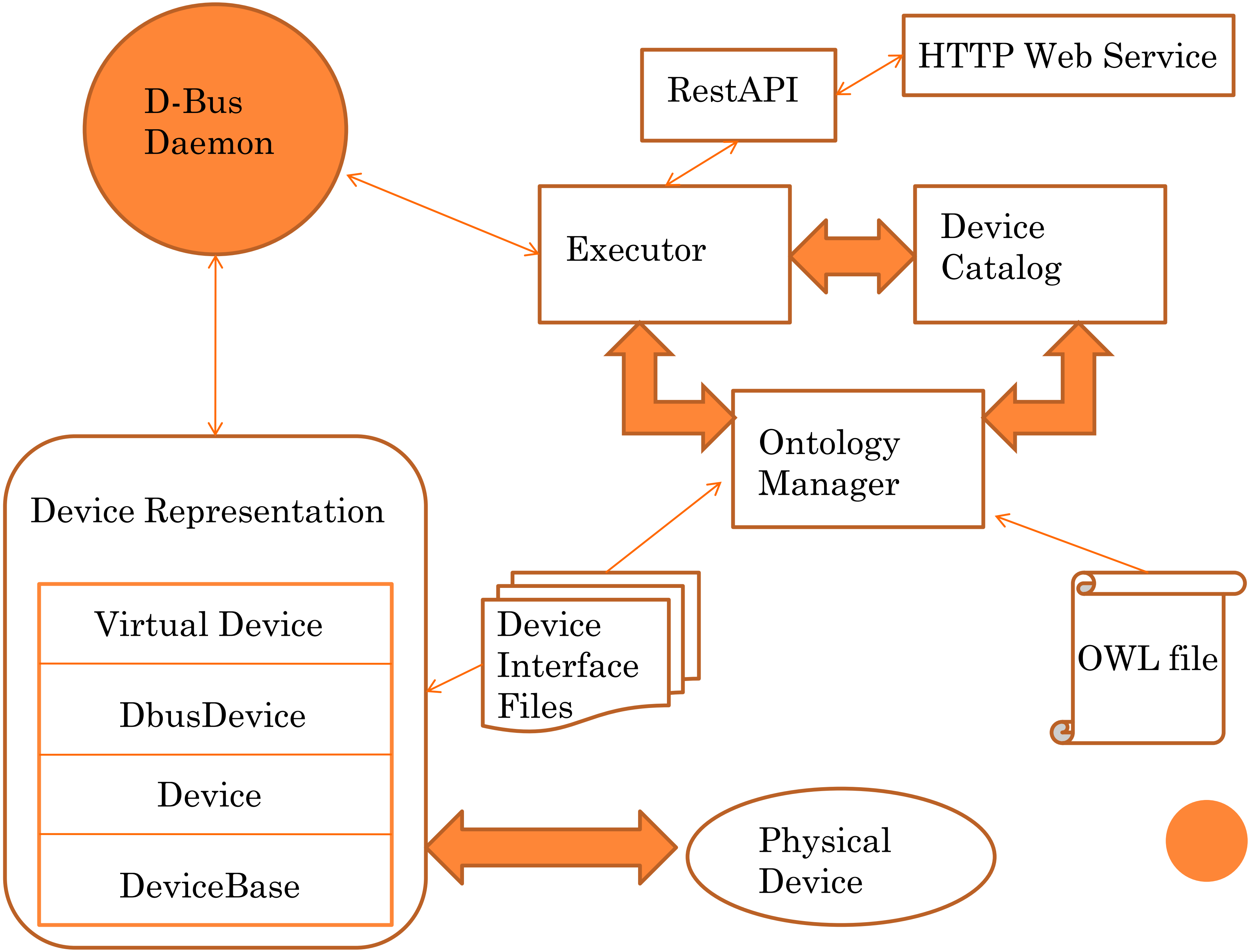
HTTP_REST_SERVER – ARCHITECTURE

- Httphandler
  - Implements HTTP web-server handling.
- RestAPI
  - Provides necessary framework to register Rest APIs.
- Executor
  - Implements RestAPIs.
  - Formats response output.

HTTP_REST_SERVER - ARCHITECTURE

- OntologyManager
  - Provides APIs to support querying of devices.
  - Processes OWL file to initialize the DeviceCatalog

- DeviceCatalog
  - Maintains map of all active devices.
  - Provides APIs for invoking methods on the devices.
  - Processes interface-xml to extract interface information.
  - Associates physical devices to their respective interface based on the Ontology.


D-Bus Daemon
RestAPI
HTTP Web Service
Executor
Device Catalog
Ontology Manager
Device Representation
Virtual Device
DbusDevice
Device
DeviceBase
Device Interface Files
OWL file
Physical Device

Why Represent Knowledge ?

- To perform high-level queries like "Where is person X now?" or "How hot/cold is the living room?" or "How many people are in the meeting room currently?"
- To perform high-level actions like "Notify user whenever he runs out of groceries, maintaining lighting and temperature conditions based on user activities and preferences, medical and health related recommendations on food and exercise, advise on purchases" and so on…

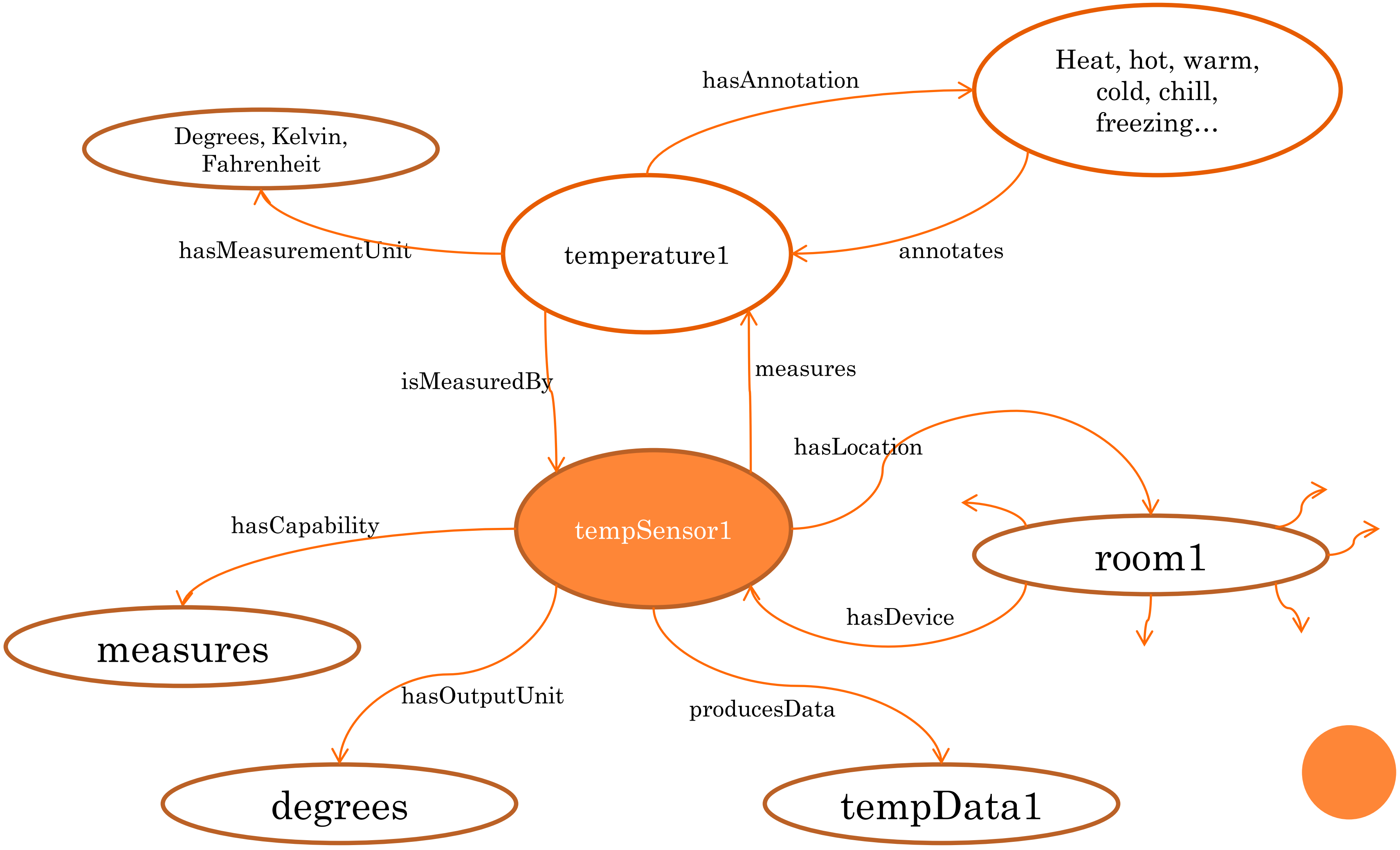
Why use Ontologies to Represent Knowledge ?

- Current Smarthome solutions are merely remote control frameworks and standards that need the user to deal with the sensors and devices manually.
- Google Nest, Apple HomeKit Framework.
- All “smart” decisions are still made by the user. They do have static rules that auto-adjust certain settings based on feedback from the sensors. But it is very limited.
- Example query: How many people are standing on the front porch?
- Current Solutions: User requests a live feed on his smartphone from the specific webcam overlooking the front door and then decides how many people are there.
- These frameworks do not “understand” the expectations of the user.
- SmartGateway Solution: Models a knowledge graph based on device capabilities, location, physical phenomenon, time frame, person, device and user-activity. Find all devices that “hasLocation” “FrontPorch” AND “hasCapability” “detects” AND “detects” “objects”. This sub-query will return the specific webcam and would query the webcam for face-count in its feed.
- The above representation is very naturally modeled and queried using Ontologies.

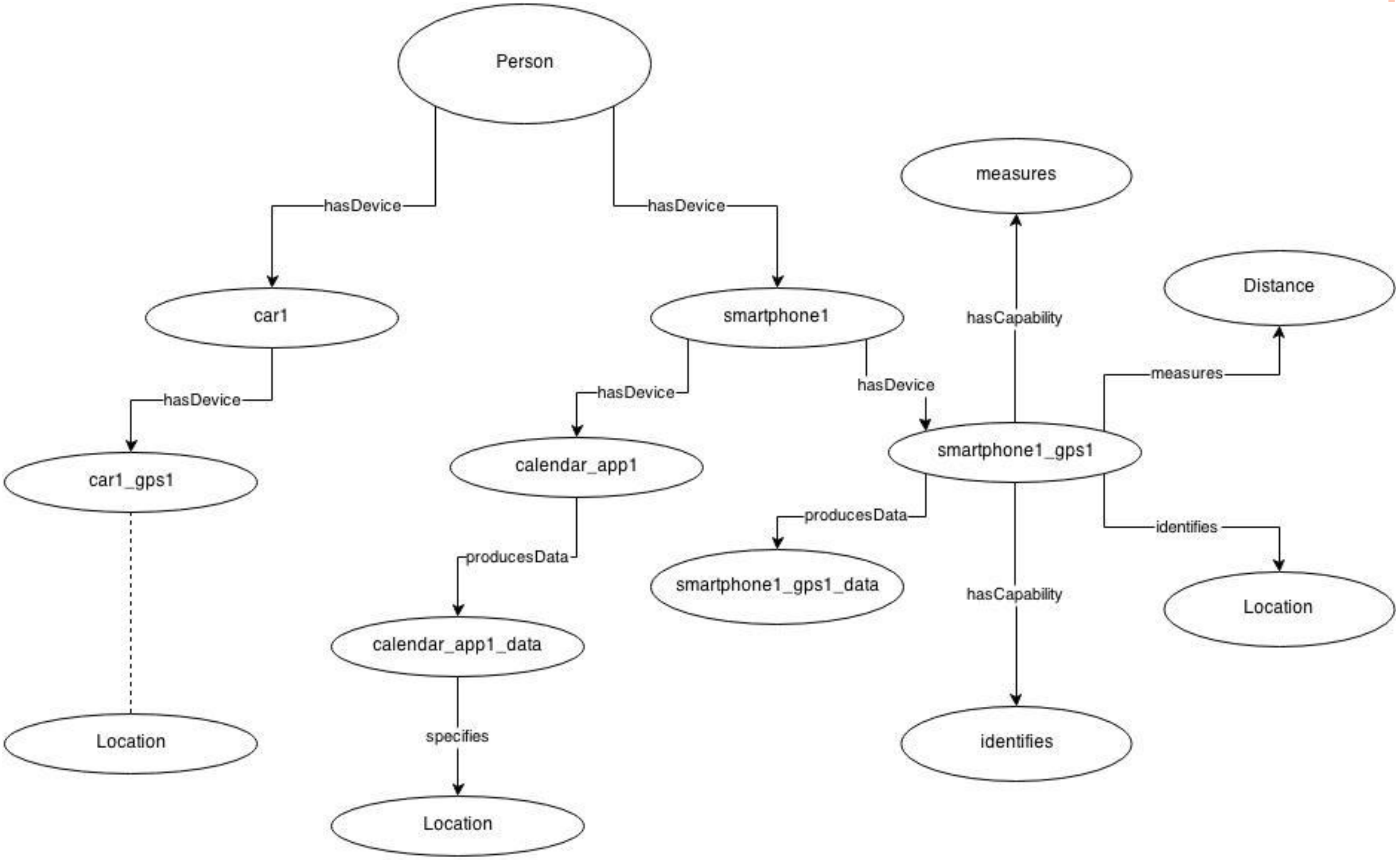
Semantic Drivers

- Till now we have seen how we can access the functionality of devices using their device drivers.
- Can we represent their capabilities in such a way that their capabilities can be queried and accessed by a machine?
- YES! We create a generic model that when implemented by a device will allow it to become a part of a larger “web” of knowledge that is machine-processable.
- Bring “understanding” to machines.
- This generic model is known as its Semantic Driver.

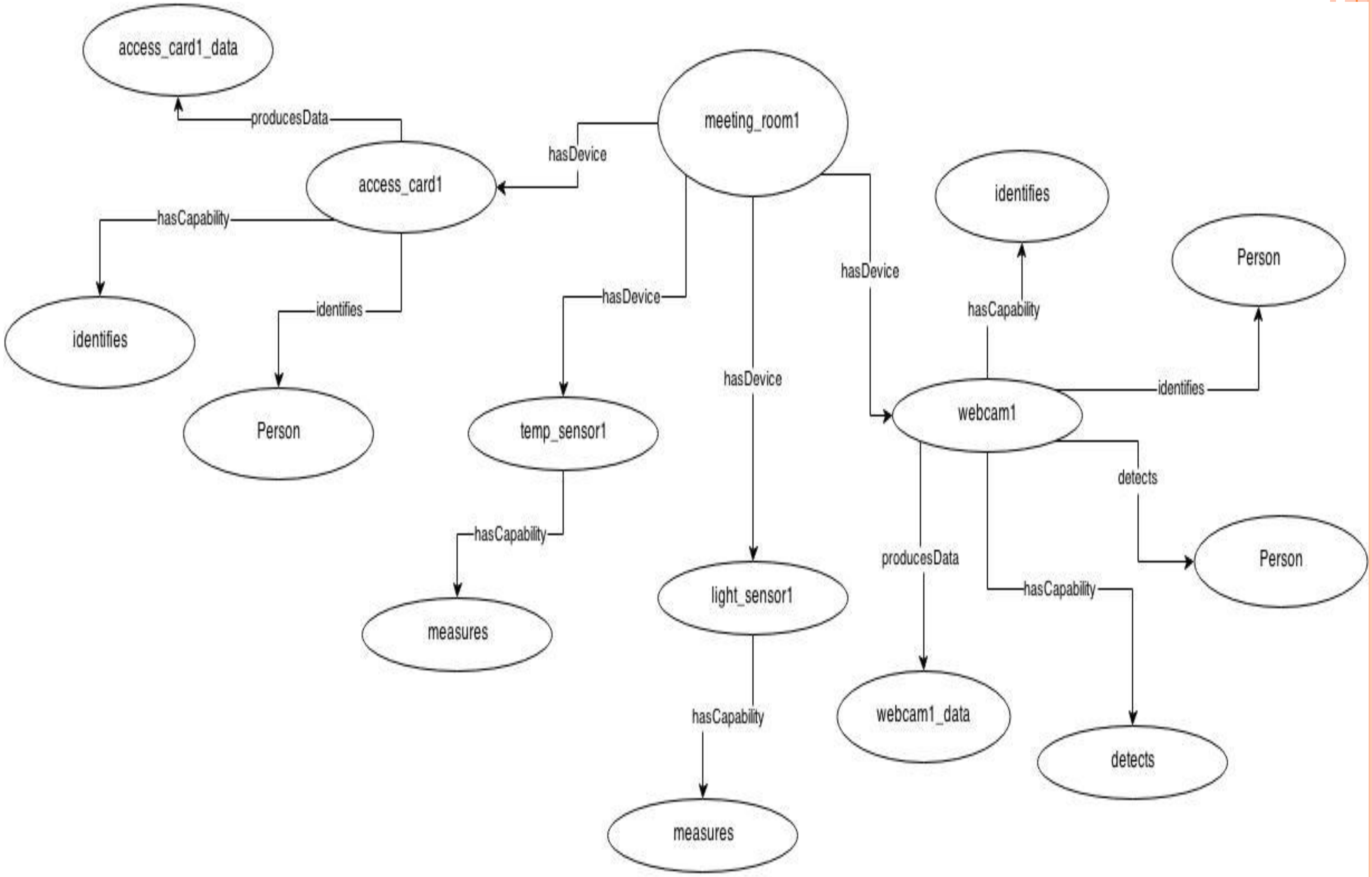
Semantic Driver Model for a Simple Temperature Sensor



WHERE IS PERSON X RIGHT NOW?



Is Person X really at that Location?



Libraries

- Glib-2.0 (-lgobject-2.0 -lglib-2.0)
  - Used for the internal representation of a D-Bus Object using Glib.
- D-bus (-ldbus-glib-1 -ldbus-1)
  - Used for D-bus support.
- Microhttpd (-lmicrohttpd)
  - Used to support the HTTP server implementation.
- OWLCPP (-lowlcpp_io -lowlcpp_logic -lowlcpp_rdf)
  - Used to read, parse and query OWL ontology files within program.
- Raptor (-lraptor)
  - Used by OWLCPP to parse the Ontology file.
- XML (-lxml2)
  - Used by OWLCPP to read the Ontology file.
- FaCT++ (-lfactpp_kernel)
  - Used by OWLCPP to support DL reasoning within the program.
- Boost (-lboost_filesystem -lboost_program_options -lboost_system)
  - Used to ease file-io, string handling and various HTTP response output formating such as JSON, XML etc.

Thank you!