IOT: COMPLEX APPLICATIONS
Yet, we use ancient methods
THREE TIERS TO RULE THEM ALL

Transforms
- 1Hz Flow and Temperature Sampling
- Data Buffering, Compression, Encryption
- Bluetooth LE, Data Forwarding, App Tracking
- Data Storage, Decryption, Acknowledgment
- Statistics, Experimentation, Inference

Controllers
- GATT/BLE
- HTTPS/REST
- SQL/R/Python

Models
- Reliable Comms.
- End-to-End Ack
- Encrypted Data
- Reliable Comms.

Views
- Shower Data
- Flow Rate
- Temperature
- Current Water usage
- Fine-Grained Water Usage
LED CONTROL - SYNCHRONIZED DURABLE

Embedded

Gateway

Cloud

LED State

LED State

LED State

Physical World

Mobile App

Web Client

View

View

LoC/files 488/3

3270/47

146/8

django
AN IOT FRAMEWORK SHOULD HAVE

• **Unified model** – entire high-level application in a single language and a single framework

• **Flexible partitioning** – reconfigure where data processing and storage occurs

• **Interaction everywhere** – easy to develop interaction

• **Deployment ready** – real code
PROGRAMING PARADIGMS

1. Enabling **SQL-like queries** over a group of nodes (Tag, Tinydb).
2. **Macroprogramming**, a high-level abstraction for apps over a set of nodes (Cosmos, Envirosuite).
3. **Stream-based programming**, higher level development on the sensory data flow (MISSA, SPITFIRE).
4. **Process-based development**, where the user programs the network as virtually-connected devices (BPMN4WSN).
5. A **prototyping frameworks** for rapid development of IoT applications (Fabryq, Exemplar).
6. Model-View-Controller (**MVC**)
MVC - BASICS

• MVC - emerged for user interfaces, adopted by web community (Ruby-on-Rails, Django, Meteor).

• Models: manages data, logic and rules of application.

• Views: output representation of the data.

• Controllers: converts inputs to commands for models and views.
DMVC FOR IOT

• **Distributed** Model View Controller

  • **Models**: manages data, logic and rules of application *across all tiers* of application.

  • **Views**: multiple fundamentally different representations, still same data.

  • **Controllers**: moving data between tiers, views and interactions on radically different platforms.
DMVC CHALLENGES

👍 Models are distributed across different devices:

- We need powerful yet simple mechanisms for automatically managing the data, storage, views, computation.

👍 The applications must be secure:

- Need some reasoning to suggest appropriate security policy and selection of encryption mechanisms.
DMVC - MODELS

- Supporting **Distributed** models:
  - **Synchronized**: data is streamed from source to sink. Data is not mutable.
  - **Replicated**: same data across all tiers. Data is always mutable.
  - Naturally, there is more model types e.g. real-time.
- **Spaces**: distributing primitives to tiers.
- Automated controllers: for networking, storage, computation.
DMVC - MODELS

Embedded

LED State

Physical World

Gateway

LED State

Mobile App

Cloud

LED State

View

Web Client
DMVC - ADDITIONAL PRIMITIVES

- **Space**: describe the properties and configuration of underlying tier. Contains means to generate device specific code.

- **Transform**: takes a model, computes on it and outputs a different model. Often boolean, filters, aggregators, basic math (avg, max, min, compare).
TO JUST FEW

class LedModel(models.Model):
    led_state = transforms.FieldFromModel(
        from_model="arduino.components.LedToggle", pin=7)

class Meta:
    replicated = True
durable = True
link = 'BLE'
• Deriving possible security policies

• DMVC allows analyze where data is accessed and what operations are used on it.

• This knowledge let’s us reason about which existing protocols can be use.
MANY BENEFITS

• Distributed Model View Controller
  • Higher level reasoning about the application and data
• Abstracting complexity:
  • we do the heavy lifting
  • we eliminate a subset of programming errors: type cast, null pointers
• One framework rather than multiple languages
THANK YOU!

QUESTIONS?

LAURYNAS RILISKIS
LAURIL@CS.STANFORD.EDU