Internet of Things
2016/2017

Architecture Styles
relevant to IoT

Johan Lukkien

John Carpenter, 1982
Guiding questions

• Which architectural styles are useful for IoT and why?
Architectural styles (patterns)

• An architectural style is a coherent set of design decisions concerning the architecture
  – a combination of a typical (de)composition
  – and typical choices for connectors, components (building blocks) and behavior

  ....a generic solution for a class of problems

• Such decomposition pertains to the structure of the system, e.g.
  – clients and servers, with correspondent distribution of functionality and behavior
  – services and orchestrators, in a Service Oriented Architecture

• We also have interaction styles, concerning just the interaction between building blocks
  – the nature of the connectors, and their organization
  – e.g., procedure calls, streaming, events
Architectural style characterization

- Defined by
  - motivation (guidelines, context) for the application of the style
    - which extra-functional properties are achieved, and how
    - which problem class is solved
  - vocabulary
    - names for components (building blocks) and connectors, and for other concepts
  - rules (constraints, responsibilities) for components and connectors
  - generic structure and behavior
    - interfaces of components, and correspondent connectors
    - data distribution, protocols, control flow, data flow

....documented as a profile (e.g. UML)

- When applied, a style yields a partial architecture
  - in fact: styles classify architectures
- Within an architecture, several styles can be applied
  - and also, several alternatives in interaction style

- Styles encourage communication, reuse, comparison (of alternatives)
• Motivation:
  – special usage of C&S aiming at
    • portability
    • independent development & deployment
    • reduction of interaction complexity
    • reliability
    • scalability

• Vocabulary
  – user agent, origin server, gateway, proxy
  – cache, layer
  – state
  – code-on-demand
  – resource, resource identifier (e.g. URI), representation, metadata

• Rules
  – stateless communication (no server interaction state), self-descriptive messages
  – response is labeled as cache-able, and can then be cached
  – uniform interface between components
    • decouples structure from functionality
  – layering
  – client functionality can be adapted by code-on-demand

REST: REpresentational State Transfer

• Structure

Vocabulary
– user agent, origin server, gateway, proxy
– cache, layer
– state
– code-on-demand
– resource, resource identifier (e.g. URI), representation, metadata

Rules
– stateless communication (no server interaction state), self-descriptive messages
– response is labeled as cache-able, and can then be cached
– uniform interface between components
  • decouples structure from functionality
– layering
– client functionality can be adapted by code-on-demand

• Typical behavior
  – as (layered) C&S, but only a single service (interface)
    • API: PUT, GET, POST, DELETE
  – calls operate on a remote state that can be inspected and modified

• Prime example: WWW

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

27-Nov-16
Basic concept of REST

• *Origin servers* store resources
• Resources can be anything
  – files, variables, devices
• Resources can be manipulated by clients
  – read (obtain a representation): GET
  – operated on / modified: POST
  – created: PUT, POST
  – deleted: DELETE
• Operations have safety requirements:
  – PUT, DELETE and GET are idempotent
• Resources are referred to by an *URI*
• Responses can sometimes be cached

Example: resources on host www.example.org:
/sensors
/sensors/light

Operations:
GET /sensors/light
GET /sensors?d=light
POST /sensors/light off
PUT /sensor//light off

*Observe that the sensor host (probably a small device) is the server*
Naming on the Internet

Universal Resource Name (URN)

```
urn:Sensei:sensinode.com:NanoSensor:N740:3a-43-ff-12-01-01
```

Universal Resource Identifier (URI)

Universal Resource Locator (URL)

```
http://www.example.org:8080/sensors?id=light
```

From presentation Shelby: CoAP: the IoT protocol
RESTful and IoT

- Simple nodes can’t do much more than record and adapt a local state
  - based on sensing
  - based on external updates
- More complex behavior can be moved to other machines
- This fits a RESTful behavior very well

- The most important concerns are:
  - complexity of involved protocols
  - overhead of data representations
    - e.g. heavy XML based protocols and representations

Example: message exchange on top of a (reliable) TCP connection has significant overhead for (small) calls
• Motivation:
  – Separate functionality (“service”), the implementation, the deployment context and the application context
  – Build applications by very late (dynamic) binding of services
  – Integration of enterprise information systems

• Vocabulary
  – SOA, service, interface, discovery, composition, binding, orchestration, choreography

• Rules
  – Application is built ad-hoc out of services that communicate in a standardized manner
    • via a network
    • see e.g. REST
  – Service is a self-contained functionality. It does not depend on state of other services, or of the system (OS, language) it is running on.
  – Services are discoverable

Service oriented (SOA) style

• Structure (conceptual):

• Typical behavior
  1. Providers publish services; Applications (“orchestrations”) discover services, and bind their interfaces
     • Service broker (registry) can exist to manage discovery process
  2. Applications send data objects through a number of services as a workflow
     • Often XML based RPC (SOAP)
     • Services are typically kept simple and focused on single task

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking
SOA and IoT

- SOA focusses on
  - separation of functionality (the service), its implementation and platform
  - on composition of services
  - and on reducing (removing) dependencies between services

- A RESTful system is in fact a SOA
  - with HTTP endpoints as services that manage resources
  - and the HTTP methods as the API

- For IoT and M2M the following are relevant:
  - advertisement and discovery of services on the network
  - binding (connecting) or calling services without human intervention
  - absence of dependency
    - (can use services into different applications without modifying their implementation)
**Publish/subscribe style**

- **Motivation:**
  - Decoupling data producer from data consumer
  - Sending data when it is available, avoiding need to poll for data.
  - Allow multiple consumers
  - Allow runtime changes of set of consumers

- **Vocabulary**
  - Publisher, subscriber, subscription, notification, topics, broker
  - Subscriber registers with publisher to receive notifications for chosen topic.

- **Rules**
  - Every subscription relates a topic to 1 subscriber.
  - Publisher can have multiple subscribers for same topic.
  - Notification goes to all subscribers; however, subscribers may specify receive policies
  - When existent, broker decouples the publishers and subscribers

- **Structure:**

- **Typical behavior**
  - Subscribers find brokers and send subscriptions for topics of interest
  - Publishers register with brokers, or are discovered
  - Notification may be only about events, or it can also contain data.
  - Broker can be omitted

- **Metaphore**
  - Newspaper/magazine subscription
  - Observer design pattern is an example of publish/subscribe style

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

27-Nov-16
P&S and IoT

- P&S changes interactions from pull to push
  - reversing the coupling in time
  - good for low resource nodes
    - can behave according to private schedule

- The use of *topics* gives a simple means to reduce communication
  - basically, a *filter*

- A broker further decouples sender and receiver
  - removes dependencies in time between sender and receiver completely
  - removes dependencies in space (referential dependency) completely

- Presence of a broker admits to solve protocol compatibility and data representation issues
Concluding

• We envision an IoT in which *fully networked* devices report to more powerful devices
  – that receive, accumulate and process the data
  – and also control these devices

• This is established by
  – advertising, discovering and binding services
  – RESTful interaction
  – publishing and subscribing

• What is required to achieve this?
  – *new bindings* of standard protocols to constrained networks
  – *new protocols* that
    • are adequate for *resource constrained devices*
    • support the wanted behavior
Things like this must be solved

- We will address two IETF protocols in particular
  - CoAP
  - 6LoWPAN
- and one OMA protocol
  - LWM2M on top of CoAP
Guiding questions

• Which architectural styles are useful for IoT and why?