From oop to … IOT systems

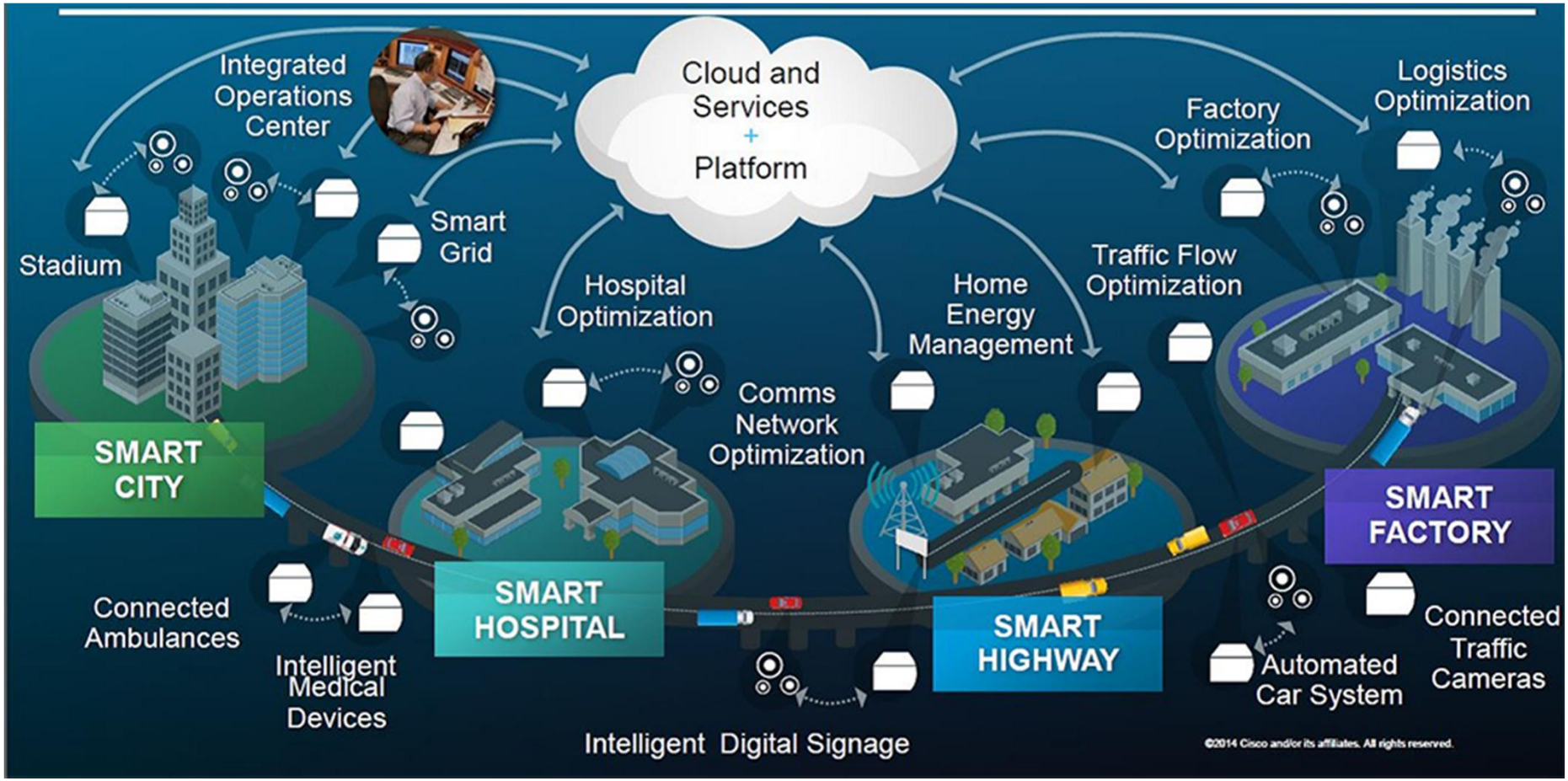
A simple (but not simplistic) case-study

https://github.com/anatali/IotUniboDemo

IOT essential

Just an overview …

The Internet of (Every)Thing (IoT, IoE)

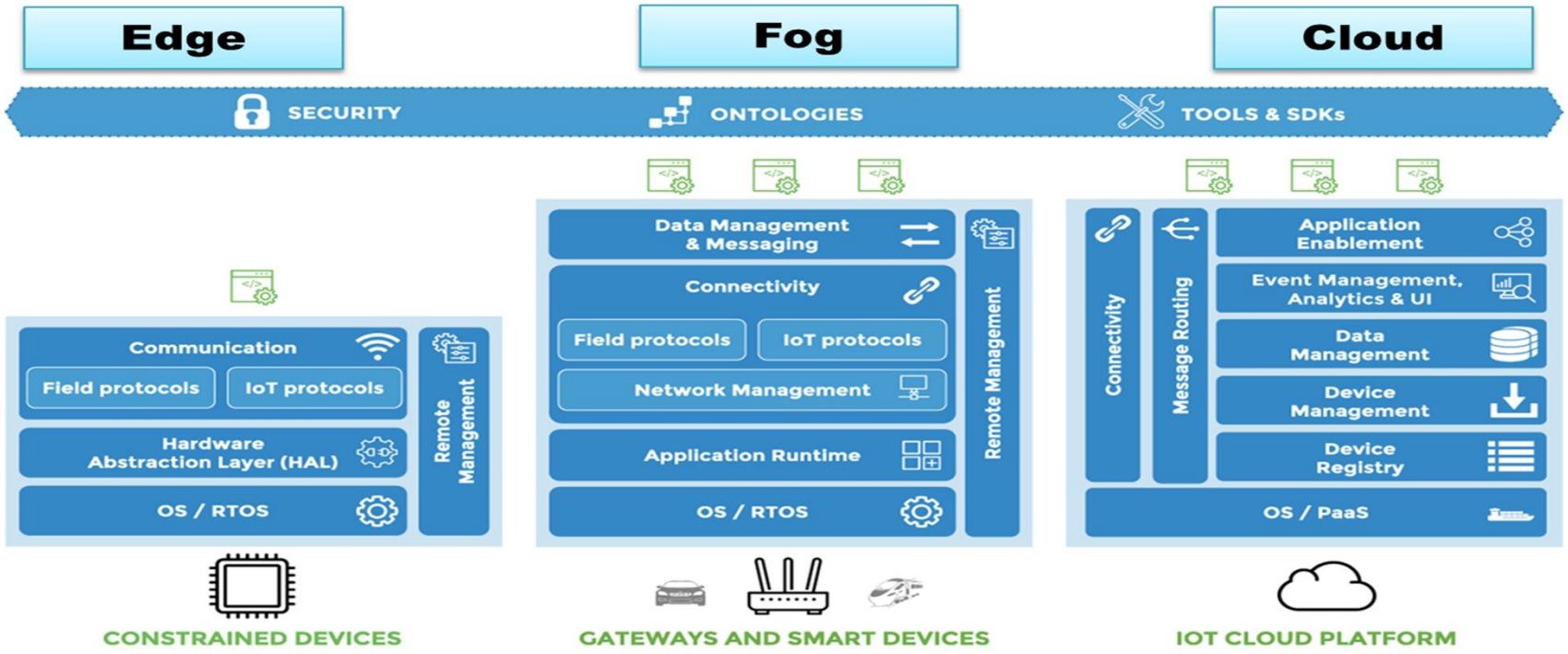


Capturing the real value of Internet-connected devices goes much further than providing connectivity, a transport scheme and databasing.

System architecture

- The Internet of Things really means the convergence of information, control and the cyber elements of physical systems.
- A key characteristic of **Systems Applications** will be the importance of how their basic functions can be combined to provide vertically focused solutions - the bulk of which will increasingly be delivered as a managed or value-added service.
- Some basic design principles must be put in place to guide the growth of a vast, distributed technological **organism** that must remain organized as it evolves according to a logic all its own. It demands that we design not only devices and networks but also information itself in ways not addressed by current IT.
- It will not be feasible to reinvent the underlying infrastructure for each new application. Instead, these applications need to integrate easily with an appropriate networking and data management foundation.

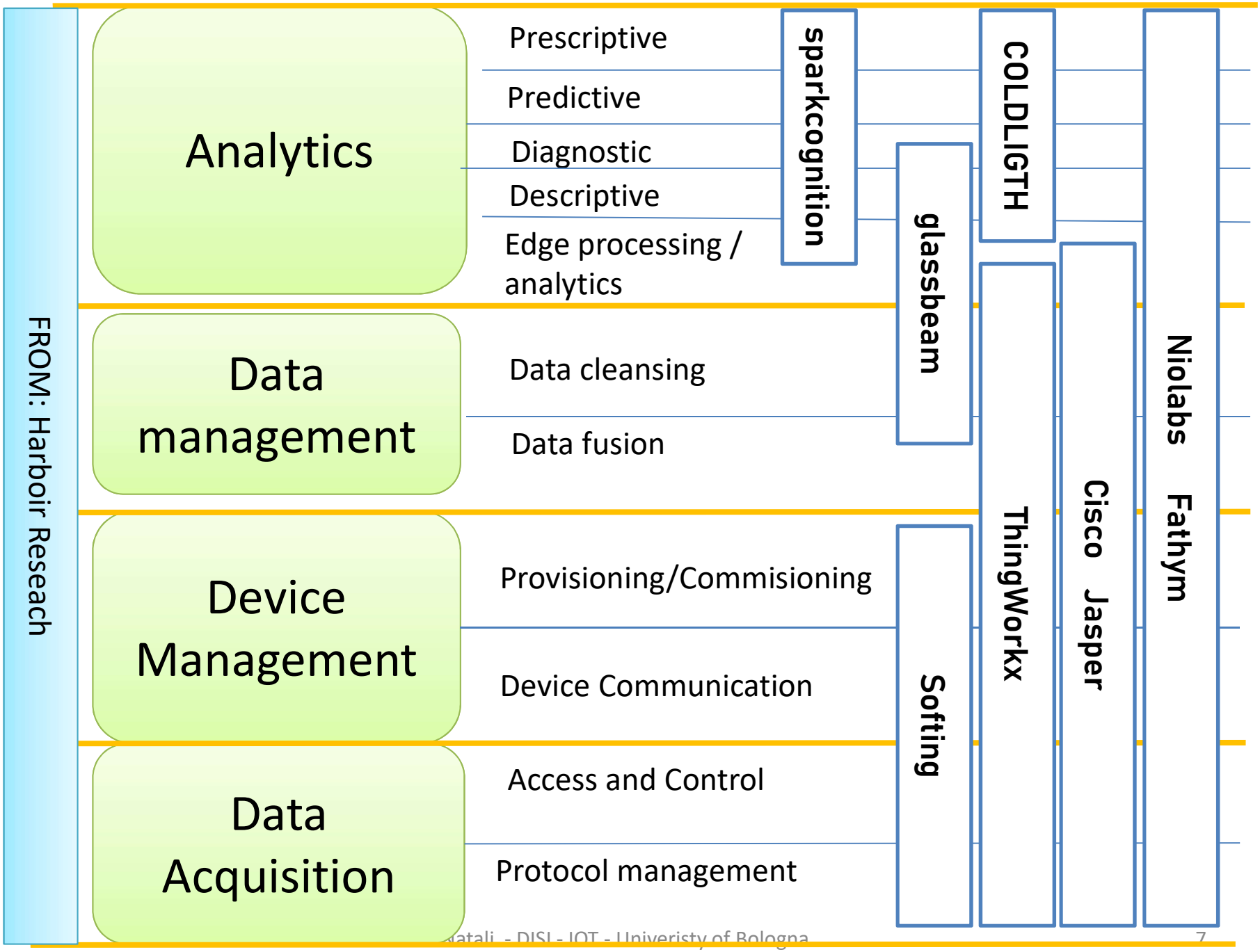
A first reference architecture

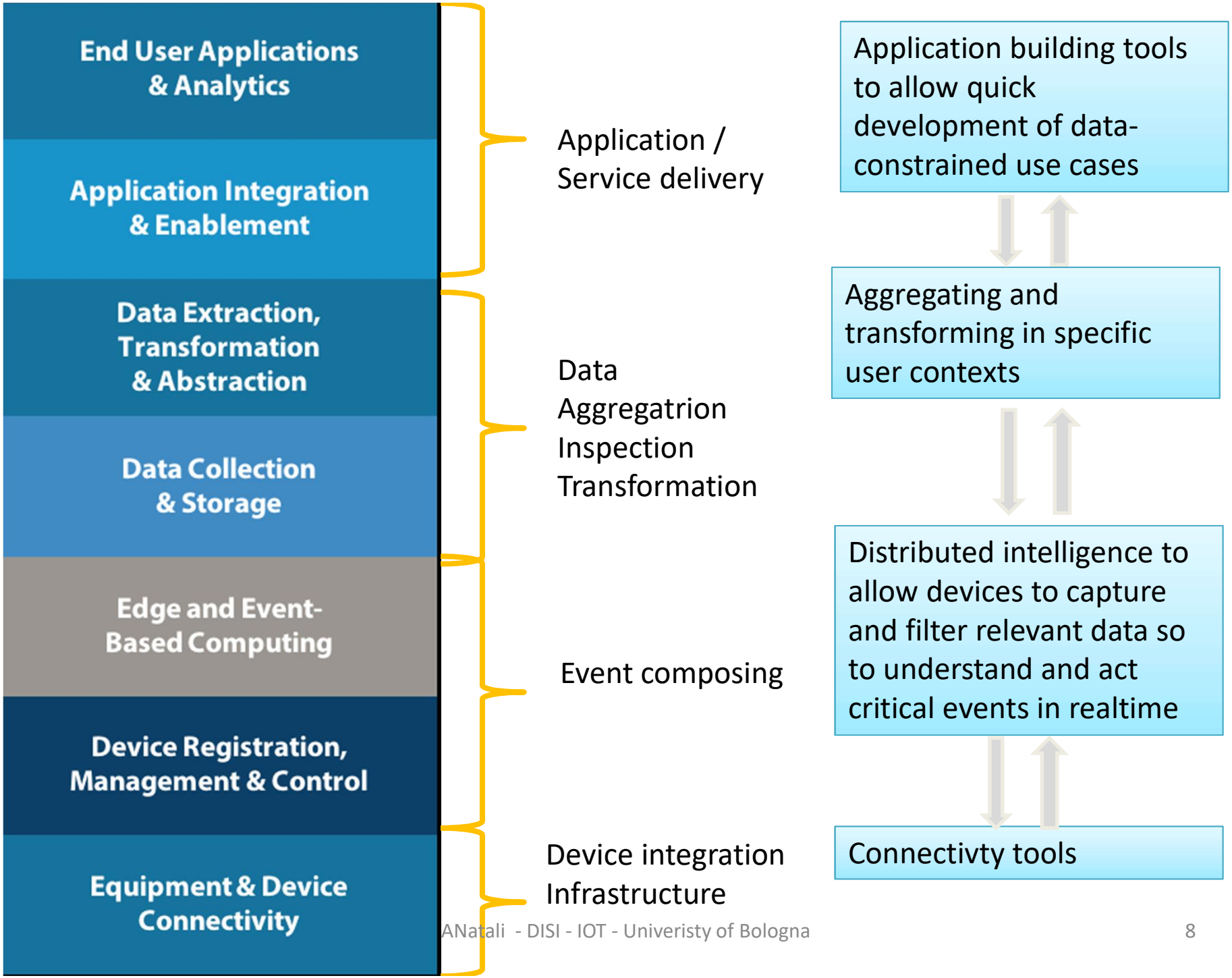


Industrial consortia

- Industrial Internet Consortium (IIC) (http://www.iiconsortium.org)
- Open Connectivity Foundations (OCF) (https://openconnectivity.org)
- OpenFog Consortium (OFC) (https://www.openfogconsortium.org)
- OPC Foundation) (https://www.opcfoundation.org)
  - (Unified Architecture)

- ODVA
- ISA
- PI (PROFIBUS, PROFINET)


FROM: Harboir Reseach
Analytics
Prescriptive
Predictive
Diagnostic
Descriptive
Edge processing / analytics
Data management
Data cleansing
Data fusion
Device Management
Provisioning/Commisioning
Device Communication
Data Acquisition
Access and Control
Protocol management
sparkcognition
glassbeam
Softing
COLDLIGTH
ThingWorkx
Cisco Jasper
Niolabs Fathym


End User Applications & Analytics
Application Integration & Enablement
Data Extraction, Transformation & Abstraction
Data Collection & Storage
Edge and Event-Based Computing
Device Registration, Management & Control
Equipment & Device Connectivity
Application / Service delivery
Data Aggregatrion Inspection Transformation
Event composing
Device integration Infrastructure
Application building tools to allow quick development of data-constrained use cases
Aggregating and transforming in specific user contexts
Distributed intelligence to allow devices to capture and filter relevant data so to understand and act critical events in realtime
Connectivty tools

The digital disruption

- Digital disruption is the change that occurs when new digital technologies and business models affect the value proposition of existing goods and services.
- It is often confused with the term disruptive technology, a term coined by Harvard Business School professor Clayton M. Christensen to describe a new technology that displaces an established technology.

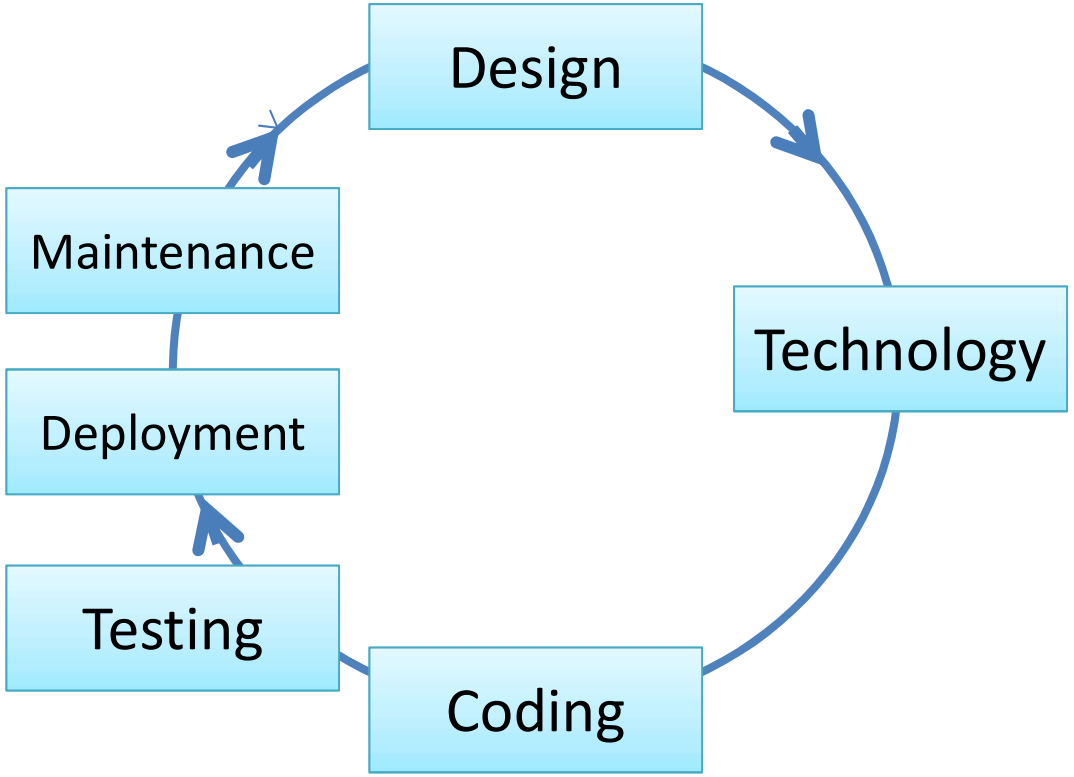
Software production

Iteration among analysis, project, implementation, delivery, maintenance

The impact of software

- *IoT is changing such fundamentals of software application that the traditional programming language constructs and principles are becoming impediments.*
- Language is the vehicle of thoughts.
- IOT applications are ***parallel***, ***asynchronous*** and ***distributed***, not sequential, synchronous or centralized.

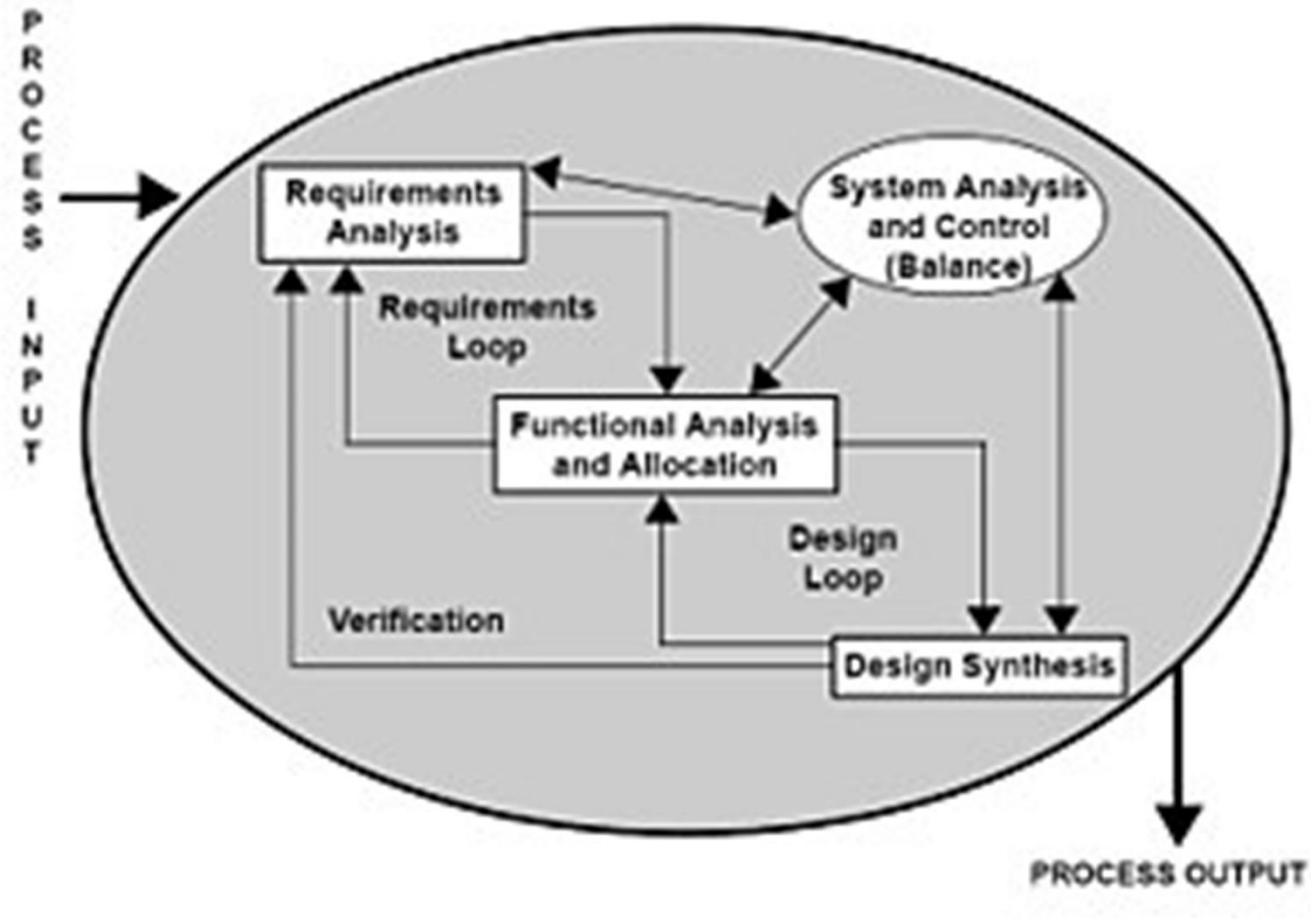
Workflow



- Oop
- Actors
- Event driven

- Java (C++ …)
- JavaScript
- Node

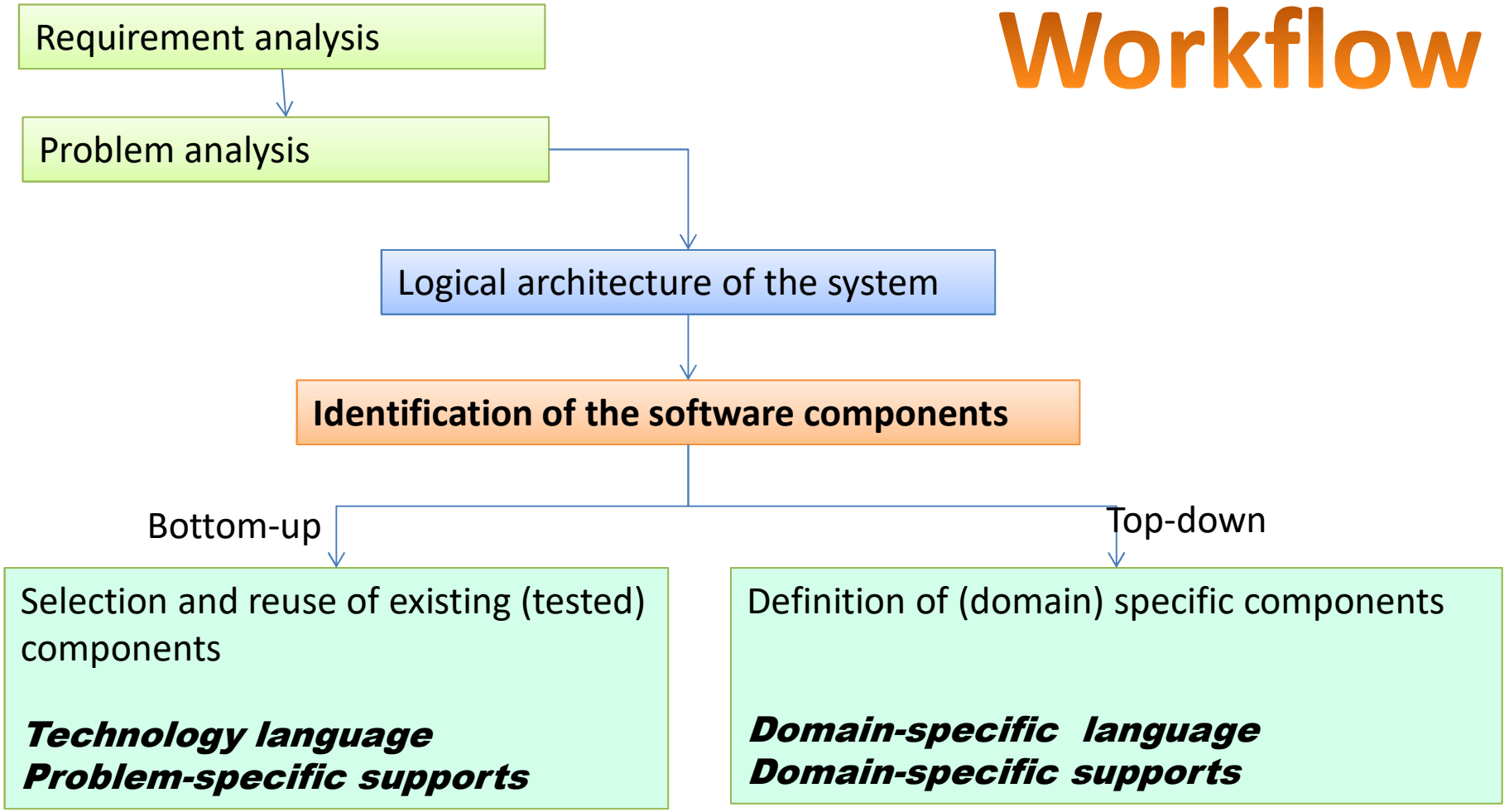
- PC (OS)
- Arduino
- Raspberry



continuous integration (**CI**) is the practice of merging all developer working copies to a shared mainline several times a day.

Grady Booch first proposed the term CI in his 1991 method,[2] although he did not advocate integrating several times a day. Extreme programming (XP) adopted the concept of CI and did advocate integrating more than once per day – perhaps as many as tens of times per day

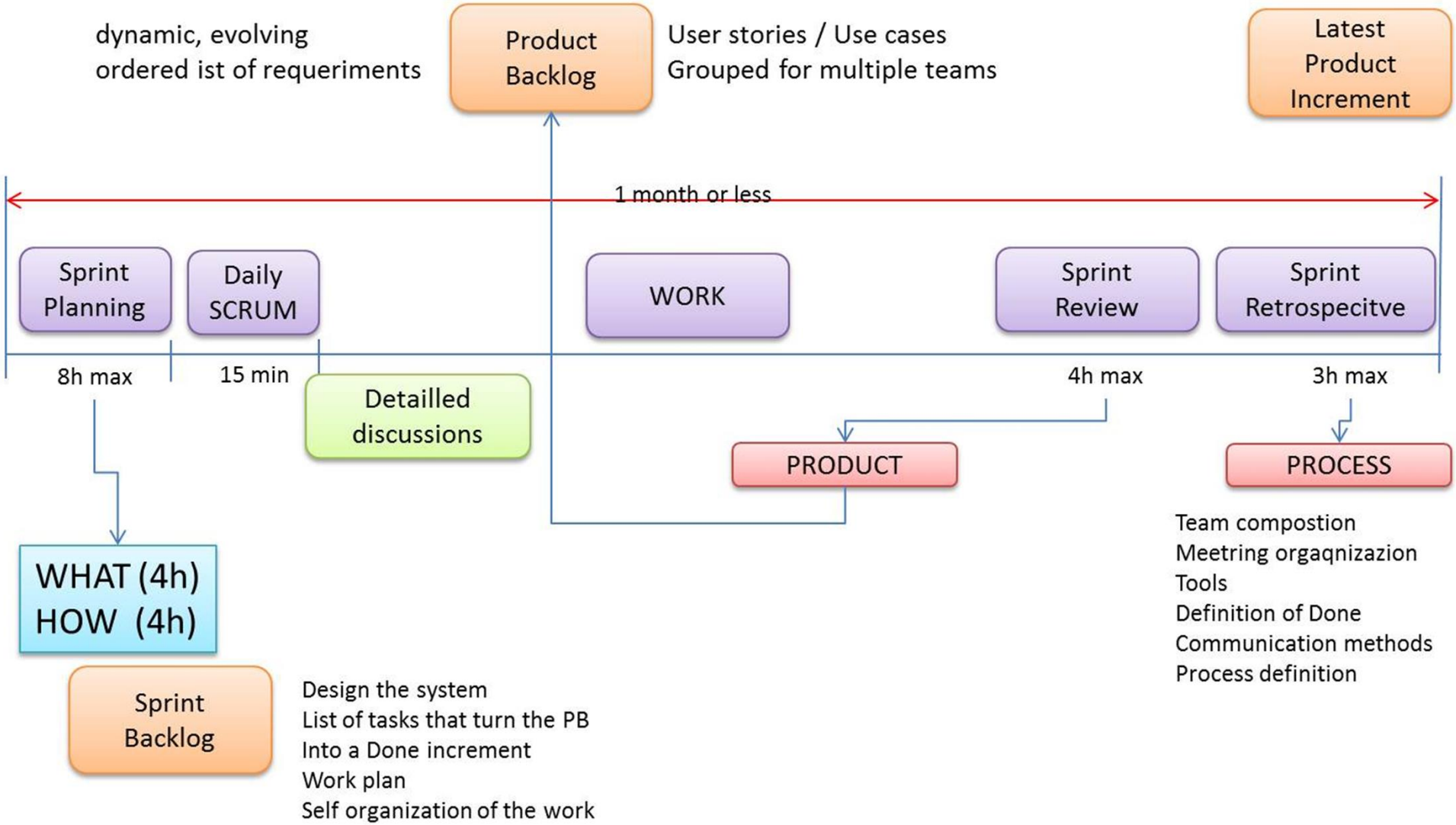
Workflow

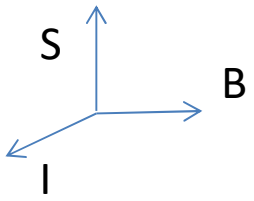


In any case:

What is a software component?
How components interact?
Which components embed the ***business logic***?

SCRUM


dynamic, evolving
ordered ist of requeriments
Product Backlog
User stories / Use cases
Grouped for multiple teams
Latest Product Increment
1 month or less
Sprint Planning
Daily SCRUM
WORK
Sprint Review
Sprint Retrospecitve
8h max
15 min
4h max
3h max
Detailled discussions
PRODUCT
PROCESS
Team compostion
Meetring orgaqnizazion
Tools
Definition of Done
Communication methods
Process definition
WHAT (4h)
HOW (4h)
Sprint Backlog
Design the system
List of tasks that turn the PB
Into a Done increment
Work plan
Self organization of the work



VISION

WHAT

BUSINESS LOGIC
DOMAIN-RELATED TERMINOLOGY

APPLICATION LEVEL

HOW

(MODEL – **CONTROL** - VIEW)

HIGH

MAPPING LEVEL

LOW

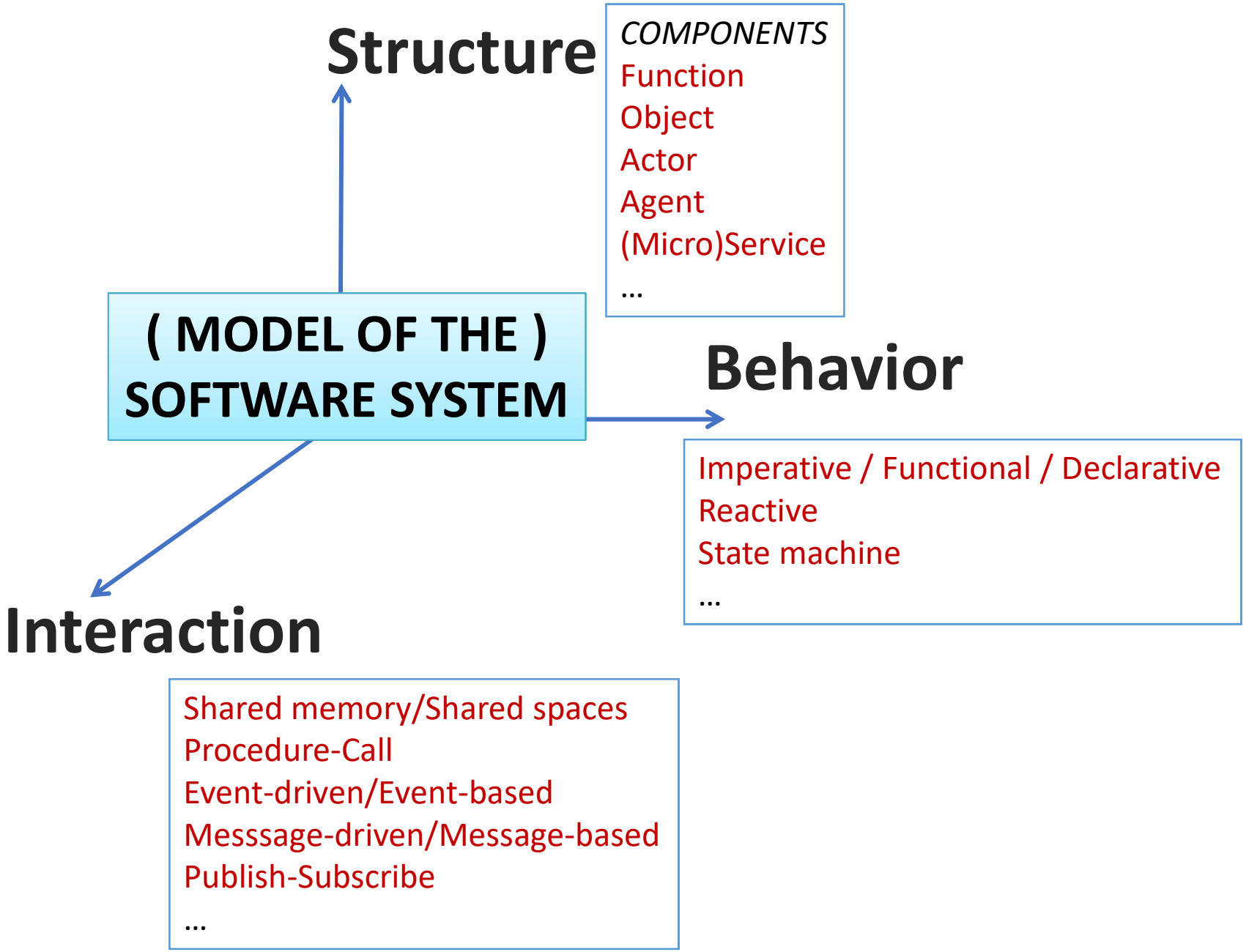
PLATFORM LEVEL

EDGE

FOG

CLOUD

CONCRETE WORLD


Structure
COMPONENTS
Function
Object
Actor
Agent
(Micro)Service
…
(MODEL OF THE)
SOFTWARE SYSTEM
Behavior
Imperative / Functional / Declarative
Reactive
State machine
…
Interaction
Shared memory/Shared spaces
Procedure-Call
Event-driven/Event-based
Messsage-driven/Message-based
Publish-Subscribe
…

IOT minimal
(one sensor, one actuator)

The ButtonLed system

Req0: accendere un Led premendo un Button

Requirement analysis

Il committente

- Cosa intende per Led?
- Cosa intende per Button?
- Cosa intende per premere?
- Cosa intende per accendere?

- Il sistema deve essere concentrato o deve/può essere distribuito?
- Il Led si trova su un oggetto in movimento?
- Vi sono vincoli sull'ambiente e sul consumo energetico?
- Vi sono requisiti di real-time?
- Sono necessari controlli di accesso/autorizzazioni?
- ….

Scenario di partenza

- Led a Button sono dispositivi fisici o virtuali
- I dispositivi fisici possono essere inseriti su dispositivi a basso costo (Arduino, Raspberry, …) e controllabili in software attraverso una o più driver
- I dispositivi virtuali sono realizzati tramite GUI e/o pagine Web

Analysis -> Project -> Implementation

POSSIBILI 'COMPONENTI'

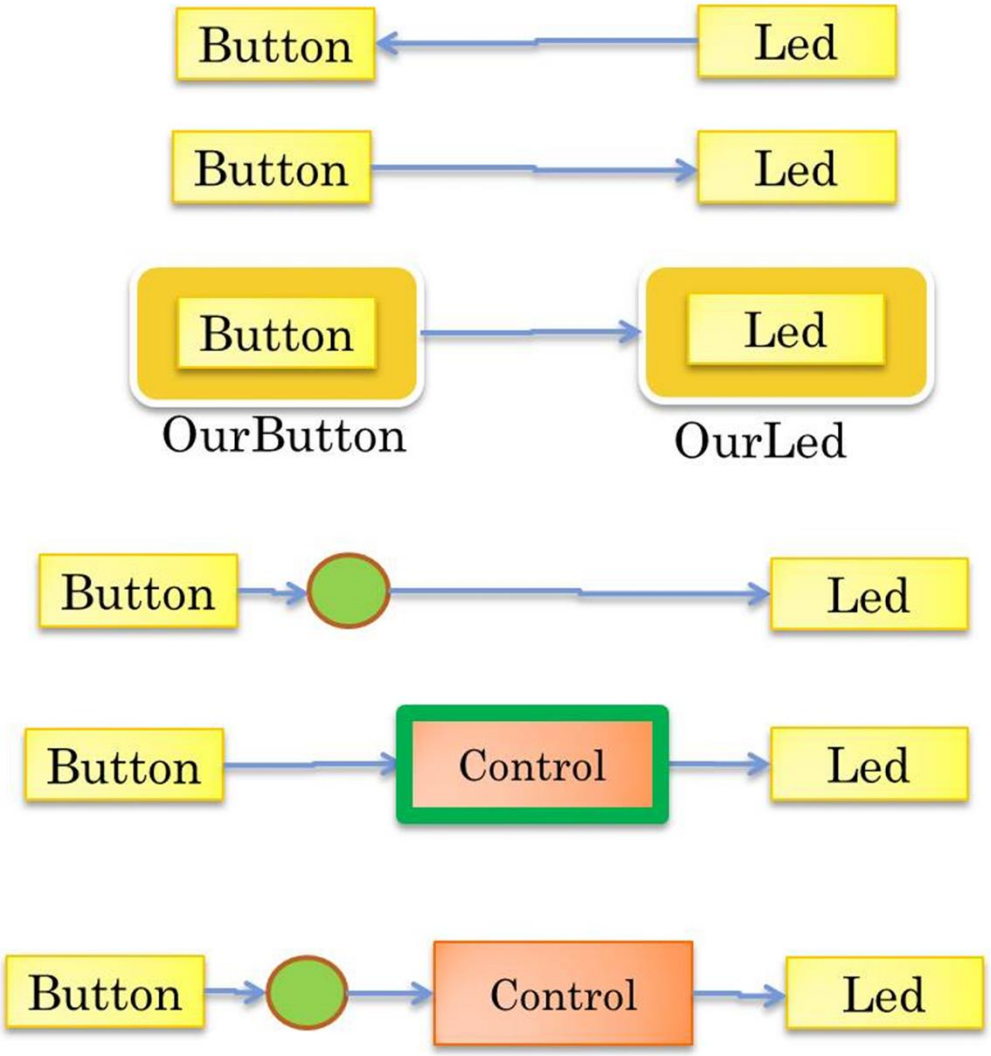
- ***Funzioni*** - vedi Js …
- ***Oggetti*** (POJO)
- ***Componenti soft*** – vedi OSGI
- ***Attori*** - vedi akka
- ***Agenti*** - vedi …
- ***Qactor*** – vedi custom unibo
- …

POSSIBILI 'Behavior'

- ***Control-based*** vedi Java, C …
- ***Event-driven*** vedi Js
- ***Message-driven*** vedi akka

POSSIBILI 'cose da fare'

- ***Dichiarazioni***
- ***Istruzioni*** ***imperative***
- ***Espressioni***
- ***Azioni*** (osservabili, timed,reactive …)



POLLING
The Led looks at the button

DIRECT CONNECTION
The `Button` does not knowm any device

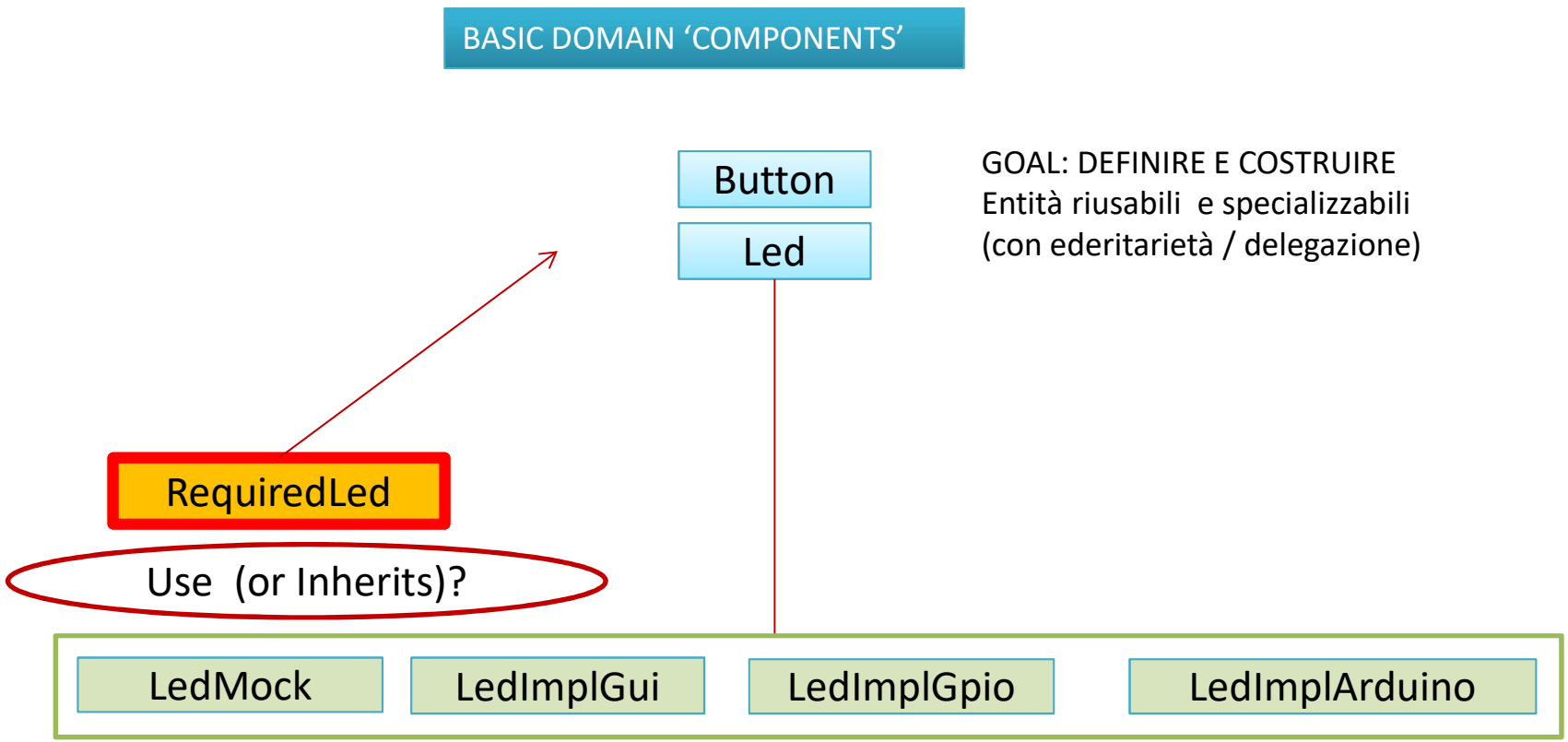
EXTENSION / WRAPPER
Ad hoc network

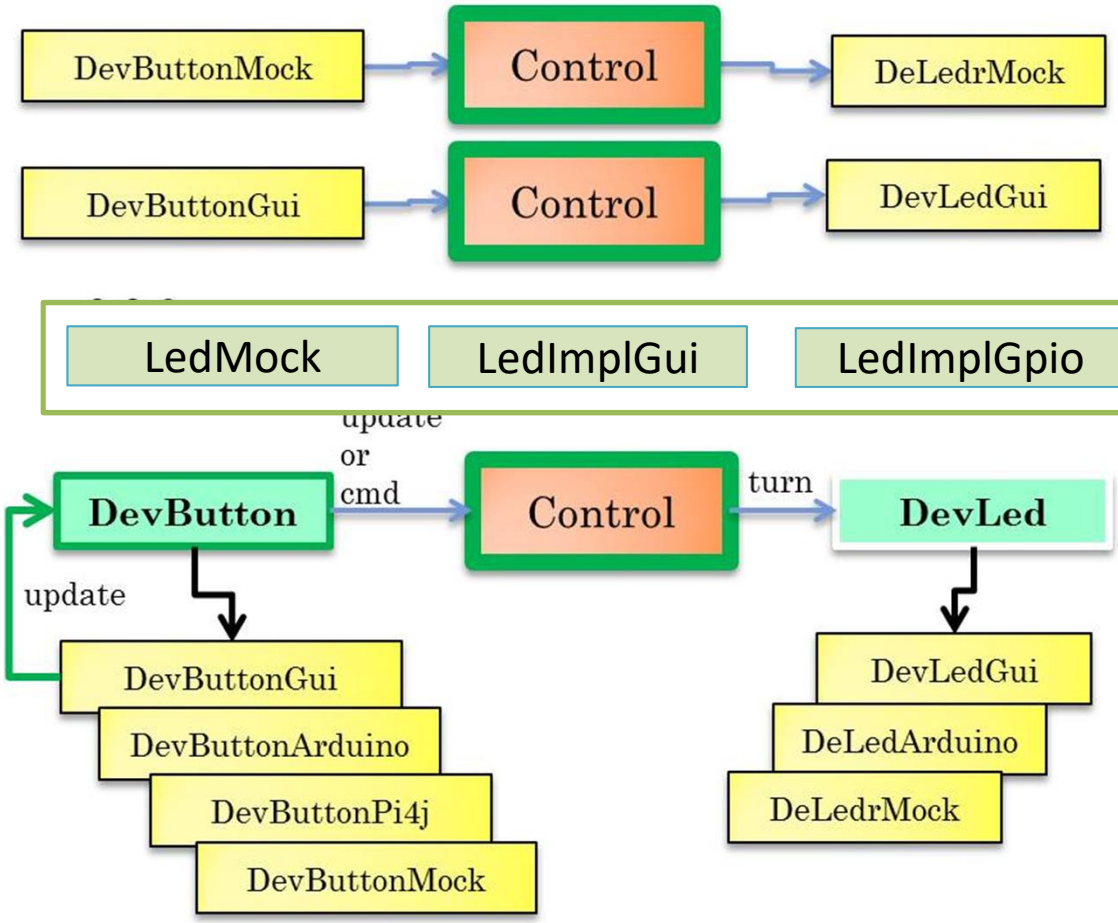
OBSERVER
The `Button` is Observable
The `Led` is called by an Observer

MEDIATOR
The `Button` is Observable
The `Control` is an Observer that defines the business logic

The `Control` is called by a specialized Observer

BASIC DOMAIN 'COMPONENTS'





BLSLHighLevellMain

An architecture for each concrete configuration …

BRIDGE

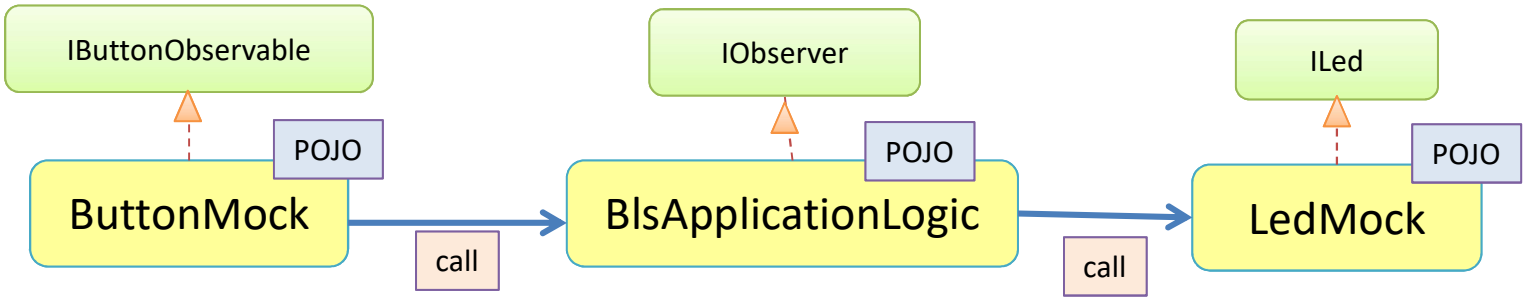
The `DevButton` is an Observable Observer (of the concrete button)

The `Control` can be an Observer

FACTORY

Each `DevButtonXXX` is created by a `DevButtonFactory` (driven by `blsConfig.properties`)

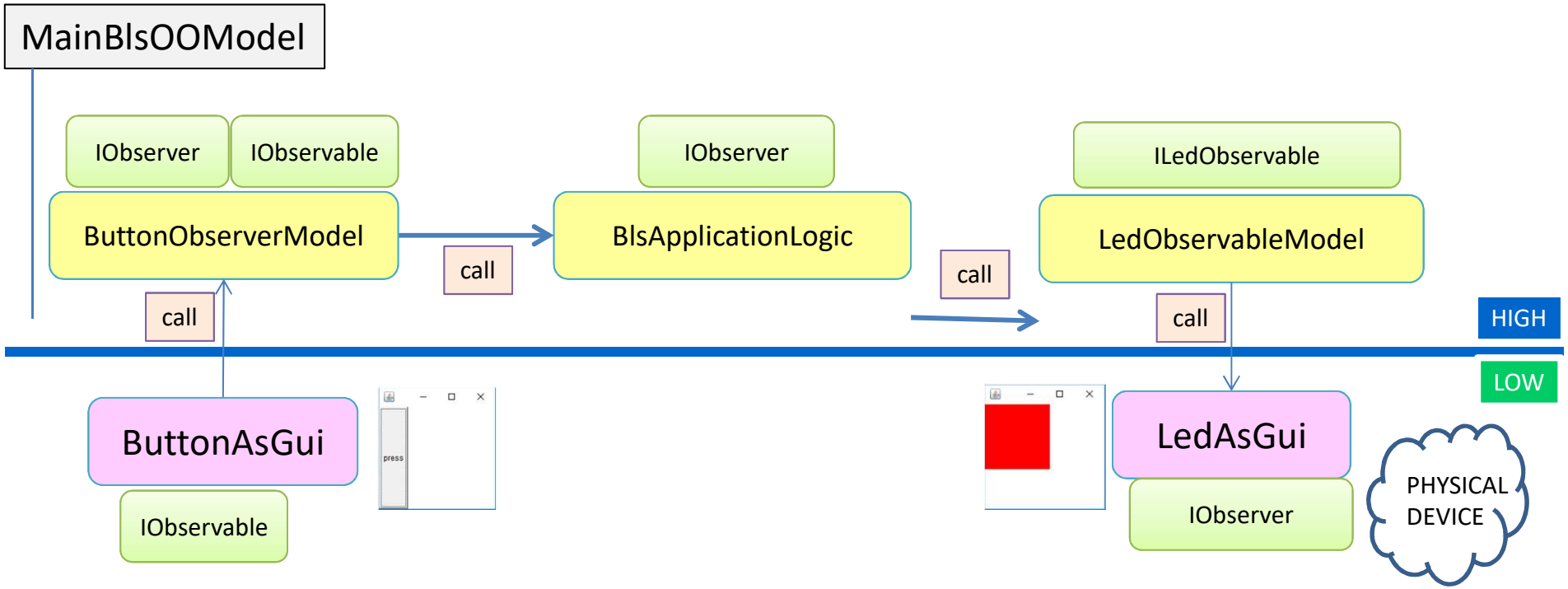
Req0: a first (object based) reference-architecure



MainBlsMockBase

KEY POINTS
- OO Design
- Analysis / Project

Req0: the observer-design patterns at work

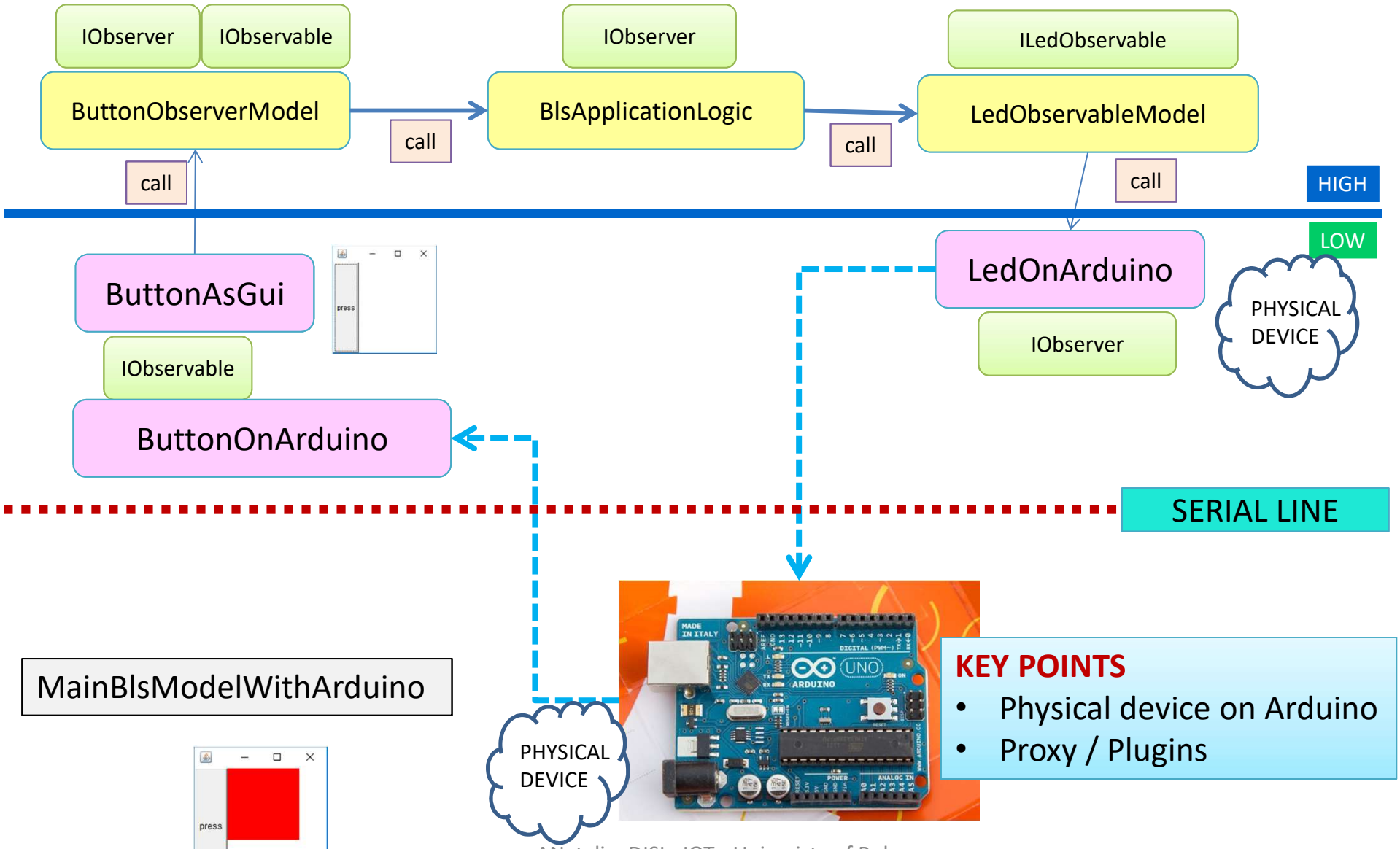


MainBlsModelWithGui

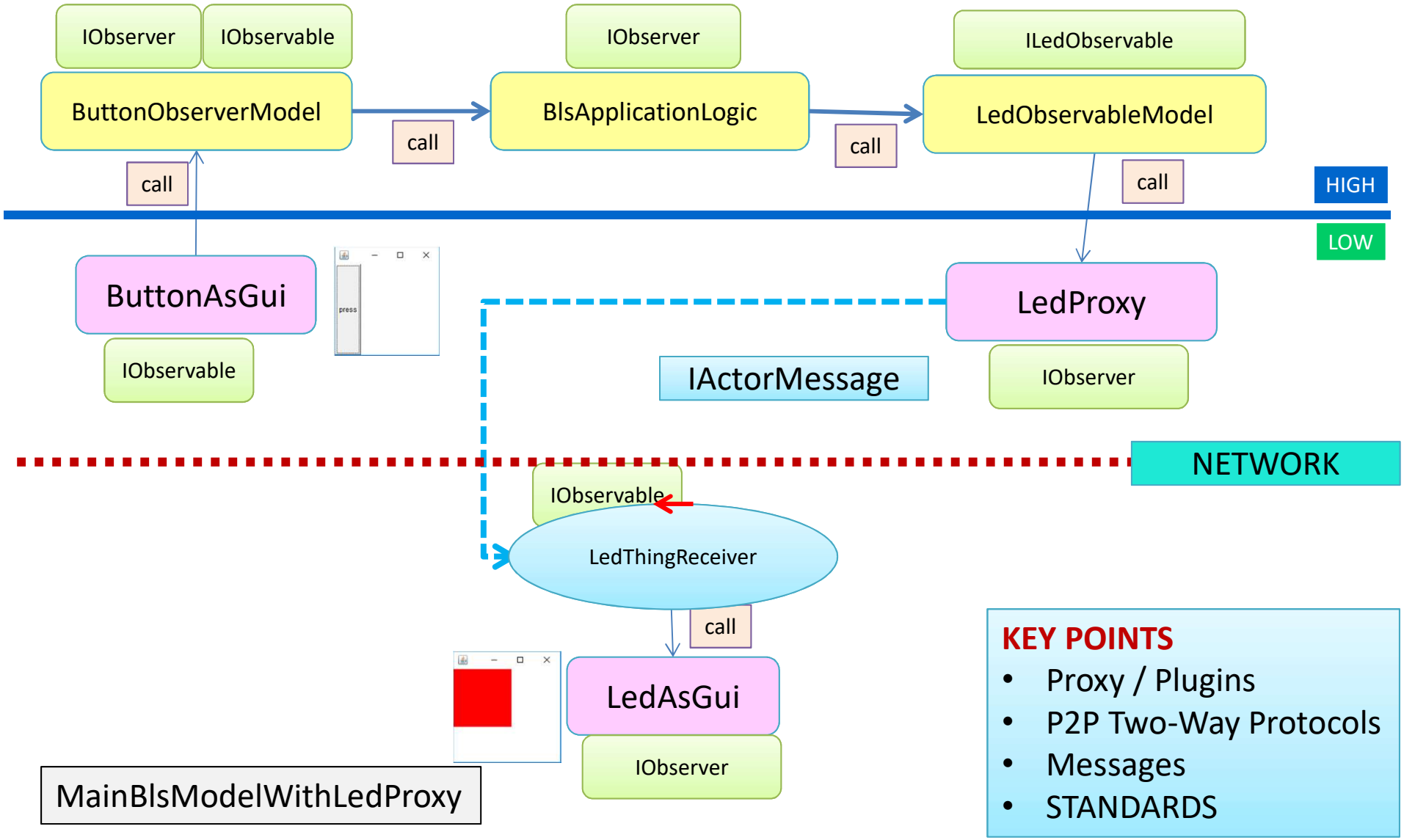
KEY POINTS

- OO Design Patterns (observer, factory ,..)
- MVC
- Frameworks

Req0: concrete devices as model observers



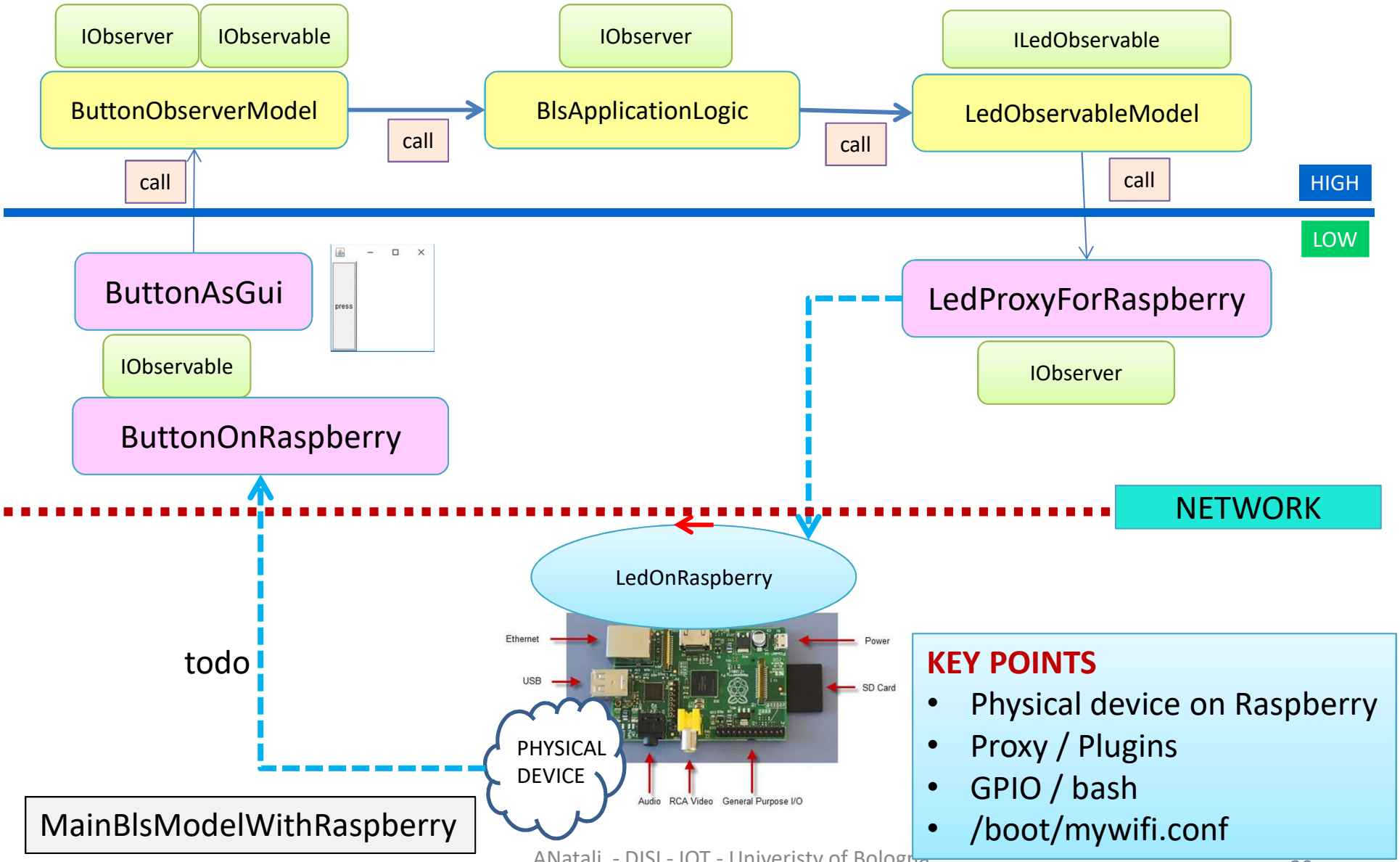
Req0: a proxy for a remote device



KEY POINTS

- Proxy / Plugins
- P2P Two-Way Protocols
- Messages
- STANDARDS

Req0: devices at the edge



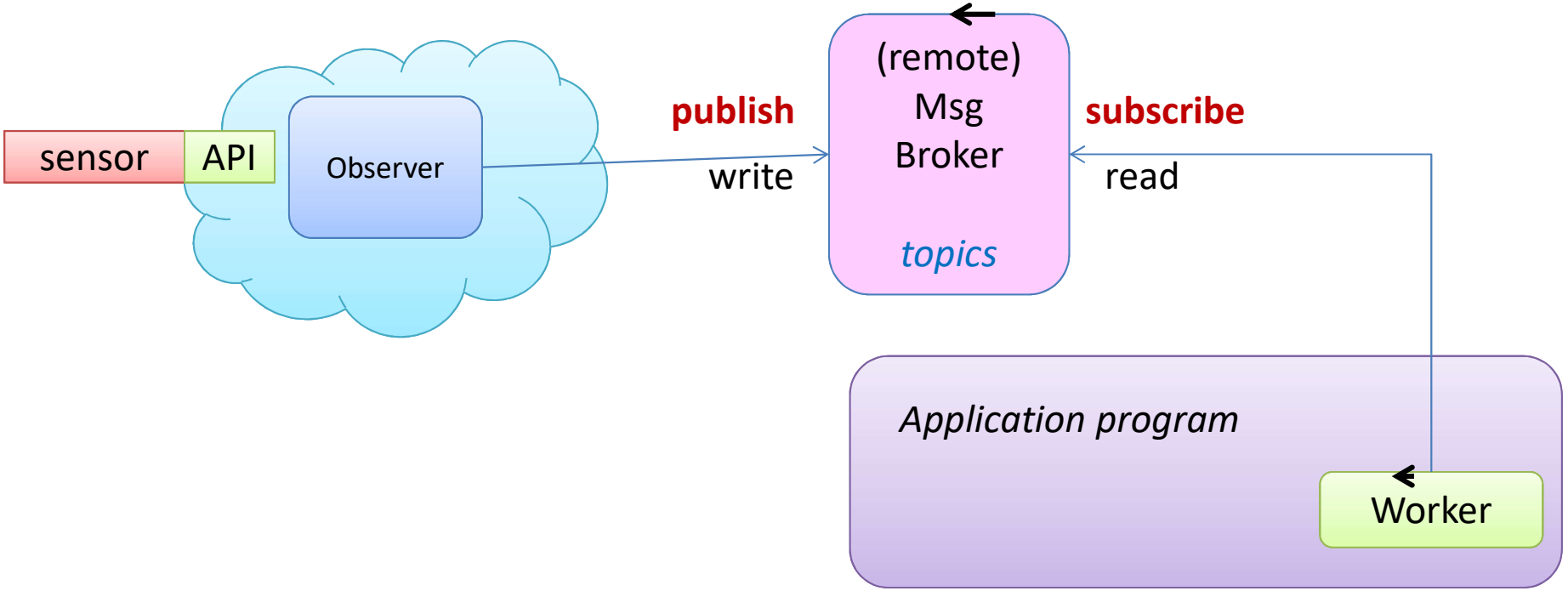
KEY POINTS

- Physical device on Raspberry
- Proxy / Plugins
- GPIO / bash
- /boot/mywifi.conf

Req1: accendere un insieme di Led premendo un Pulsante

MOM

Publish–subscribe pattern



The MQ Telemetry Transport (MQTT) is an ISO standard (ISO/IEC PRF 20922) supported by the OASIS organization.
Eclipse Mosquitto™ is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 3.1 and 3.1.1.

Mosquitto on Docker

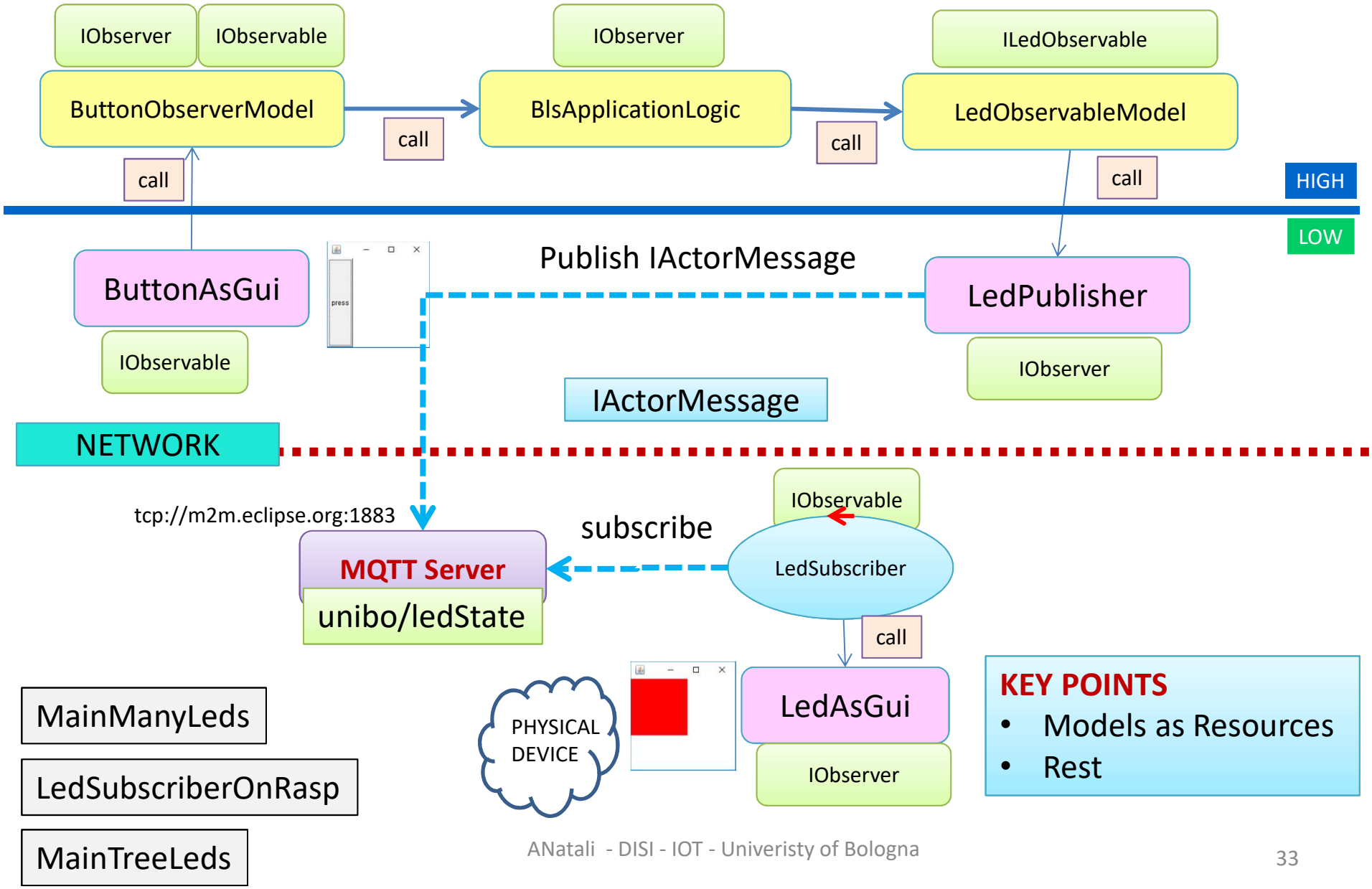
- Images (docker pull …)
  - docker images
- Container (a runnable instance of an image)
  - docker run -p 8484 -a stdin -a stdout -i -t --name natdocker node:nat /bin/bash
  - docker ps
  - docker start /stop --container

docker run -ti -p 1883:1883 -p 9001:9001 eclipse-mosquitto
Run for MQTT + websocket:
docker run -d -p 1883:1883 -p 9001:9001 --name=mosquitto sourceperl/mosquitto

docker ps -a
docker start d8a5f1fefc5b
docker ps
docker exec -it d8a5f1fefc5b /bin/bash wizardly_ride

Activate a MQTT server on localhost:1883 (Docker images)
docker run -ti -p 1883:1883 -p 9001:9001 eclipse-mosquitto

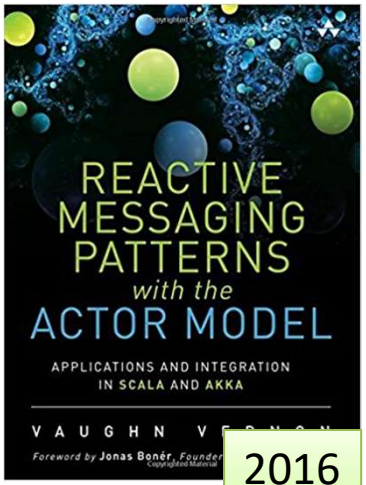
it.unibo.bls18

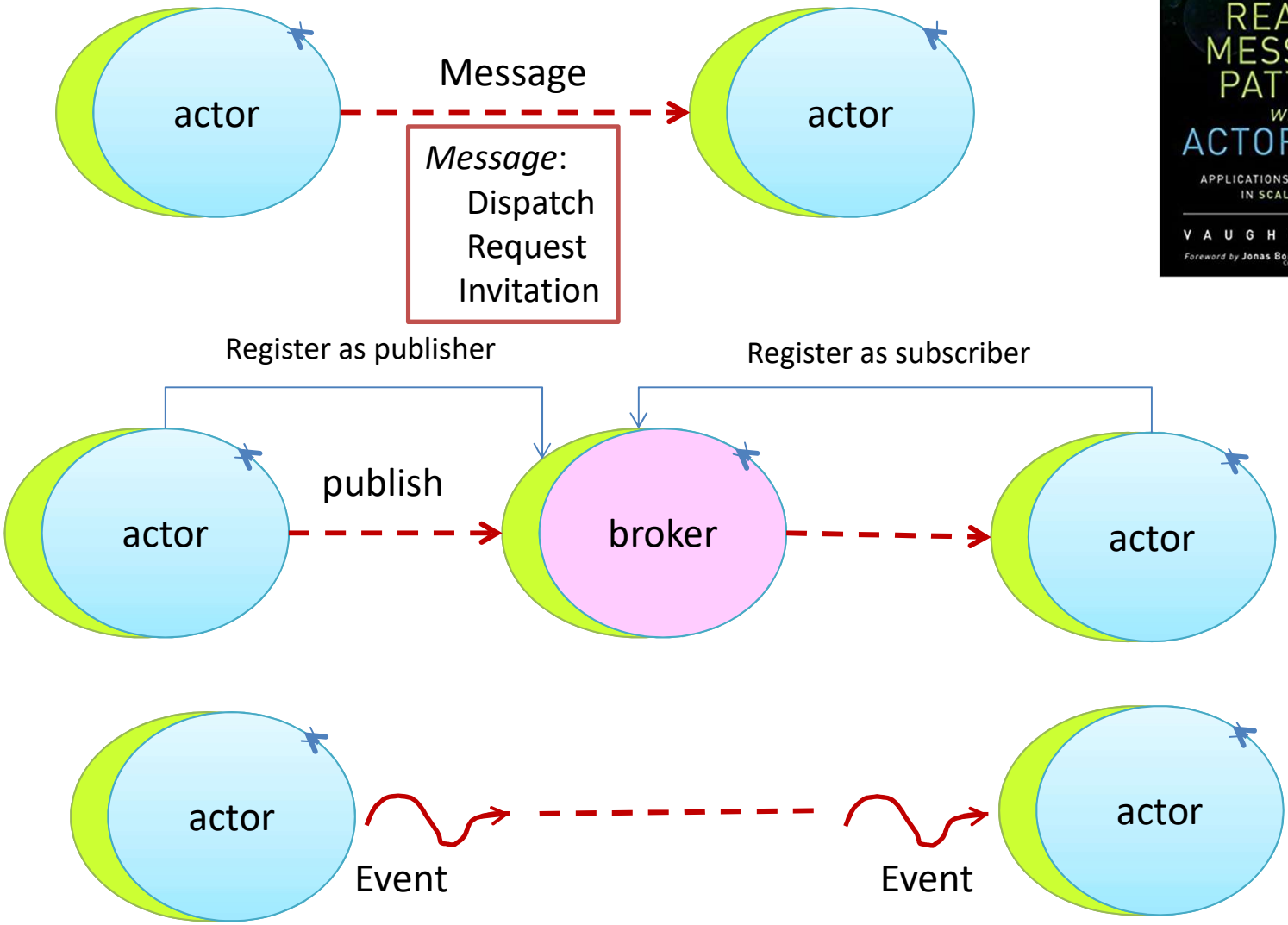


KEY POINTS at the end of phase1

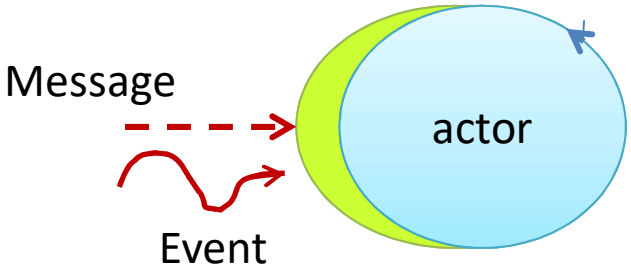
- SISTEMI (DISTRIBUITI ETEREOGENEI
- ARCHITETTURE – DESIGN PATTERN
- STANDARD DI INTERAZIONE/COMUNICAZIONE:
  - Messaggi (dispatch, invitation, request, …)
  - Eventi (messaggi ‘senza destinatario’???)
  - Protocolli P2P (TCP, UDP, CoAP, HTTP, …) o publish/subscribe
  - Payload (vocabolari)
  - M2M, Man2M, M2Man (ManToMan)
- SCHEMI DI COMPORTAMENTO
  - Message-Event BASED (**FSM**)
  - Message-Event DRIVEN

A new paradigm





Behaviour



The actor can handle messages/events:

1) as a FSM
2) as a reactor

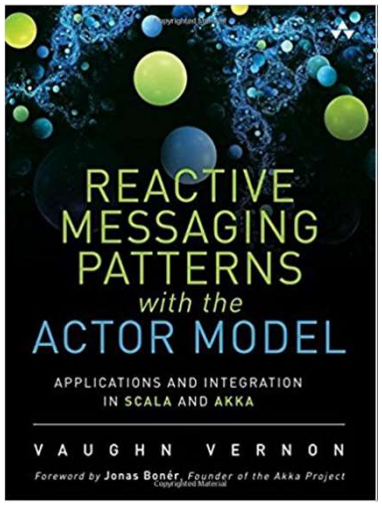
Events can be emitted by *asynchronous operations*

write on a file
send a request
…

Strategic Information Systems

SIS

Software produces business competitive advantage (*strategic results*).



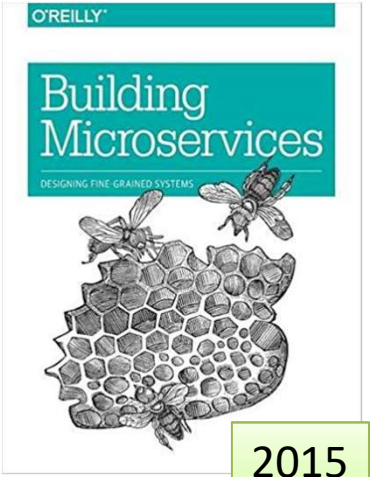
The Actor model allows us to design software models that are:

- responsive, resilient, elastic,message-driven
- more understandable
- able to employ both sequential and functional programming techniques in a highly dynamic environemt
- able to abstractely increase the distributiviy of computation as it evolves, by dynamically creating new actors
- integrable with existing systems

- Beyond performance and scalability, we need to reflect the mental model of business.
- The Actor model allows us to intersect with the DDD-based approach.

Microservices

- **Microservices** is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services: services should be fine-grained and the protocols should be lightweight.
- Microservices emerged from: **domain-driven design, conitnuos delivery, on-demand virtualization, infrastructure automation, small autonomous teams, systems at scale**.
- Microservices improve modularity and makes the application easier to understand, develop and test. It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.It also allows the architecture of an individual service to emerge through continuous refactoring. Microservices-based architectures enable continuous delivery and deployment.

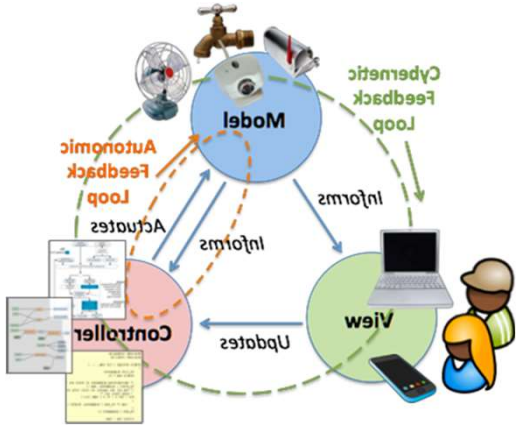


2015

Distributed systems have become more fine-grained in the past 10 years, shifting from code-heavy monolithic applications to smaller, self-contained microservices.
With lots of examples and practical advice, this book takes a **holistic view** of the topics that system architects and administrators must consider when building, managing, and evolving **microservice architectures**.

IL SISTEMA BLS (IOT minimal)

- **DOPO** avere costruito il sistema:
  - Cosa è un LED? Cosa è un BUTTON?
  - Cosa fa il sistema?
  - Come fa un utente a sapere quello che fa?
  - IL PATTERN MVC

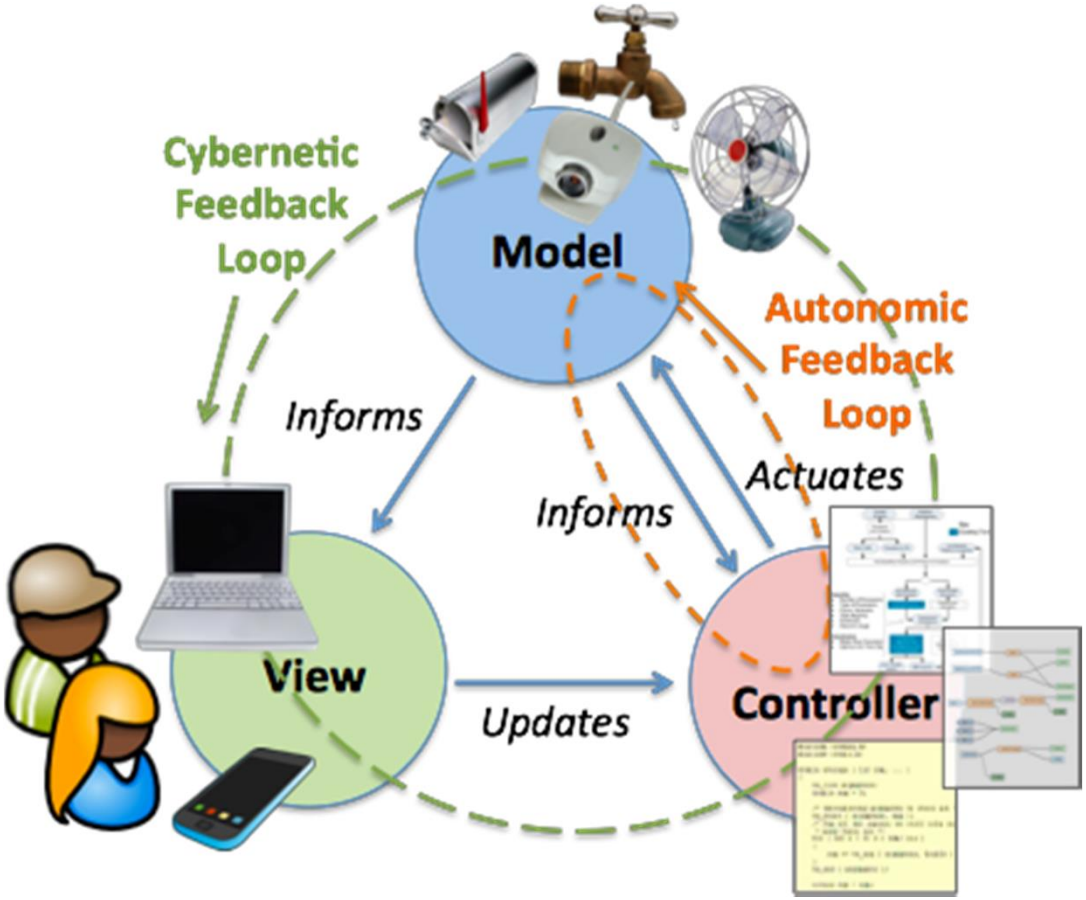


Il pattern MVC

Una guida per molte soluzioni vincenti

MVC

The **Model-VIew-Controller macro pattern** provides a framework for the structured division of responsibility between people and software in IoT applications. It also provides a framework for high level interoperability between data sources, control elements, and UI elements.



The **Model** is a representation or an abstraction of the physical things and their attributes, which *informs* a Controller.

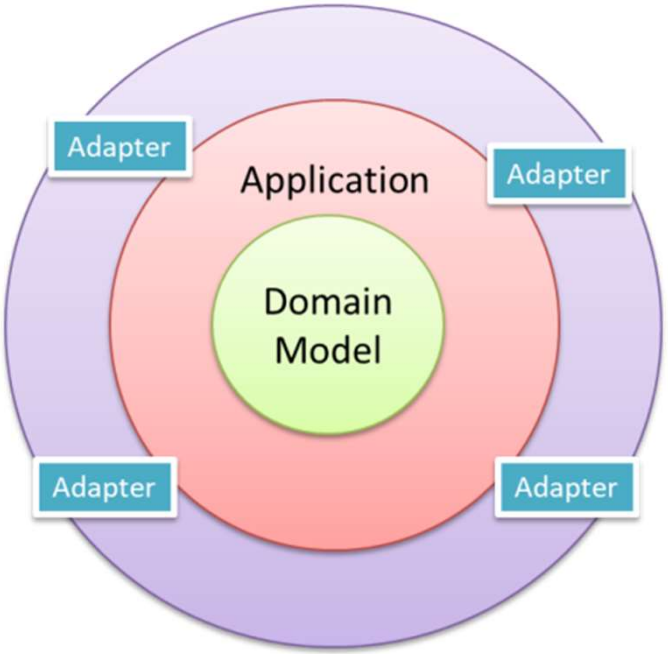
The **Controller** is software which makes *actuation* decisions based on the information, and sends actuation commands to the thing using it's modeled affordances.

The software goal is to maintain a desired state of the thing through it's model.

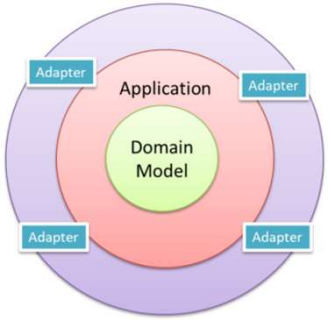
SISTEMI: una visione ‘olistica’

- VISTE DI UN SISTEMA
  - Dal di fuori (cosa fa)
  - Dal di dentro (come è fatto)
- ARCHITETTURA ESAGONALE

The connection between the *inside* and the *outside* part of the system is realized via abstractions called *ports* and their implementation counterparts called *adapters*.

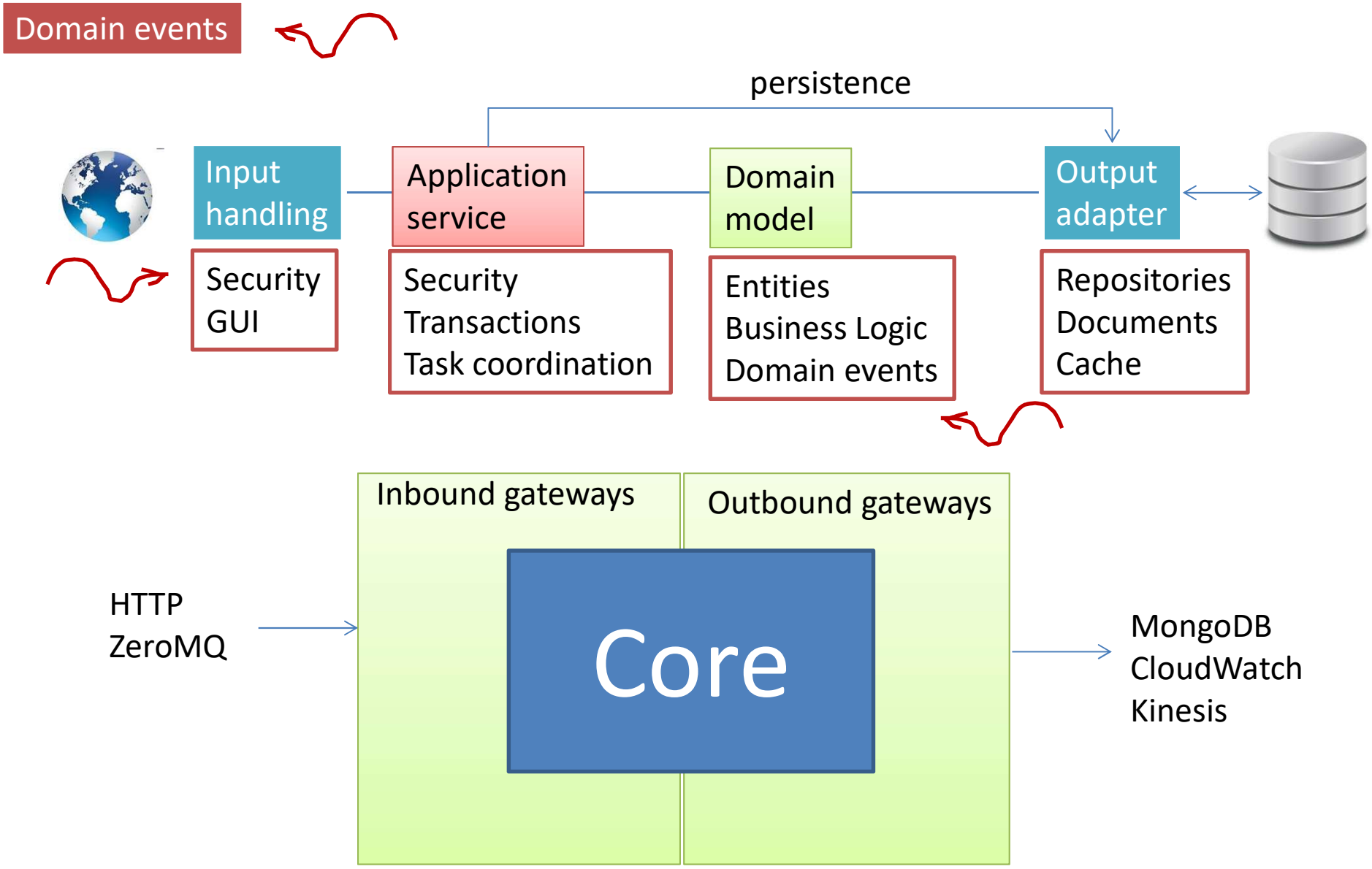


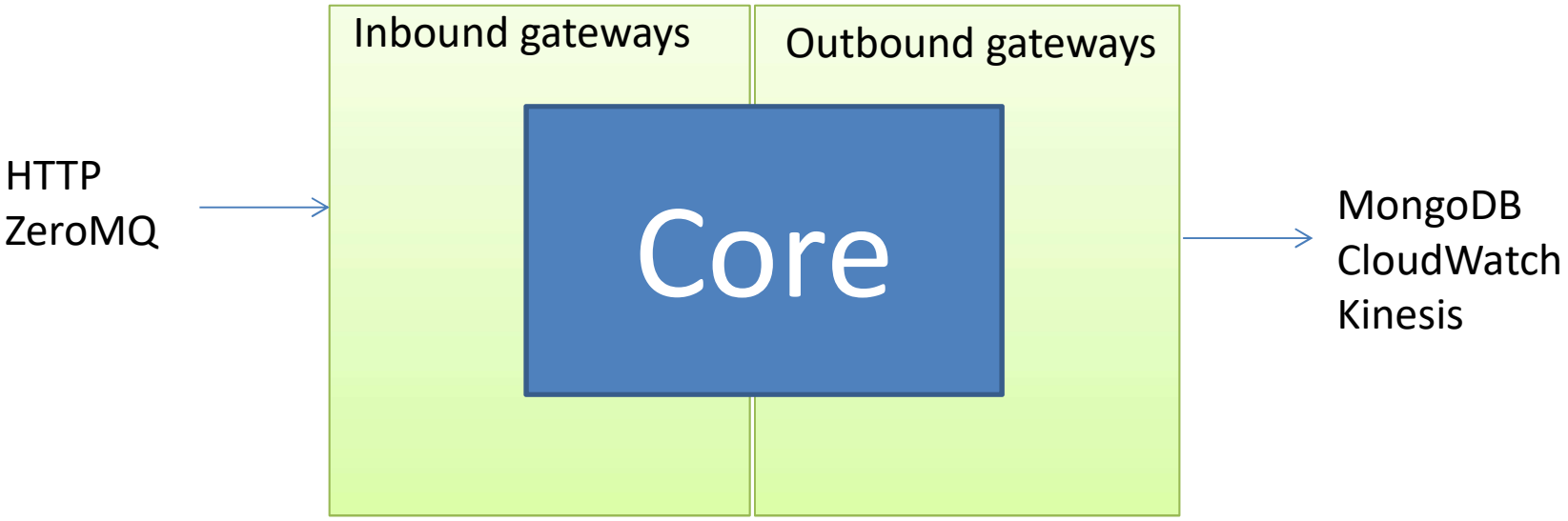
Hexagonal Architecture (Port-Adapter)



Alistair Cockburn

- Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.
- As events arrive from the outside world at a port, a technology-specific adapter converts it into a usable procedure call or message and passes it to the application. The application is blissfully ignorant of the nature of the input device.
- When the application has something to send out, it sends it out through a port to an adapter, which creates the appropriate signals needed by the receiving technology (human or automated).
- The application has a semantically sound interaction with the adapters on all sides of it, without actually knowing the nature of the things on the other side of the adapters.


Domain events
persistence
Input handling
Application service
Domain model
Output adapter
Security
GUI
Security
Transactions
Task coordination
Entities
Business Logic
Domain events
Repositories
Documents
Cache
Inbound gateways
Outbound gateways
HTTP
ZeroMQ
Core
MongoDB
CloudWatch
Kinesis



https://dzone.com/articles/hexagonal-architecture-it-works

Core components never violate the "only leave the process via a gateway" rule. All out-of-process access (including logging) must flow through a gateway.

Simplifying testing is the primary benefit with the ability to swap out both inbound and outbound integration points a close second.

Implement inter-service gateways using ZeroMQ which is a messaging system that can operate both in-memory and distributed.
In theory, you could move a previously in-process service into its own microservice with only minor changes to the ZeroMQ configuration.

Accessi REST al modello

Gli standard pagano …

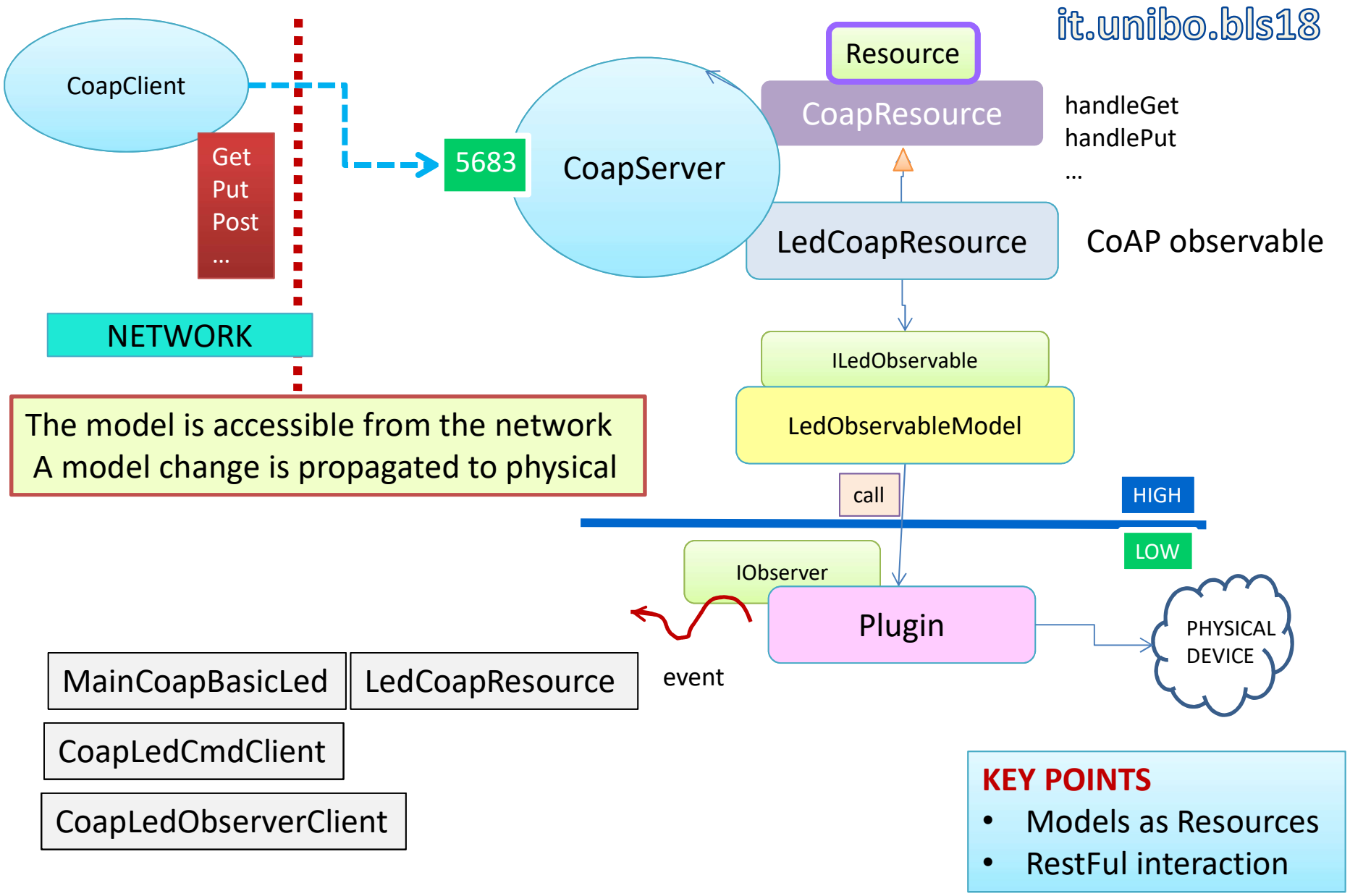
Req2: accedere a un Led via web/REST

REST = **RE**presentational **S**tate **T**ransfer: is an *architectural style*

1. Definiamo il modello come CoAp-resource osservabile
2. Inseriamo la CoAp-resource entro un CoAp-server (port 5683)
3. La CoAp-resource usa come observer il LedObservableModel già definito
4. Accediamo alla CoAp-resource attraverso un CoAp-client osservatore (**CoapLedObserverClient**) che invia GET e PUT
5. Modifichiamo il LedObservableModel attraverso un sotto-sistema. Le modifiche verranno osservate dalla CoAp-resource che aggiorna tutti i client osservatori
6. Osserviamo le modifiche della CoAp-resource lato client

Coap Californium

- A CoAP server hosts a tree of Resources which are exposed to clients by means of one or more Endpoints which are bound to a network interface.
- CoapResource is a basic implementation of a resource.
- CoapResource uses four distinct methods to handle requests: handleGET(), handlePOST(), handlePUT() and handleDELETE().
- Each resource is allowed to define its own executor.
- CoapResource supports CoAP's *observe mechanism*. Enable a CoapResource to be observable by a CoAP client by marking it as observable with **setObservable(boolean**).
- The class **ResourceObserver** has nothing to do with CoAP's observe mechanism but is an implementation of the general observe-pattern.


it.unibo.bls18
CoapClient
Get
Put
Post
...
5683
CoapServer
Resource
CoapResource
handleGet
handlePut
...
LedCoapResource
CoAP observable
NETWORK
ILedObservable
LedObservableModel
call
HIGH
LOW
IObserver
Plugin
PHYSICAL DEVICE
event
The model is accessible from the network
A model change is propagated to physical
MainCoapBasicLed
LedCoapResource
CoapLedCmdClient
CoapLedObserverClient
KEY POINTS
• Models as Resources
• RestFul interaction

Inherits or use? → The resource USES the model

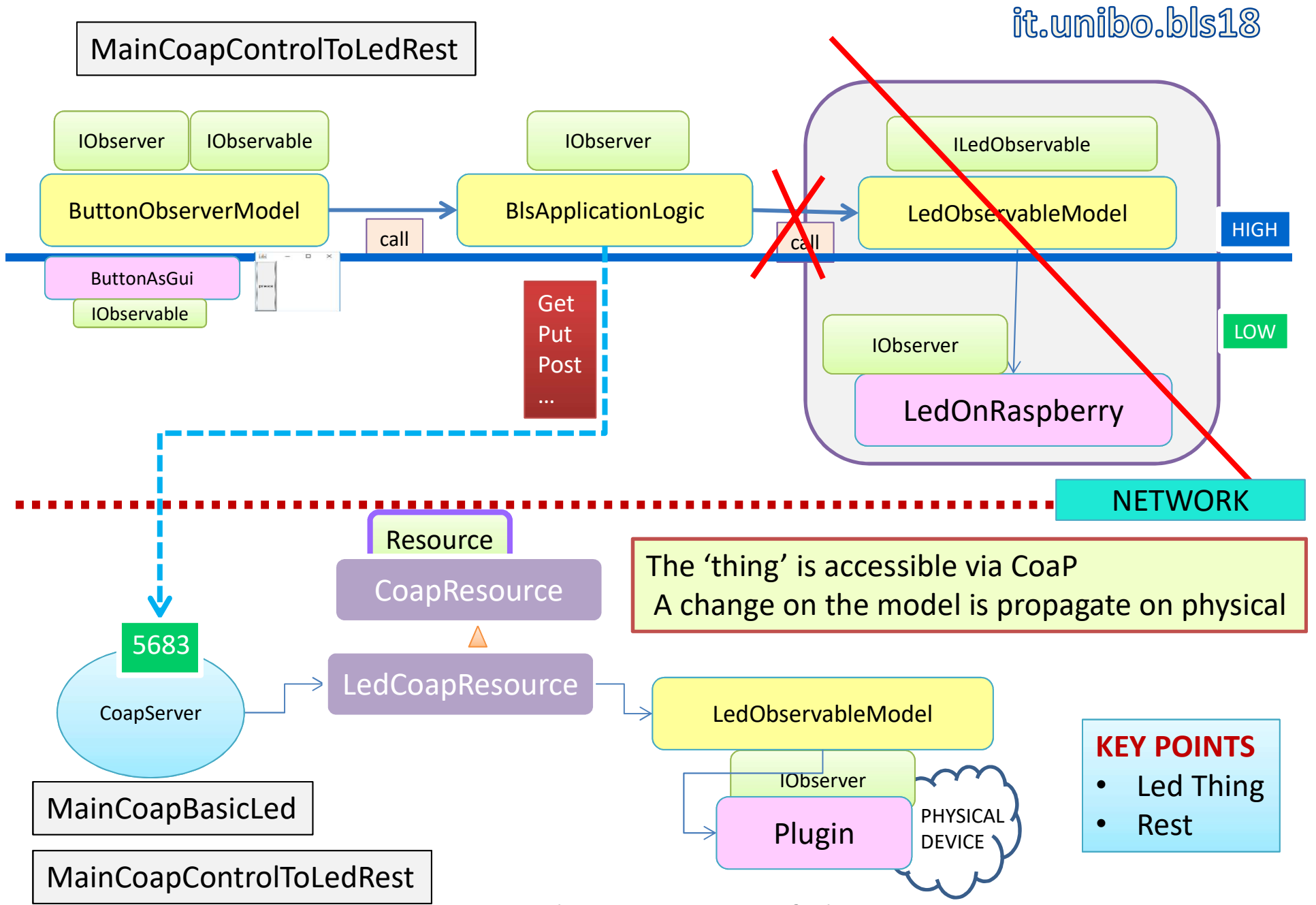
```
public class LedCoapResource extends CoapResource implements IObserver{
privateILedObservable ledModel ;

public LedCoapResource(String name, ILedObservable model ) {
            super(name);
            ledModel = model;
}
  @Override //CoapResource
   public void handleGET(CoapExchange exchange) {
       exchange.respond( ResponseCode.CONTENT, getValue(), MediaTypeRegistry.TEXT_PLAIN) ;
   }
   @Override //CoapResource
   public  void handlePUT(CoapExchange exchange) {
       try {
      String value = exchange.getRequestText();//new String(payload, "UTF-8");
         if( value.equals("switch"))  switchValue();   else setValue(value);
         exchange.respond(CHANGED,  value);
      } catch (Exception e) {
          exchange.respond(BAD_REQUEST, "Invalid String");
      }
   }
```

```
protected void setValue(String v) {
  if( v.equals("true")) ledModel.turnOn();
  else ledModel.turnOff();
}
```

La disponibilità del modello come CoAp-resource ci permette di definire una '**LedThing**' accessibile via rete (**MainCoapBasicLed, MainLedCoapOnRaspberry**) e

1. di definire la business logic cone CoAp-client del modello (**BIsApplicationLogicCoap**)

1. di definire un server HTTP (**serverHttpToCoap.js**) che rende visibile via Web lo stato corrente della **LedThing**


it.unibo.bls18
MainCoapControlToLedRest
IObserver
IObservable
ButtonObserverModel
ButtonAsGui
IObservable
call
IObserver
BlsApplicationLogic
Get
Put
Post
...
call
ILedObservable
LedObservableModel
HIGH
LOW
IObserver
LedOnRaspberry
NETWORK
Resource
CoapResource
The 'thing' is accessible via CoaP
A change on the model is propagate on physical
5683
CoapServer
LedCoapResource
LedObservableModel
IObserver
Plugin
PHYSICAL
DEVICE
KEY POINTS
• Led Thing
• Rest
MainCoapBasicLed
MainCoapControlToLedRest

Il WEB

(From) Lamp (to) Mean

Req3: accedere a un Led mediante una pagina WEB e visualizzare lo stato del Led nella pagina

LAMP is an archetypal model of web service stacks, named as an acronym of the names of its original four open-source components: the GNU/Linux operating system, the Apache HTTP Server, the MySQL relational database management system (RDBMS), and the PHP programming language.

MEAN is a free and open-source JavaScript software stack for building dynamic web sites and web applications.
The MEAN stack is MongoDB, Express.js, AngularJS (or Angular), and Node.js. Because all components of the MEAN stack support programs are written in JavaScript, MEAN applications can be written in one language for both server-side and client-side execution environments.

MainCoapControlToLedRest

2

IObserver IObservable

ButtonObserverModel

call

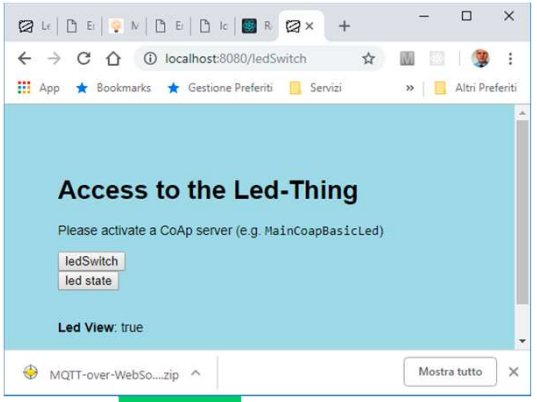
IObserver

BlsApplicationLogicCoap

ButtonAsGui

IObservable

Get
Put
Post
...



8080

3

serverHttpToCoapMqttNoMongo

subscribe

5683

CoapServer

LedCoapResource

publish true / false

unibo/qasys

MQTT Server

MainCoapBasicLed

1

LedObservableModel

IObserver

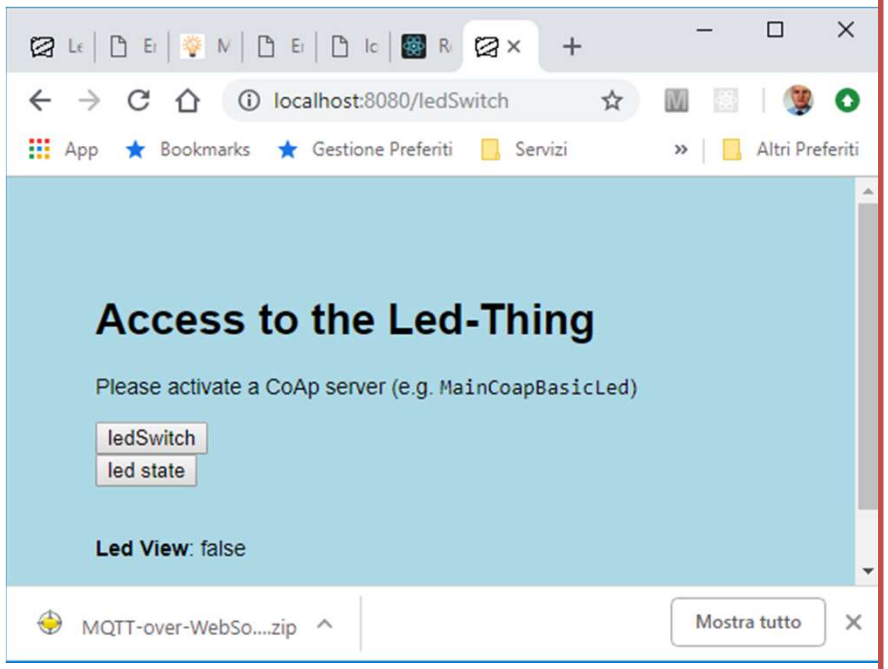
LedAsGui

KEY POINTS

- Front end server
- JavaScript
- WebSockets

curl -X GET http://localhost:8080/Led
curl -X PUT -d "true" http://localhost:8080/ledSwitch

1. **MainCoapBasicLed**: rende disponibile il Led come risorsa Coap (LedThing) inserendo in un server Coap la risorsa LedCoapResource
2. **MainCoapControlToLedRest**: crea un (sotto)sistema che modifica la LedThing usando un ButtonGui
3. **serverHttpToCoap.js**: fornisce un front-end web alla LedThing che
   - Offre all'utente pulsanti per modificare/ispezionare la LedThing
   - Offre una vista aggiornata del valore corrente della LedThing usando una WebSocket

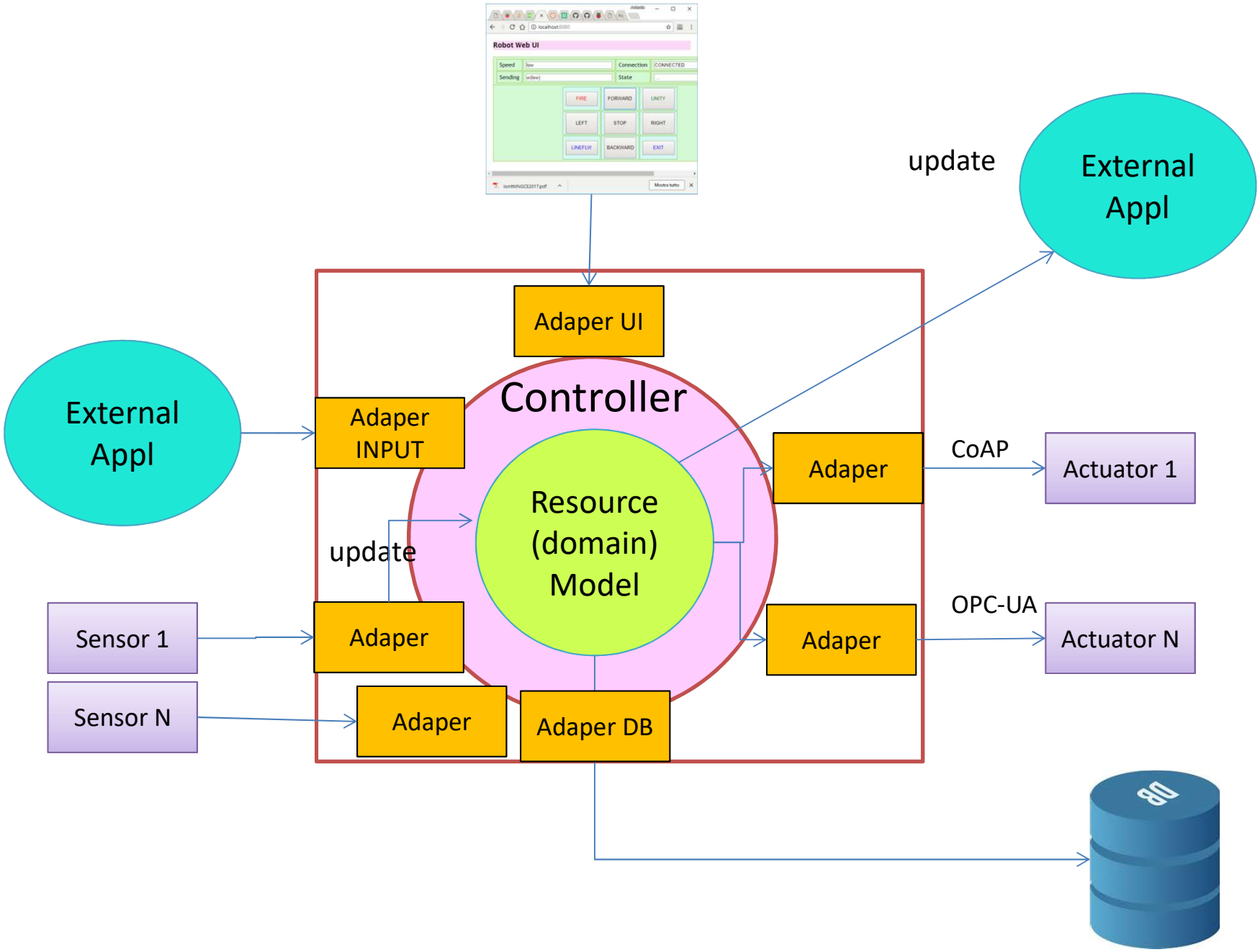


LedCoapResource

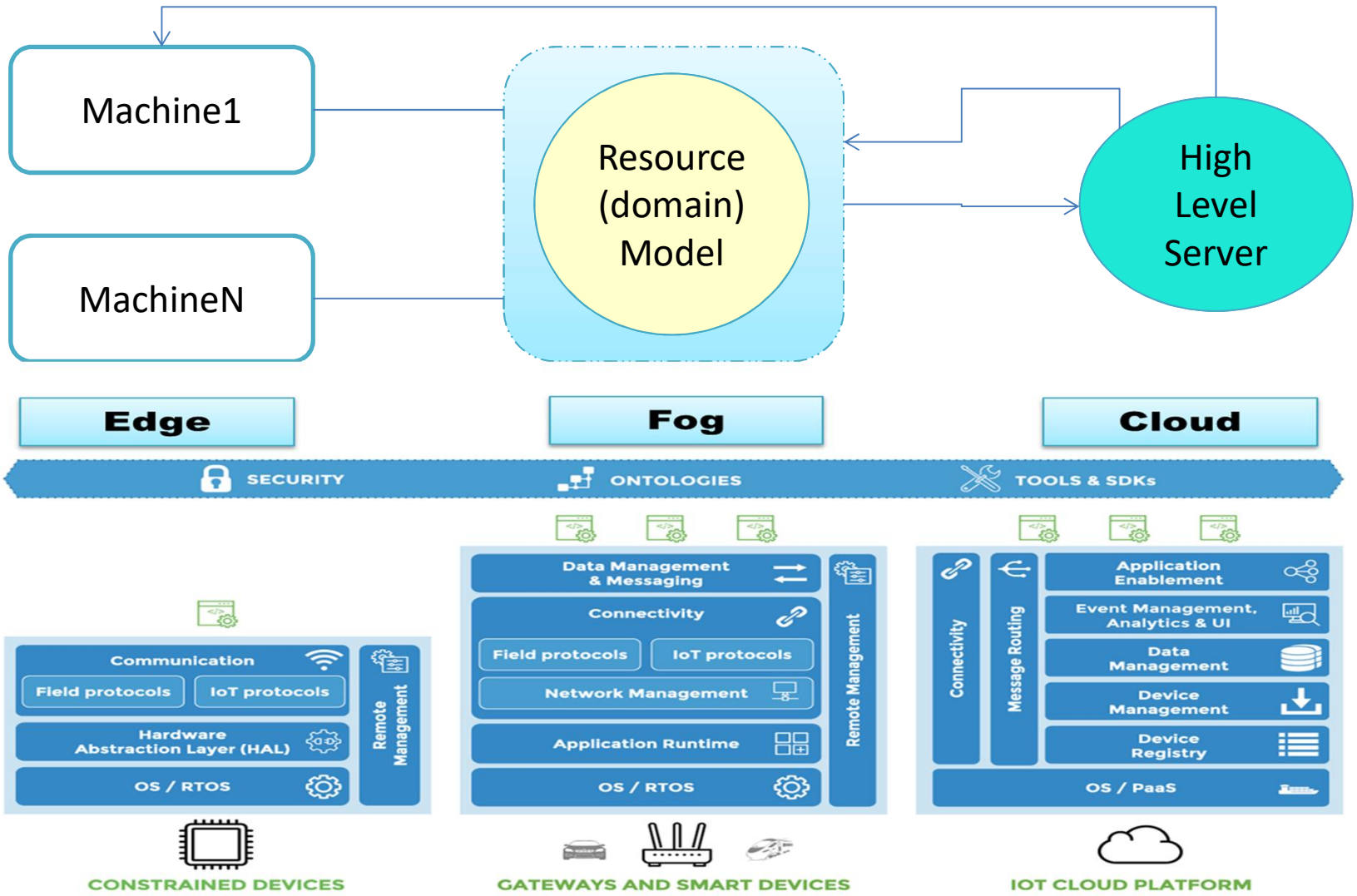
1. Risponde alle richieste GET/PUT per url=/Led
2. modifica lo stato del Led
   - come observer di ledmodel
   - in risposta a una PUT
3. agisce come observable
   - rispetto al led concreto
   - rispetto ai clienti Coap
4. Pubblica via MQTT le modifiche di stato sulla topic **unibo/qasys.** Queste sono ricevute dal server che aggiorna le pagine HTML via WebSocket

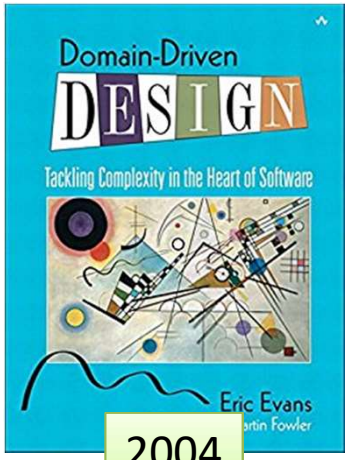
Models as (domain) resources

Towards Domain-Driven design


Robot Web UI
update
External Appl
Adaper UI
External Appl
Controller
Adaper INPUT
Resource (domain) Model
Adaper
CoAP
Actuator 1
update
Sensor 1
Adaper
Adaper
OPC-UA
Actuator N
Sensor N
Adaper
Adaper DB

The business logic





2004

Software developer ↔ Designer ↔ Domain expert

DDD

- Developers are insulated from the domain experts. If a developer does not understand a concept, it is likely the implementation will not accurately reflect the domain.
- Developers without solid design principles will produce a code that is hard to understand or change – the oppostive of agility. (pg. xxiij)

Domain model: a rigorously organized and selective abstraction of the knowledge in a domain's expert head (pg. 3).
One model should underlie implementation, design and team communications (pg. 41)
The model is the backbone of a language used by all team members (pg.4, 26).

- iterate a single model to reflect a shared understanding across domain experts, designers and developers
- establisha common language, i.e. a UBIQUITOUS LANGUAGE (pg. 24)

Code as a design document does have its limits (pg. 38). A document should explain the concepts of the model and must be involved in project activities (pg.39)

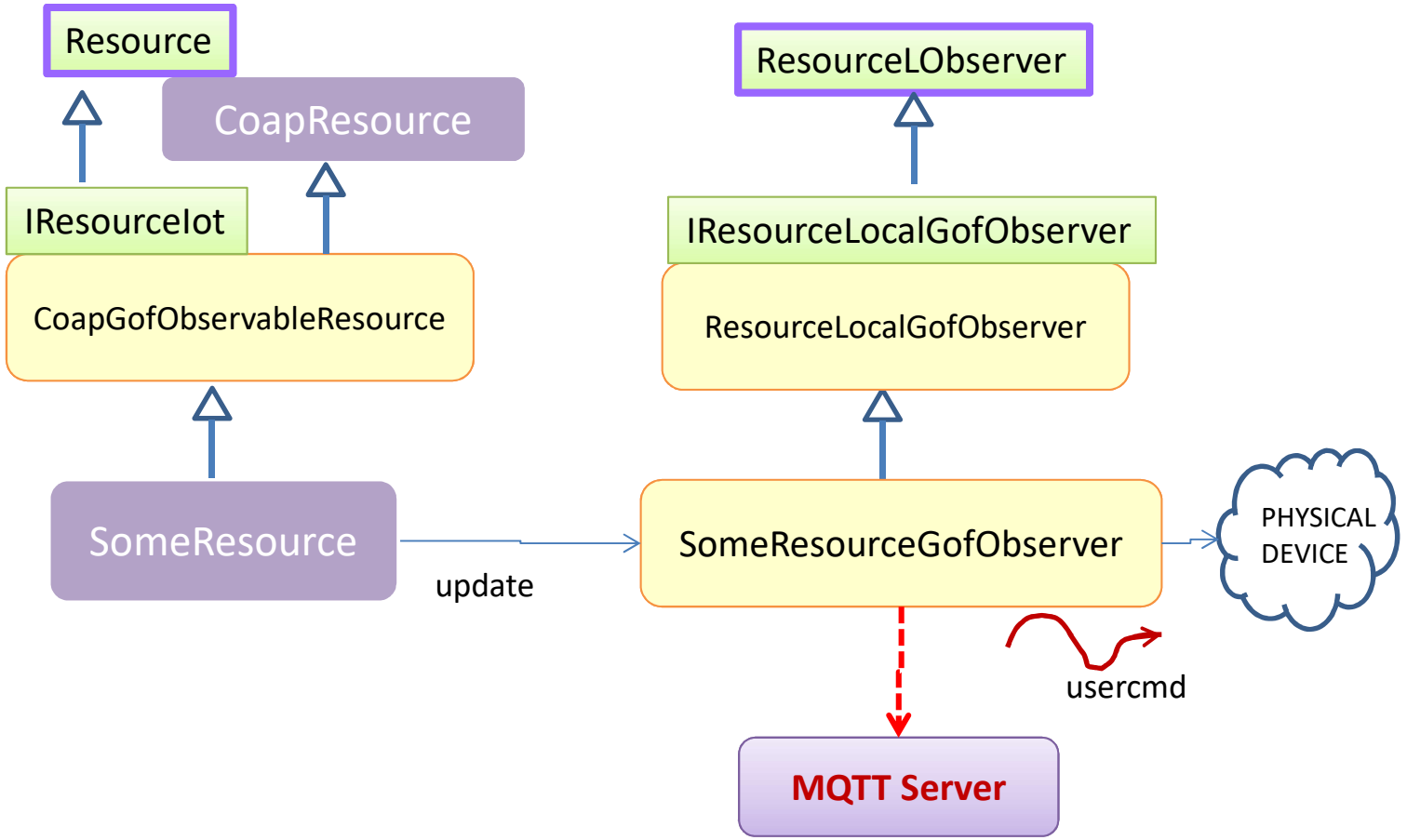
Effective domain modelers are knowledge crunchers. Continuous learning takes place between domain experts, designers and developers (pg. 15)
(OO) MODEL-DRIVEN DESIGN pg. 47

Inherits or use? → The resource IS the model

```
public  class LedResource extends CoapGofObservableResource {
public static final String resourcePath = "led";
private String value = "false";
   public LedResource() {
      super(resourcePath);
   }
@Override //CoapResource
   public void handleGET(CoapExchange exchange) {
                exchange.respond( value );
   }
 @Override //CoapResource
   public  void handlePUT(CoapExchange exchange) {
   try {
         value = exchange.getRequestText();
          setValue(value);
          exchange.respond(CHANGED,  value);
      } catch (Exception e) {
           exchange.respond(BAD_REQUEST, "Invalid String");
      }
  }
}
```

```
 @Override // CoapGofObservableResource
public void setValue(String v) {
  value = v ;
  update(value);//notify the GOF observer
}
```

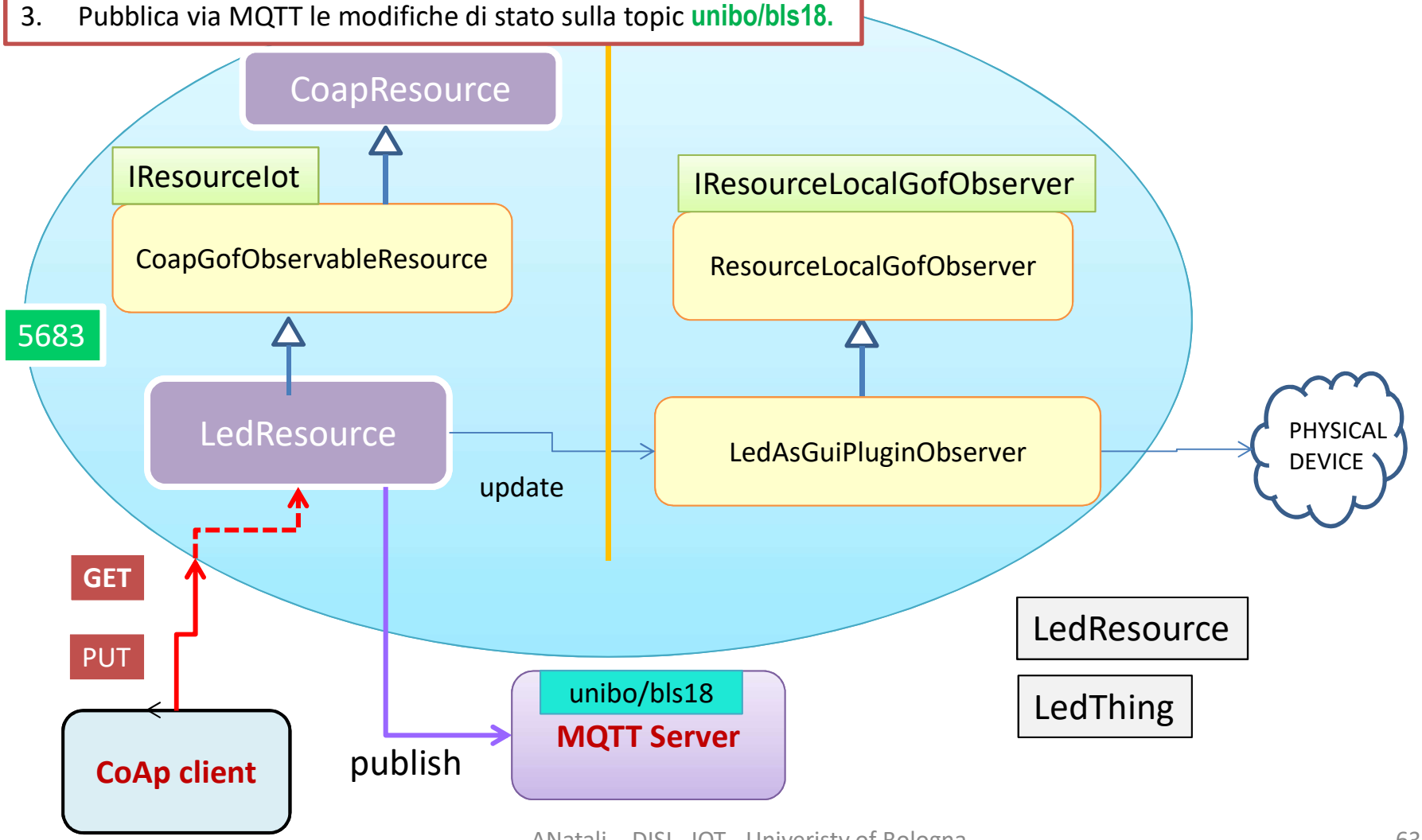
Observable (CoAP) resource



Led Thing

LedResource

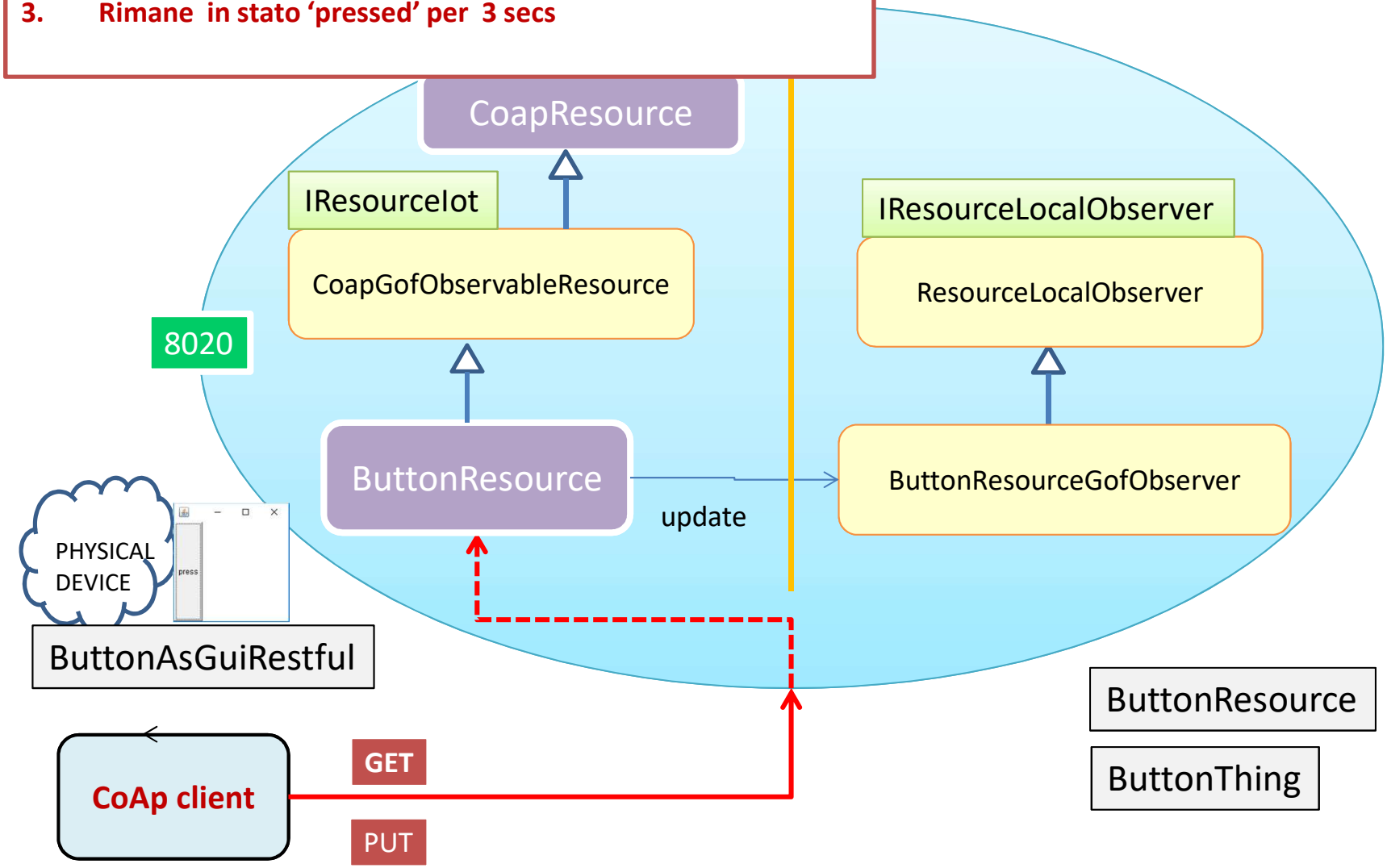
1. Risponde alle richieste GET/PUT per url=/Led
1. modifica lo stato del Led in risposta a una PUT
2. agisce come observable
   - rispetto al led concreto
   - rispetto ai clienti Coap
3. Pubblica via MQTT le modifiche di stato sulla topic **unibo/bls18.**



Button Thing

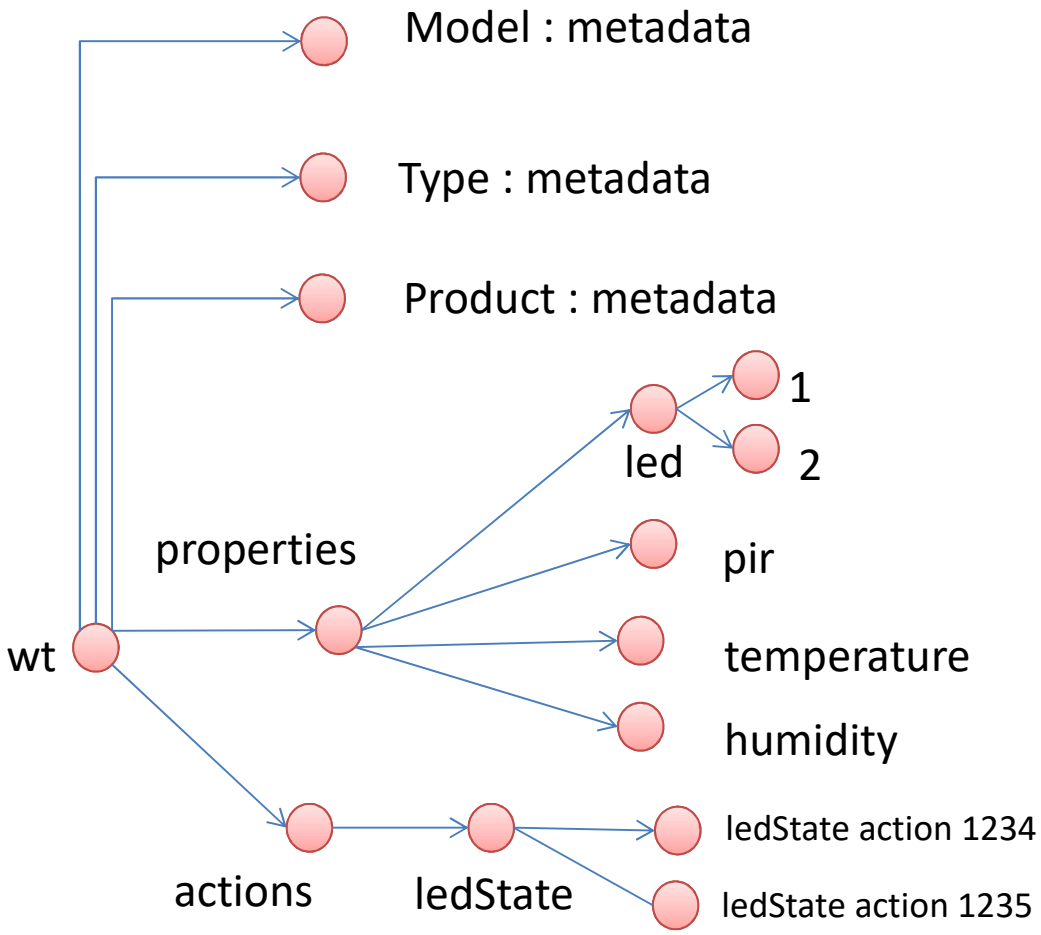
ButtonResource

1. Risponde alle richieste GET/PUT per url=/Button
2. agisce come observable
   - rispetto ai clienti Coap
   - rispetto a osservatori GOF.
3. **Rimane in stato 'pressed' per 3 secs**

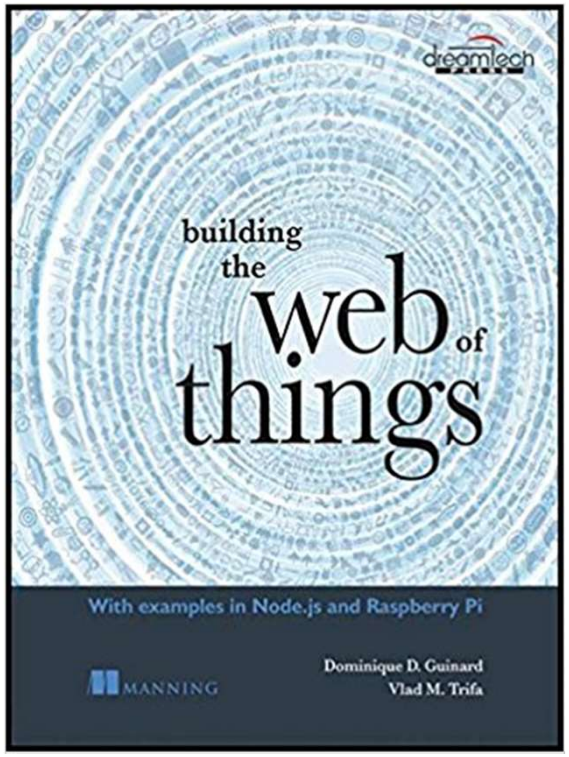


The Web of Things

Web Thing Model



http://gateway.webofthings.io/properties/pir



http://model.webofthings.io W3C

Terminology

- **WT:** Web Thing
- **Model**: metadata that defines various aspects of a WT, such as name, description, or configurations.
- **Properties**: represent the internal state of a WT. A property is a collection of data values that relate to some aspect of the WT. Properties can be modified through actions.
- **Actions**: functions offered by a WT (e.g. 'open','close','enable','scan',...).
- **ExternalThing**: a (Application Layer) gateway to other devices that don't have an internet connection.

MVC-based servers providing an API

A **model** is the representation of a specific category of data, or entity, within the application. A Model should not know about the rest of the application.

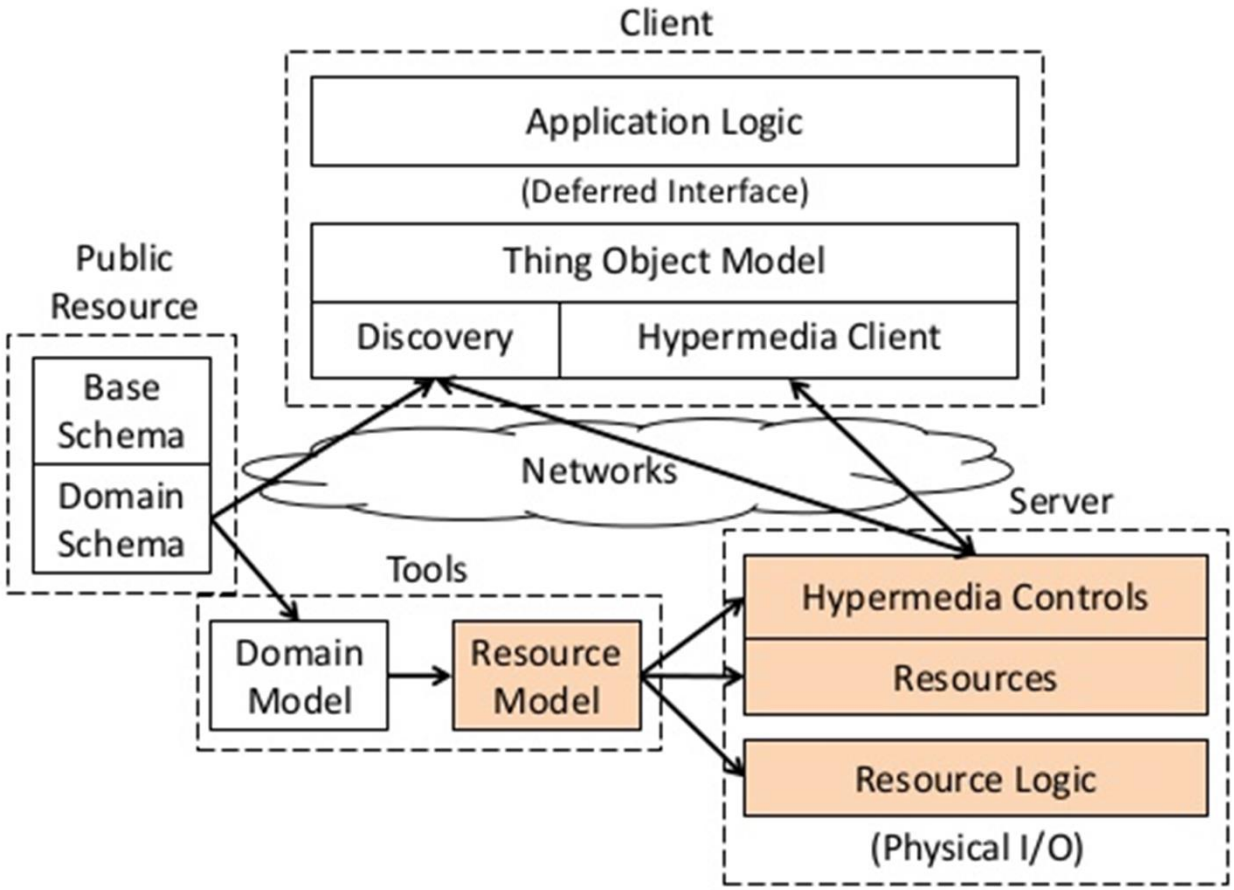
The **View** should contain logic for rendering structured data. The View should not have knowledge of the rest of the application. A view engine (EJS, PUG) can be defined as a module that does the actual rendering of views.

The **Controller** uses the Models and Views to accept and handle inputs and commands from the user.

MVC server handles requests by delegating the work to proper controllers. It may be a good idea to extract the business logic out to a service class and treat the controller as a method of accepting and returning (http) requests. The answer wiil be a status code and some (JSON) data.

Controller that access a resources with **RESTFul API** (to perform common CRUD functions (create, read, update, delete) do not depend any more on data representation.

Resource Model



WOT design

- **Resource design**. Identify the functionality or services of a Thing, and organize the hierarchy of these services.
- **Integration strategy**. Choose a pattern to integrate Things to the internet and the web. The main patterns are:
  - **Direct integration pattern** (REST on device). A device can directly connect to the internet, e.g. via Wi-Fi or Ethernet.
  - **Gateway integration pattern**. Resource-constrained devices can use non-web protocols (e.g. ZigBee, Bluetooth) to talk to a more powerful device (the gateway), which then exposes a REST API for those non-web devices.
  - **Cloud integration pattern**. An extension of the gateway pattern where the gateway is a remote server that devices and applications access via the Internet.
- **Representation design**. Decide which representations will be served for each resource.
- **Interface design**. Decide which commands are possible for each service, along with which error codes.
- **Resource linking design**. Decide how the different resources are linked to each other.

WoT

←

Integration patterns: REST on device, Gateway (CoAP), Cloud (MQTT)

Resource model design : (ontology) tree, knowledge base, ..

Representation design : json, **prolog**, java, HTML, MessagePack, …

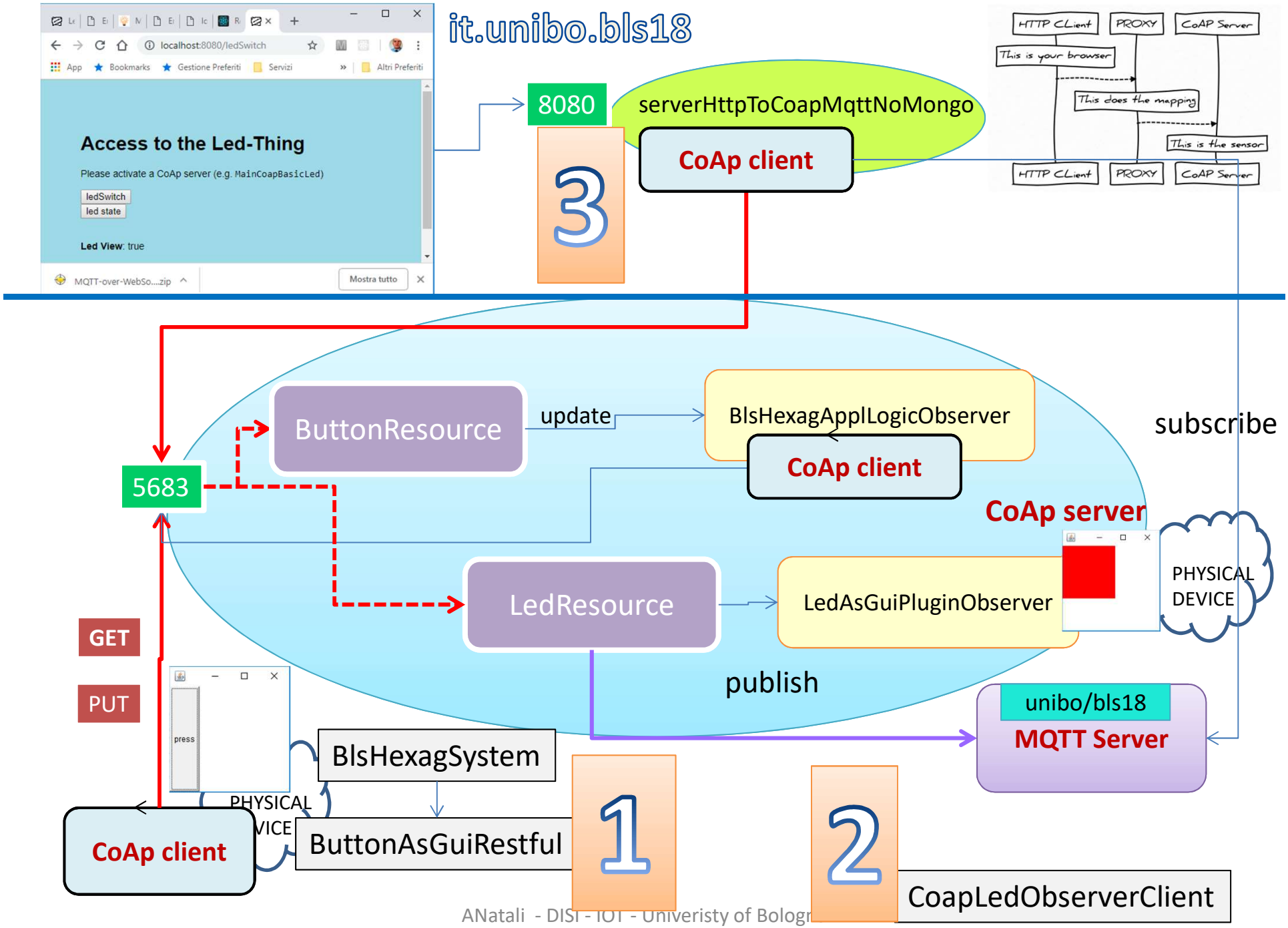
Interface design : GET, PUT, etc

Resourse binding design : HAETOAS (web linking,..), findability, …

HATEOAS (Hypermedia as the Engine of Application State): state changes within an application happen by following links, which meets the self-contained-messages constraint. Links enable clients to discover related resources, either by browsing in the case of a human user following links on pages, or by crawling in the case of a machine.

GOALS

- Define a server in Node
- Provide a RESTful API for resources (data and things)
- Define a sensor plugin to update properties in the resource model
- Define an actuator plugin to observe the model and update properties when changing the state after an action has been executed.

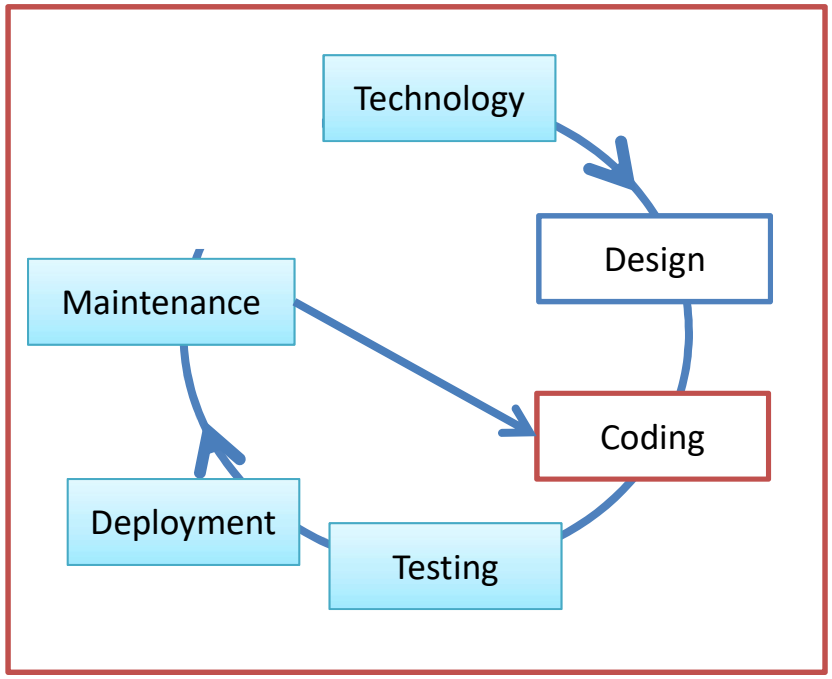

it.unibo.bls18
localhost:8080/ledSwitch
Access to the Led-Thing
Please activate a CoAp server (e.g. MainCoapBasicLed)
ledSwitch
led state
Led View: true
8080
serverHttpToCoapMqttNoMongo
CoAp client
3
HTTP CLient
PROXY
CoAP Server
This is your browser
This does the mapping
This is the sensor
ButtonResource
update
BlsHexagApplLogicObserver
CoAp client
subscribe
5683
CoAp server
LedResource
LedAsGuiPluginObserver
PHYSICAL DEVICE
GET
PUT
publish
unibo/bls18
MQTT Server
press
BlsHexagSystem
PHYSICAL DEVICE
CoAp client
ButtonAsGuiRestful
1
2
CoapLedObserverClient

MODEL DRIVEN SOFTWARE

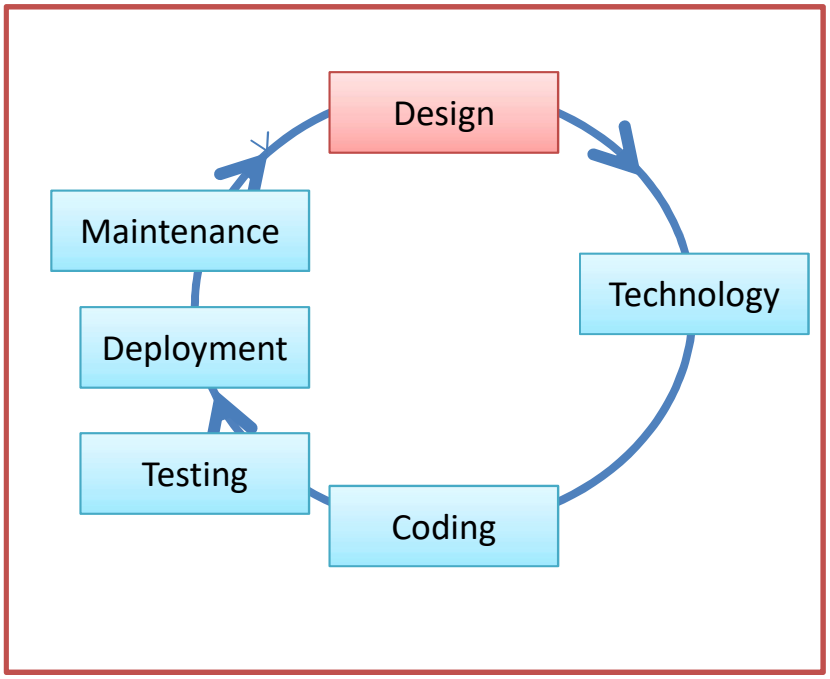
Towards software factories

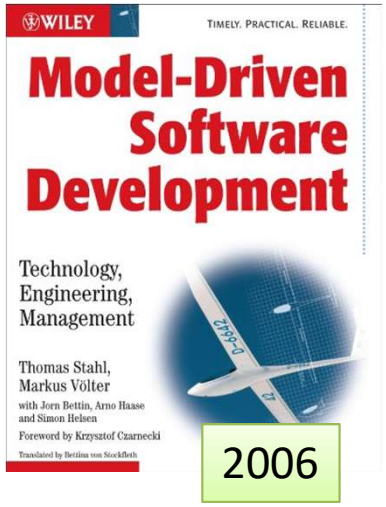
MDSD

Tecnhology-based



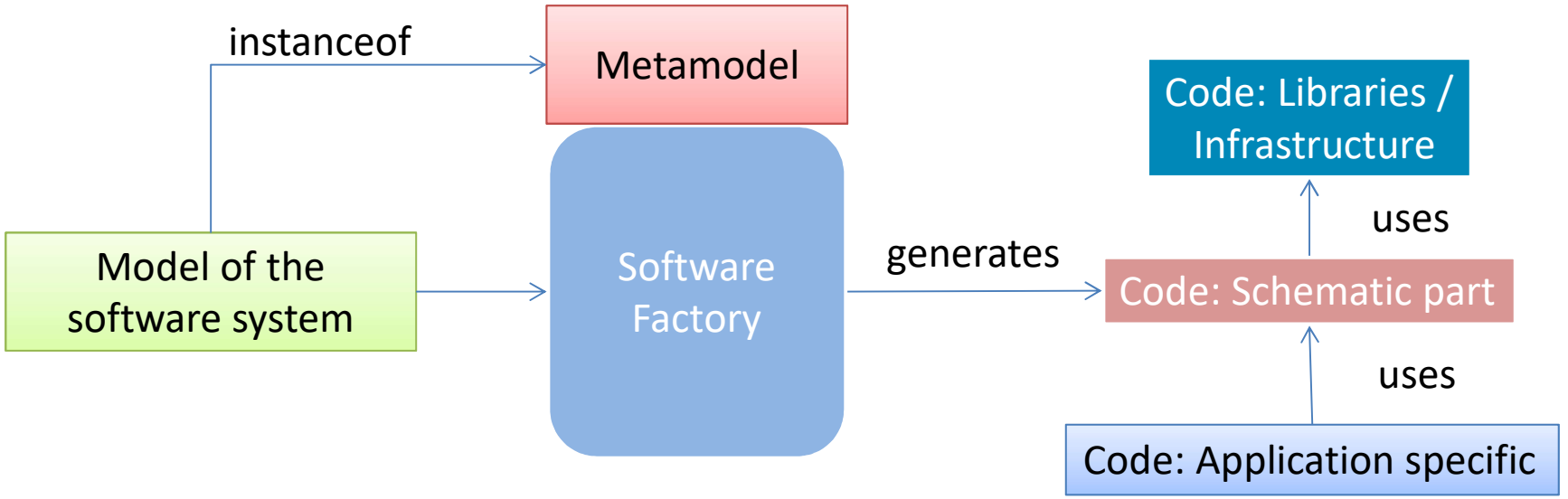
Model-based

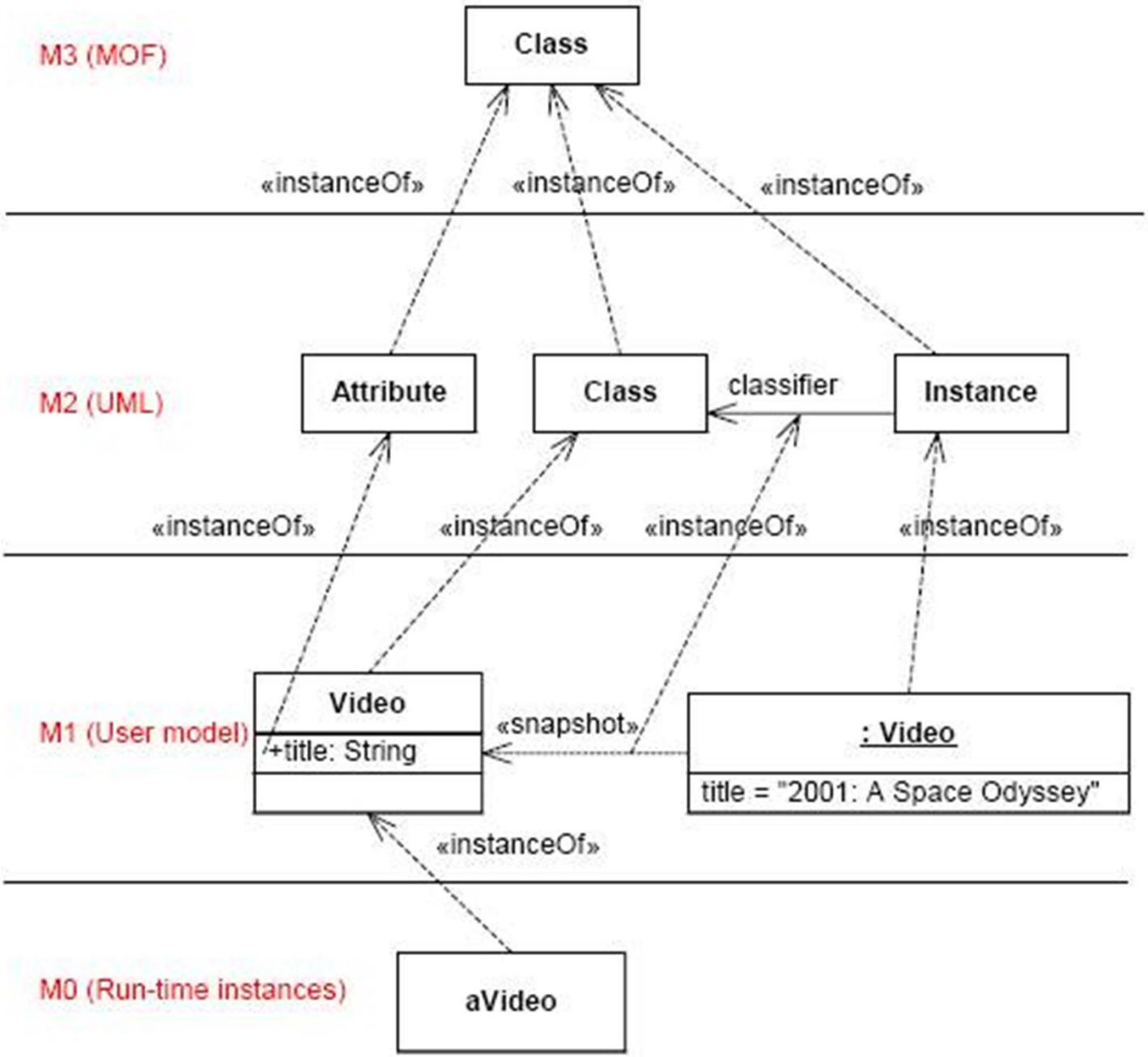




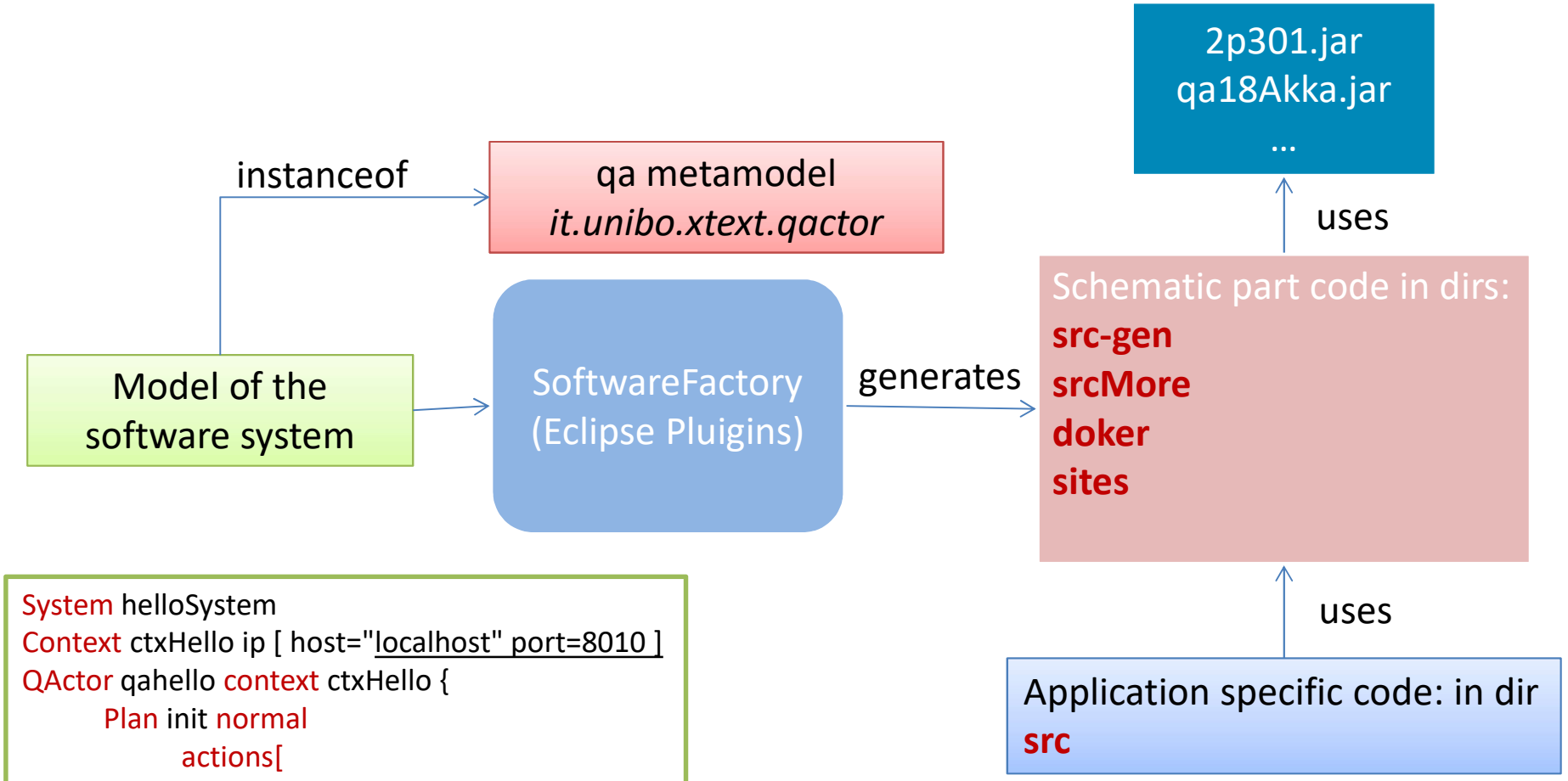
- Model-Driven Software Development (MDSD) puts **analysis and design models** on par with code.
- Models do not constitute documentation, but are considered equal to code, as their implementation is automated.
- The goal of the book is to convince you, the reader, that MDSD is a practicable method today, and that it is superior to conventional development methods in many cases.

MDSD

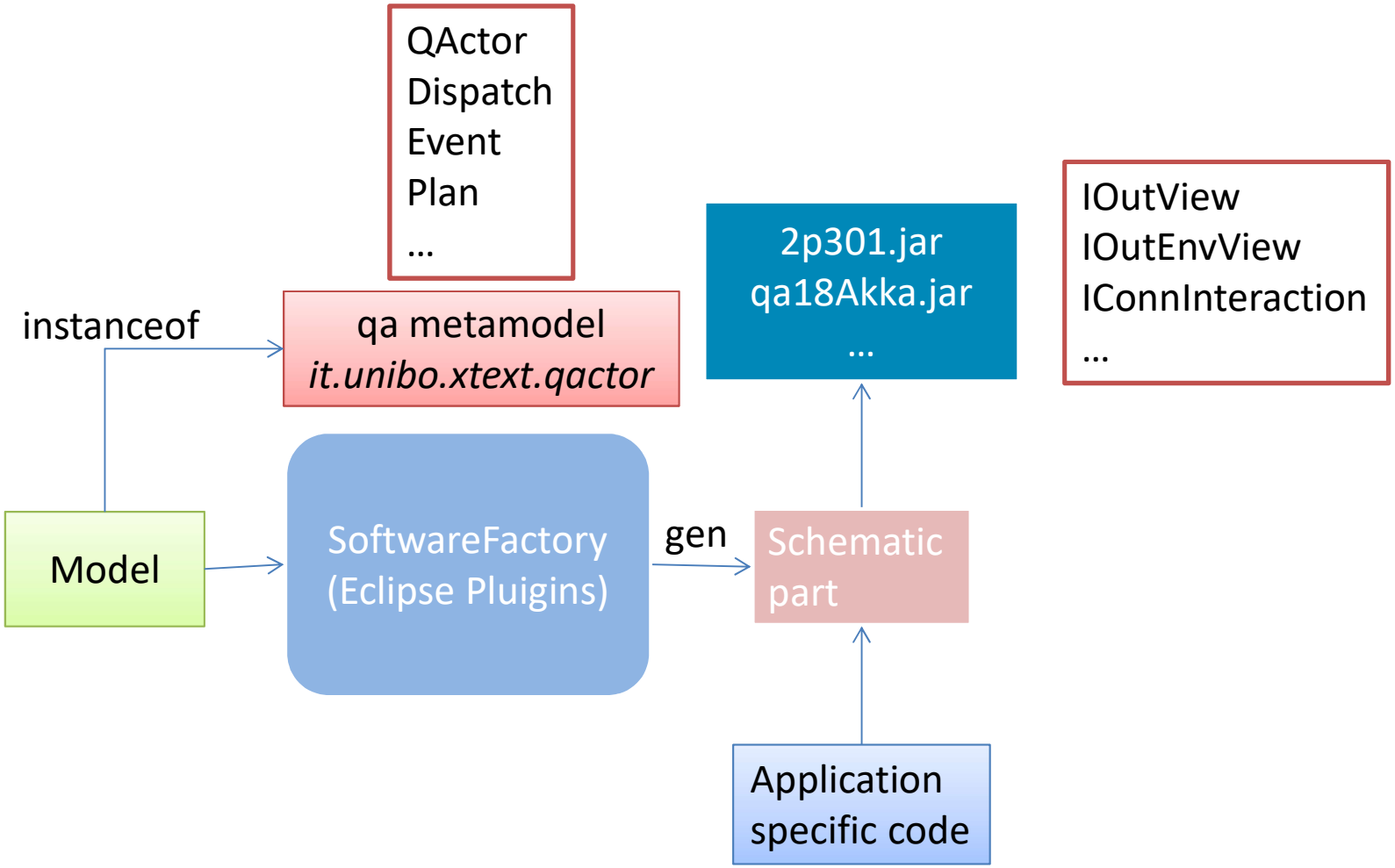

instanceof
Metamodel
Code: Libraries / Infrastructure
uses
Model of the software system
Software Factory
generates
Code: Schematic part
uses
Code: Application specific


M3 (MOF)
Class
«instanceOf»
«instanceOf»
«instanceOf»
M2 (UML)
Attribute
Class
classifier
Instance
«instanceOf»
«instanceOf»
«instanceOf»
«instanceOf»
M1 (User model)
Video
+title: String
«snapshot»
: Video
title = "2001: A Space Odyssey"
«instanceOf»
M0 (Run-time instances)
aVideo

QActors




```
System helloSystem
Context ctxHello ip [ host="localhost" port=8010 ]
QActor qahello context ctxHello {
        Plan init normal
                actions[
                    println("Hello world!!"
                ]
}
```


QActor
Dispatch
Event
Plan
…
2p301.jar
qa18Akka.jar
…
IOutView
IOutEnvView
IConnInteraction
…
instanceof
qa metamodel
it.unibo.xtext.qactor
Model
SoftwareFactory
(Eclipse Pluigins)
gen
Schematic
part
Application
specific code

working - it.unibo.bls18.qa/src/bls0.qa - Eclipse

e Edit Navigate Search Project Run Window Help

Quick Access

Package Expl... JUnit

> it.unibo.bls18 [IotUniboDemo ma
> it.unibo.bls18.qa [IotUniboDemo
> src-gen
> it.unibo.bls0button
> it.unibo.bls0control
> it.unibo.bls0Ctx
> it.unibo.bls0led
> it.unibo.blsMvcCtx
> it.unibo.btuuonmockgui
> it.unibo.ledarduino
> it.unibo.ledmockgui
> it.unibo.mvccontroller
> it.unibo.qatemperature
> it.unibo.tofronted
> src
> it.unibo.bls0button
> it.unibo.bls0control
> it.unibo.bls0led
> it.unibo.blsMvcCtx
it.unibo.btuuonmockgui
> it.unibo.custom.gui

Git Repositories

> [NO-HEAD] - C:\.git
> IotUniboDemo [master] - C:\aLab
Branches
Tags
References
Remotes

blsMvc.qa build_blsMvcCtx.gradle *bls0.qa customBlsGui.java

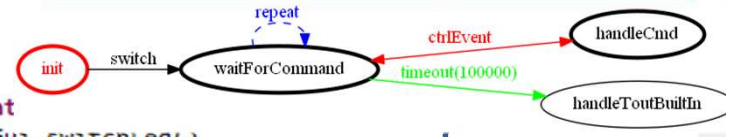
```
4  System bls0
5  Event ctrlEvent   : ctrlEvent( CATEG, NAME,CMD   )
6  Event local_click : clicked(N)
7
8  Context bls0Ctx ip [ host="localhost"  port=8019 ]
9
10 QActor bls0control context bls0Ctx  {
11     Plan init normal [ println( bls0control(starts)  )   ]
12         switchTo waitForInputEvent
13     Plan waitForInputEvent[ println("waitForInputEvent") ]
14         transition stopAfter 6000000
15             whenEvent local_click -> handleClickEvent
16         finally repeatPlan
17     Plan handleClickEvent resumeLastPlan [   printCurrentEvent;
18         emit ctrlEvent : ctrlEvent(leds, led1, switch)
19     ]
20 }
21 QActor bls0button context bls0Ctx {
22     Plan init normal [ println( bls0button(starts)  ) ;
23         javaRun it.unibo.custom.gui.customBlsGui.createCustomButtonGui()
24     ]
25 }
26 QActor bls0led context bls0Ctx  {
27     Plan init normal [  println( ledmockgui(starts)  ) ;
28         javaRun it.unibo.custom.gui.customBlsGui.createCustomLedGui()
29     ]
30         switchTo waitForCommand
31     Plan waitForCommand[  ]
32         transition stopAfter 100000
33             whenEvent ctrlEvent -> handleCmd
34         finally repeatPlan
35     Plan handleCmd resumeLastPlan[ printCurrent
36         javaRun it.unibo.custom.gui.customBlsGui.switchLed()
37     ]
```

Declaration of Messages and Events

Declaration of Computational nodes

Actor (on a computational node)

States of the actor (as a Moore FSM)



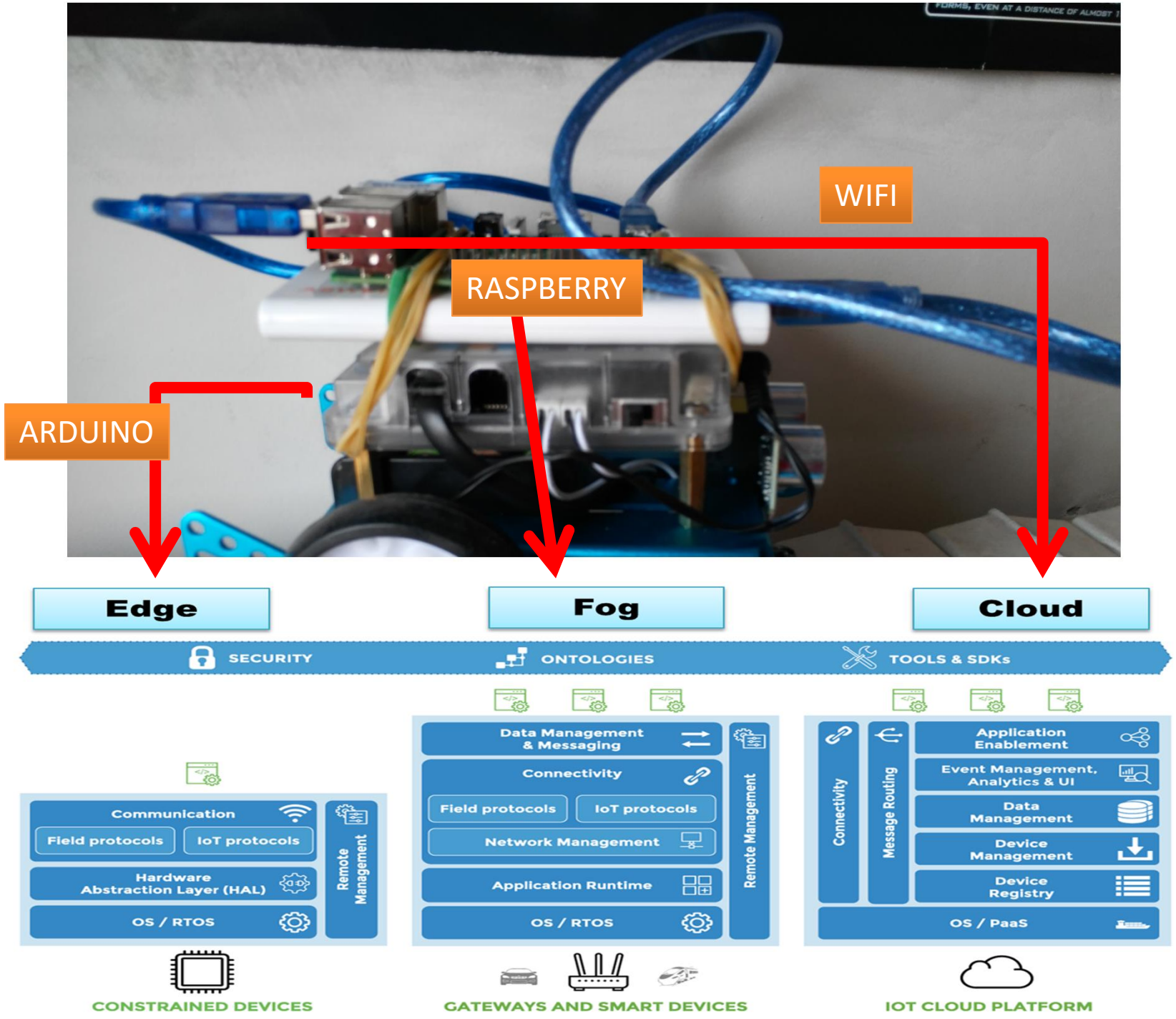
Problems Javadoc Declaration Console Terminal Git Staging

MainBls0Ctx [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (27 ott 2018, 09:31:26)

Writable Insert 30 : 9 730M of 828M

Beyond IOT minimal

DDR Robot as a thing


WIFI
RASPBERRY
ARDUINO
Edge
Fog
Cloud
SECURITY
ONTOLOGIES
TOOLS & SDKs
Communication
Field protocols
IoT protocols
Hardware Abstraction Layer (HAL)
OS / RTOS
Remote Management
Data Management & Messaging
Connectivity
Field protocols
IoT protocols
Network Management
Application Runtime
OS / RTOS
Remote Management
Connectivity
Message Routing
Application Enablement
Event Management, Analytics & UI
Data Management
Device Management
Device Registry
OS / PaaS
CONSTRAINED DEVICES
GATEWAYS AND SMART DEVICES
IOT CLOUD PLATFORM

Requirements:

see RequisitiRobotCleaner.pdf