Progettazione avanzata di software di controllo industriale

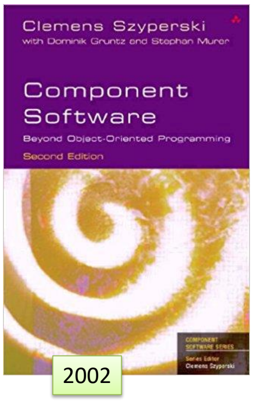
Elettric80

https://github.com/anatali/lss0

Sviluppo di sistemi software

PROGETTAZIONE, COMPONENTI, RIUSO, TESTING

Software components

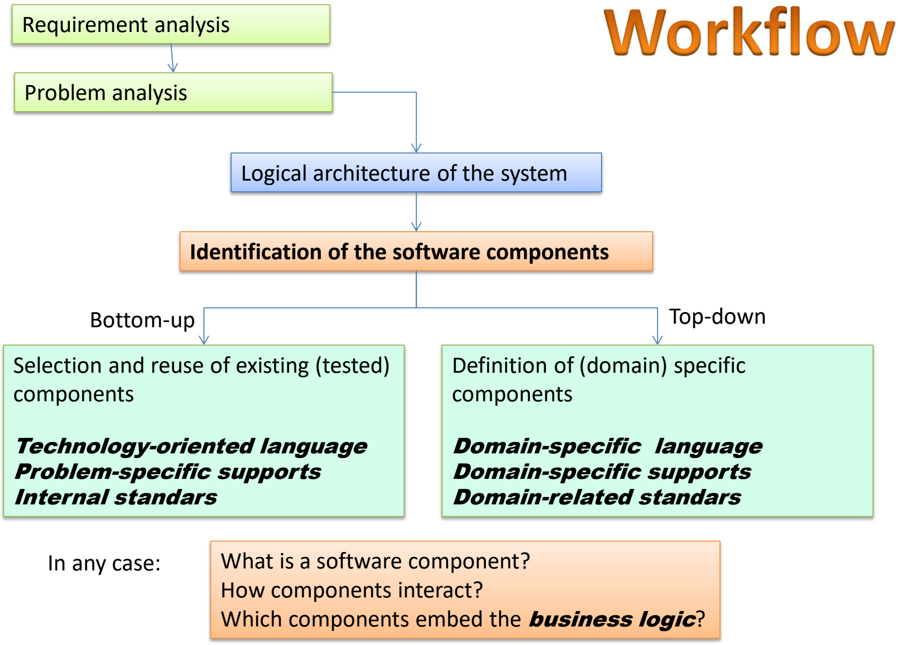

Clemens Szyperski
with Dominik Gruntz and Stephan Murer
Component Software
Beyond Object-Oriented Programming
Second Edition
COMPONENT SOFTWARE SERIES
Series Editor Clemens Szyperski

2002

The book gives us an objective survey of the component landscape, blended with unique insights into the market forces that influence deployment and in-depth coverage of real problems and their solutions.
Highlights of the Second Edition include:

- A comprehensive update of market-leading technologies including COM+, CORBA, EJB and J2EE
- New sections evaluating the strengths and weaknesses of emerging technologies like .NET, the CORBA Component Model, XML Web Services, showing how they work together with components and XML-related standards
- New examples in C# in addition to Java and Component Pascal

Workflow



Requirement analysis

Problem analysis

Logical architecture of the system

Identification of the software components

Bottom-up

Top-down

Selection and reuse of existing (tested) components

Technology-oriented language
Problem-specific supports
Internal standars

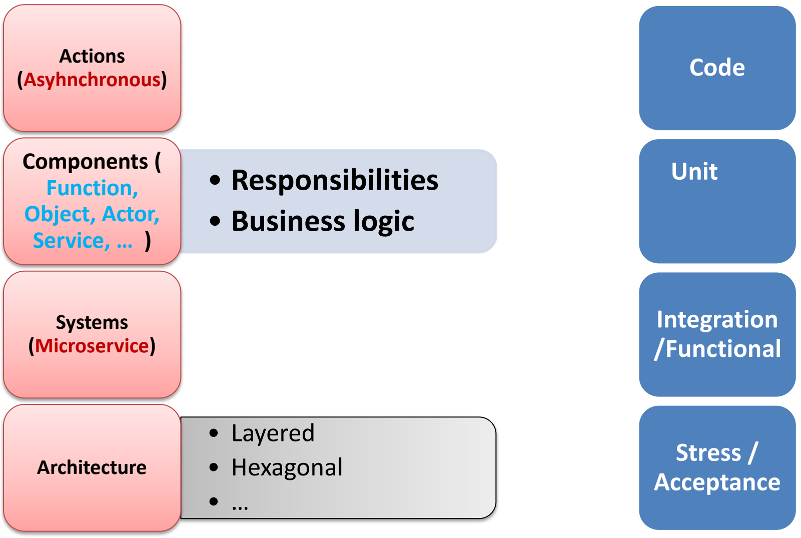
Definition of (domain) specific components

Domain-specific language
Domain-specific supports
Domain-related standars

In any case:

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

Testing



Actions (Asyhnchronous)

Components (Function, Object, Actor, Service, …)

- Responsibilities
- Business logic

Systems (Microservice)

Architecture

- Layered
- Hexagonal
- …

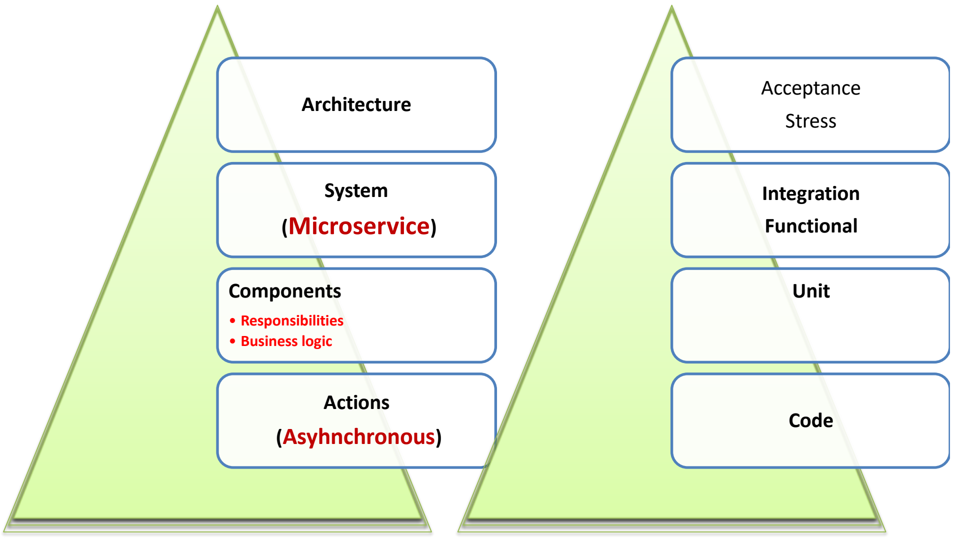
Code

Unit

Integration /Functional

Stress / Acceptance

Testing



Architecture

System (Microservice)

Components
- Responsibilities
- Business logic

Actions (Asyhnchronous)

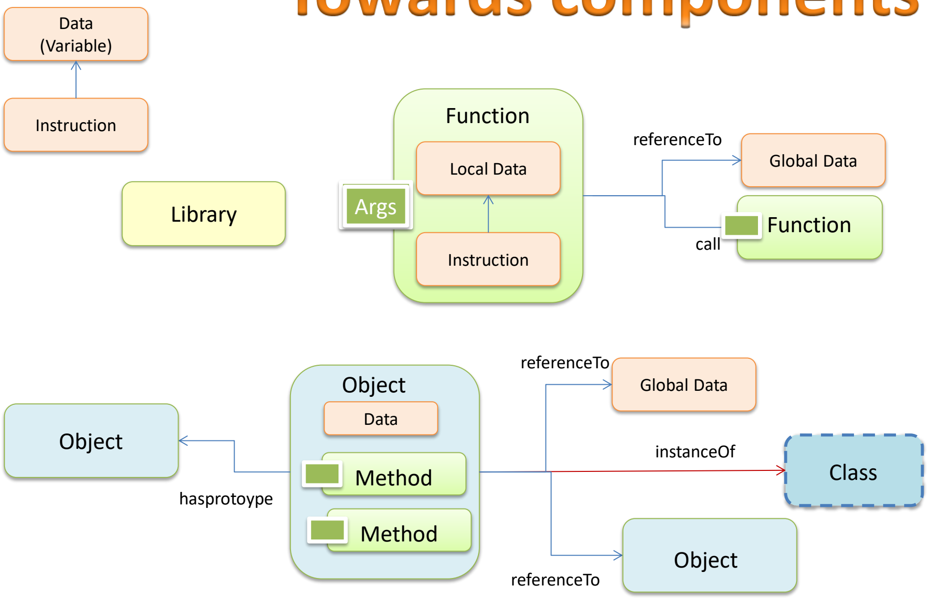
Acceptance Stress

Integration Functional

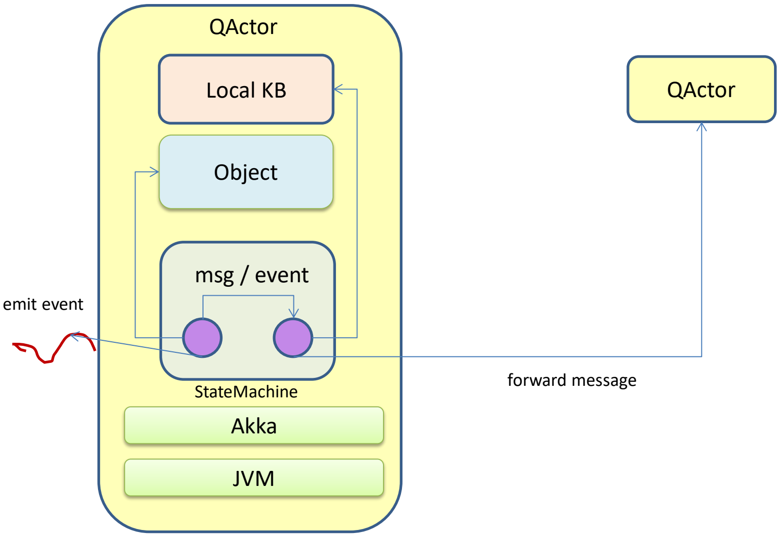
Unit

Code

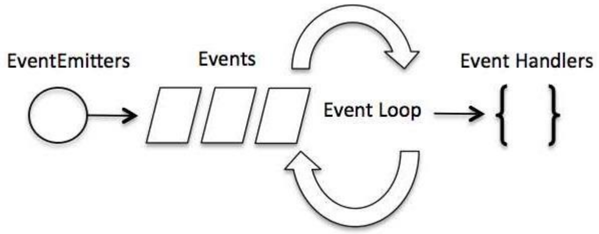
Towards components



Beyond procedure calls



Event loop



```
setTimeout(  function(){ console.log(“1000a1”); console.log(“1000a2”);  } , 1000 );

setTimeout(  function(){ console.log(“1000b1”); console.log(“1000b2”);  } , 1000 );

setTimeout(  function(){ console.log(“500”);  } , 500);
```

Fact asynch

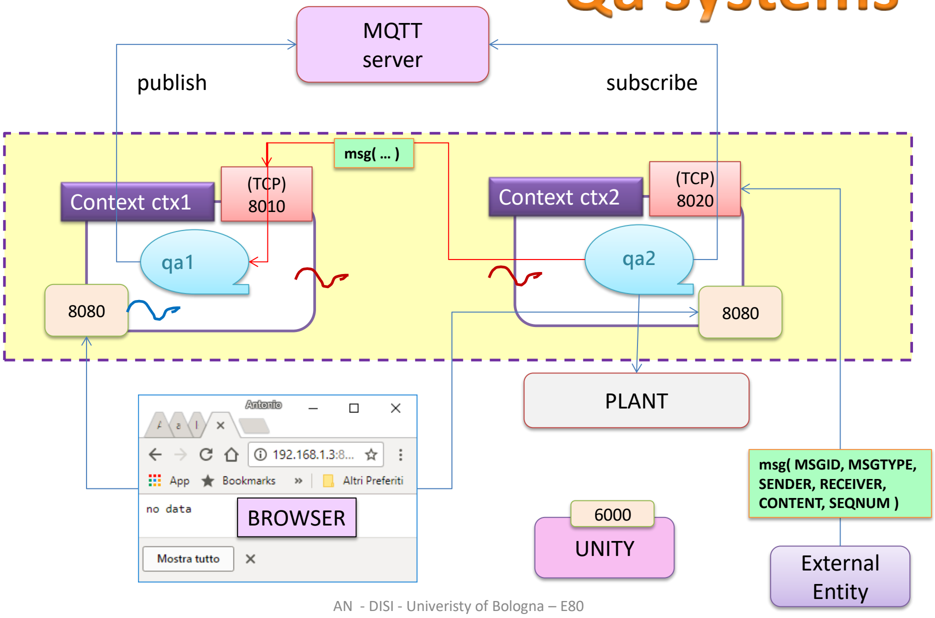
```
factAsynch = function( n,  callback ){ factIterAsynch(n,n,1,callback); }
factIterAsynch = function( n, n0, v, callback ){
var res = n*v;          //ACCUMULATOR
        console.log( "factIterAsynch n0=" + n0 + " n=" + n, " v=" + v + " res=" + res);
        if( n == 1 ) callback( "factIterAsynch(" + n0 + ") RESULT="+res );
        else setTimeout( function(){  factIterAsynch( n-1, n0, res, callback ) ; } , 0 );
 }
console.log("START");
console.log("CALL= ", factAsynch(4, console.log) );
factAsynch(6,console.log);
console.log("END");
```

```
START
factIterAsynch n0=4 n=4  v=1 res=4
CALL=  undefined
factIterAsynch n0=6 n=6  v=1 res=6
END
factIterAsynch n0=4 n=3  v=4 res=12
factIterAsynch n0=6 n=5  v=6 res=30
factIterAsynch n0=4 n=2  v=12 res=24
factIterAsynch n0=6 n=4  v=30 res=120
factIterAsynch n0=4 n=1  v=24 res=24
factIterAsynch(4) RESULT=24
factIterAsynch n0=6 n=3  v=120 res=360
factIterAsynch n0=6 n=2  v=360 res=720
factIterAsynch n0=6 n=1  v=720 res=720
factIterAsynch(6) RESULT=720
```

Fibonacci asynch

```
fibonacciAsync  = function( n, callback ){
if( n==1  || n == 2 || n == 3 ) {  callback( n  ); }
else{
console.log( "fibonacciAsync for " + n   );
 process.nextTick(function() {
      fibonacciAsync( n -1 , function(val1){
      process.nextTick(function() {
          fibonacciAsync( n -2, function(val2){
                 callback(  val1 + val2  );
          });
        });
      });
    });
  }}
console.log("fibAsynch  STARTS ");
fibonacciAsync(10, console.log);
console.log("fibAsynch  ENDS ");
```

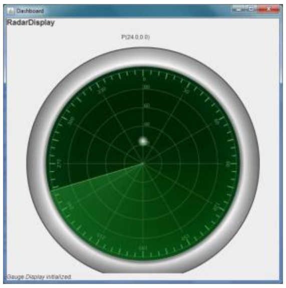
Qa systems



CaseStudy1

Si vuole realizzare un sistema software capace di presentare su un personal computer convenzionale l'immagine grafica di uno schermo radar (detto *radarGui*)

Il sistema deve fare in modo che su *radarGui* venga visualizzata l'informazione acquisita da una o più sorgenti di diverso tipo, tra cui un sonar HC-SR04 connesso a un RaspberryPi o a un Arduinio.



L'immagine risultante potrebbe essere del tipo mostrato nella figura in cui viene visualizzato un punto a distanza 24 cm in direzione 0.0.
radarGui deve permettere di visualizzare punti a distanza compresa tra cm 0 e cm 80 lungo direzioni comprese tra 0 e 180, utilizzando il quadrante di destra della figura.

PRIMA ANALISI : si tratta di realizzare un sistema software distribuito ed eterogeneo

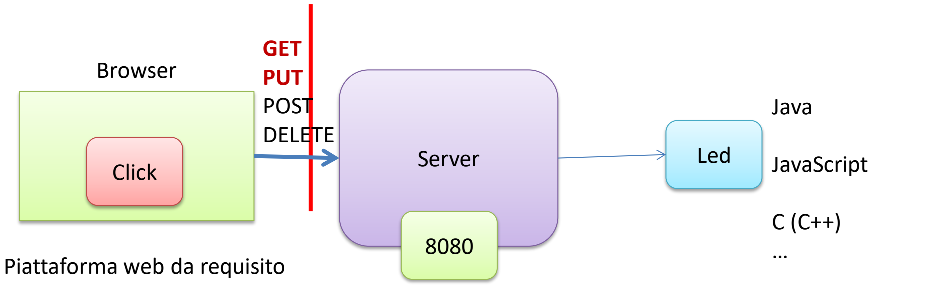
AN - DISI - Univeristy of Bologna – E80

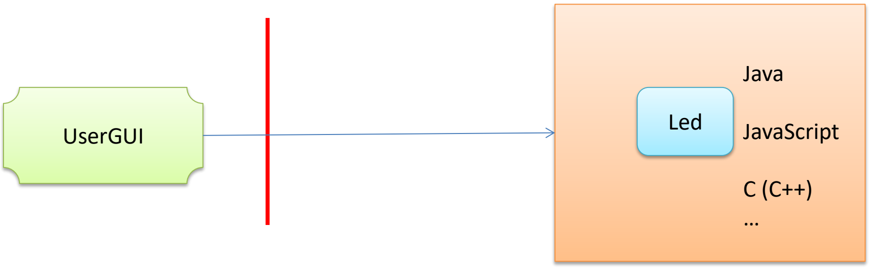
ButtonLed system

Project it.unibo.qa.nodeserver
https://github.com/anatali/lss0

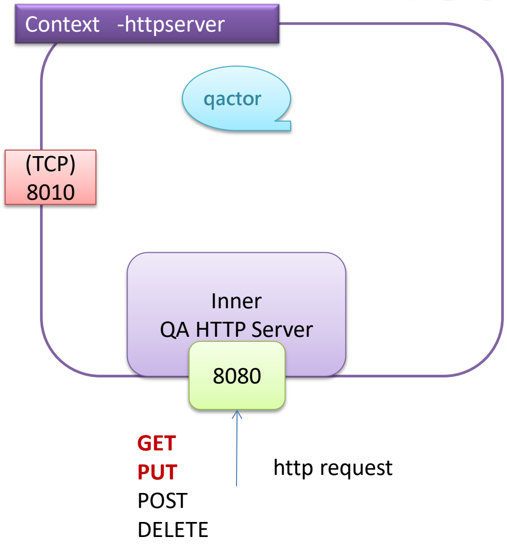
blsHlCustom: a BLS system on PC /Rasp
blshlBlink a system that executes reactive actions
blsHlBlinkReactiveWeb
blsHlNode a system that works with Node
helloMqtt a system that does publish/subscribe

LED

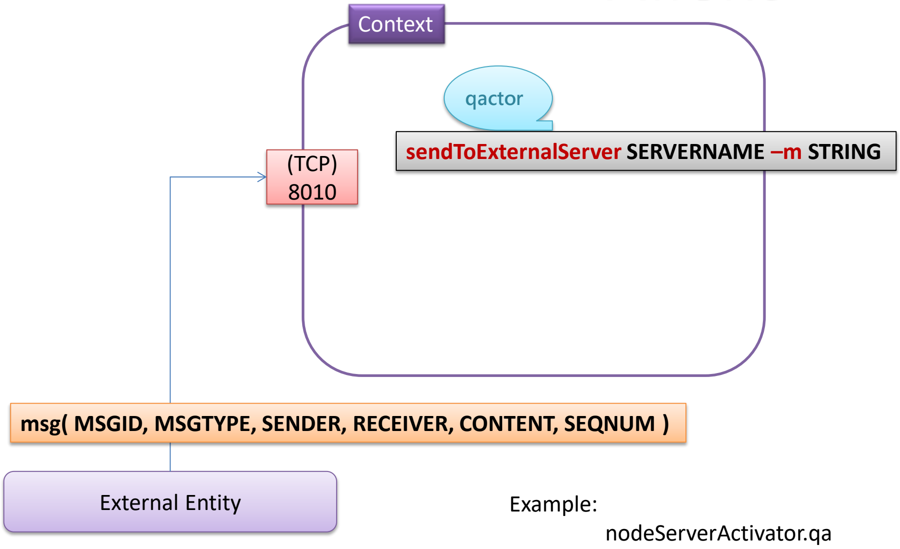




Web



Aliens

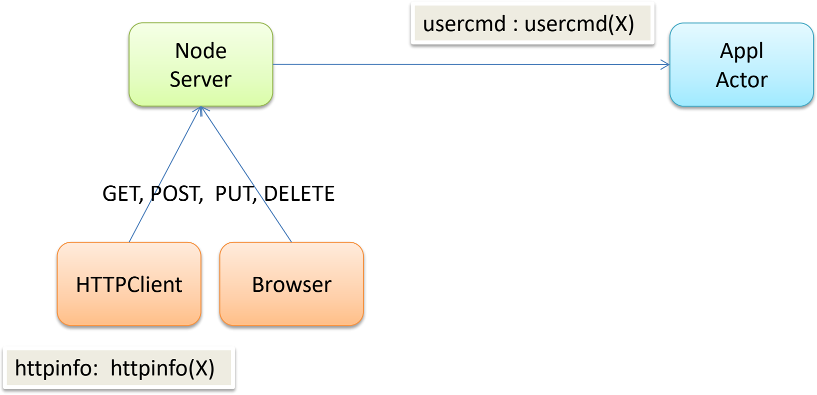


sendToExternalServer SERVERNAME –m STRING

msg(MSGID, MSGTYPE, SENDER, RECEIVER, CONTENT, SEQNUM)

Example:

nodeServerActivator.qa
httpClient.qa



usercmd : usercmd(X)

Node Server

Appl Actor

GET, POST, PUT, DELETE

HTTPClient

Browser

httpinfo: httpinfo(X)

blsHlCustom

A button-led system working on a PC

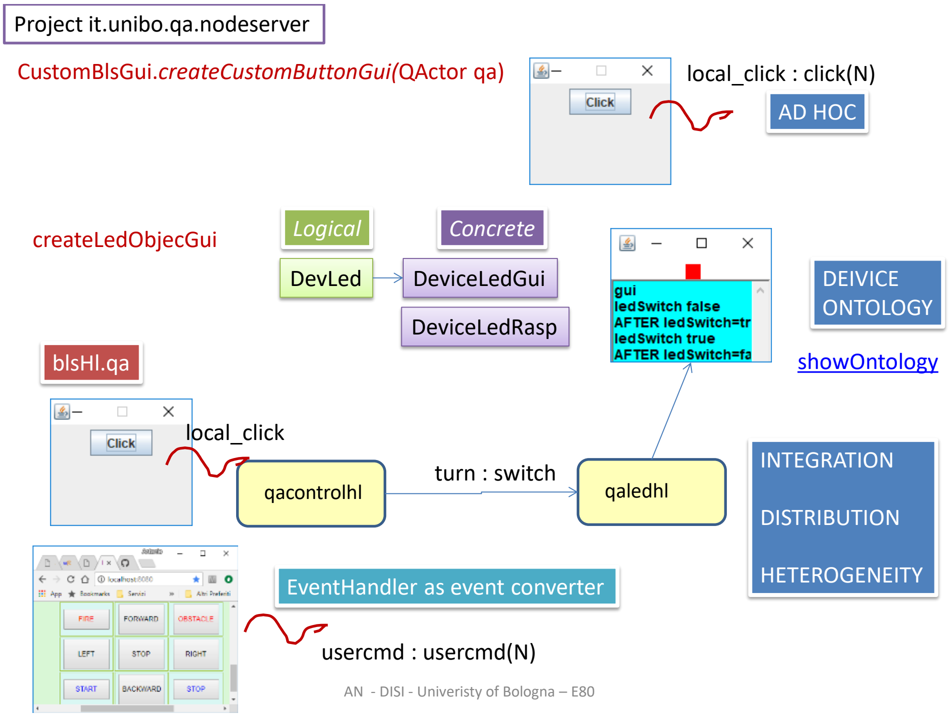
1. it.unibo.buttonLed.components. DevLed
2. it.unibo.buttonLed.components.DeviceLedImpl
3. it.unibo.custom.led. LedFactory
4. it.unibo.custom.button. ButtonFactory
5. blsHLCustom.qa
6. --
7. srcMore/it.unibo.ctxBlsHlCustom/QActorWebUI.html
8. Context ctxBlsHlCustom ip [host="localhost" port=8029] -httpserver
9. --
10. Events and Event-conversion

blsHlNode

A button-led system working in Node on a PC and on Raspberry

The Led on PC writes the current value on a file

1. it.unibo.qa.nodeserver\node\blsOop\Led.js
2. it.unibo.qa.nodeserver\node\blsOop\LedImplPc.js
3. it.unibo.qa.nodeserver\node\blsOop\LedHlPc.js
4. blsHLNode.qa(a qactor that interacts with a Led implemented in Node)
5. --
6. it.unibo.qa.nodeserver\cmd.txt (updated by LedHlPc.js | next: gpio)
7. --
8. it.unibo.qa.nodeserver\node\blsOop\LedHlRasp.js
9. it.unibo.qa.nodeserver\node\blsOop\LedImplGpiojs



Project it.unibo.qa.nodeserver

CustomBlsGui.*createCustomButtonGui(*QActor qa)

local_click : click(N)

AD HOC

createLedObjecGui

Logical

Concrete

DevLed

DeviceLedGui

DeviceLedRasp

```
gui
ledSwitch false
AFTER ledSwitch=tr
ledSwitch true
AFTER ledSwitch=fa
```

DEIVICE ONTOLOGY

showOntology

blsHl.qa

local_click

qacontrolhl

turn : switch

qaledhl

INTEGRATION

DISTRIBUTION

HETEROGENEITY

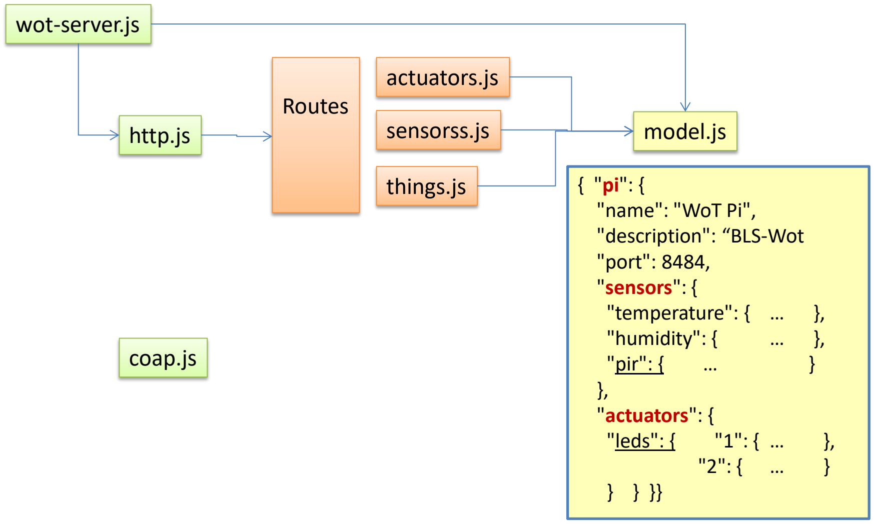
EventHandler as event converter

usercmd : usercmd(N)

blsWot

A ButtonLed system working as a Wot system

1. it.unibo.wot\nodeServerRest\servers\http.js
   - it.unibo.wot\nodeServerRest\resources\model.js
   - it.unibo.wot\nodeServerRest\resources\resources.json
   - ---
   - it.unibo.wot\nodeServerRest\routes\actuators.js
   - it.unibo.wot\nodeServerRest\routes\sensors.js
   - it.unibo.wot\nodeServerRest\routes\things.js
2. it.unibo.wot\nodeServerRest\servers\coap.js
3. it.unibo.wot\nodeServerRest\plugins\internal\ledsPlugin.js
   - it.unibo.wot\nodeServerRest\nat\observableFactory.js
   - it.unibo.wot\nodeServerRest\nat\TcpClientToQaNode.js
4. it.unibo.wot\nodeServerRest\plugins\external\coapPlugin.js
5. it.unibo.wot\nodeServerRest\wot-server.js
6. wotRestServerNode.qa ( )
7. ---
8. it.unibo.wot\src\it\unibo\rest\clienthttp\RestClientHttp.java
9. ---



```
{ "pi": {
   "name": "WoT Pi",
   "description": “BLS-Wot
   "port": 8484,
   "sensors": {
     "temperature": {   ...     },
     "humidity": {          ...     },
     "pir": {        ...              }
   },
   "actuators": {
     "leds": {       "1": {  ...       },
                    "2": {     ...       }
     }   } }}
```

Applications

Software Design (and development)

New design

- Domain driven design, Small (autonomous) teams, Continuos integration
- Bounded Contexts,
- Microservices,

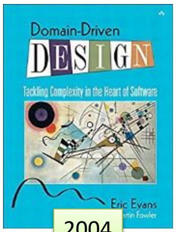
An approach to delivering SOA by building fine-grained services to support business capabilities that are distributed and organized as functional domains

- Communications based on (core) business concepts,
  No more data (anemic CRUD services) but business capabilites
- No more (only) layered architectures but hexagonal architectures

- Communications standards : the REST architecture style
  Self-descriptive messages / Stateless interactions

- Evolutionary architectures (requirements, componenyts, views)
  (Discoverability, **Coonectivity**, **Reactivity**, **Safety**, **Interoperability**, Delegation, Scalability, Collaboration, **Usability**, Marketability)

DDD



2004

Software developer
Designer
Domain expert

- 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).

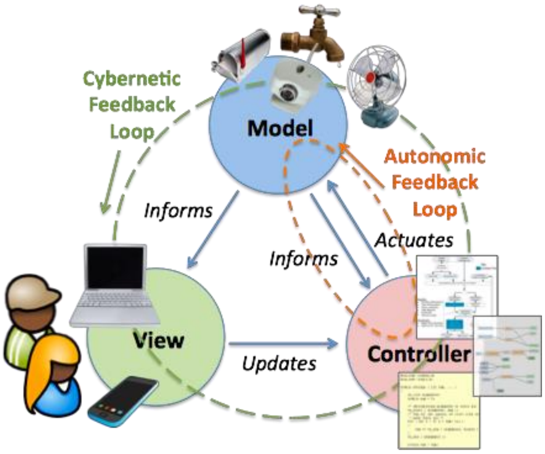
- iterate a single model to reflect a shared understanding across domain experts, designers and developers
- establisha common language, i.e. a UBIQUITOUS LANGUAGE (pg. 24)

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

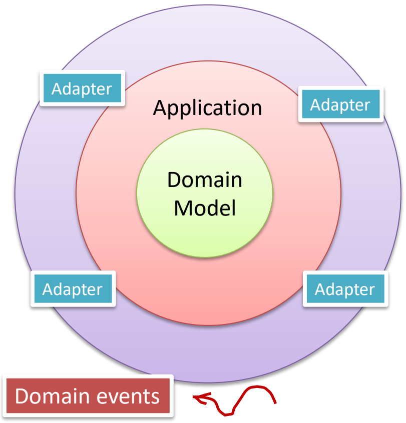
MVC

The **Model-VIew-Controller macro pattern** provides a framework for the structured division of responsibility between people and software. 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.

Architecture



Adapter

Application

Adapter

Domain Model

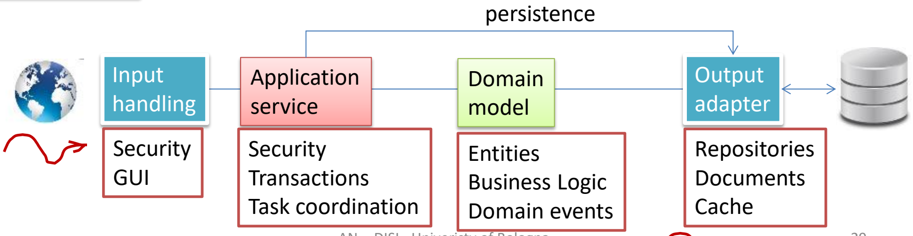
Adapter

Adapter

Domain events

A Ports and Adapters architecture style, or a Hexagonal Architecture, makes a clear separation between the domain model and the devices used for inputs and outputs.

No technology concerns, e.g. HTTP contexts or database calls, are referenced in the domain, allowing changes in technology to be made without affecting the domain.



persistence

Input handling

Application service

Domain model

Output adapter

Security
GUI

Security
Transactions
Task coordination

Entities
Business Logic
Domain events

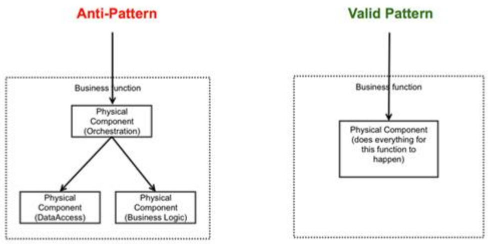
Repositories
Documents
Cache

Microservices (anti)patterns

https://www.infoq.com/articles/seven-uservices-antipatterns

Services must align clearly to a business capability and should not try to do something outside of their boundary.
Functional separation of concern is vital for architecture to govern otherwise it will destroy the agility, performance, and scalability and ended up in establishing a tightly coupled architecture, resulting in delivery entropy and cohesion chaos.
Don't have services separated by technical concerns instead they must be separated based on the business capability.

Try to look at a service as one atomic business entity, which must implement everything to achieve the desired business functionality. The self-contained services are more autonomous and scalable than the layered services.
It's perfect to re-write some common code across multiple services, that's fine and it's a good trade-off to keep the autonomy level.

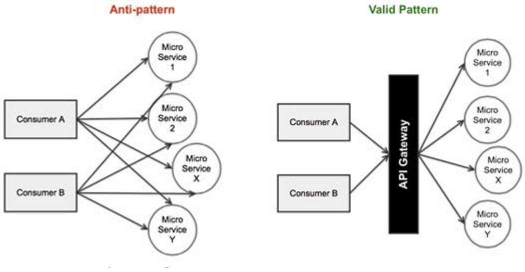

Anti-Pattern
Business function
Physical Component (Orchestration)
Physical Component (DataAccess)
Physical Component (Business Logic)
Valid Pattern
Business function
Physical Component (does everything for this function to happen)

Microservices (anti)patterns

Give independence to your services. Every service that you deliver must have a test suite, which should cover all the service functionality, security, performance, error handling, and consumption driven testing for every current and future consumer.

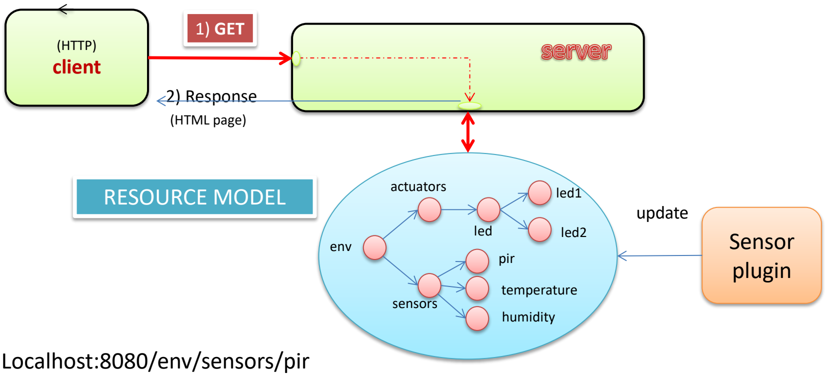
Continuous deployment, if you have not done so, is a must investment and a cultural change that every enterprise should aim for. At least, if you don't have a way to automatically test and deploy – do not do micro-services.

Invest in API Management solutions to centralize, manage and monitor some of the non-functional concerns and which would also eliminate the burden of consumer's managing several microservices configurations. API gateway can be used orchestrate the cross-functional microservices that may reduce round trips for web applications.



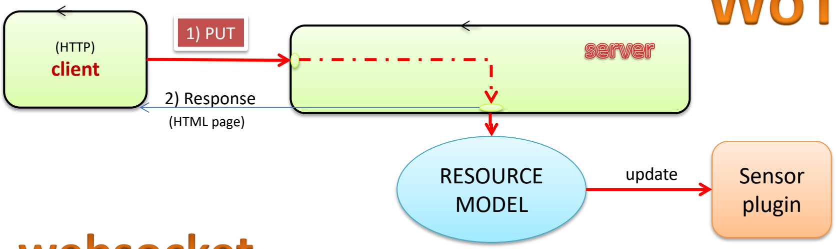
WoT

Integration patterns: REST on device, Gateway (CoAP), Cloud (MQTT)
Resource model design : (ontology) tree, knowledge base, ..
Representation design : json, prolog, HTML, MessagePack, …
Interface design : GET, PUT, etc
Resourse binding design : HAETOAS (web linking,..), findability, …

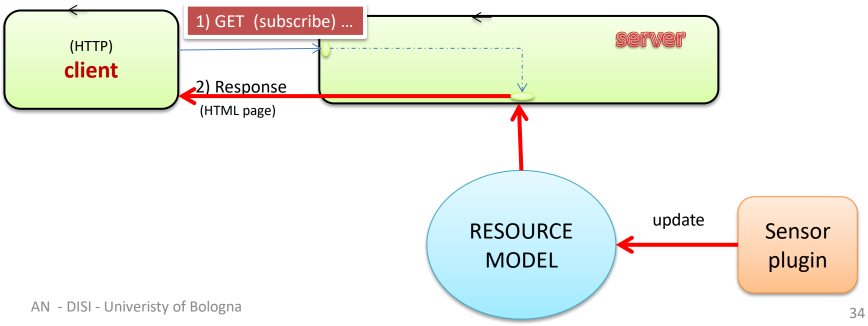


Localhost:8080/env/sensors/pir

WoT



websocket



Challenges

New **challenges** (Overcorming disruption)

- Service Orchestration vs. service Choreography
- UI fragments and UI composition
- Reporting (event data pump)
- Monitoring (specialized subsystems),
- Distributed transactions,
- Eventual consistency,
- Compensating transactions,
- Distributed transactions,

Splitting the monolite

- The Strangler Application Pattern
- https://www.ibm.com/developerworks/cloud/library/cl-strangler-application-pattern-microservices-apps-trs/index.html

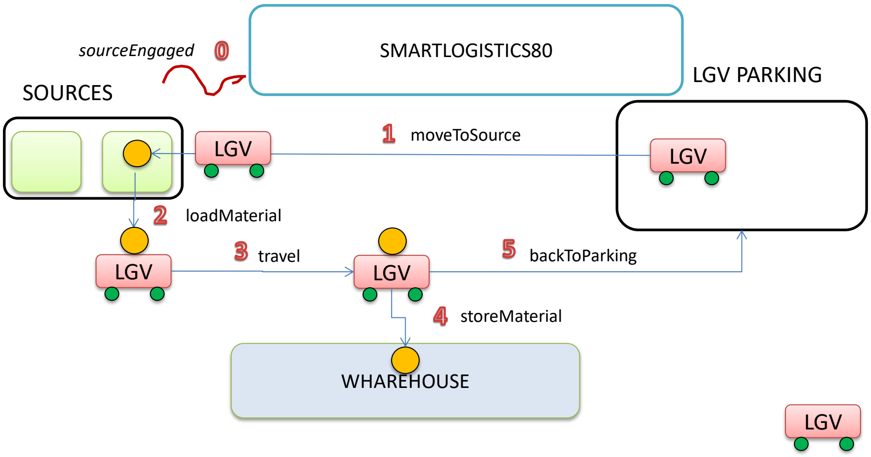
Case study

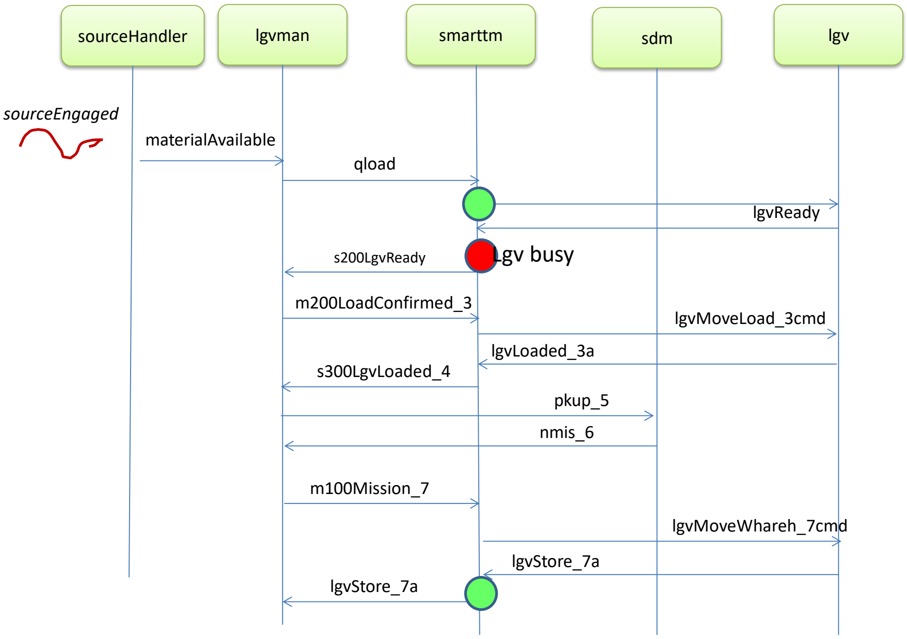
Requirements

Our company must build a software system (named SMARTLOGISTICS80)
that must store in automatic way physical material put on
one of its material sources into a proper cell of a Warehouse.
More specifically the system must:

R0) detect the presence of new material on a source and select a free LGV (LASER GUIDED VEHICLE) in the parking area
R1) move the selected LGV from the parking area to the source
R2) check when the material is loaded on the LGV
R3) select a cell in the Warehouse and drive the LGV from the source to the Warehouse
R4) check when the material is stored in the selected cell of the Warehouse
R5) move the LGV from the Warehouse to the parking area

Requirements





sourceHandler

lgvman

smarttm

sdm

lgv

sourceEngaged

materialAvailable

qload

lgvReady

s200LgvReady

Lgv busy

m200LoadConfirmed_3

lgvMoveLoad_3cmd

lgvLoaded_3a

s300LgvLoaded_4

pkup_5

nmis_6

m100Mission_7

lgvMoveWhareh_7cmd

lgvStore_7a

lgvStore_7a

AN - DISI - Univeristy of Bologna – E80