HEADS
Heterogeneous and Distributed Services for the Future Computing Continuum

ThingML: A Modelling Language for the Internet of Things

HEADS Project
Olivier Barais
University of Rennes 1
barais@irisa.fr

SINTEF

Outline

- Background and Motivations
- ThingML Contribution
- The ThingML Language
- ThingML Transformations
- ThingML Demo
- Summary

Outline

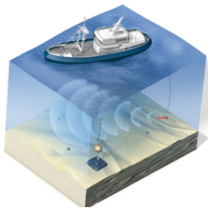
- **Background and Motivations**
- ThingML Contribution
- The ThingML Language
- ThingML Transformations
- ThingML Demo
- Summary

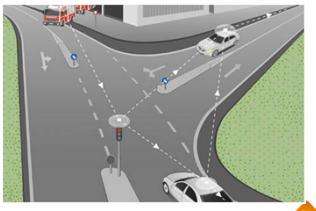
Context: HD-Services

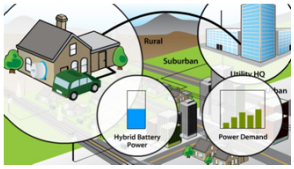
- Heterogeneous and Distributed Services
  - **Heterogeneous:** The infrastructure on which the service runs is composed of a set of different nodes and networks.
    - The "Future Computing Continuum" which ranges from microcontroller based sensors and devices to cloud.
  - **Distributed:** The implementation of the services is composed of a set of independent processes communicating asynchronously.
    - Truly distributed services implementation is required in order to provide useful and reliable services which take advantage of the infrastructure.

Examples

- Health domain and ambient assisted living
- Energy domain and smart grids
- Environmental monitoring and oil and gas
- Safety in hazardous environments
- Intelligent Transport Systems (ITS)
- ...







Fields of IoT

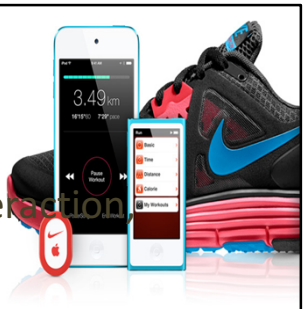
SEI Summer School, July 2015

http://heads-project.eu

Fields of IoT

Wearable

Quantified self



Form : Stuff you wear, minimal interaction,
often connected your phone
but not your typical mobile device.

Motivations : body/health information.

Withings JAWBONE BASIS fitbit

Wearable - Watches

Wearable … but not really an IoT.
More like a mobile device. Note

- Intel/TagHeuer/Google collab
- Intel MICA Smartband.

Wearable - Glasses

Fields of IoT

Smart building / Home automation

SEI Summer School, July 2015

HEADS
http://heads-project.eu

Google - Nest

HEADS
http://heads-project.eu

Sen.se

sense mother.

Fields of IoT

Non humanoid robots and drones

Nabaztag

Parrot Minidrones

Fields of IoT

Embedded – Old is New

Arcade / smart surfaces

MASH MACHINE

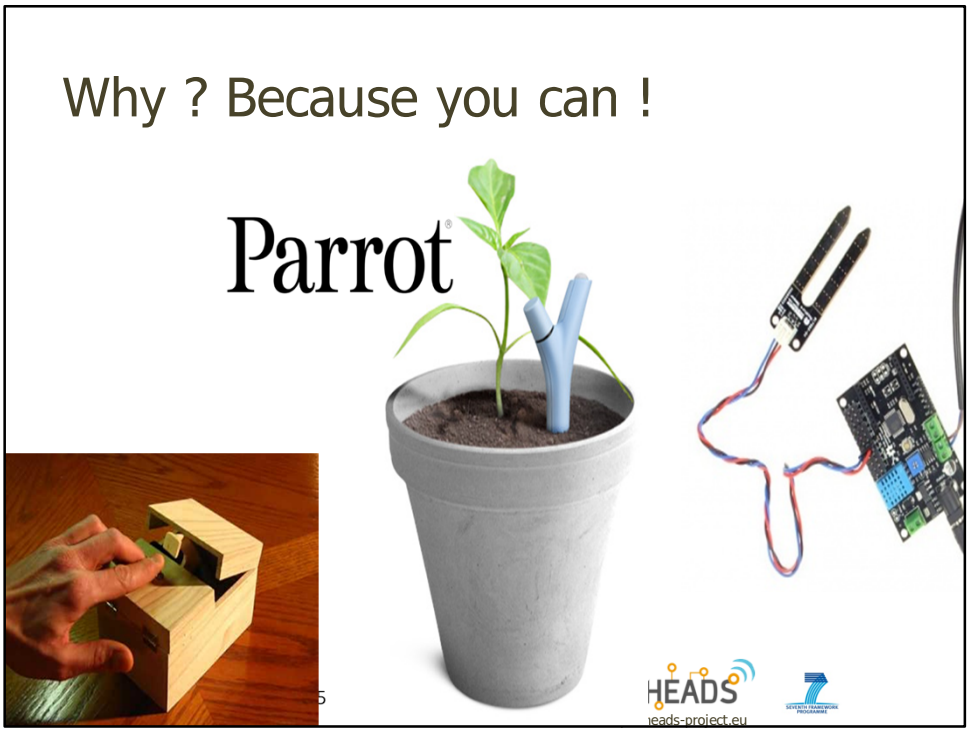
Digital signage / Smart furniture

Fields of IoT

For the fun, the lulz

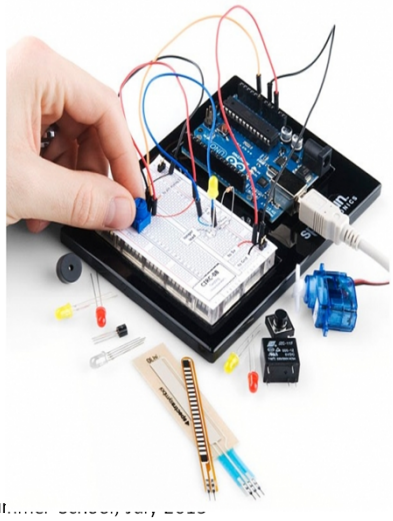
SEI Summer School, July 2015

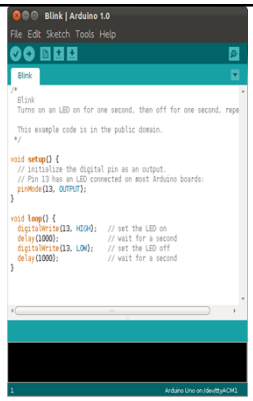
http://heads-project.eu



	Arduino	Raspberry Pi	Intel Edison	ST Nucleo	ESP8266
Cost	**$3-$30**	**~$35**	**$40**	**$8**	**~$3**
Wifi	Shield ($60) or ESP8266	USB Dongle	Built-in	Shield ($60) or ESP8266	Built-in
Programming	**C/C++**	**Python/Java/C++, Javascript, …**	**C, C++, Javascript, Python, Java, ..**	**C, C++**	**C++/Lua**
Code distribution	USB/ SPI/ Serial	In-Situ	In-Situ	USB/ SPI/ Serial	Serial/ OTA
Storage	Built-In	SD-Card	Built-In	Built-In	Built-In
I/O	Arduino Uno: 13 GPIO/ 6 ADC	17 GPIO	20 GPIO (PWM, SPI, one-wire) • 6 analog inputs. 1 UART (Rx/Tx).	GPIO (50) 12-bit ADC with 16 Timers (8) I2C (3), SPI (3) USART (3)	10 GPIO/ 1 ADC
Power consumption	12V 50mA Deep sleep ?	1A 5V	**peak**: 984mW using Wifi 5GHz band : 680mW **using serial port**: 346mW **Idle** 88mW **powered** off 45mW	128 µA/MHz. In Stop mode, the power consumption can be as low as 9 µA. 3,3V	3,3V Transmit 802.11b, CCK 11Mbps, POUT=+18.5dBm 197mA Standby 0.9mA Deep sleep 10 uA

Arduino





Raspberry Pi

Evaluation of IoT boards

Microcontroller vs Processor

Arduino is using a 8-bit microcontroller. It's simple and predictable for people new to software development.

But it's impossible to install a full system like linux, so it's limited by nature.

To have a full OS with network, the programming language of your choice and a lot of potential, a processor is required.

Digital / Analog IO

If you develop on your laptop, you can only plug devices with high level IO like USB. But most IoT sensors are a lot simpler than that, using low level analog and digital IO.

To be really useful as an IoT platform, you must have digital and/or analog IO.

Note : Some IoT platforms only have digital IO, others like Intel Galileo and Intel Edison have both digital and analog IO.

Graphics

Many wearable projects like watches have a touch screen, or a projector for glasses. They look like a lot like small mobile devices, and require graphics.

But for all the other IoT projects, you don't need and don't want graphics.

Remember : display = interaction = brain attention share = not scalable.

Networking

Network connectivity is important in IoT.

You may take a platform without wireless internet and add a dongle for prototyping, but :

- It's harder to switch to production
- Power optimizations are limited
- Dongle often propose limited features : Bluetooth instead of Bluetooth Low Energy.
- Integration with software-OS is not always trivial.

Advice : take platforms with great networking inside for prototyping then production.

Form factor

It's convenient to have large boards with lots of IO for prototyping. But you'll need to design a new board for production. It takes a lot of money, time and skills.

Or you take a board with a modular design : your compute module has all the complex parts you won't redesign, and the simper connectivity board can be replaced or designed easily.

Power features

A big difference between IoT prototypes and production is total power consumption.

You would not plug a 3G dongle on a desktop and call it a mobile phone, right ? Same for IoT.

You need :

- a very efficient processor,
  with advanced sleep/hibernation features
- power optimized wireless
- great integration of all parts
- lots of software, driver and OS optimization

OS

We all like to prototype with our desktop OS.

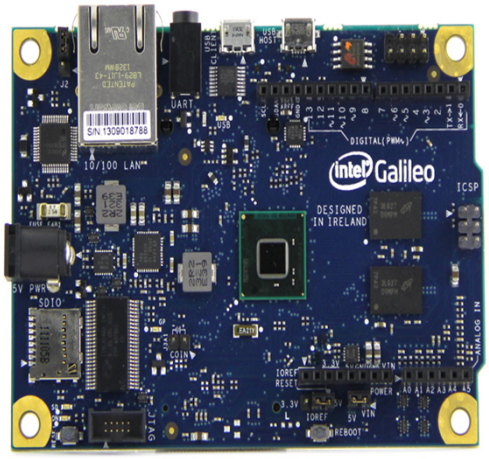
It can be a big linux distro like Ubuntu, Windows 10, OSX … it's easy, all the packages are readily available.

But a professional grade embedded project requires to start from scratch, perhaps use a substitute of glic, control each piece of code added to the system and integrate with a large team of software developers.

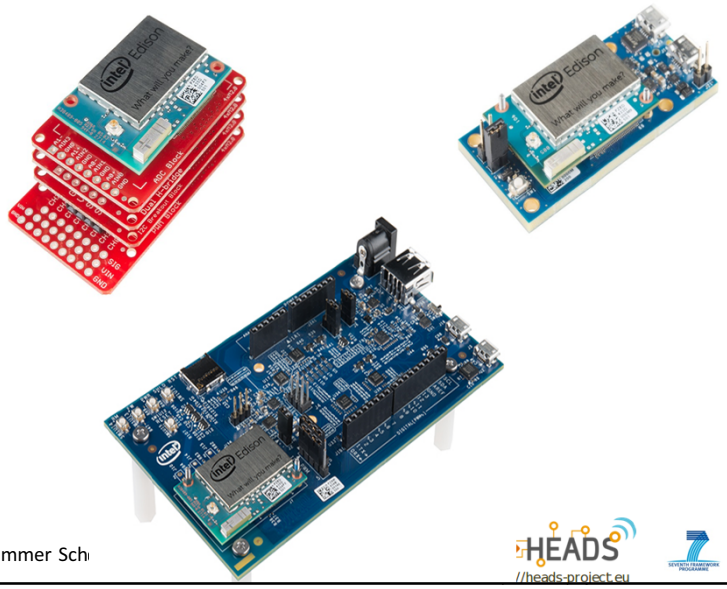
A typical open source OS for professional projects is Yocto.

Intel IoT boards

Intel Galileo



Intel Edison



Intel Curie

Intel IoT boards

Intel Galileo Specifications

Intel Galileo Specs

Intel Quark System-on-Chip (SoC) x1000 with a 32bit core running at 400MHz.

Connections : mini-PCI Express (for Wifi-Bluetooth module), 100Mb Ethernet port, Micro-SD slot, RS-232 serial port, USB Host, USB Client.

Form factor : Arduino compatible pins.

Intel Galileo Specs

The board can run :

- Linux Yocto by default from Intel.
  (Yocto is an open source linux used by embedded professionals)
- Debian variant by the community.
- Arduino style code by an emulator.
- Windows by Microsoft.
- In some cases, WindRiver solutions.

Intel IoT boards

Intel Edison Specifications

Intel Edison Specs

22 nm Intel SoC that includes :

- a dual-core, dual-threaded Intel Atom CPU at 500 MHz, running linux
- a 32-bit Intel Quark microcontroller at 100 MHz, running RTOS

Memory : 1 GB LPDDR3

Storage : 4 GB eMMC

Wifi : a/b/g/n

Bluetooth : 4.0 Low Energy

Intel Edison Specs

40 GPIO :

- SD card : 1 interface
- UART : 2 controllers (1 full flow, 1 Rx/Tx)
- I2C : 2 controllers
- SPI : 1 controller with 2 chip selects
- I2S : 1 controller
- GPIO : Additional 12 (with 4 capable of PWM)
- USB 2.0 : 1 OTG controller

Form factor : modular. The connectivity boards are available from Intel, Sparkfun, …

Intel Edison Specs

The board can run :

- Linux Yocto by default from Intel.
  (Yocto is an open source linux used by embedded professionals)
- Debian variant Ubilinux by the community.
- Arduino style code by an emulator.

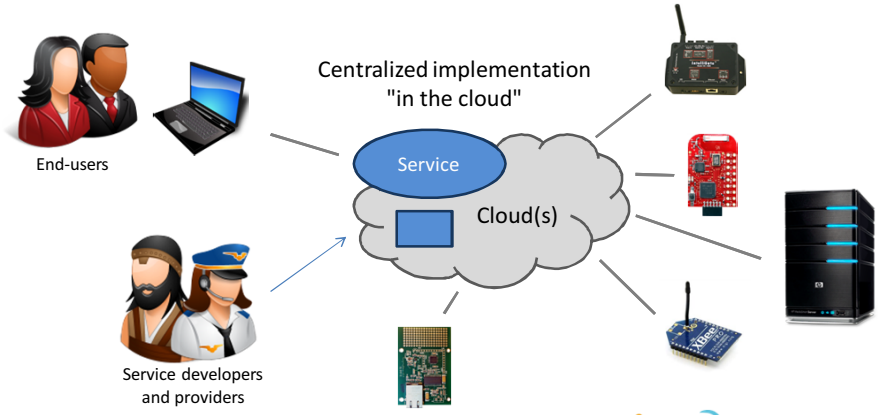
SEI Summer School, July 2015

HEADS
http://heads-project.eu

	Arduino	Raspberry Pi	Intel Edison	ST Nucleo	ESP8266
Cost	**$3-$30**	**~$35**	**$40**	**$8**	**~$3**
Wifi	Shield ($60) or ESP8266	USB Dongle	Built-in	Shield ($60) or ESP8266	Built-in
Programming	**C/C++**	**Python/Java/C++, Javascript, ...**	**C, C++, Javascript, Python, Java, ..**	**C, C++**	**C++/Lua**
Code distribution	USB/ SPI/ Serial	In-Situ	In-Situ	USB/ SPI/ Serial	Serial/ OTA
Storage	Built-In	SD-Card	Built-In	Built-In	Built-In
I/O	Arduino Uno: 13 GPIO/ 6 ADC	17 GPIO	20 GPIO (PWM, SPI, one-wire) • 6 analog inputs. 1 UART (Rx/Tx).	GPIO (50) 12-bit ADC with 16 Timers (8) I2C (3), SPI (3) USART (3)	10 GPIO/ 1 ADC
Power consumption	12V 50mA Deep sleep ?	1A 5V	**peak**: 984mW using Wifi 5GHz band : 680mW **using serial port**: 346mW **Idle** 88mW **powered** off 45mW	128 µA/MHz. In Stop mode, the power consumption can be as low as 9 µA. 3,3V	3,3V Transmit 802.11b, CCK 11Mbps, POUT=+18.5dBm 197mA Standby 0.9mA Deep sleep 10 uA

Why "HD-Services" ?

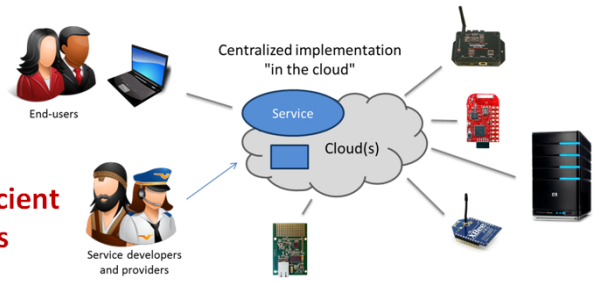
- Isn't Internet of Things about having everything connected and available in the cloud?



Limitations of centralized approaches

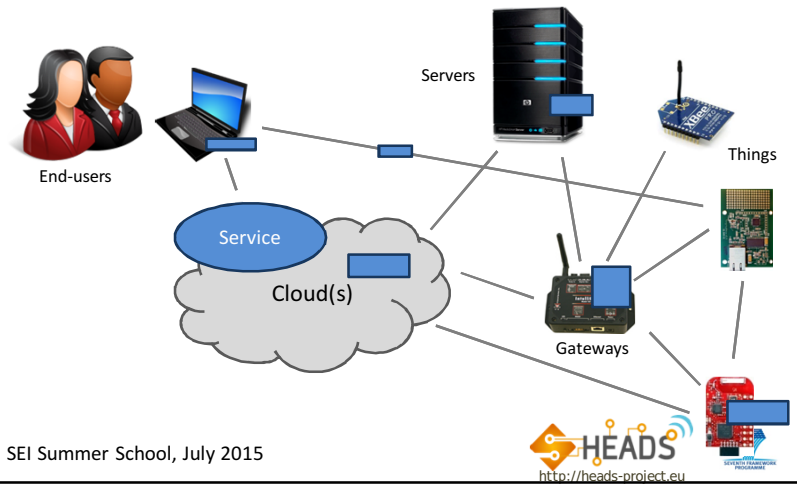
- Very easy to develop, evolve and maintain but…
  - Underexploits "Things" capabilities
  - Does not allow real-time or critical services
  - Not resource efficient (bandwidth)
  - Not robust
  - Does not scale

Good solution when possible but not sufficient in many realistic cases



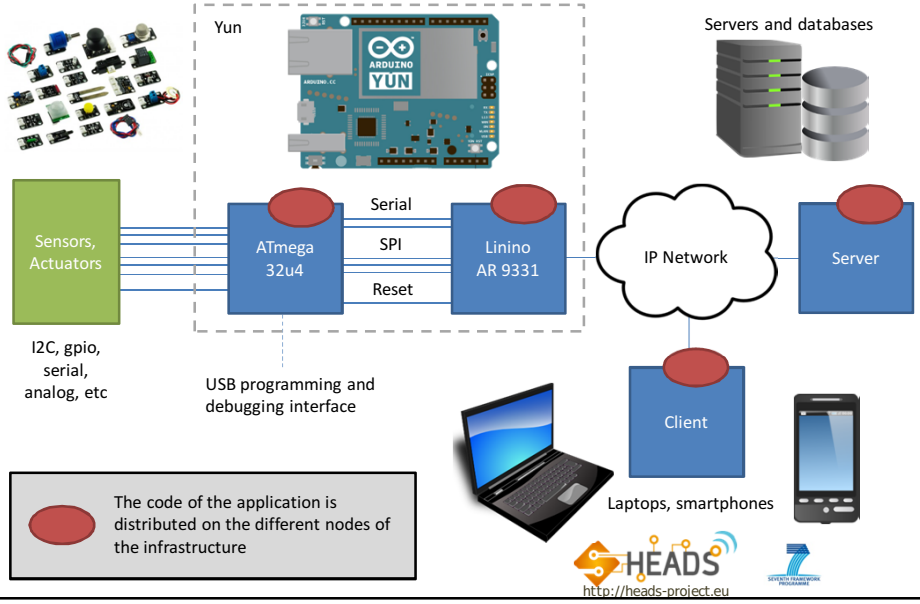
Distributing the implementation

- The service implementation is distributed to exploit the infrastructure



HEADS
http://heads-project.eu

Simple IoT Infrastructure Example

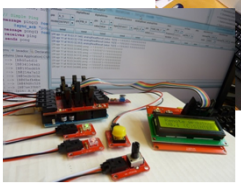


HEADS
http://heads-project.eu

Experimental platforms and "lab"

- Cloud (Amazon, Flexiant, Rackspace, etc)
- Mini-Cloud (Openstack + Docker)
- Android (Java + Android)
- Cubietruck "cloud" (Linux + Docker)
- Raspberry Pi (Linux)
- Arduino Yun (dd-wrt linux + AVR µC)
- Arduino (AVR µC)
- TI ARM/MSP µC
- Esp8266
- Intel Edison

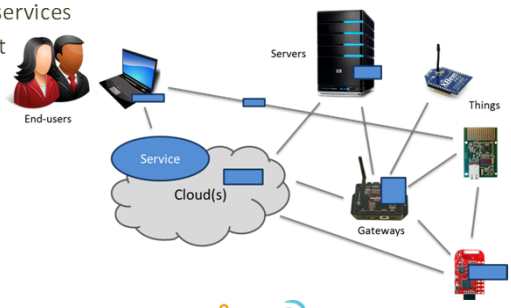
- Home automation and wearable devices



Benefits of HD-Services

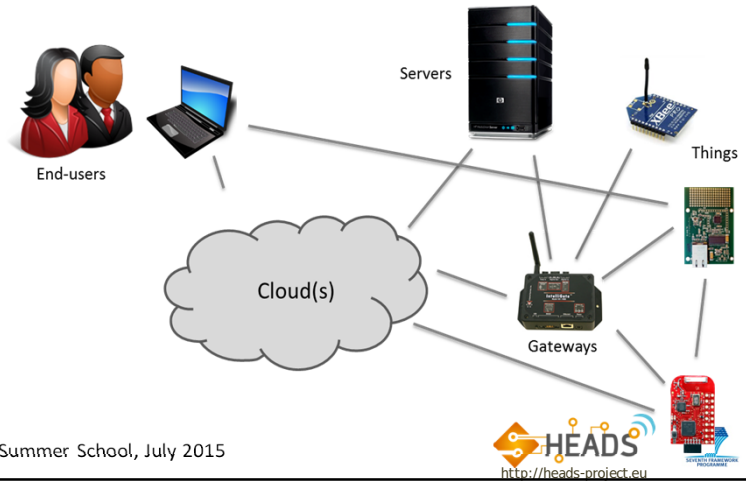
- Complex to develop, lots of different skills involved but…
  - Allows fully exploiting the features of each platforms
  - Allow for local and/or decentralized decision making
  - Robust to partial and/or temporary failures
  - Push processing close to data sources
  - Allow for real-time and critical services
  - Can scale in a "big data" context

In practice for more and more real-world services are HD-Services



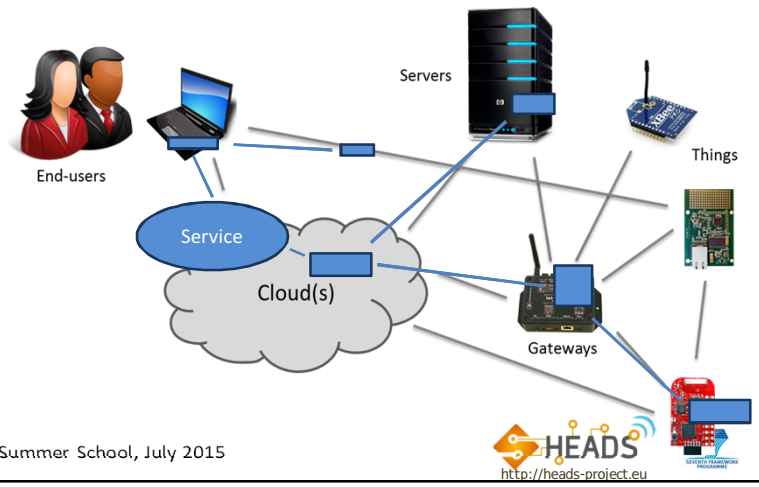
What are the problems? (1/6)

- Here is an example infrastructure



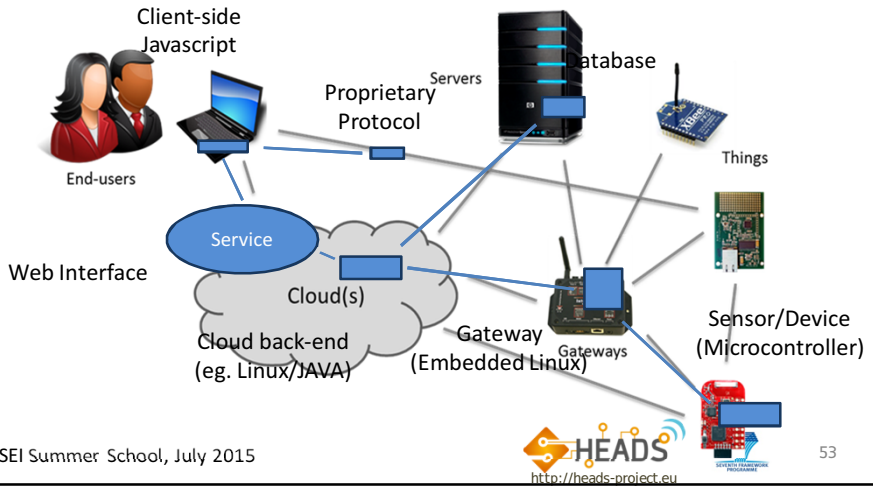
What are the problems? (2/6)

- Here is the software components needed for the service



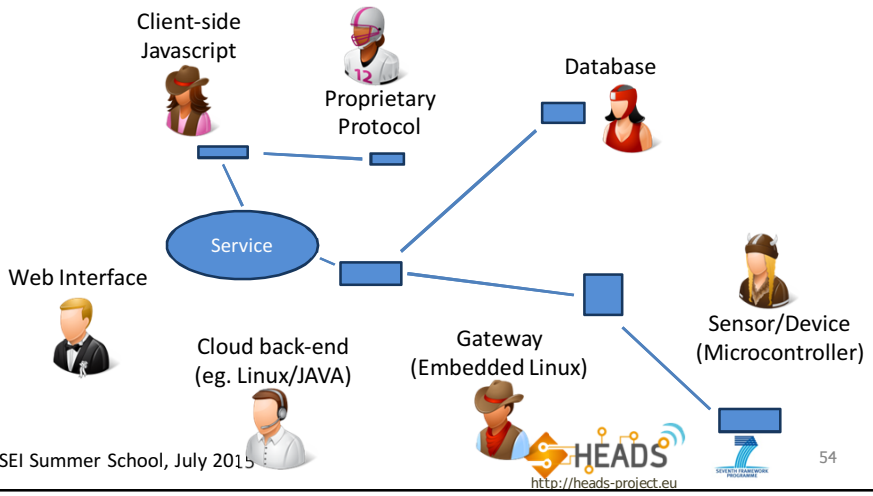
What are the problems? (3/6)

- Heterogeneous infrastructure and technologies are needed



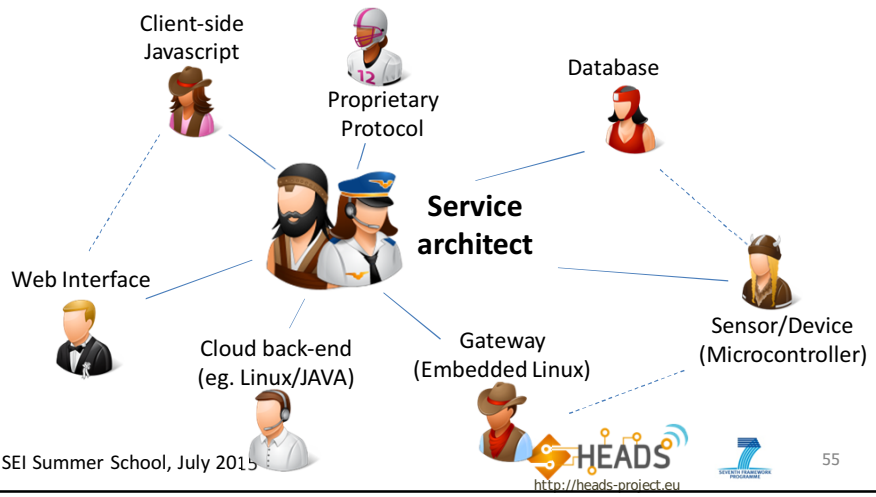
What are the problems? (4/6)

- A lot of different expertise are needed
  - Both for development and runtime deployment/maintenance



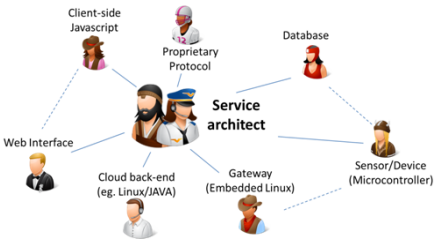
What are the problems? (5/6)

- Someone needs to coordinate all experts
  - Design the different components, their functionality and interactions



What are the problems? (6/6)

- Large heterogeneous teams need to collaborate
  - A service architect / developer
  - Many "platform experts"
  - Complex and expensive
  - Unavailable to small actors
- Service maintenance and evolutions
- Infrastructure is dynamic
  - Constant evolution/adaptation
- (Early) Validation?
- Software reuse?



Challenging and expensive

State of the practice

- **State of the art / practice**
  - Solution 1: Centralized service which uses devices "as-is"
    - Most common practice. Simple but restrictive.
  - Solution 2: Avoid problems by carefully selecting platforms
    - For which software frameworks pre-exist (eg. Arduino libs / shields)
  - Solution 3: Hide behind an homogeneous software layer
    - OS + generic or specific middleware platforms (eg. JAVA/JVM)
  - Solution 4: Custom develop manually all pieces of software
    - Can exploit full potential but very expensive (eg. automotive)
  - Solution 5: Fully fledged Model-Driven "PIM/PSM" approach
    - Good separation of concerns but impractical and too exclusive
- Non of the above allow exploiting the HDS to its full potential
- **Our Goal: Combine MDE techniques and runtime techniques into a practical software engineering approach which retains the main benefits of each of them**

Why?

Challenges

- **Provide a support for a multi-viewpoints approach for IoT on top of Eclipse**
  - **A kind of Eclipse Polarsys Capella for IoT ;)**
- **Viewpoints examples**
  - **Manage development of large applications**
    - **Typescript example**
  - **Manage the deployment and evolution of large scale distributed applications**
    - **Kevoree example (www.kevoree.org)**
  - **Event streaming management**
  - **Reactive programming (ThingML.org)** ***TODAY***

Why?

HEADS Goal

- Provide tools and methods
  - For each actor to concentrate on his task
  - For decoupling the tasks of different actors
  - Using **state of the art software engineering practices**
    - Modularity, reusability, runtime deployment, continuous integration, validation, etc…
  - Cost efficient and practically usable
    - No large overhead, integrated with legacy systems, etc…

Outline

- Background and Motivations
- **ThingML Contribution**
- The ThingML Language
- ThingML Transformations
- ThingML Demo
- Summary

What is ThingML ?

- A DSL to model distributed reactive systems
  - IoT systems, embedded systems, sensor networks, ...
- Components, State machines and action language
  - « Main stream » MDE
- Contribution of ThingML
  - « Complete » action language
  - Slots, Mixins and Aspects instead of Inheritance and Composites
  - Enforced encapsulation and actors semantics
- Target Platforms and Applications
  - MDE for resource constrained systems (microcontrollers, IoT)
  - Development of applications distributed across heterogeneous hardware
  - Other types of reactive systems?

Why ThingML ?

- Typical MDE benefits
  - Reduce development, maintenance and evolution costs
  - Perform verifications and analysis on the models
  - Model application at a platform independent level
- No existing approach can deal with microcontrollers
  - ThingML can run on hardware less than 1ko of RAM
- No existing approach is really platform independent
  - Since actions are written in the target language

ThingML Goals

- Provide tools and methods
  - For each actor to concentrate on his task
  - For decoupling the tasks of different actors
  - Using state of the art software engineering practices
    - Modularity, reusability, runtime deployment, continuous integration, validation, etc…
  - Cost efficient and practically usable
    - No large overhead, integrated with legacy systems, etc…

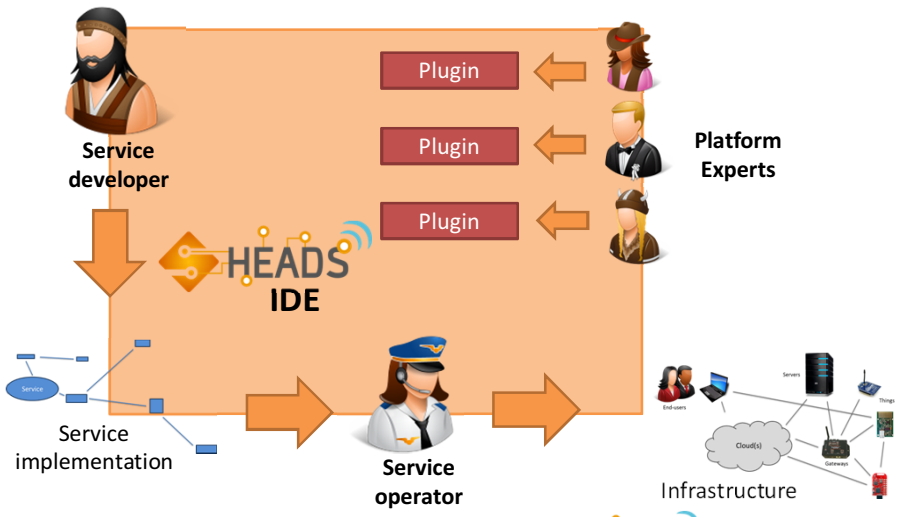
The ThingML tools

- Based on Eclipse / EMF Metamodel
- Textual Syntax with EMFText
  - For good usability and productivity
  - To keep the development cost of the editor(s) reasonable
- Graphical exports (graphML, graphviz, …)
- Static well formedness and type checker
- **Equivalent** compilers for a set of platforms
  - C/C++ for different microcontrollers, linux, embedded linux
  - Java for computers, smartphones, …
  - Javascript (NodeJS)
  - Maybe others if needed
- Generators for communication channels
- Easy to distribute ThingML IDE
  - Standalone and lightweight IDE
  - Eclipse plugins

Devolopping the ThingML tools

- Technologies
  - Eclipse / EMF and EMFText for metamodels and editors
  - Scala for constraints, transformation and code generation
  - Swing lightweight standalone editor
- Continuous integration process (using our thingml.org cloud server)
  - Maintain a code repository : Github open-source forge based on git
  - Automate the build : Maven build tool + Jenkins server
  - Make the build self-testing : Maven + JUnit
  - Everyone commits to the baseline every day : Github
  - Every commit (to baseline) should be built : Github triggers Jenkins
  - Keep the build fast : About 2 minutes at this point
  - Test in a clone of the production environment : Maven
  - Make it easy to get the latest deliverables : Archiva, Jenkins web interface
  - Everyone can see the results of the latest build : Jenkins web interface
  - Automate deployment : Java Web Start (JNLP)

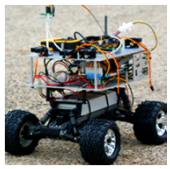
ThingML in the HEADS IDE

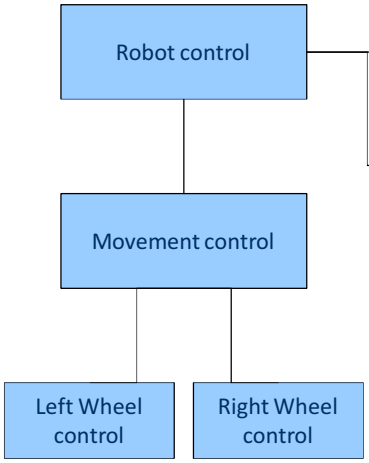


Outline

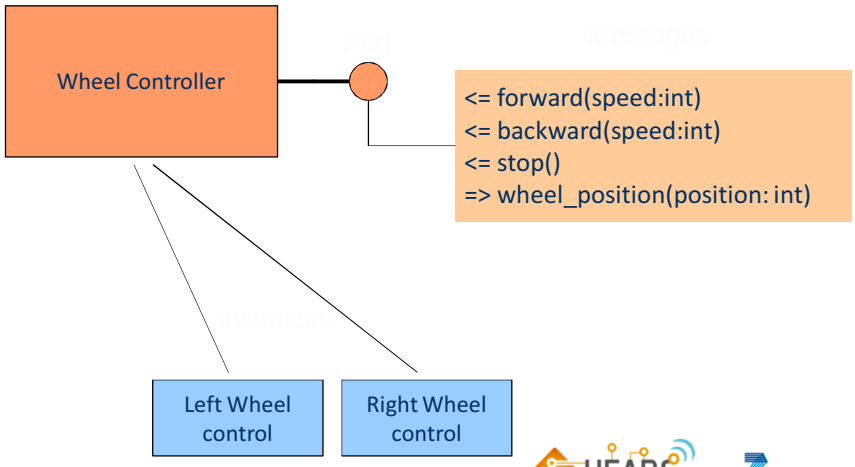
- Background and Motivations
- ThingML Contribution
- **The ThingML Language**
- ThingML Transformations
- ThingML Demo
- Summary

ThingML: Architecture Model

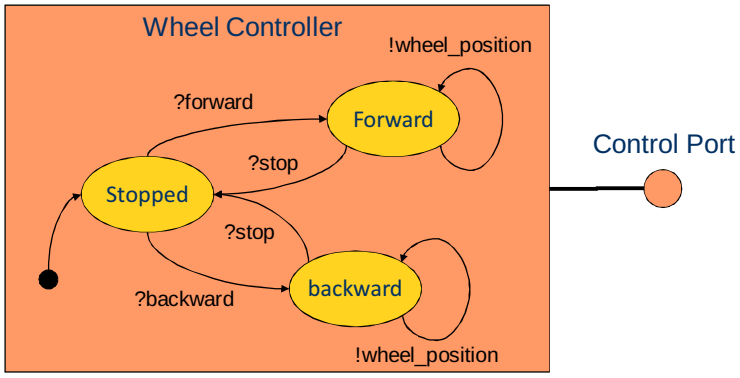




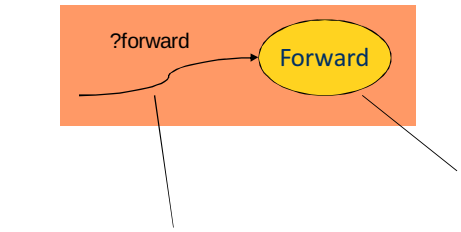
ThingML: Component



ThingML: State Machines



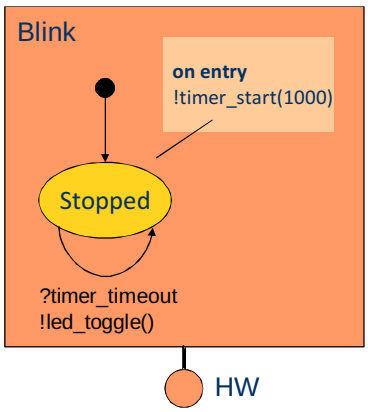
ThingML: Action Language




```
on entry do
  reset_wheel_position()
  motor_start()
end
...
```

```
action do
  motor_set_speed(speed)
  motor_set_direction(FW)
end
```

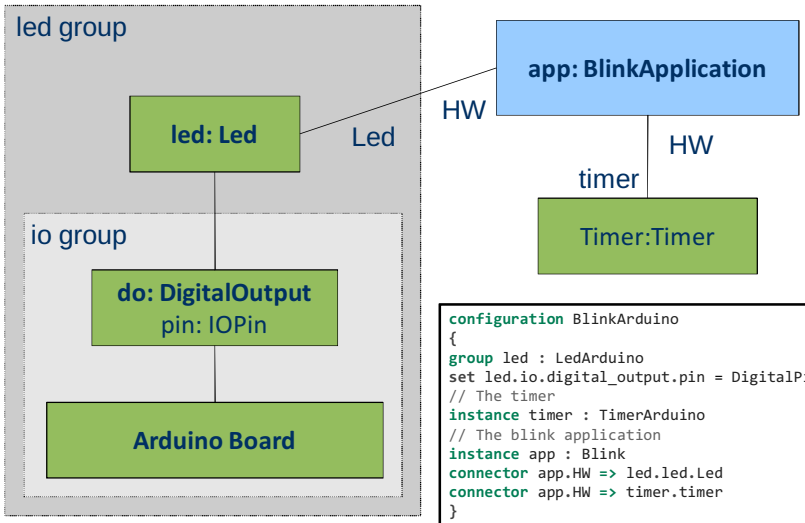

Blink example state machine




```
<= timer_timeout()
=> led_toggle()
=> timer_start(delay:int)
```

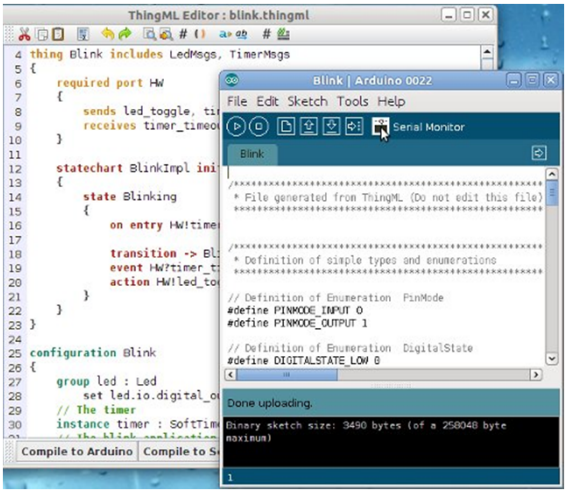
```
thing Blink includes LedMsgs, TimerMsgs
{
    required port HW
    {
        sends led_toggle, timer_start
        receives timer_timeout
    }

    statechart BlinkImpl init Blinking
    {
        state Blinking
        {
            on entry HW!timer_start (1000)

            transition -> Blinking
            event HW?timer_timeout
            action HW!led_toggle ()
        }
    }
}
```

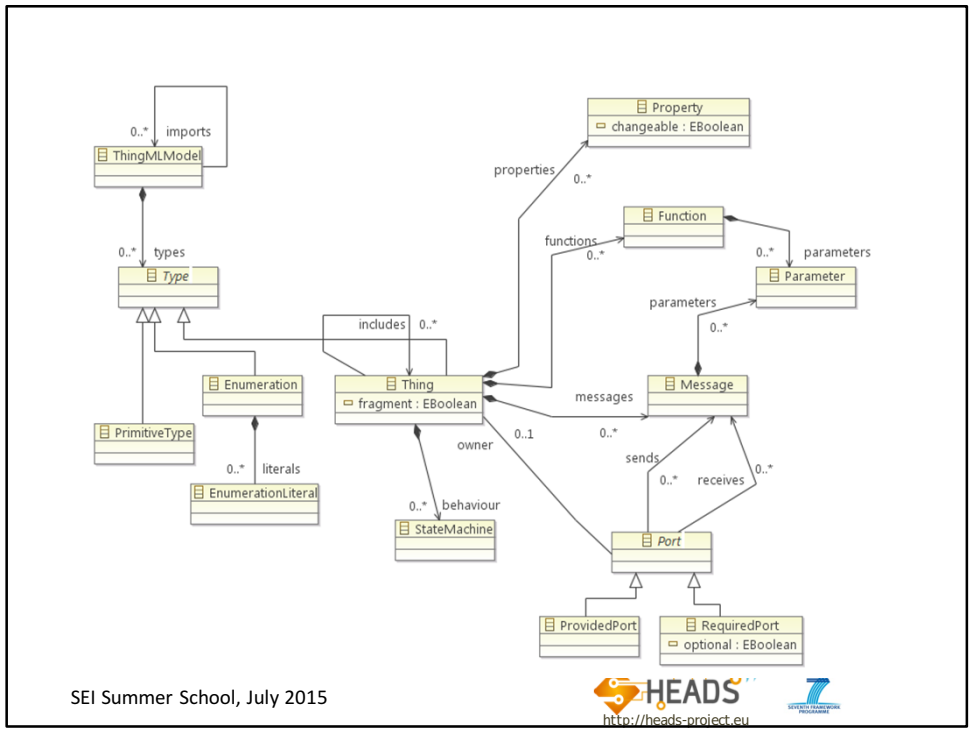
Blink example and instance groups

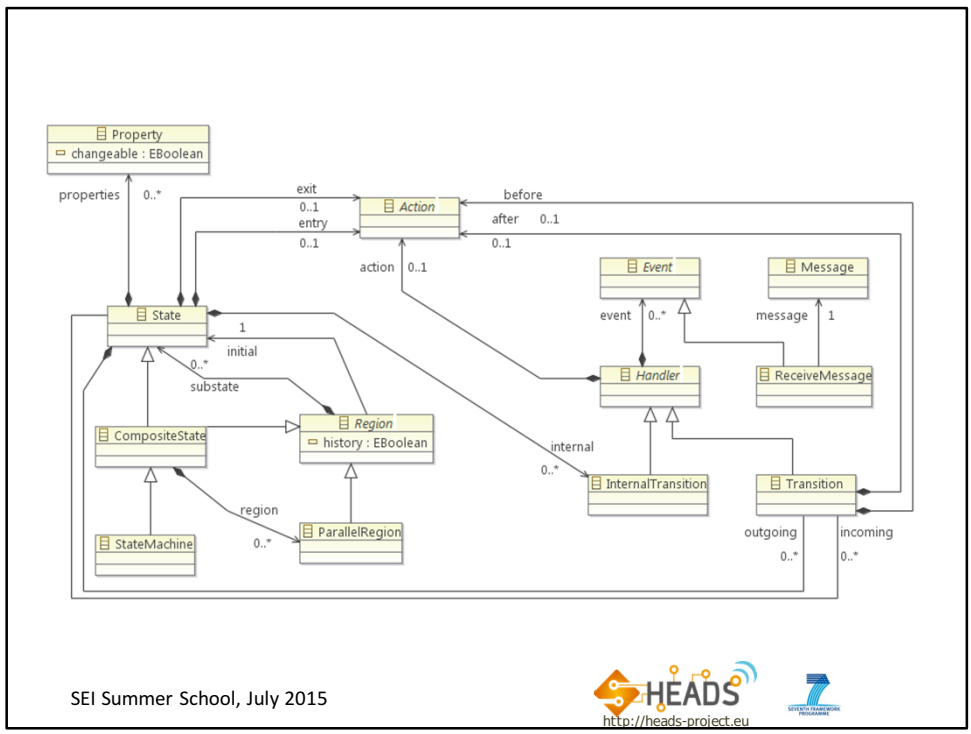


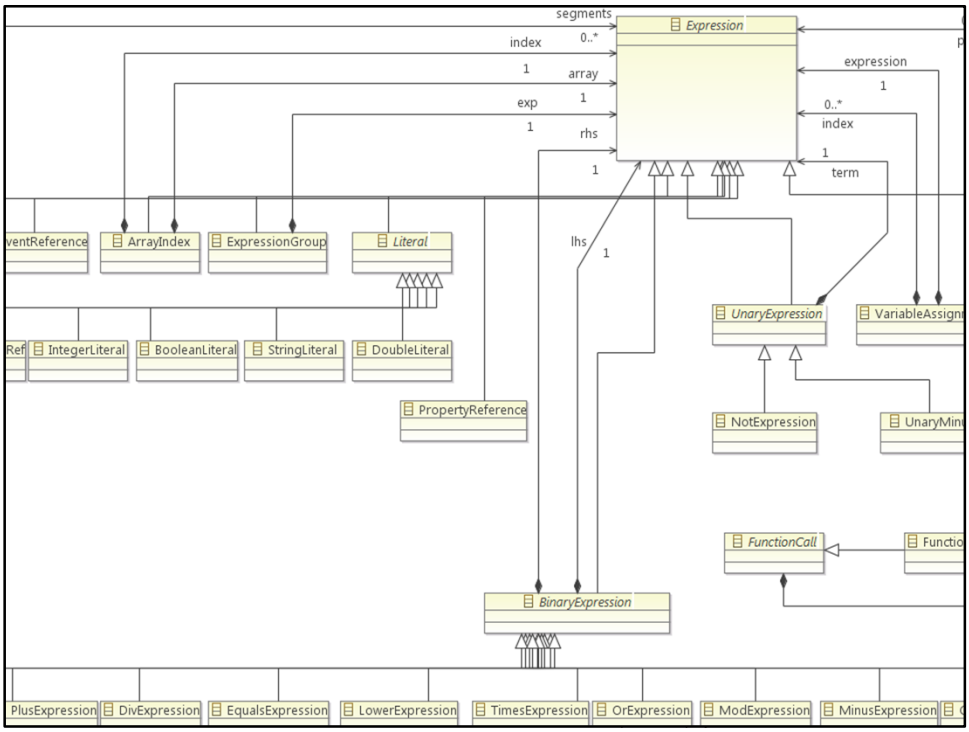

```
configuration BlinkArduino
{
group led : LedArduino
set led.io.digital_output.pin = DigitalPin:PIN_13
// The timer
instance timer : TimerArduino
// The blink application
instance app : Blink
connector app.HW => led.led.Led
connector app.HW => timer.timer
}
```

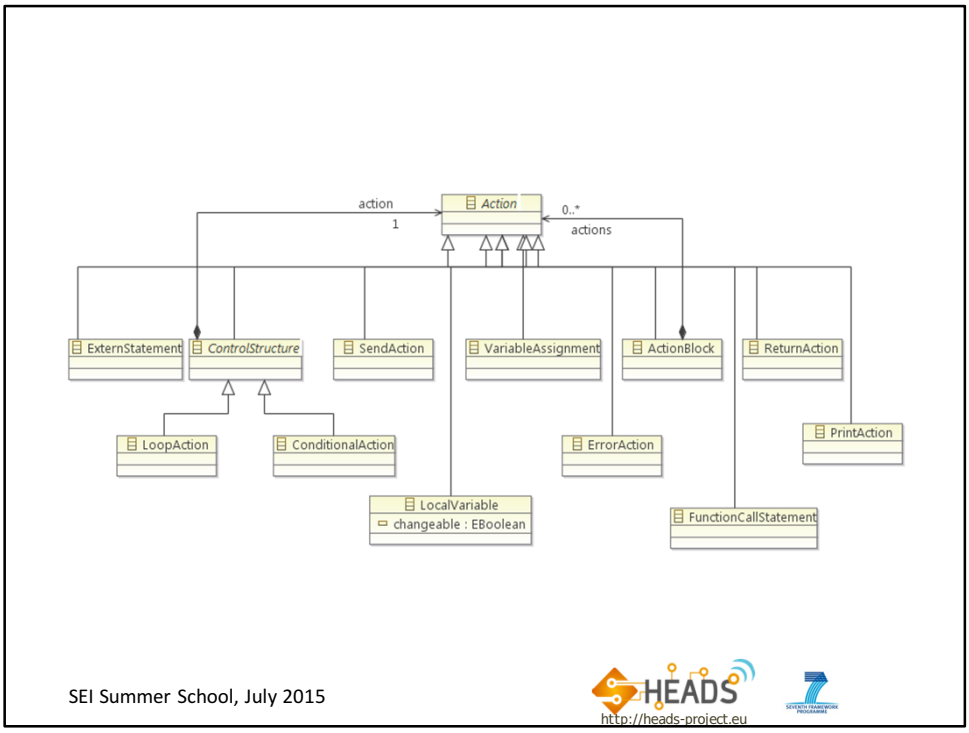

ThingML Editor

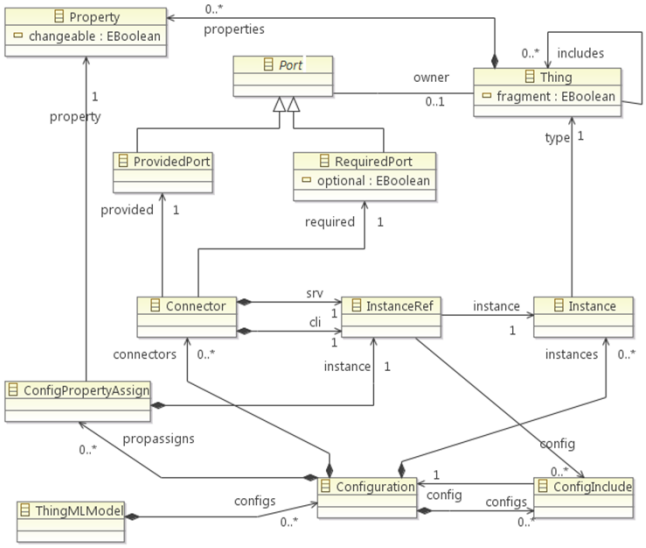



Property
changeable : EBoolean
properties
0..*
imports
ThingMLModel
types
Type
functions
Function
parameters
Parameter
includes
Enumeration
Thing
fragment : EBoolean
messages
Message
PrimitiveType
literals
EnumerationLiteral
owner
behaviour
StateMachine
sends
receives
Port
ProvidedPort
RequiredPort
optional : EBoolean
SEI Summer School, July 2015
HEADS
http://heads-project.eu


Property
changeable : EBoolean
properties
exit
entry
Action
before
after
action
Event
Message
event
message
State
initial
substate
Handler
ReceiveMessage
CompositeState
Region
history : EBoolean
internal
InternalTransition
Transition
region
ParallelRegion
StateMachine
outgoing
incoming
SEI Summer School, July 2015
HEADS
http://heads-project.eu


segments
Expression
index
0..*
1
array
1
exp
1
1
rhs
1
expression
1
0..*
index
1
term
ventReference
ArrayIndex
ExpressionGroup
Literal
lhs
1
UnaryExpression
VariableAssign
Ref
IntegerLiteral
BooleanLiteral
StringLiteral
DoubleLiteral
PropertyReference
NotExpression
UnaryMin
FunctionCall
Functio
BinaryExpression
PlusExpression
DivExpression
EqualsExpression
LowerExpression
TimesExpression
OrExpression
ModExpression
MinusExpression

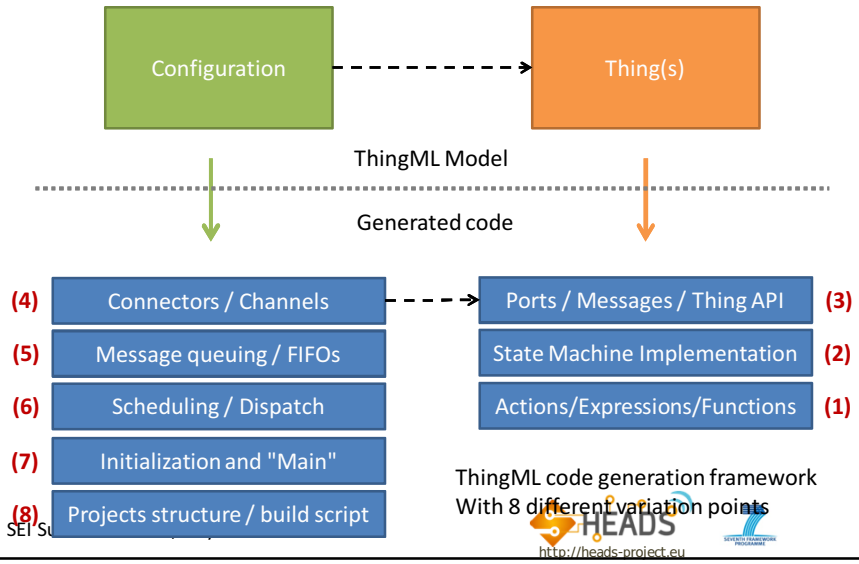

action
1
Action
0..*
actions
ExternStatement
ControlStructure
SendAction
VariableAssignment
ActionBlock
ReturnAction
LoopAction
ConditionalAction
ErrorAction
PrintAction
LocalVariable
changeable : EBoolean
FunctionCallStatement
SEI Summer School, July 2015
HEADS
http://heads-project.eu



Outline

- Background and Motivations
- ThingML Contribution
- The ThingML Language
- **ThingML Transformations**
- ThingML Demo
- Summary

ThingML code generation framework



ThingML code generation framework
With 8 different variation points

(1) Actions / Expressions / Functions

- Scope
  - Depends only on the target language
  - Can be reused for different platforms
- Implementation
  - Visitor on the ThingML meta-model
- Customizable by
  - Implementing a new visitor for a new language
  - Inheriting from an exiting visitor and overriding some of its methods

(2) State machine implementation

- Scope
  - Specific to a specific state machine implementation strategy.
  - Can generate either the complete state machine in the target language or leverage a state machine framework on the target platform
- Implementation
  - Abstract state machine code generator
  - A set of reusable helpers to calculate states, transitions and events according to the common ThingML semantics.
- Customization
  - Implement the abstract state machine generator

(3) Ports / Messages / Thing APIs

- Scope
  - Depends on the language best practices
  - Depends on how components should be "packaged" on the target platform
    - Can generate any custom API for the Things
    - Can generate towards exiting middleware / OS
  - Can/should produce "manually usable" APIs
  - Different generators can be used for different things
- Implementation
  - Visitor on the "Thing" part of the metamodel
  - Helpers to collapse fragments and gathers all the elements of a thing (messages, ports, functions, etc).
- Customization
  - Implement a new visitor for a new target language / platform
  - Inherit from an existing visitor for light customization

(4) Connectors / channels

- Scope
  - Depends on how messages are transported from on thing to the next using the Things APIs
  - Can be local and/or remote, includes the serialization, transport through networks and deserialization
  - Different generators can be used for different ports
- Implementation
  - Abstract generator for serialization, deserialization and transport
- Customization
  - Implement new concrete generators
  - Easy to reuse serialization and just override transport

(5) Message Queuing / FIFOs

- Scope
  - Asynchronous behaviour of messages
  - Can target existing message frameworks or middleware or use custom made FIFOs
  - Different generators can be used for different ports
- Implementation
  - Abstract generator which can be customized
  - Helpers to calculate the sets of messages to be handled (combines fragments and prunes unused messages).
- Customization
  - Inherit and implement the abstract generator

(6) Scheduling / Dispatch

- Scope
  - Implements the main loop of the program, schedules the activation of the components and dispatches the incoming messages
  - Relies on underlying OS and libraries of the target platform.
  - Can generate a custom scheduler for microcontroller applications.
- Implementation
  - Template + Helper
- Customization
  - Create of modify an existing template

(7) Initialization and "main"

- Scope
  - Generate the entry point and initialize the components and connectors
  - Depends on the target languages and traget frameworks
- Implementation
  - Template + Helper providing the set of components and connectors to instantiate
- Customization
  - Create or modify a template

(9) Project structure / build script

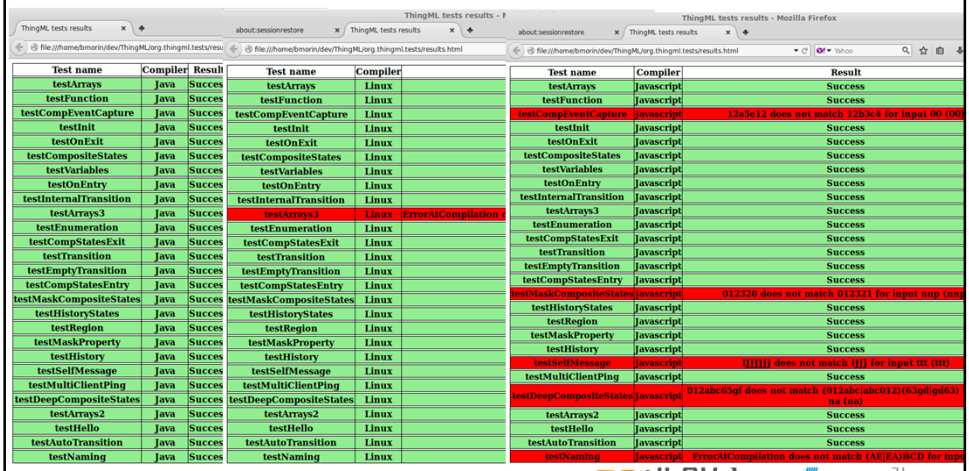
- Scope
  - Produce the right file structure, additional project files and/or build scripts
  - Can be customize to fit a specific target environment (makefiles, maven files, etc)
- Implementation
  - Abstract generator with access to buffers containing all the generated code.
- Customization
  - Create a concrete generator. Possibility to use templates.

Consistency checking

- A suite of tests (27) written in ThingML
  - Takes characters as inputs (or nothing)
  - Generates characters as outputs
- A set of platform specific harness (also in ThingML)
  - For C/Linux, Java, Node.js
  - Write outputs into a file (or simply crash if severe bug)
- Discussion
  - Testing ThingML using ThingML: possible bugs that hide each others…
  - …less and less probable as the number of compilers augments

Current test results

- Java: 100%, C/Linux:96%, Node.js (started 10/14): 81%, now 100%



ThingML tests results - Mozilla Firefox

Test name	Compiler	Result
testArrays	Java	Succes
testFunction	Java	Succes
testCompEventCapture	Java	Succes
testInit	Java	Succes
testOnExit	Java	Succes
testCompositeStates	Java	Succes
testVariables	Java	Succes
testOnEntry	Java	Succes
testInternalTransition	Java	Succes
testArrays3	Java	Succes
testEnumeration	Java	Succes
testCompStatesExit	Java	Succes
testTransition	Java	Succes
testEmptyTransition	Java	Succes
testCompStatesEntry	Java	Succes
testMaskCompositeStates	Java	Succes
testHistoryStates	Java	Succes
testRegion	Java	Succes
testMaskProperty	Java	Succes
testHistory	Java	Succes
testSelfMessage	Java	Succes
testMultiClientPing	Java	Succes
testDeepCompositeStates	Java	Succes
testArrays2	Java	Succes
testHello	Java	Succes
testAutoTransition	Java	Succes
testNaming	Java	Succes

Test name	Compiler	
testArrays	Linux	
testFunction	Linux	
testCompEventCapture	Linux	
testInit	Linux	
testOnExit	Linux	
testCompositeStates	Linux	
testVariables	Linux	
testOnEntry	Linux	
testInternalTransition	Linux	
testArrays3	Linux	ErrorAtCompilation
testEnumeration	Linux	
testCompStatesExit	Linux	
testTransition	Linux	
testEmptyTransition	Linux	
testCompStatesEntry	Linux	
testMaskCompositeStates	Linux	
testHistoryStates	Linux	
testRegion	Linux	
testMaskProperty	Linux	
testHistory	Linux	
testSelfMessage	Linux	
testMultiClientPing	Linux	
testDeepCompositeStates	Linux	
testArrays2	Linux	
testHello	Linux	
testAutoTransition	Linux	
testNaming	Linux	

Test name	Compiler	Result
testArrays	Javascript	Success
testFunction	Javascript	Success
testCompEventCapture	Javascript	12a5e12 does not match 12b3c4 for input 00 (00)
testInit	Javascript	Success
testOnExit	Javascript	Success
testCompositeStates	Javascript	Success
testVariables	Javascript	Success
testOnEntry	Javascript	Success
testInternalTransition	Javascript	Success
testArrays3	Javascript	Success
testEnumeration	Javascript	Success
testCompStatesExit	Javascript	Success
testTransition	Javascript	Success
testEmptyTransition	Javascript	Success
testCompStatesEntry	Javascript	Success
testMaskCompositeStates	Javascript	012320 does not match 012321 for input nnp (nnp
testHistoryStates	Javascript	Success
testRegion	Javascript	Success
testMaskProperty	Javascript	Success
testHistory	Javascript	Success
testSelfMessage	Javascript	IJJJJJJ does not match IJJJ for input ttt (ttt)
testMultiClientPing	Javascript	Success
testDeepCompositeStates	Javascript	012abc65gf does not match (012abc|abc012)(63gd|gd63)na (na)
testArrays2	Javascript	Success
testHello	Javascript	Success
testAutoTransition	Javascript	Success
testNaming	Javascript	ErrorAtCompilation does not match (AE|EA)BCD for inp

Outline

- Background and Motivations
- ThingML Contribution
- The ThingML Language
- ThingML Transformations
- **ThingML Demo**
- Summary

DEMO !

Outline

- Background and Motivations
- ThingML Contribution
- The ThingML Language
- ThingML Transformations
- ThingML Demo
- **Summary**

Summary

- ThingML is open-source
  - http://www.thingml.org
- There are some tutorials and examples
- The language is stable
  - Based on state of the art modelling concepts
  - No significant changes in the meta-model in the last year
  - Includes generic extension mechanisms for experimenting
- Currently 4 fully supported targets
  - Arduino, Posix, Java, NodeJS
- The comparator framework is being re-writen
  - Will allow for easy customization of the generated code

Thanks for your attention!

- Questions?

- More questions: barais@irisa.fr franck.fleurey@sintef.no
- More info: http://www.thingml.org

Consortium

Outline

- Background and Motivations
- Developing with Intel Edison
- **Typescript on top of Intel Edison**
- Node-red with Intel Edison
- Distributed deployment using Kevoree

Why?

HEADS
Heterogeneous and Distributed Services for the Future Computing Continuum

Using **state of the art software engineering practices**

A support of typescript for Intel Edison

A viewpoint for service developer

Why?

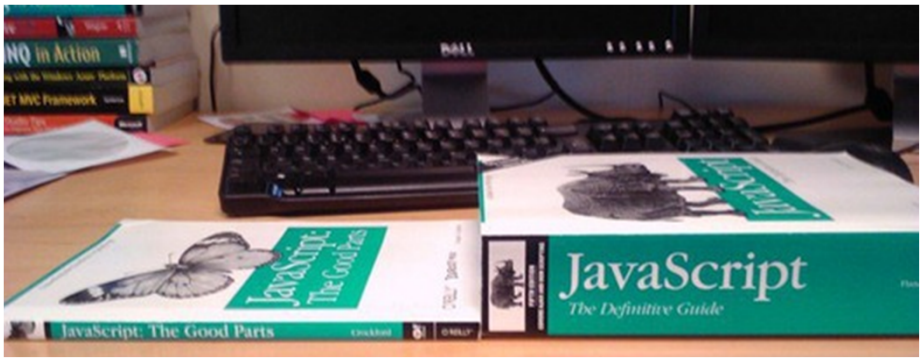
What is TypeScript?

- Free and open source, strongly supported by Microsoft
- Based on ecmascript 4 + ecmascript 6
- Created by the father of C# Anders Hejlsberg
- A superset of JavaScript
- To answer why we need JavaScript+, we need to understand what's wrong with vanilla JavaScript

Why?

What is the problem ?

- Why do people hate working in JavaScript?



Using state of the art software engineering practices ;)

Why?

What is the problem ?

- JS is designed for small things
- We now use to do big things
- But JavaScript is not suited for building large applications

- Your JavaScript code gets complex; it becomes extremely unwieldy

SEI Summer School, July 2015

HEADS
http://heads-project.eu

Why?

Let's look at TypeScript

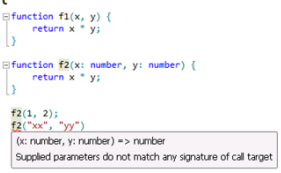
- To get started with TypeScript, grab it from http://typescriptlang.org
- Let's look at TypeScript, first the core concept…

Why?

TypeScript - first glance - optional strong type checking

- ```
  // js
  function f(x, y) {
    return x * y;
  }
  ```
- ```
  // ts
  function f(x : number, y : number) : number {
    return x * y;
  }
  ```

```
// Type information is enforced in design and
// compile time, but removed at runtime
```



Why?

TypeScript features

- Static Type Checking
- Modules and Export
- Interface and Class for traditional Object Oriented Programming

- Works with all your existing JavaScript libraries
- Low learning overhead compared to similar JavaScript systems (*CoffeeScript* or *Dart*)

- Amazing Visual Studio, eclipse or IntelliJ tooling
- Outstanding team and refactoring scenarios

Expected benefits?

Summary - why TypeScript

- Have to learn one more thing - there is a learning curve, very easy if you already know JavaScript, or if you know C# or Java very well.
- You still have to learn JavaScript - Understanding how TypeScript converts to JavaScript will teach you better JavaScript
- Some definition files don't exist or incomplete, but I think this will vanish very quickly. These aren't hard to write if you really need something.

- Modules and classes enable large projects and code reuse
- Compile-time checking prevents errors
- Definition files for common JavaScript libraries makes them very easy to work with, and provides strong type checking
- Source map debugging makes debug easy

HEADS
Heterogeneous and Distributed Services for the Future Computing Continuum

How ?

MRAA

- Libmraa is a C/C++ library with bindings to JavaScript & python to interface with the IO on Galileo, Edison & other platforms, with a structured and sane API where port names/numbering matches the board that you are on

- We need an interface definition for libmraa (mraa.d.ts)
  - Generate mraa.d.ts from .h file

https://github.com/HEADS-project/mraa_hpp2ts_generator

  - Or get it from github

git clone https://github.com/HEADS-project/mraa

Test it

- First install the node.d.ts

> npm install tsd -g #https://github.com/borisyankov/DefinitelyTyped

and next you can download it in typing
> tsd query node -a install #Download node.d.ts

Use the following command to compile your typescript:
> tsc --module commonjs AioA0.ts

next you can install mraa in this folder:

> npm install mraa

finally you can run the samples e.g. AioA0:

> sudo node AioA0.js

Using MRAA definition for typescript

```
///<reference path='typings/node/node.d.ts' />
///<reference path='../../src/typescript/mraa.d.ts' />

var m = require('mraa'); //require mraa
console.log('MRAA Version: ' + m.getVersion()); //write the mraa version to the console

var analogPin0 = new m.Aio(0); //setup access analog inpuput pin 0
var analogValue = analogPin0.read(); //read the value of the analog pin
console.log(analogValue); //write the value of the analog pin to the console
```

More complex example

```
///<reference path='collections/collections.d.ts' />
///<reference path='node/node.d.ts' />
///<reference path='node/express.d.ts' />
///<reference path='mraa.d.ts' />

import fs = require("fs")
import http = require("http")
import path = require("path")
import mraa= require("mraa")
import express = require("express")
import index = require("./routes/index")
import user = require("./routes/user")

var col = fs.readFileSync('./collections/collections.js','utf8');
eval(col);
....
```


Initial conclusion - if I have to make a decision for you...

- If you see yourself using more JavaScript. You have to look at TypeScript.
- If you and your team has to work on JavaScript together, you have to look at TypeScript.
- Once you've done the initial hard work and converted a project. You can't stand going back.

Your turn

Outline

- Background and Motivations
- Developing with Intel Edison
- Typescript on top of Intel Edison
- **Node-red with Intel Edison**
- Distributed deployment using Kevoree

Why Node-Red?

We have
- Processors for editing Words
- Spreadsheets for working with Numbers
- Powerpoint for arranging Pictures and Ideas

We don't have a simple tool for coordinating Events
- Business events - status of processes, alerts from machines
- Social events - tweets, alerts
- Internet of Things events - temperatures, weather, lights, doors

- Something that anyone can use to build
  - situational applications

... Node-RED can fill that gap

Node-RED is

- an application composition tool.
- a lightweight proof of concept runtime.
- easy to use for simple tasks.
- simple to extend to add new capabilities and types of integration.
- capable of creating the back-end glue between social applications.
- a great way to try
- "can I just get this data from here to here ?"
- "...and change it slightly on the way"

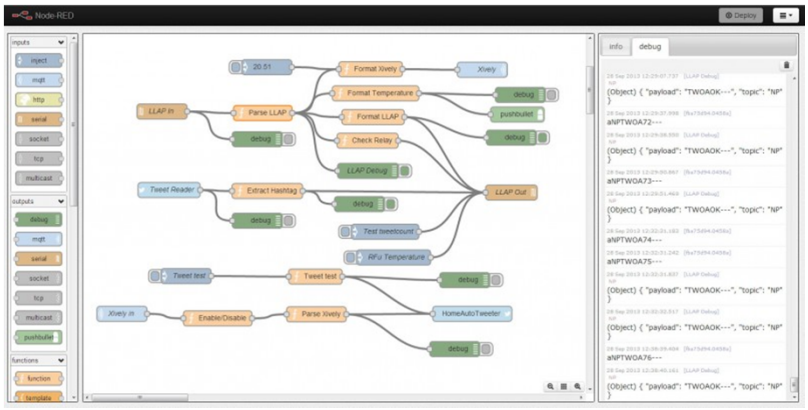
Node-RED is NOT

- an enterprise strength application runtime.
- a dashboard with widgets.
- a mobile application builder.
- the answer to life the universe and everything...
  - not yet anyway
- Node-Red is already capable of connecting to many things,

Foundations

- node.js, v8 engine - fast
- event driven, asynchronous io - it's all about events
- single threaded, event queue - great for fairness
- javascript front and back - only the one language

Node-Red



Extensible

- Easy to wrap any npm module into a palette node*
- Each node defined in a pair of files:
  - .js : server—side behaviour
  - .html : appearance in editor 8: help

** once we get the API documented*

A visual tool for wiring the internet of things

- New developers and education
  - Short learning curve
  - Easy to use
  - Low barrier to entry
- App Developers
  - Rapid prototyping
  - Easy to extend with richer/bespoke functionality
  - tools and applications
- Community Developers
  - Open standards
  - Flexibility
  - Ability to share
- Hardware Hackers
  - Easy to integrate with existing Runs on Raspberry Pi, Beaglebone, other low power devices
  - Works with Arduino, etc
  - Easy to add devices

Your turn

Outline

- Background and Motivations
- Developing with Intel Edison
- Typescript on top of Intel Edison
- Node-red with Intel Edison
- **Distributed deployment using Kevoree**

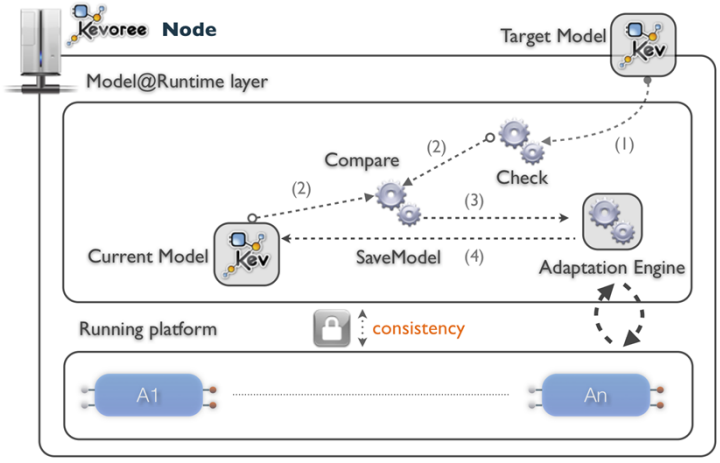
Next steps

- Complex deployment of HD-Services
  - A configuration language for managing module deployment and reconfiguration

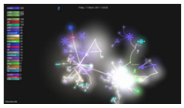
Kevoree (www.kevoree.org)

- Kevoree project aims at supporting dynamic adaptation in distributed system (What Benjamin calls Management layer)
  - **MDE@Runtime**
    - Shared model representation for distributed nodes
    - Offline & online operation, *compute@Model* level, apply @Runtime
  - **Component-based**
    - Actor semantics on each ports to closely separate component behavior
    - Communication semantics between component in channel
    - Support component reconfiguration (parametric, architectural, behavioral
  - **Continuous Design, type definition continuous definition**
    - Hot (re-)deploy & provisioning
  - **Heterogeneity management with NodeType**

Kevoree general overview



An OSGi-like framework for HD-Services








Kevoree

M@R Runtimes for Distributed and Heterogeneous adaptive systems

Your turn

Thank you!

- Questions?