UMR IRISA

Model Driven Engineering for the Internet of Things

Prof. Jean-Marc Jézéquel

Director of IRISA

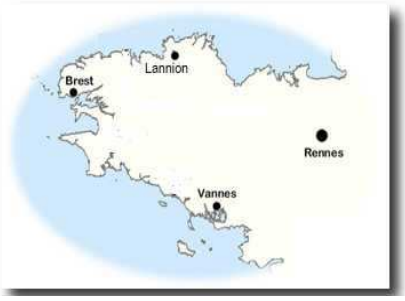
jezequel@irisa.fr

http://people.irisa.fr/Jean-Marc.Jezequel/

UNIVERSITÉ DE RENNES 1

IRISA is the Common Research Lab in Informatics in Brittany of:

- 4 sites in Brittany
  - *750 people, 35 teams*



- Two main research domains

Computer science

Automatic, signal/image processing and robotics

- Many new topics addressed

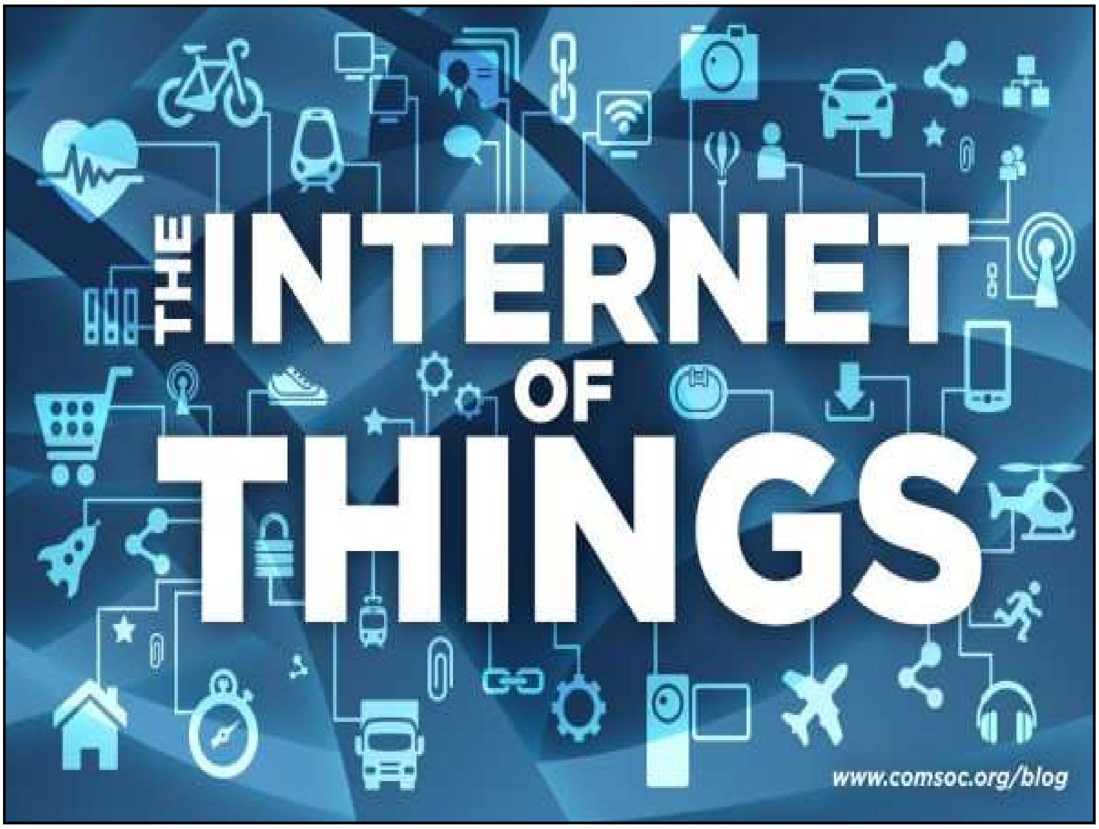
Cloud computing, Web services, Use and SSH, Green computing, Neuroimaging, Security, Home automation…

UMR IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

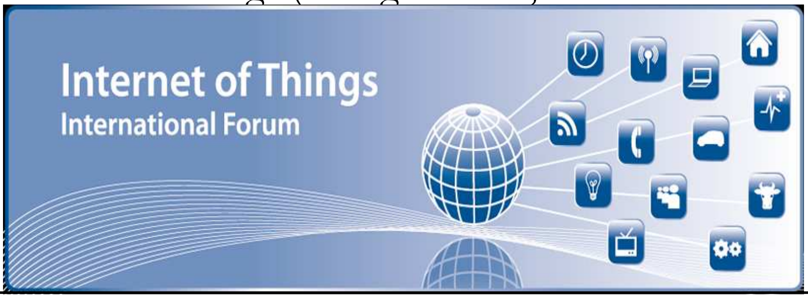
Outline

- Background on IoT and Adaptive Systems
- Principle of Models @ runtime
- Kevoree: A Concrete implementation of Distributed Models@Runtime for IoT



Internet of Things

- 75 billions of connected devices by 2020
  - **Everything** is connected!
- Encompasses
  - Smart home/building/hospital/office/farm
  - Intelligent cars/trucks…
  - Wearabble things (Google Glass, fitness monitors…)



5

A multi-billion dollar industry to be

- Example scenario (from Control4):
  - My house will check the outdoor temperature and note that it's near freezing. Knowing that I leave the house at 8am, it would open the garage door five minutes before and start my smart car so it's nice and toasty when I'm ready to head to the office.
- Endless possible applications
  - When everything is connected to everything else
    - Including IoS and the Cloud
  - And driven with smart phones

6

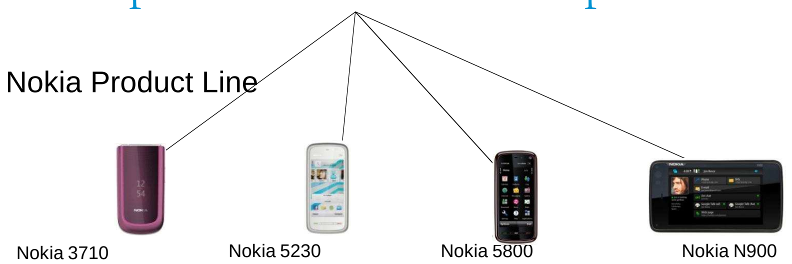
Software for the IoT

- Eternal systems
  - Cannot be stoped/rebooted upon upgrade
- Many, many different applications
  - Though large subsets are shared among applications
  - Aka *product lines*
- Development process: must be better than *Agile*
  - Not enough programmers to care for changes in
    - User needs, Hardware (failures, new hardware…)
- Towards Hyper-Agility *(Autonomic Computing)*
  - For Dynamically Adaptive Systems

Background: Software Product Lines (SPL)

- Families of related software products
- The products have a *common* part and a *variable* part

Nokia Product Line



Nokia 3710	Nokia 5230	Nokia 5800	Nokia N900
• color = {plum}	• color = {white,black}	• color = {red, blue, black}	• color = {black}
• keypad,large keys	• touch screen	• keypad, touch screen, stylus	• full keyboard, touch screen, stylus
• MMS	• MMS	• MMS	• MMS
• GPS	• GPS	• GPS	• GPS
• TCP/IP	• TCP/IP	• TCP/IP	• TCP/IP
• FM Radio	• FM Radio	• FM Radio	• Internet calling

Variable features:
- color
- input method
- radio
- Internet calling

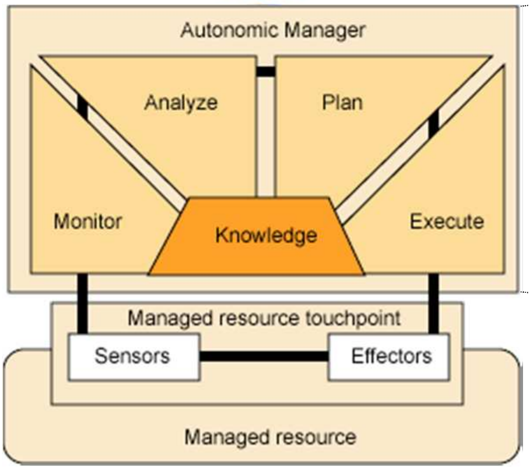
…

Common features

Background: Agile Manifesto

- Manifesto for Agile Software Development
  - **Individuals and interactions**
    - over processes and tools
  - **Working software**
    - over comprehensive documentation
  - **Customer collaboration**
    - over contract negotiation
  - **Responding to change**
    - over following a plan

Background: Autonomic Computing



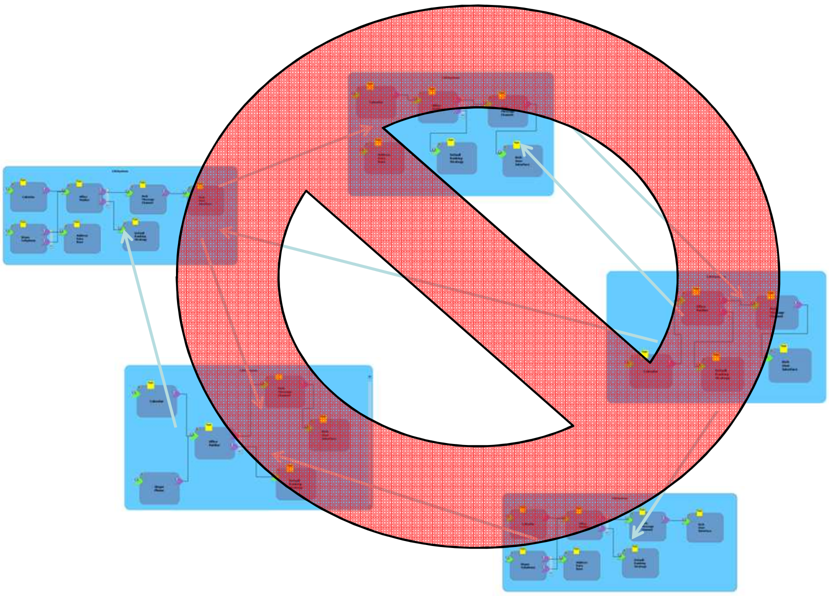
Beyond Agility: Adapting to User Needs, Continuously

Jeffrey O. Kephart, David M. Chess, *"**The Vision of Autonomic Computing**"*, IEEE Computer, vol. 36, no. 1, pp. 41-50, Jan. 2003, doi:10.1109/MC.2003.1160055

Issues of Dynamic Adaptive Software (DAS) for IoT

- Development:
  - Adaptation logic often embedded into application logic
  - Adaptation logic hard-coded using low-level APIs
  - Readability, maintainability, and communication with other stakeholder not easy
- Exponential growth of possible configurations
  - N features, N tending to be larger and larger
    - Since any feature can be there/not be there
    - 2^N potential program configurations, 2^N x (2^N-1) transitions
- Reliability: Lives may depend on the system
  - System used by « normal » people (not computer scientists)
    - Rebooting the system cannot be a solution
  - how to validate every configuration before actual adaptation, with no possibility to specify all the possible configurations

Dynamic Adaptive System (DAS) as a state-machine



Hyper-Agility: using MDE @ runtime

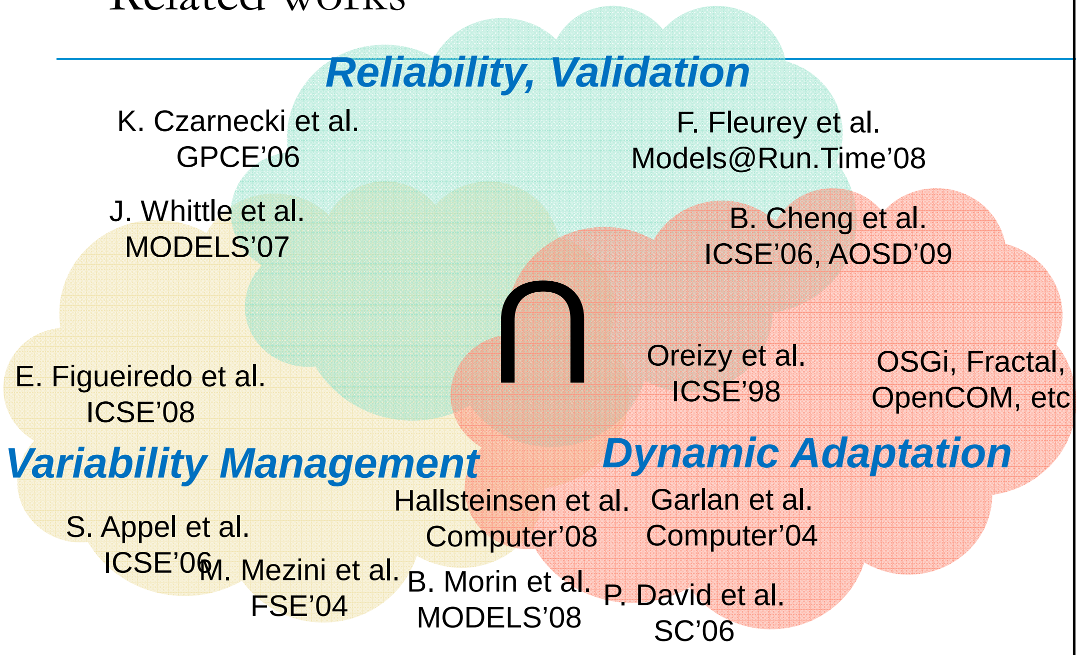
Adopting a DSPL approach

- Focus on **variability**, not on configurations
  - **variability modeling**: Feature Diagrams, CVL
- Build (derive) configurations from **models**
  - when needed: JIT, On demand at runtime, caching…
- **Validate** configurations before actual adaptation
- **Automate** the reconfiguration process

➢ **Think of it as the Agile Manifesto @ runtime**

- Individuals and interactions
- Working software
- Customer collaboration
- Responding to change

Related works



Reliability, Validation

K. Czarnecki et al. GPCE’06

F. Fleurey et al. Models@Run.Time’08

J. Whittle et al. MODELS’07

B. Cheng et al. ICSE’06, AOSD’09

∩

E. Figueiredo et al. ICSE’08

Oreizy et al. ICSE’98

OSGi, Fractal, OpenCOM, etc

Variability Management

Dynamic Adaptation

Hallsteinsen et al. Computer’08

Garlan et al. Computer’04

S. Appel et al. ICSE’06

M. Mezini et al. FSE’04

B. Morin et al. MODELS’08

P. David et al. SC’06

Outline

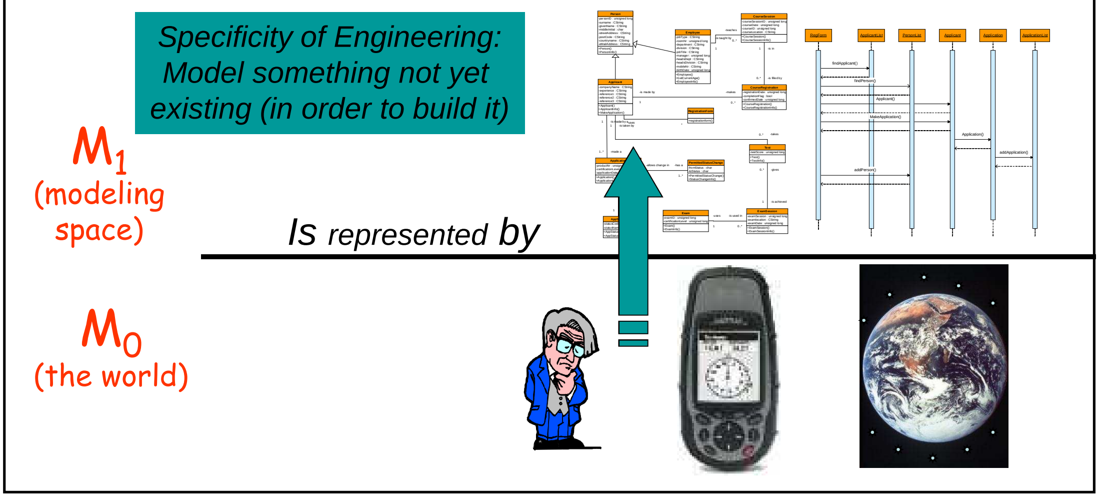
- Background on IoT and Adaptive Systems
- Principle of Models @ runtime
- Kevoree: A Concrete implementation of Distributed Models@Runtime for IoT

Why modeling: master complexity

- Modeling, in the broadest sense, is the *cost-effective use of something in place of something else for some cognitive purpose*. It allows us to use something that is *simpler*, *safer* or *cheaper* than reality instead of reality for some purpose.
- A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.
  - *Jeff Rothenberg.*

Modeling in Science & Engineering

- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*



Specificity of Engineering: Model something not yet existing (in order to build it)

M_1 (modeling space)

Is represented by

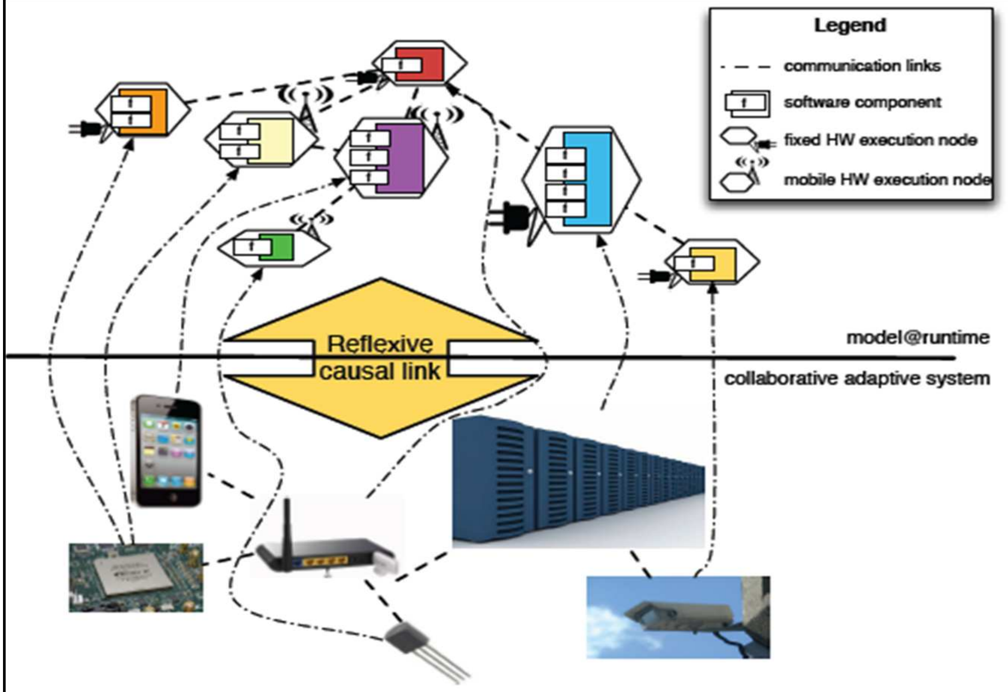
M_0 (the world)

Model and Reality in Software

- Usualy in Engineering, Models & Systems have different natures (bridge drawings and concrete bridge)
  - Sun Tse: *Do not take the map for the reality*
  - Magritte

- Whereas a program *is a* model
- Software Models: from contemplative to productive

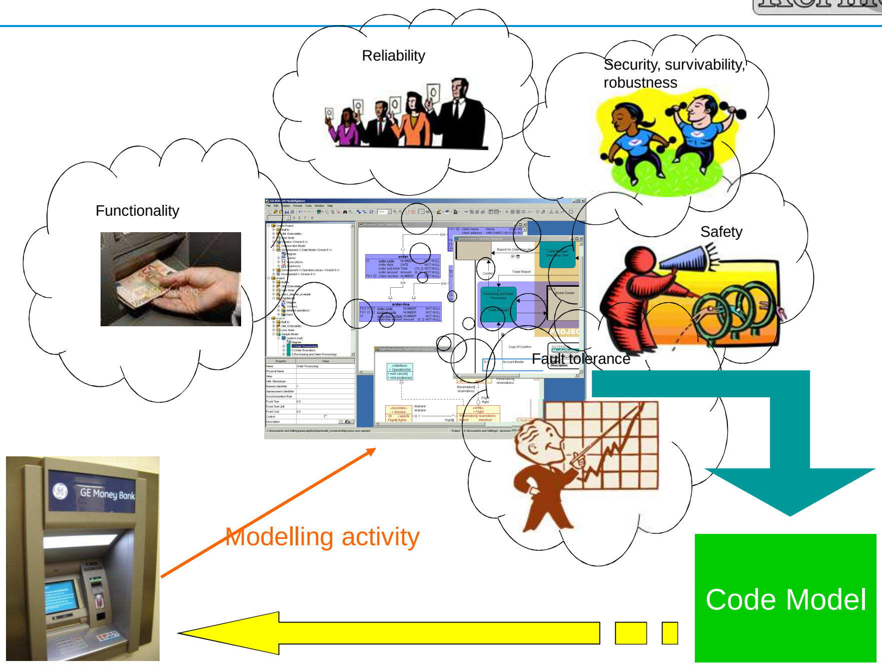
Models @ runtime



Models @ runtime and Reflexion

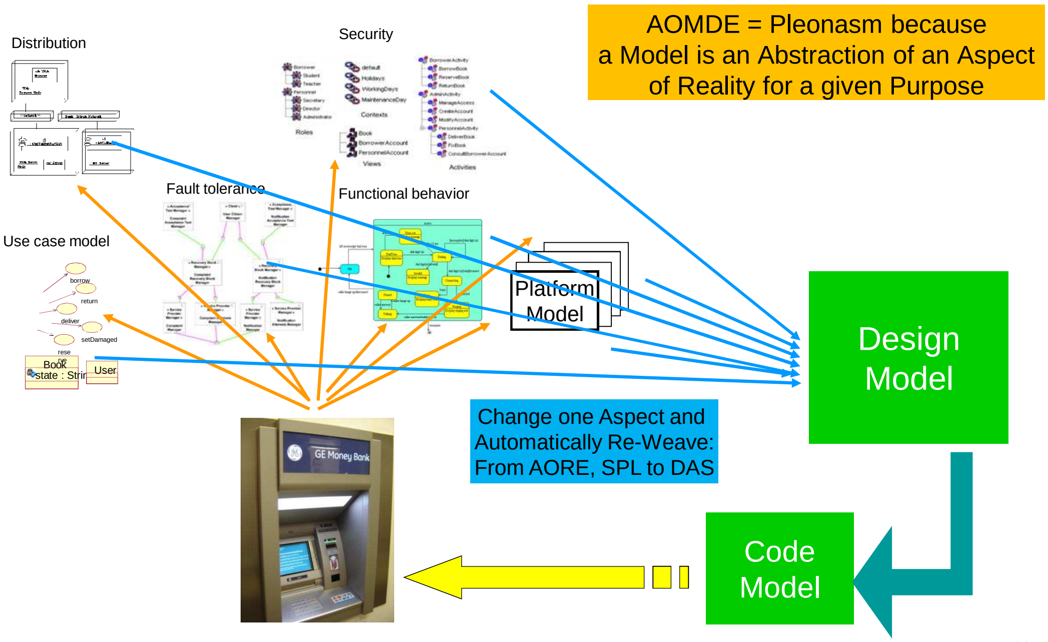
- Just a classical reflexion mecanism
  - But using standard MDE technology instead of hard-coded ad hoc mechanisms
- Reflexion = Introspection + Intersession
  - Introspection: Code -> Model
  - Intersession: Model -> Code
  - In between: room to manipulate models => **MDE**
    - Explore design space, validate potential solutions…
    - Once suitable target model is found, enact it!
  - i.e. mimic the way we humans do ***think***!

Naive Model Driven Engineering



Aspect Oriented Model Driven Engineering

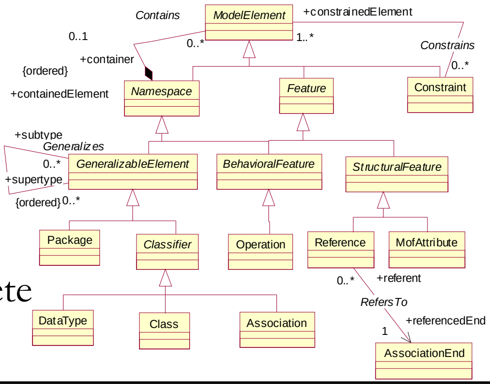
AOMDE = Pleonasm because a Model is an Abstraction of an Aspect of Reality for a given Purpose



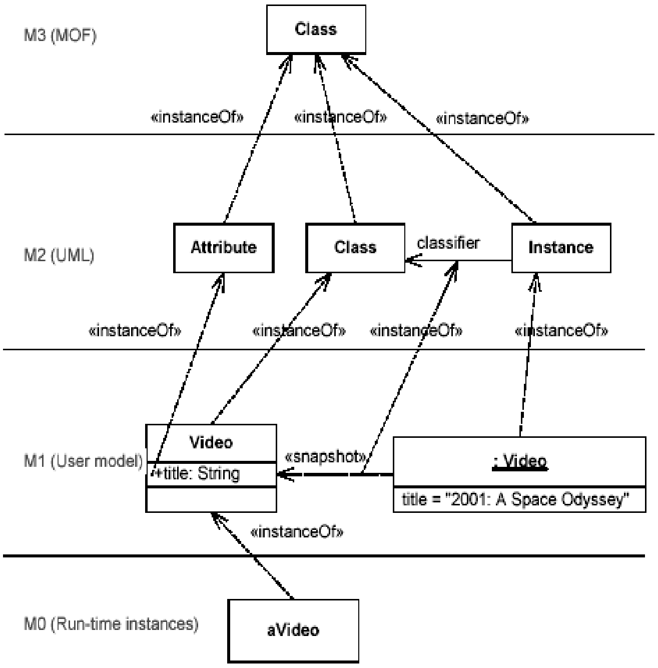
Change one Aspect and Automatically Re-Weave: From AORE, SPL to DAS

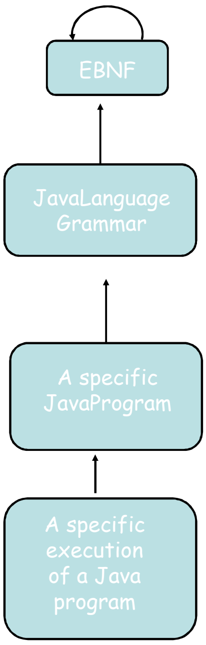
Assigning Meaning to Models

- A model *is no longer* just
  - fancy pictures in e.g. UML to decorate your room
  - a graphical syntax for C++/Java/C#/...
- Tools must be able to manipulate models
  - Let's make a model of what a model is!
  - => ***meta-modeling***
    - & meta-meta-modeling..
  - (Model=aspect) => incomplete

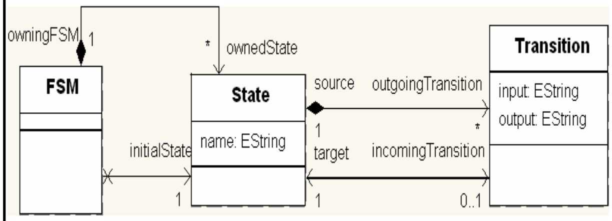


The 4 layers in practice





Key idea: Composition at M2 level



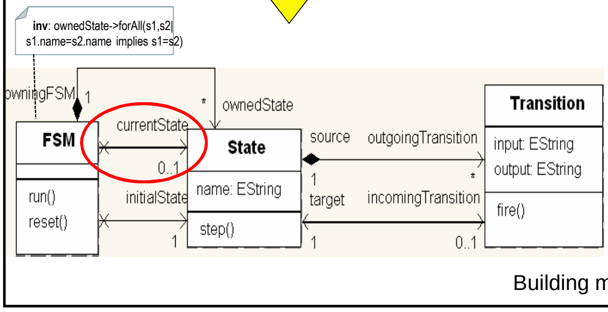
```
context FSM
inv: ownedState->forAll(s1,s2|
 s1.name=s2.name implies s1=s2)
```

```
aspect class FSM{
reference currentState : State
operation run() : Void is do
...
end
...
}
aspect class Transition {
operation fire() : EString is do
source.owningFSM.currentState := target
result := output
}
```

require "FSM.ecore"

require "FSM.ocl"

require "FSMbehavior.kmt"



Model of the resulting composition

Building modular and efficient DSL tools with Kermeta

Kernel Meta-modeling Language

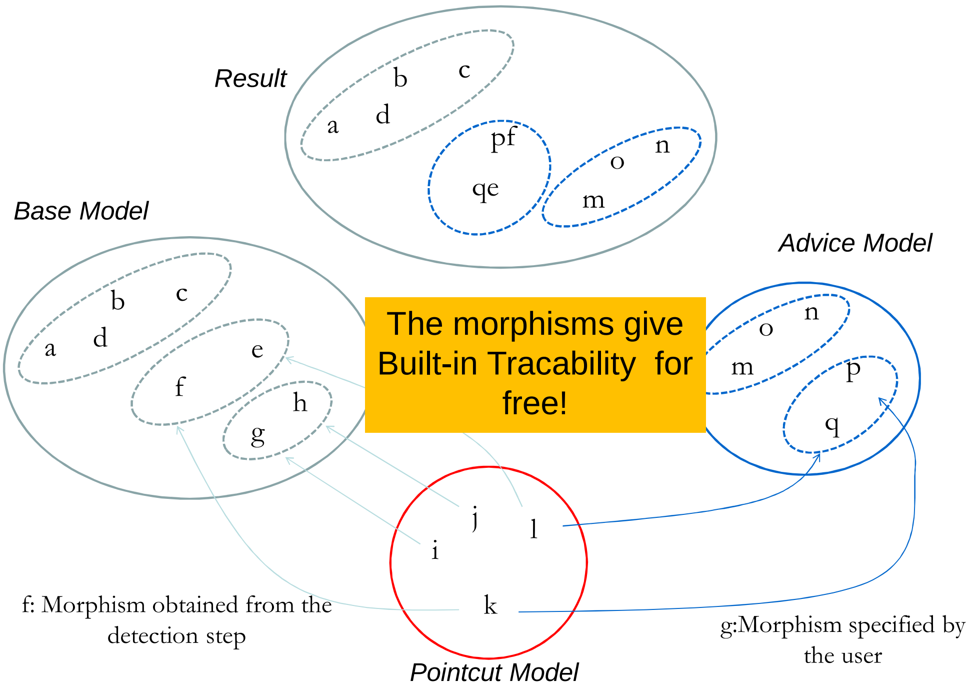
A tool to build tools to build software

- Kermeta is a Model-Oriented Language
  - based on an AO/OO executable meta-modeling paradigm
    - Static typing, generics, functions objects, reflection…
  - First language where models are first class entities
    - Allows interesting questions to be asked: e.g.; what is the type of a model?
- Executable meta-modeling allows:
  - specification of abstract syntax, static semantic (OCL) and dynamic semantics, connection to the concrete syntax.
  - model and meta-model simulation and prototyping
  - model transformation, design level aspect weaving (SmartAdapter, Geko)
- A Kermeta compiler is available to deliver JARs without any dependency with Kermeta

(AO) MDE
=
(AO) Modeling
+
Composition

Why? When? What? Where? How?

Generic Weaving in AOM



The morphisms give Built-in Tracability for free!

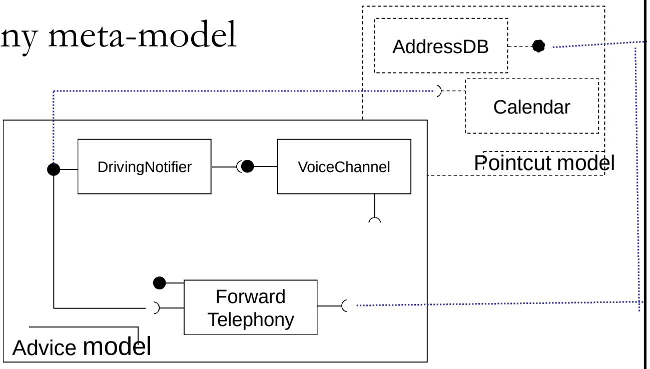
f: Morphism obtained from the detection step

g:Morphism specified by the user

Weaving M1 Aspects: SmartAdapters

Triskell Metamodeling Kernel www.kermeta.org Kermeta

- SmartAdapters: a generic framework for AOM
  - Independent from any meta-model
- Aspect Model
  - What?
    - ➡Advice model = architecture fragment
  - Where?
    - ➡Pointcut model = architecture fragment
  - How?
    - ➡Composition protocol = imperative script



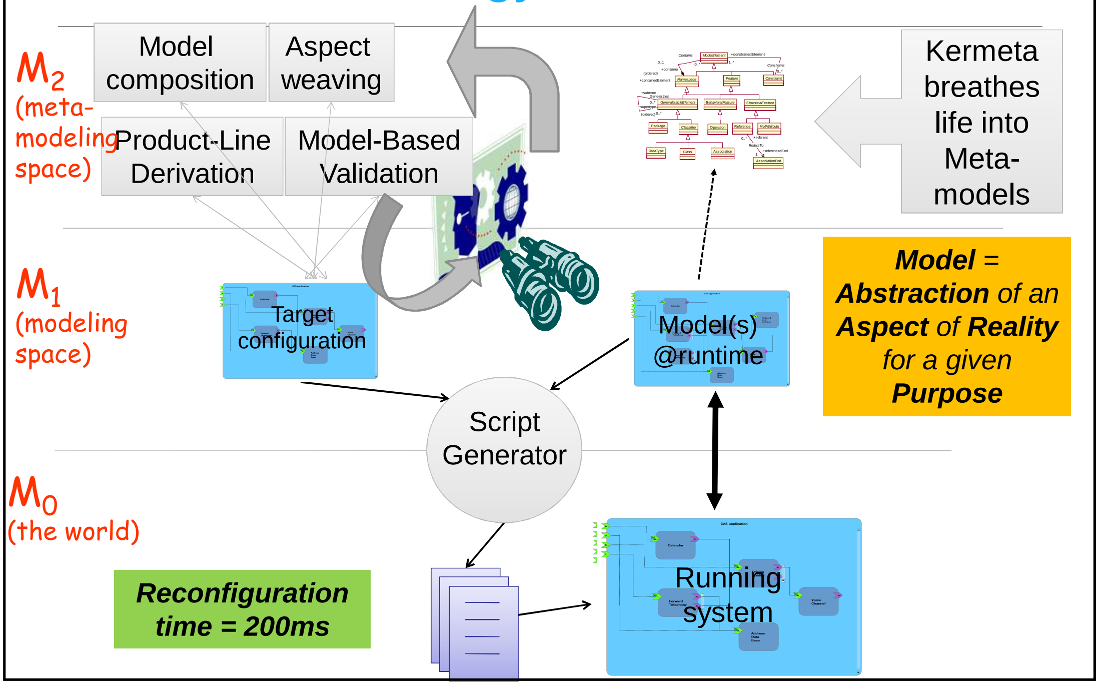
29

DiVA Technical Approach

(FP7 STREP)

- Separate the application-specific functionality from the adaptation concerns in requirements: (D)SPL
  - Feature Model, decorated with Quality Attributes
  - Reasoner works with (*context* x *Feature Model QA*)
- Associate fragments of architecture to Features
  - aka *advice*, + where it can be woven: aka *pointcut*
- Model driven techniques
  - used to raise the level of abstraction of the model of the context
    - updated with CEP
  - model composition (aspect weaving)
    - To compose new architectures on the fly
  - automatic generation of platform reconfiguration script

Models@runtime: Using MDE technology at runtime



M_2 (meta-modeling space)

Model composition

Aspect weaving

Product-Line Derivation

Model-Based Validation

Kermeta breathes life into Meta-models

M_1 (modeling space)

Target configuration

Model(s) @runtime

Model = ***Abstraction*** *of an* ***Aspect*** *of* ***Reality*** *for a given* ***Purpose***

Script Generator

M_0 (the world)

Running system

Reconfiguration time = 200ms

Models@runtime

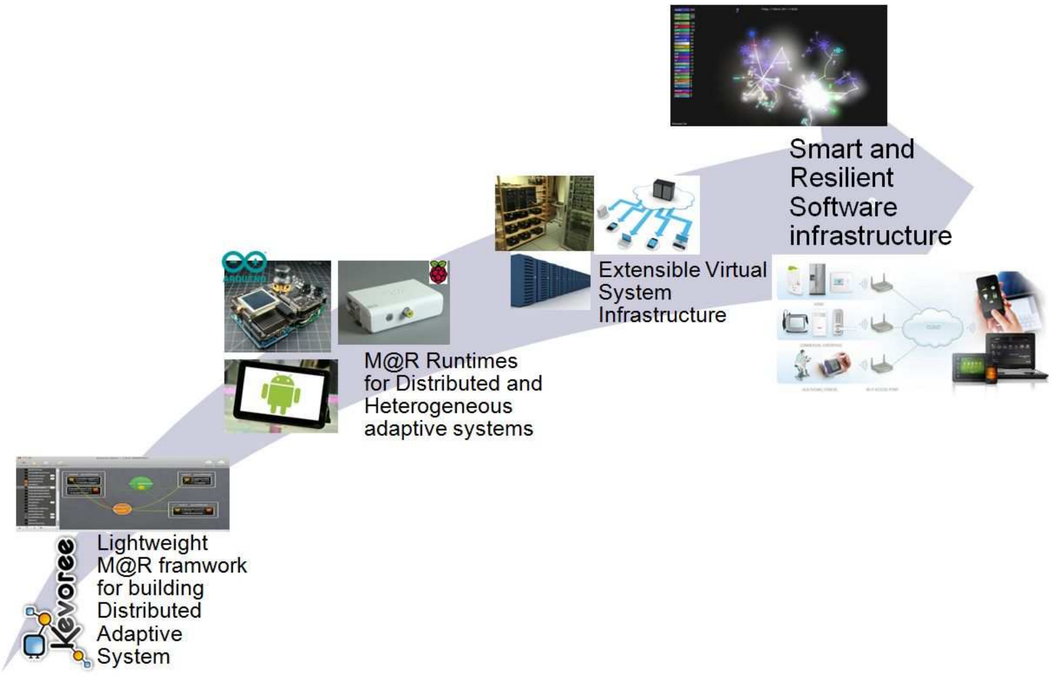
- Aspect as variability units raised at the model level
  - No explicit representation of ALL possible configurations *(n instead of 2^n)*
  - Configurations obtained by weaving most adapted aspects *(on demand @runtime)*
- MDE for automation of model composition
  - Model Based Validation, Generation of reconfiguration scripts
- It works for real!
  - DiVA: Airport Crisis Management, CRM, Home Automation
  - Publications
    - Morin et al., Models@runtime, *IEEE Computer* 10/2009
    - Brice Morin, Olivier Barais, Grégory Nain, and Jean-Marc Jézéquel. -- Taming Dynamically Adaptive Systems with Models and Aspects. -- In *31st International Conference on Software Engineering (ICSE'09)*, Vancouver, Canada, May 2009.
- Follow up => Kevoree = Model@Runtime platform for **distributed** dynamic adaptive system

Outline

- Background on IoT and Adaptive Systems
- Principle of Models @ runtime
- Kevoree: A Concrete implementation of Distributed Models@Runtime for IoT

33

Kevoree overview

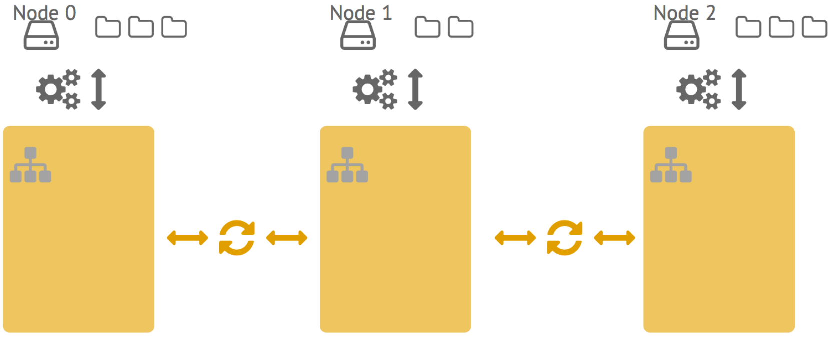

Kevoree
Lightweight M@R framwork for building Distributed Adaptive System
M@R Runtimes for Distributed and Heterogeneous adaptive systems
Extensible Virtual System Infrastructure
Smart and Resilient Software infrastructure

What Is Kevoree ?

- Concrete implementation of Distributed Models@Runtime
- Environment of Development and Execution
- Component-Based and highly Distributed Software Systems
- Leverages Model Driven Engineering *(through KMF)*
- Framework to build your own Dynamic Distributed System…

Distributed Model@Runtime

- One global model replica /node, local causal link
- DDAS synchronization = M@R replicas synchronization
- Synchronization policy depends on case study *(strong, eventual,…)*



Model@Runtime for DDAS

- **MDE@Runtime**
  - Architecture model shared across distributed nodes
  - Offline & online operation, compute@Model, apply @Runtime
- **Component-based**
  - Communication semantics between components in channels
  - Continuous Design (Hot (re-)deploy & provisioning)
- **Heterogeneity management** with NodeType

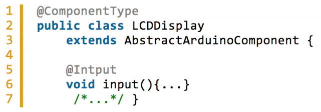
node JS
Java
ANDROID
Cloud
ARDUINO

Background: Software components

- Independent & Self-Contained piece of Software
- Conforms to a Component Model
- Deployment unit
- Functional unit
- Provides / Requires Services
  - defined by a set of methods
- Life-cycle based elements
- **Goals: Abstraction, Separation of concerns, Reuse**

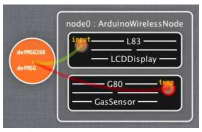
CBSE and Model@Run.time

- Based on a **component** model and software industry best practices
- Framework: Code ->Model and Model ->Code *(continuous design)*

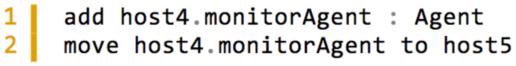


```
@ComponentType
public class LCDDisplay
    extends AbstractArduinoComponent {

    @Intput
    void input(){...}
     /*...*/ }
```



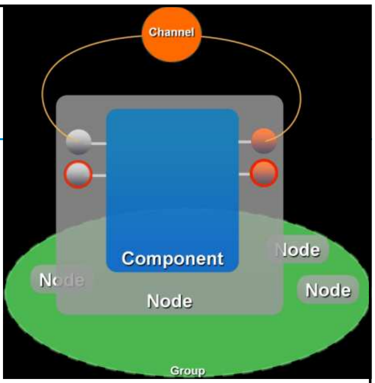
- High level **DSL** to write distributed system adaptations



```
add host4.monitorAgent : Agent
move host4.monitorAgent to host5
```

- Independence of components development and adaptation

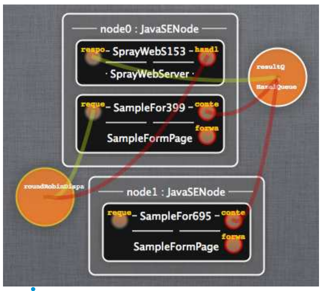
Main concepts in Kevoree



- **Component**
  - Encapsulates domain features
- **Channel**
  - Encapsulates communication semantics
- **Group**
  - Encapsulates model@runtime dissemination semantics
- **Node**
  - Encapsulates adaptation semantics

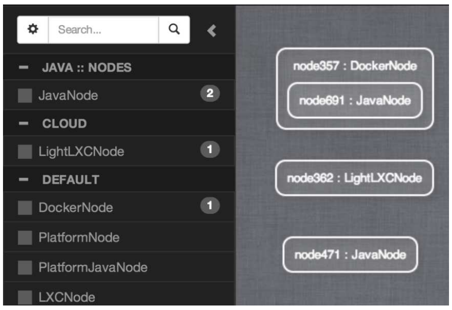
Graphical language

- Explicit topology, components, channels,...
- Nodes: execution platform
  - local adaptation
- Components: functional units
  - Communications *via* via ports
- Channels: communication semantics
  - encapsulate complex communication (N-N)



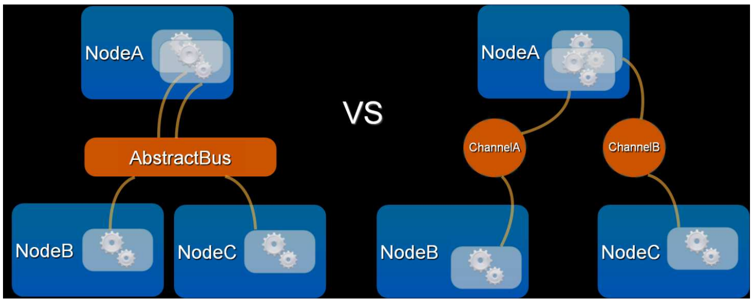
NodeType and Node: runtime mapping

- **NodeType used to declare deploy target**
  - JavaNode
  - AndroidNode
  - DockerCloudNode
  - LXCCloudNode
  - Arduino
  - [...]



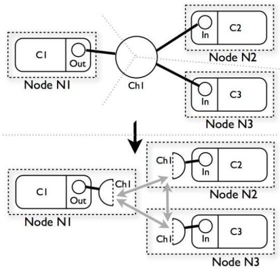
Channel Type

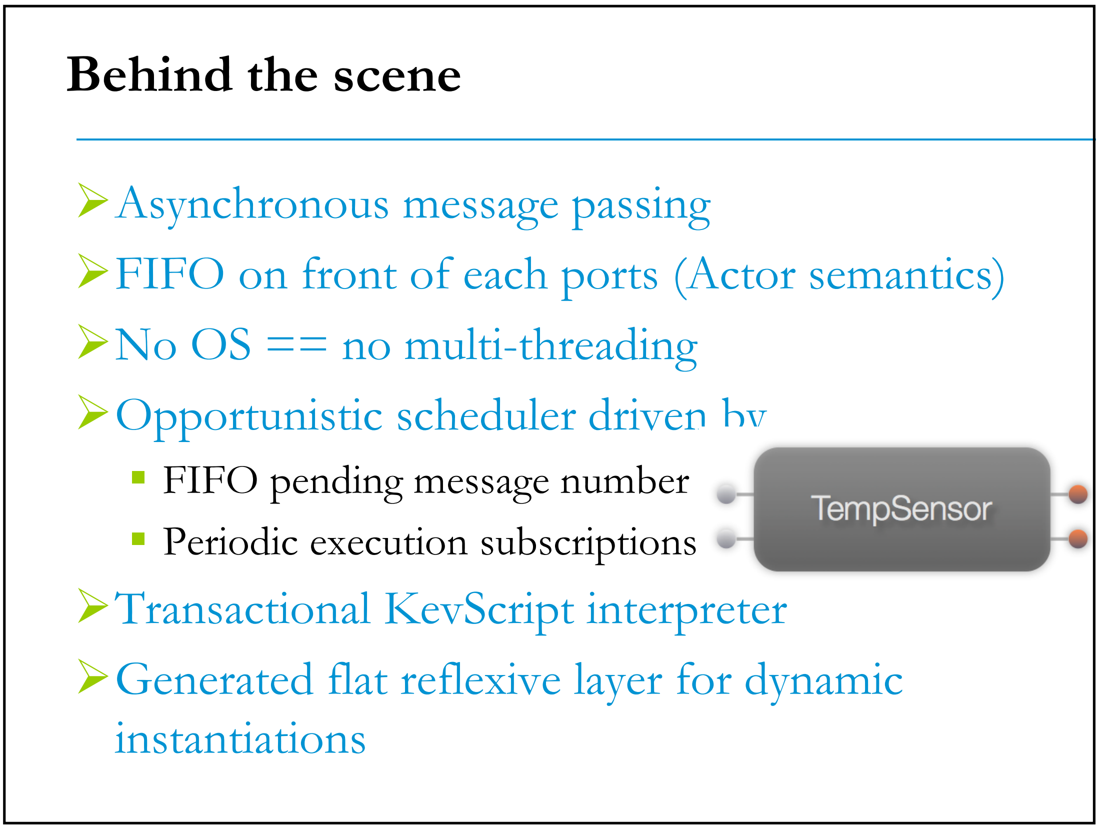
- Extract communication semantics from component
- Explicit complex distribution scenario
- Model communication semantics



Instance fragmentation and concurrency

- Component
  - Actor semantic on each port, channel
  - Instances are isolated in dedicated process
- Channel (aka connector)
  - Fragmented on each linked node
  - Actor semantic on each fragments
  - Local diffusion responsibility

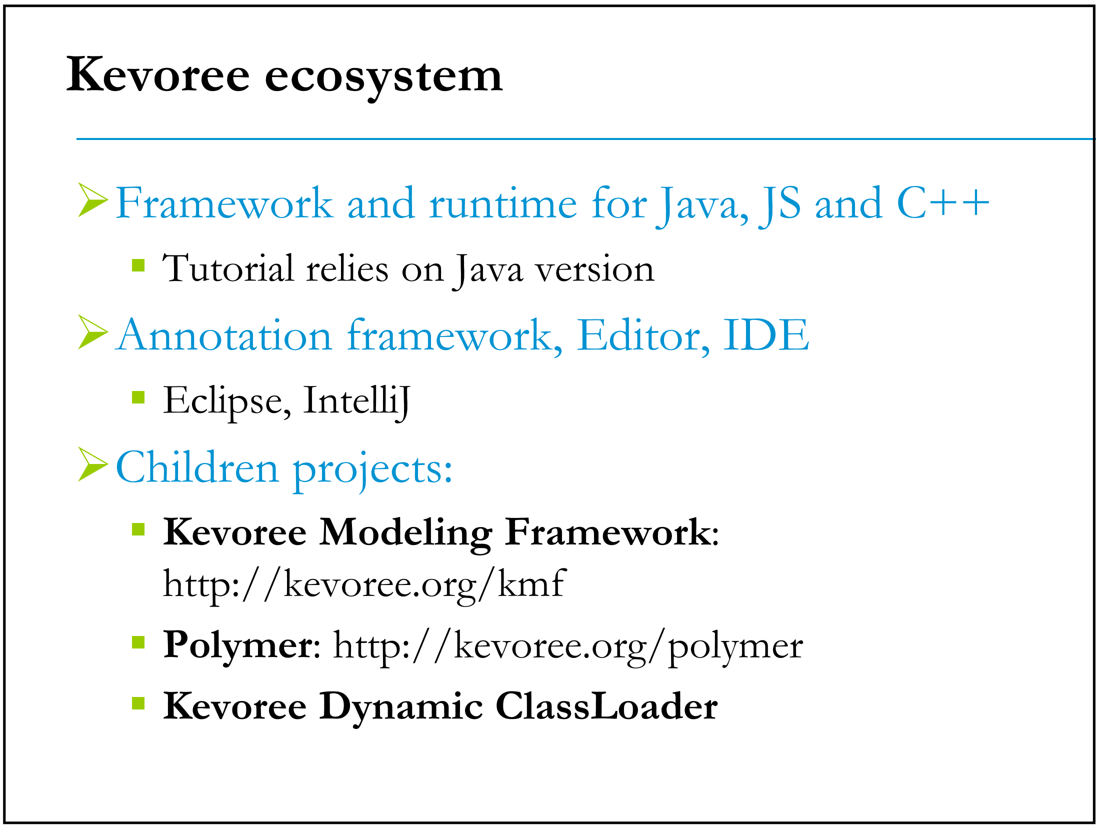




Behind the scene

- Asynchronous message passing
- FIFO on front of each ports (Actor semantics)
- No OS == no multi-threading
- Opportunistic scheduler driven by
  - FIFO pending message number
  - Periodic execution subscriptions
- Transactional KevScript interpreter
- Generated flat reflexive layer for dynamic instantiations

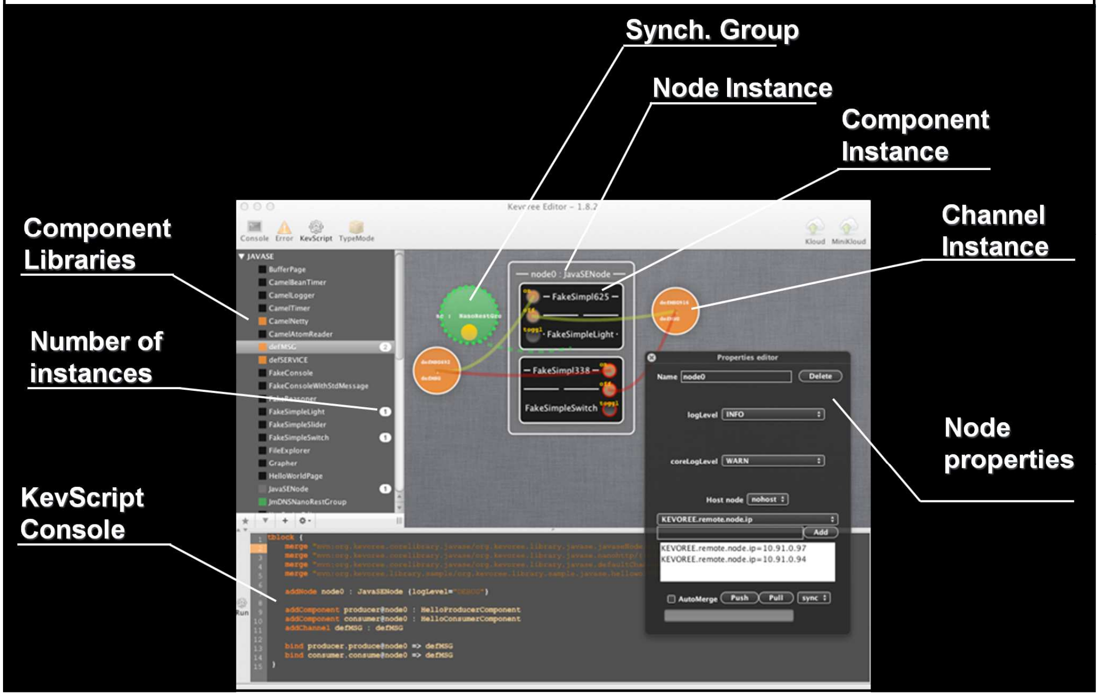
TempSensor



Kevoree ecosystem

- Framework and runtime for Java, JS and C++
  - Tutorial relies on Java version
- Annotation framework, Editor, IDE
  - Eclipse, IntelliJ
- Children projects:
  - **Kevoree Modeling Framework**: http://kevoree.org/kmf
  - **Polymer**: http://kevoree.org/polymer
  - **Kevoree Dynamic ClassLoader**

Kevoree Editor



Name node0

logLevel INFO

coreLogLevel WARN

Host node nohost

KEVOREE.remote.node.ip

KEVOREE.remote.node.ip=10.91.0.97
KEVOREE.remote.node.ip=10.91.0.94

```
tblock {
    merge "mvn:org.kevoree.corelibrary.javase/org.kevoree.library.javase.javaseNode
    merge "mvn:org.kevoree.corelibrary.javase/org.kevoree.library.javase.nanohttp/(
    merge "mvn:org.kevoree.corelibrary.javase/org.kevoree.library.javase.defaultCha
    merge "mvn:org.kevoree.library.sample/org.kevoree.library.sample.javase.hellow

    addNode node0 : JavaSENode {logLevel="DEBUG"}

    addComponent producer@node0 : HelloProducerComponent
    addComponent consumer@node0 : HelloConsumerComponent
    addChannel defMSG : defMSG

    bind producer.produce@node0 => defMSG
    bind consumer.consume@node0 => defMSG
}
```

KevoreeFramework, KevScript

- Scripting language to manipulate models
  - Comments -> // this is a comment
  - Namespace -> namespace space42
    attach node0, node1 space42
  - Repository -> repo
    org.sonatype.org/foo/bar?a=b&c=d
  - Include -> include
    mvn:org.kevoree.library.java:org.kevoree.library.java.javaNode:3.0.0
  - add -> add node0, node1: JavaNode
    add node0.comp0: ToyConsole

KevoreeFramework, KevScript

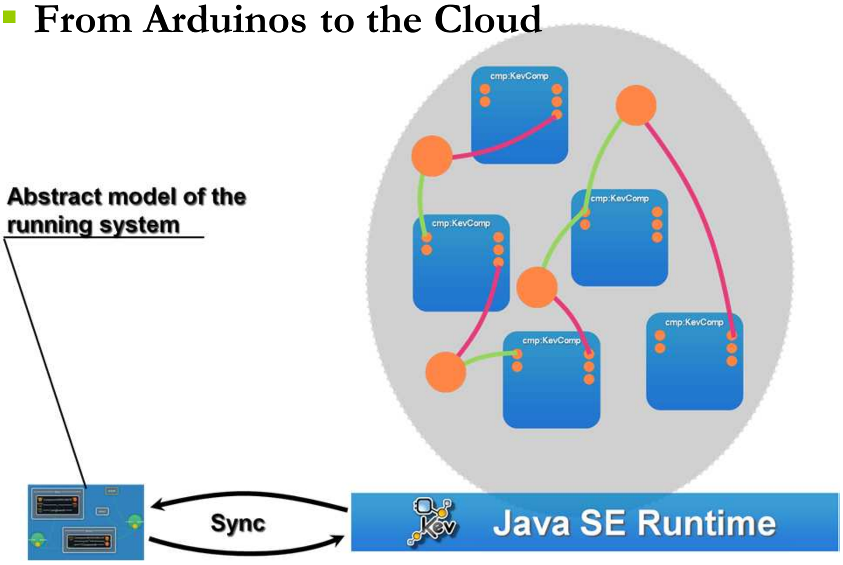
➢ Scripting language to manipulate models

- remove -> remove node0
  remove node0.comp0
- move -> move node0.comp0 node1
- set -> set node0.comp0.foo = "bar"
- bind/unbind -> (un)bind node0.base.output chan0
- (de)attach -> (de)attach node0 sync
- network -> network node0 192.168.0.1

Manipulating the model => reconfiguration

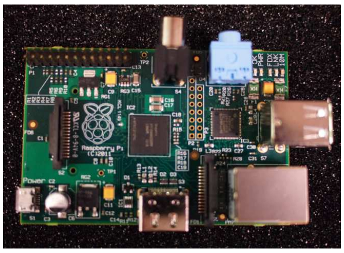
➢ Unifying the user's view

- **From Arduinos to the Cloud**



Kevoree in the small: Devices diversity

- New mobility usages boost low consumption computer development
- FPGA, Softproc (VHDL)
- Micro-controller (uC, C code)
- System on chip (SoC)
  (Raspberry project, Java, Python, Javascript)
- Smartphones (android, Java)
- Low consumption Cloud VM based on ARM



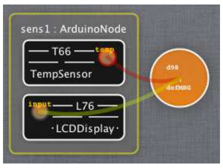
Challenges of adaptation on µC

- µC nodes not powerful enough to support all adaptation strategies!
  - Dynamic adaptation on µC faces several challenges
    - For comfort and critical uses, downtime must be minimal
- Volatile memory usage must fit in few kB
- Persistent memory writes are limited
- Reconfiguration must be statefull, to recover from power failure
- use more powerful sibling to pre-compile adaptations

µKev Component model

- Type definition/instance (what/where)
- Component with ports (in/out) and parameters
- Communication channels between ports (bindings)
- Write code (C/Processing/Java) => Instantiate in a model

```
@ComponentType
public class TempSensor
    extends AbstractPeriodicArduinoComponent {
    //behavior code
    @Param
    int pin;
    @output
    Temp temp;
}
```



- Generate firmware for types + KevScript for instances

Kevoree in the large: Managing clouds

- Kevoree node is a container of components but also of nodes !
- A parent node is responsible for child's nodes life cycle (start/stop)
- NodeType refines adaptation strategies
- See the Cloud as a hierarchy
  IaaSNode -> PaaSNode -> SaaSNode

IaaS and PaaS nodes

- Add/Remove node capability
- Available NodeType
  - MiniCloud: (Add/Remove Java Virtual Machine)
  - FreeBSD Jails: (Add/Remove VM in the same kernel space)
  - SmartOS Zone: (same as Jail)
  - KVM: Hypervisor management
  - Amazon EC2: Add/Remove EC2 Virtual Machine

KevScript as a cloud modeling language

- KevoreeScript abstracts model manipulation
  - IaaS adaptation

```
//add IaaSPlatform
addNode iaasKVM21: KVMNode {ip="10.0.1.5"}
//add two user node
addNode userNode1: JavaNode
addNode userNode2: UbuntuNode {version="12.04"}
//attach to IaaSNode42
addChild userNode1,userNode2@iaasKVM21
```

  - PaaS adaptation

```
addNode worker2@IaaSNode44: JavaNode {cpufreq="800mhz",arch="32bits",ram="2gb"}
addNode worker3@IaaSNode44: JavaNode {cpufreq="800mhz",arch="32bits",ram="2gb"}
moveComponent *@worker1 => worker2
removeNode worker1@IaaSNode42
```

Kevoree Modeling Framework: Optimized for use at runtime

Metamodel

- generate lightweight object structure with concise API for manipulating models in Javascript, Kotlin, Java, Scala, C+++
- generate helpers: XMI/JSON loader, XMI/JSON serializer, cloner, diff operator, merger ..

Features

- Efficiency with Lazy Cloning (20x faster than EMF)
- Persistence, Events
- No Dependencies
- Native Mechanisms for Time-related data

Models synthesis and space exploration in Kevoree

```
engine = new GeneticEngine<Cloud>();
engine.setAlgorithm(
          GeneticAlgorithm.EpsilonCrowdingNSGII);
engine.addOperator(new AddNodeMutator());
engine.addOperator(new RemoveNodeMutator());
engine.addOperator(new AddSoftwareMutator());
engine.addFitnessFuntion(new CloudCostFitness());
engine.addFitnessFuntion(new CloudLatencyFitness());
engine.setMutationSelectionStrategy(
          MutationSelectionStrategy.SPUTNIK_ELITIST);
engine.setMaxGeneration(300);
engine.run();
```

Polymer framework

- Sputnik hyper-heuristics and batteries included

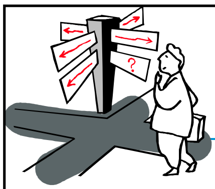
Application on CREOS reasoning engine *(greedy/full search)*

- Application on CLOUD computing elasticity *(genetic)*

Application on POST SmartHome reasoning engine *(greedy/genetic)*

Some Kevoree project outputs

- ➢ Initial implementation for Java (Java, Scala, KotLin) *(2010,2011)*
- ➢ Management of Android devices *(2012)* (firefighters tactical decision system)
- ➢ Peer to peer synchronization (Gossip, VectorClock) *(DAIS'12)*
- ➢ Management of embedded IoT nodes (C, Arduino, AVR 8bits) *(CBSE'12)*
- ➢ M@R driven Clouds (Amazon EC2, OpenStack) *(2012-2013)*
- ➢ Home automation gateway *(2012-2013)*
- ➢ Web Javascript runtime *(2013)*



Wrap-up

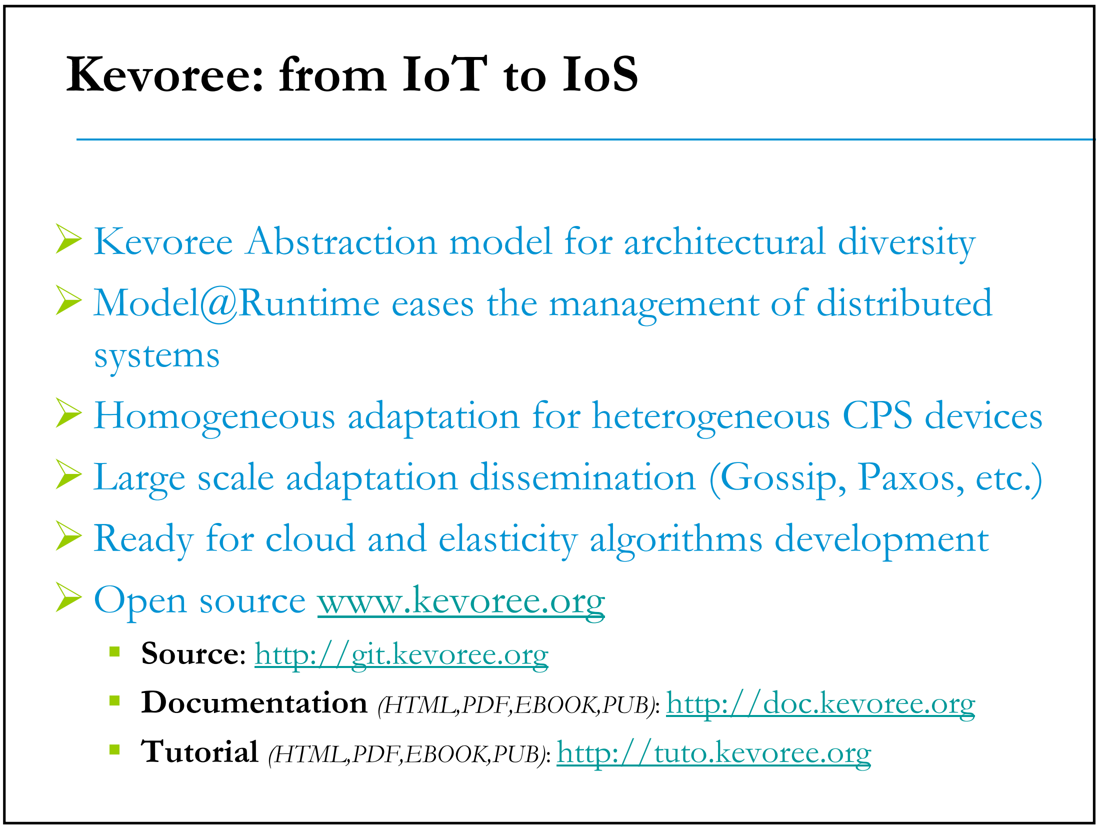
- Background on IoT and Adaptive Systems
- Principle of Models @ runtime
- Kevoree: A Concrete implementation of Distributed Models@Runtime for IoT

Models @ runtime

- (D)SPL approach to tame
  - The combinatorial explosion of configurations
  - The quadratic explosion of transitions
- Models @ runtime to reflect configurations
  - Runtime validation before adapting the running system
  - Simple roll-back
- MDE to automate reconfiguration
  - Generation of safe reconfiguration scripts

Kevoree overview

- Heterogeneity management with notion of NodeType
  - Java Node, Dalvik Node, Arduino Node, Cloud Node (Jails/*BSD, JCloud, mini-cloud, EC2)
- Component-based
  - Component, Port, Channel, Node, Group, …
  - Actor semantics on each port to separate component behavior
  - Communication semantics between components using channels
  - Hot (re-)deploy & provisioning
- Models@Runtime in a Distributed Context
  - Shared model representation for distributed nodes
  - Offline & online operation, compute@Model level, apply @Runtime
    - Reconfiguration using platform mecanisns, from classLoader, DLL to reflashing
- Publications
  - A Dynamic Component Model for Cyber Physical Systems François Fouquet; Olivier Barais; Noël Plouzeau; Jean-Marc Jézéquel; Brice Morin; Franck Fleurey *15th Int. ACM SIGSOFT Symposium on Component Based Software Engineering*, Jul 2012, Italy.
  - Dissemination of reconfiguration policies on mesh networks François Fouquet; Erwan Daubert; Noël Plouzeau; Olivier Barais; Johann Bourcier; Jean-Marc Jézéquel *DAIS 2012*, Jun 2012, Stockholm, Sweden.



Kevoree: from IoT to IoS

- Kevoree Abstraction model for architectural diversity
- Model@Runtime eases the management of distributed systems
- Homogeneous adaptation for heterogeneous CPS devices
- Large scale adaptation dissemination (Gossip, Paxos, etc.)
- Ready for cloud and elasticity algorithms development
- Open source www.kevoree.org
  - **Source**: http://git.kevoree.org
  - **Documentation** *(HTML,PDF,EBOOK,PUB)*: http://doc.kevoree.org
  - **Tutorial** *(HTML,PDF,EBOOK,PUB)*: http://tuto.kevoree.org