ThingML: A Domain Specific language for Cyber Physical Systems / Internet of Things

Franck Fleurey
SINTEF
franck.fleurey@sintef.no
Internet of Things / Cyber-Physical Systems

- 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)
- ...
Model-Driven Software Engineering

- Use models to support developers in the implementation of the system
  - Target users are CPS developers
  - Models are used to generate code

MDE can be used in many other phases
- Requirements analysis, Safety / Security analysis, Design
- Target users are different too
Topics of the lecture

• SE Challenges for IoT / Cyber-Physical Systems
• The ThingML approach / toolset
  ▪ Applying MDE for building CPS
  ▪ As a platform to support Research
• Experiences and lesson learned
  ▪ Building and tooling a DSML
  ▪ Practical use of MDE with developers with different backgrounds
• Questions
Typical IoT Scenario

- System under development
- External service
- Legacy
- Partner
- Proprietary product
- Things
I2C, gpio, serial, analog, etc

ATmega 32u4

Linino AR 9331

Serial

SPI

Reset

USB programming and debugging interface

IP Network

Servers and databases

Laptops, smartphones

Yun

Servers and databases

Client

The code of the application is distributed on the different nodes of the infrastructure
Challenges

- 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
State of the art / practice

- State of the art / practice
  - Solution 1: Centralized service which uses devices "as-is"
    - Most common practice. Simple but restrictive.
  - Solution 2: Hide behind an homogeneous software layer
    - OS + generic or specific middleware platforms (eg. JAVA/JVM)
  - Solution 3: Avoid problems by carefully selecting platforms
    - For which software frameworks pre-exist (eg. Arduino libs / shields)
  - Solution 4: Custom develop manually all pieces of software
    - Can exploit full potential but very expensive (eg. automotive)

- Use MDE to abstract from platforms and different code generators to generate for different targets
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
    o Modularity, reusability, runtime deployment, continuous integration, validation, etc...
  ▪ Cost efficient and practically usable
    o No large overhead, integrated with legacy systems, etc...
ThingML Versions / Iterations

• Iteration 1 (~2009)
  ▪ "3 layers architecture", generate APIs
  ▪ Re-use existing modelling languages / tools
  ▪ Target mostly resource constrained devices

• Iteration 2 (~2012)
  ▪ "Complete" DSML (with action language)
  ▪ Full code generation + communication APIs
  ▪ Target wider range of platforms

• Iteration 3 (~2015)
  ▪ "Blending" of target platform and legacy code
  ▪ Code generation framework with plug-in mechanism
ThingML Versions / Iterations

• Iteration 1 (~2009)
  ▪ "3 layers architecture", generate APIs
  ▪ Re-use existing modelling languages / tools

• Iteration 2 (~2012)
  ▪ "Complete" DSML (with action language)
  ▪ Full code generation + communication APIs

• Iteration 3 (~2015)
  ▪ "Blending" of target platform and legacy code
  ▪ Code generation framework with plug-in mechanism
Iteration 1 Use Cases (1/2)

**Seismic changes**
The search for oil nearly always involves seismic imaging – determining what lies underground or beneath the seafloor by evaluating echoes from blasts of sound.

In recent years, offshore seismic operators have put a few twists on the process.

**The basics**
Conventional offshore seismic surveying uses a single vessel towing arrays of sensors called **streamers** in a series of parallel lines.

- **Seismic survey vessel**
- **Sound waves**
- **A specialized air gun** on the vessel sends sound waves through the water and into the seafloor.
- **Buoy**

The sound echoes back and is captured by the streamers, with the results stored for evaluation by geophysicists, petroleum engineers and other experts using high-power computers.

**New variations:**
1. The vessel sails in linked circles over a target area.
2. A team of up to four vessels sails parallel patterns.

In both approaches, the process captures more echoes, providing more detailed data than in past surveys. The result is clearer 3-D images.

Another method uses nodes placed on the ocean floor to capture sound waves, cutting out the need for streamer cables. This method produces the most detailed images, but is much more expensive.

Sources: BP, Schlumberger, Fairfield Nodal

Ken Ellis: Houston Chronicle
Iteration 1 Use Cases (2/2)
ThingML Iteration 1

- **Domain Model**
  - APP 1
  - APP 2
  - Driver 1
  - Driver 2
  - Driver 3

- **Applications**
  - Domain (Nodes and Messages)
  - Hardware specific drivers

- **Hardware**
  - Sensors
  - Nodes
  - Actuators
  - Networks

- **Platform specific models**

- **State Machines**
Step 1: Modeling the domain

```
system SimpleDomain;

device Button[2..2]
    @avr_pooling "true"
    @avr_init "true"
{
    address button_id : Integer
    incoming press();
    incoming release();
}
device Timer[0..2]
    @avr_pooling "true"
    @avr_init "true"
{
    address timer_id : Integer
    outgoing start(delay : Integer);
    outgoing cancel();
    incoming timeout();
}
device Light[0..4]
    @avr_pooling "false"
    @avr_init "true"
{
    address light_id : Integer
    multicast outgoing setColor(red : Integer, green : Integer, blue : Integer);
    multicast outgoing fadeColor(red : Integer, green : Integer, blue : Integer);
    multicast outgoing blinkColor(red : Integer, green : Integer, blue : Integer);
    multicast outgoing setFadingDelay(delay : Integer);
    multicast outgoing setBlinkingParams(delay : Integer, repeat : Integer);
}
datatype Integer
    @avr "uint_8";
```

AVR CPU (ATMega168)

Light bus

Light 1

Light 2

Light 3

Light 4

Button 1

Button 2

Timer
Step 2: Generation of the domain layer

```c
#include "timer.h"

void init_timer() {
    // TODO: Initialize the driver here
}

void pool_timer() {
    // TODO: Implement the pooling here
}

void timer_send_start(int timer_id, int delay) {
    // TODO: Send start message
}

void timer_send_cancel(int timer_id) {
    // TODO: Send cancel message
}

(void) timer_receive_timeout_listener = 0x0;

void register_timer_receive_timeout_listener(void (*listener)(int)) {
    timer_receive_timeout_listener = listener;
}

void timer_receive_timeout(int timer_id) {
    if (timer_receive_timeout_listener != 0)
        timer_receive_timeout_listener(timer_id);
}
```

```
Timer[0..2]
@avr_pooling "true"
@avr_init "true"

address timer_id : Integer
outgoing start(delay : Integer);
outgoing cancel();
incoming timeout();
```
Step 3: Using Existing tools for modeling the application

IAR visualSTATE
Importing the domain in IAR visualSTATE

- Generated from the EDAP domain model

```c
VS_VOID light_send_SetFadingDelay(VS_INT light_id, VS_INT delay),
VS_VOID light_broadcast_SetFadingDelay(VS_INT delay),
VS_VOID light_send_SetBlinkingParams(VS_INT light_id, VS_INT delay, VS_INT repeat),
VS_VOID light_broadcast_SetBlinkingParams(VS_INT delay, VS_INT repeat),
VS_VOID timer_send_start(VS_INT timer_id, VS_INT delay),
VS_VOID timer_send_cancel(VS_INT timer_id);

@Event
  timer_receive_timeout(VS_INT timer_id),
  button_receive_press(VS_INT button_id),
  button_receive_release(VS_INT button_id);
```
Imported domain in IAR visualSTATE
Step 4: Modeling the application
Step 5: Putting it together (1)

Complete the hardware driver code manually

- Use IAR visualSTATE built-in code generator for AVR controllers

- The generated domain layer handles all interactions between the application generated code and the hardware driver

- Compile C code with avr-gcc
Step 5: Putting it together (2)

The «main» just needs to initialize the application

```c
void receive_timeout(int timer) {
    ControllerVSDeduct(timer_receive_timeout, timer);
}

void receive_button_press(int button) {
    ControllerVSDeduct(button_receive_press, button);
}

void receive_button_release(int button) {
    ControllerVSDeduct(button_receive_release, button);
}

void init_application() {
    // The application listens for button events
    register_button_receive_press_listener(&receive_button_press);
    register_button_receive_release_listener(&receive_button_release);
    // Register for timer events
    register_timer_receive_timeout_listener(&receive_timeout);

    ControllerVSInitAll();
    ControllerVSDeduct(SE_RESET);
}
```
Iteration 1 Findings

- A set of case studies have been implemented
- Working with different developers
- UML Profile vs Domain Specific Language
- Graphical vs Lexical Concrete Syntax
- Generate code skeletons
- Quality of generated code
- Using Existing MDE and UML tools
- Template based code generators
- Work / Integrate a set of tools
ThingML Versions / Iterations

- **Iteration 1 (~2009)**
  - "3 layers architecture", generate APIs
  - Re-use existing modelling languages / tools

- **Iteration 2 (~2012)**
  - "Complete" DSML (with action language)
  - Full code generation + communication APIs

- **Iteration 3 (~2015)**
  - "Blending" of target platform and legacy code
  - Code generation framework with plug-in mechanism
Iteration 2

• One integrated modelling language
  ▪ More generic architecture models
• Includes a platform independent action language
• Includes a simple "kick-down" to integrate manually written code in the model
• Generated "complete" modules
ThingML Approach

Abstract HD-Services

Model of the services

Inputs

Code generator
Code generator
Code generator
Code generator

Framework for
resources-constrained
devices

Framework for
Internet enabled
devices

Code generator
Code generator

Deploy

Concrete HD-Services

Case study implementation

Cloud(s)

Servers, databases

Gateways

Things, devices

Smartphones

End-users

HD Service Developer

Platform Experts

Design HD-services

Loose coupling

Add new devices and platforms
Iteration 2 Use Case

**Push-Pull Control (PPC) Cables**

PPC cable-based actuation system controls the movements of the orthosis joints in the sagittal plane. 3 actuators per orthosis leg are placed on the mobile platform while the PPC cables are flexible links to the joints that transfer the rotational movement of the motors to specifically designed orthotic joints.

**Powered Orthosis**

The powered orthosis assists the patient’s lower limb joint motions. There are 6 DoFs at each leg: 3 in the hip, 1 in the knee, and 2 in the ankle joints. The hip, knee and ankle joint motions on the sagittal plane are active DoFs; the hip DoFs in the transverse and frontal planes and the ankle DoF in the transverse plane are passive.

**Operating Modes**

**Learning**

Learning of the "therapist-assisted gait" as reference for the corrective mode.

**Corrective**

Adaptation of robotic support according to patient's performance and patient's state.

**Linear Unit**

The linear unit enables both lifting and lowering of the patient's body and side to side movement of the patient's body. It is equipped with two servo-positioning motors for the actuation of the vertical axis which are chosen in order to provide partial body weight support. One servo motor is used for the actuation of the horizontal axis.

**Pelvis Link**

The pelvis link is the connection between the linear unit, attached to the platform, and the powered orthosis. The pelvis link ensures that the patient's pelvis can rotate in the frontal and transverse planes. The F/T sensor for measurement of the interaction force between the patient and the CORBYS system is placed within the pelvis link.

**Mobile Platform**

The mobile platform will provide mobility for a patient. It serves as a component carrier for the linear unit, motors for the orthosis, central power supply, and other CORBYS system modules such as the safety module and the computers upon which the modules of the control architecture run.
Iteration 2 Use Case

HSS Controller

Chest Unit Driver

IMU Sensor Driver

EMG Driver

HSS Front-end

Engineer GUI

Parameter Server

CORBYS GPN

Bluetooth

BLE

Sensor_Data_HSS

Heart_Beat

Config_Data

HSS_Chest_Phi
HSS_Chest_IMU
HSS_Chest_ECG

HSS_Back_TH
HSS_Back_IMU

HSS_Env_TH
HSS_Env_IMU

HSS_EMG_TLS
HSS_EMG_BL
HSS_EMG_TR
HSS_EMG_BR

Chest Unit Sensors

IMU Sensors

EMG Sensors

1x

2x

4x
Iteration 2 Use Case

Server
(Registration, Discovery, Composition, Execution, Visualization)

Visualisation

Adaptive Applications (mobile or not)

Adapters from commercial sensor networks to sensapp

Commercial gateway (hardware)

Commercial Sensors Temperature, energy, ...

Commercial home automation devices

Commercial Gateway (Software)

Adapter

Adapter

TI Adaptive ThingML Sensors Temperature, Light, ...

TI ThingML Intelligent Gateway

ThingML Waveman Bridge

Commercial Sensors Temperature, energy, ...

Visualisation

Commercial home automation devices

Iteration 2 Use Case
ThingML: Architecture Model

- Robot control
  - Movement control
    - Left Wheel control
    - Right Wheel control
  - Distance Sensor
    - Collision Sensor
ThingML: Component

Wheel Controller

<= forward(speed:int)
<= backward(speed:int)
<= stop()
=> wheel_position(position: int)

Left Wheel control
Right Wheel control
ThingML: State Machines

Wheel Controller

- Stopped
  - ?stop
  - ?forward
- Forward
  - ?stop
  - !wheel_position
- backward
  - ?backward
  - !wheel_position

Control Port
ThingML: Action Language

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

```
?forward
```

```
Forward
```

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

```
...}
```
http://www.thingml.org

What is ThingML?

ThingML is a modeling language for embedded and distributed systems. It is developed by the Networked Systems and Services department of SINTEF in Oslo, Norway.

ThingML stands for “Thing” Modeling Language as a reference to the so called Internet of Things.

The idea of ThingML is to develop a practical model-driven software engineering tool-chain which targets resource constrained embedded systems such as low-power sensor and microcontroller based devices.

ThingML is developed as a domain-specific modeling language which includes concepts to describe both software components and communication protocols. The formalism used is a combination of architecture models, state machines and an imperative action language.
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 (UML)
- 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)
  - Under development: Lua, C#
- Generators for communication channels
- Easy to distribute ThingML IDE
  - Standalone and lightweight IDE
  - Eclipse plugins
Arduino Programming with **ThingML**

```java
void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}
```
Hello World!

```java
void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.println("Hello World!");
    while (1) {}  // Infinite loop
}
```
Simple LED Blink

```c
void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```
More Realistic LED Blink

```cpp
const int ledPin = 13;

int ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 1000;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
    digitalWrite(ledPin, ledState);
  }
}
```
Iteration 2 Results (1/2)

- Several case studies successfully implemented
- CORBYS Robot Human Sensory System
  - Implemented in ThingML
  - Integrated to the complete robotics system
  - Maintained / evolved by "external" developers
  - Currently being re-used in a different context
- Tool and concrete syntax is usable
- Generated code has the required "quality"
Iteration 2 Results (2/2)

- Code generators needs to be customized
  - The generic generators almost fits but...
  - Slightly different target platforms
  - Different projects constraints / legacy code
- Remaining complexity has to do with
  - Waiting, counting and combining events
  - Implementing communication
- Need for methodology / guidelines
ThingML Versions / Iterations

• **Iteration 1 (~2009)**
  - "3 layers architecture", generate APIs
  - Re-use existing modelling languages / tools

• **Iteration 2 (~2012)**
  - "Complete" DSML (with action language)
  - Full code generation + communication APIs

• **Iteration 3 (~2015)**
  - "Blending" of target platform and legacy code
  - Code generation framework with plug-in mechanism
What are the challenges? (1/6)

- Here is an example infrastructure
What are the challenges? (2/6)

• Here is the software components needed for the service
What are the challenges? (3/6)

• Heterogeneous infrastructure and technologies are needed.
What are the challenges? (4/6)

• A lot of different expertise are needed
  ▪ Both for development and runtime deployment/maintenance
What are the challenges? (5/6)

- Someone needs to coordinate all experts
  - Design the different components, their functionality and interactions
What are the challenges? (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?
ThingML Approach

Abstract HD-Services

Model of the services

Inputs

Code generator
Code generator
Code generator
Code generator

Resources-constrained devices

Framework for

Code generator
Code generator

Internet-enabled devices

Deploy

Deploy

smartphones

Things, devices

Gateways

Servers, databases

Cloud(s)

Case study implementation

Concrete HD-Services

Design HD-services

HD Service Developer

Loose coupling

Platform Experts

Add new devices and platforms
Code generation framework

Need to provide a framework which can be customized by developers with very different background

https://github.com/SINTEF-9012/ThingML/tree/master/compilers
ThingML code generation framework

- Configuration
- Thing(s)
- ThingML Model
- Generated code

1. Ports / Messages / Thing API
2. State Machine Implementation
3. Actions/Expressions/Functions
4. Connectors / Channels
5. Message queuing / FIFOs
6. Scheduling / Dispatch
7. Initialization and "Main"
8. Projects structure / build script

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 one 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 target 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 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%
Ongoing / Future Work

• Complex-Event Processing
• Library of communication protocols
• Runtime deployment and management
• Static analysis / ThingML checker
(Complex) Event Processing

React on critical events (alerts)

Drivers, wrappers

Tame "big data" directly at the source

Coarse-grain behavior

CEP

ECA

Composite State Machines

Imperative programming

Other software systems

Physical world
Generating the Communication layer

• By default all messages are considered as asynchronous and are sent with a "send and forget" policy
• Annotation to refine communication semantics

```cpp
// Simple Ping
message ping() @code "66"
    @sync_ack "pong" @timeout "1000" @retry "3";
message pong() @timeout "67";
receives ping
sends pong

// Get the data from individual sensors
message GetTemperature() @code "1"
    @sync_response "TemperatureValue\$v" @timeout "500" @retry "0";
message TemperatureValue(v : Integer)
    @code "2";
```
Event Processing
CEP meta-model extension
Summary

- ThingML is open-source
  - [http://www.thingml.org](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 target languages
  - Arduino, Posix, Java, NodeJS
- The code generation framework allows for easy customization of the generated code
Thanks for your attention!

- Questions?

- More info: [http://www.thingml.org](http://www.thingml.org)
- Code: [https://github.com/SINTEF-9012/ThingML](https://github.com/SINTEF-9012/ThingML)