A Language-based Approach to Interoperability of IoT Platforms
The challenge

IoT platforms frequently take the shape of **vertical solutions** that rely on a single communication technology stack.
A language-based Approach for Interoperability of IoT platforms

The challenge

IoT platforms frequently take the shape of vertical solutions that rely on a single communication technology stack.
The challenge

IoT platforms frequently take the shape of vertical solutions that rely on a single communication technology stack.
The challenge

IoT platforms frequently take the shape of **vertical solutions** that rely on a single communication technology stack.
The challenge

IoT platforms frequently take the shape of **vertical solutions** that rely on a single communication technology stack.

---

A language-based Approach for Interoperability of IoT platforms

---

Stefano Pio Zingaro

stefanopio.zingaro@unibo.it
The challenge

IoT platforms frequently take the shape of **vertical solutions** that rely on a single communication technology stack.
The challenge

IoT platforms frequently take the shape of vertical solutions that rely on a single communication technology stack.
Possible solutions

To facilitate the development of IoT applications

• **Low Level Abstractions**
  - A. Middlewares
  - B. Smart Gateways (Meshlium, etc.)

• **High Level Abstractions (our approach)**
  - C. Integration Frameworks (Eclipse IoT)
  - D. APIs (WoT from W3C)
Possible solutions

To facilitate the development of IoT applications

• Low Level Abstractions
  A. Middlewares
  B. Smart Gateways (Meshlium, etc.)

• High Level Abstractions (our approach)
  C. Integration Frameworks (Eclipse IoT)
  D. APIs (WoT from W3C)

NOT A DEDICATED LANGUAGE!
The proposed solution

In order to achieve a better integration we propose a programming language:

- Programmers can easily change the transport and application protocols (even at runtime!)
- The language can automatically marshal and unmarshal data formats as required (e.g. HTTP/JSON to CoAP/JSON)
- The complexity of guaranteeing interoperability among protocols is managed by the programming language
The challenge

We focus on integration at the transport and application layer.
An Example of Interoperability

Suppose we want to replace the technology used:

- **Static replacement** — because of the deployment of new, heterogeneous devices in a pre-existing system.
- **Dynamic replacement** — to support a changing topology of disparate mobile devices.
Another Example of Interoperability

COAP over UDP
TEMPERATURE SENSOR

HTTP over TCP
TEMPERATURE SENSOR

Send Temperature

COLLECTOR

Send Temperature
Another Example of Interoperability

A language-based Approach for Interoperability of IoT platforms

```
/* Unique behavior */
main {
    ...
    receiveTemperature(data);
    ...
}
```
Another Example of Interoperability

A language-based Approach for Interoperability of IoT platforms

```c
/* Unique behavior */
main {
    ...
    receiveTemperature(data);
    ...
}
```

```c
/* Deployment for HTTP/TCP */
inputPort HTTP_Collector {
    Location: "socket://collector.net:8000"
    Protocol: http
    OneWay: receiveTemperature(string)
}
```
Another Example of Interoperability

A language-based Approach for Interoperability of IoT platforms

```cpp
/*
* Deployment for CoAP/UDP
*/
inputPort CoAP_Collector {
    Location: "datagram://local.me:5683"
    Protocol: coap
    OneWay: receiveTemperature( string )
}

/*
* Deployment for HTTP/TCP
*/
inputPort HTTP_Collector {
    Location: "socket://collector.net:8000"
    Protocol: http
    OneWay: receiveTemperature( string )
}

/*
* Unique behavior
*/
main {
    ...
    receiveTemperature( data );
    ...
}
```
Another Example of Interoperability

Use different communication modalities in a uniform way

 CORS

/*
 * Deployment for CoAP/UDP
 */

inputPort CoAP_Collector {
  Location: "datagram://coap.me:5683"
  Protocol: coap
  OneWay: receiveTemperature( string )
}

/*
 * Deployment for HTTP/TCP
 */

inputPort HTTP_Collector {
  Location: "socket://collector.net:8000"
  Protocol: http
  OneWay: receiveTemperature( string )
}

/*
 * Unique behavior
 */

main {
  ...
  receiveTemperature( data );
  ...
}
Another Example of Interoperability

Use different communication modalities in a uniform way

Easily switch between them
The **JIoT** Project — Main Contribution

Integrating IoT related technologies **into the** **Jolie** Service Oriented Programming Language:

- The **Publish/Subscribe** communication pattern
- The **MQTT** Application Layer protocol
- The **UDP** Transport Layer protocol
- The **CoAP** Application Layer protocol
MQTT integration in Jolie

1. **Publish/Subscribe communication paradigm**
   Adding a generic publish/subscribe meta-channel that bridges *between end-to-end style* of communications *and publish/subscribe* interactions.

2. **Message Queue Telemetry Transport (MQTT)**
   Encoding/Decoding MQTT messages (*Netty* based)
CoAP and UDP integration in Jolie

1. **User Datagram Protocol (UDP)**
   • Adding Listener and Sender classes based on **Netty**
   • Extend the semantic to support un-reliable communications

2. **Constrained Application Protocol (CoAP)**
   Encoding/Decoding CoAP messages (**nCoAP** based)
**JIoT Interpreter**

**Last Release of the interpreter can be found at:**
http://cs.unibo.it/projects/jolie/jiot.html

- Requires JRE 1.8+
- Download jiot-1.0.3.jar
- Open a console and run:

  ```java
  java -jar jiot-1.0.3.jar
  ```
JIoT Sources

JIoT is an open source project developed as an extension to Jolie

The sources for the project can be found at:
https://github.com/stefanopiozingaro/jolie/tree/next
Recap – The Service Oriented Paradigm

• Everything is a service
• A service is an application that offers functionalities through operations
• A service can invoke another service by calling one of its operations
Recap — What is Jolie?

A Service-Oriented Programming Language

![Diagram showing the relationship between Service-Oriented and Object-Oriented concepts: Services vs. Operations on the left, Objects vs. Methods on the right.](slide)
Deployments
Enabling Communications
Communication Ports in Deployments

**Services** communicate through **Ports**

**Ports** give access to an **Interface**

**Interface** be a set of **Operations**

An **Output Port** is used to invoke interfaces exposed by other services

An **Input Port** is used to expose an interface
A closer look on ports - Protocols

- A protocol **defines the format** in which the data is sent (**encoded**) and received (**decoded**)
- In Jolie **protocols are described by names** and possibly some **parameters**:

  - json/rpc
  - sodep
  - https
  - soap
  - http \{ .debug = true \}
Behaviours

Composing Interactions
Interactions via Operations

Input Operations

- `oneWay(req)`
- `reqRes(req)(res)`

// code block

Output Operations

- `oneWay@Port(req)`
- `reqRes@Port(req)(res)`
Case Study — Smart Home Example

A language-based Approach for Interoperability of IoT platforms

TEMPERATURE SENSOR

Get Temperature 68 °F

COLLECTOR

Set Temperature 70 °F

HVAC

68 °F

70 °F

Stefano Pio Zingaro
stefanopio.zingaro@unibo.it
Case Study — Smart Home Example

End to end communication using CoAP over UDP
Case Study – Smart Home Example
Publish/Subscribe communication using MQTT over TCP
Case Study — Smart Home Example

/*
 * Let's define a common interface for the different equipped thermostats,
 * using the same operations
 */

interface ThermostatInterface {
    OneWay: setTemp( int )
    RequestResponse: getTemp( void )( int )
}
Case Study — Smart Home Example

/*
 * The common behaviour
 */

main {
    getTemp@Thermostat( ) ( temp );
    if ( temp > 81 ) {
        setTemp@Thermostat( 75 )
    } else if ( temp < 59 ) {
        setTemp@Thermostat( 72 )
    }
}
Case Study — Smart Home Example

```c
/*
 * The final, common behavior
 */
main {
    getTemp@Thermostat( )( temp );
    if ( temp > 81 ) {
        setTemp@Thermostat( 75
    } else if ( temp < 59 ) {
        setTemp@Thermostat( 72
    }
}
```

REQUEST-RESPONSE pattern

Outbound RequestResponse communications send a message and wait for a reply.
A language-based Approach for Interoperability of IoT platforms

Case Study – Smart Home Example

/* The final, common behavior */

main
{
  getTemp@Thermostat( )( temp );
  if ( temp > 81 ) {
    setTemp@Thermostat( 75 );
  } else if ( temp < 59 ) {
    setTemp@Thermostat( 72 );
  }
}

ONE-WAY pattern

Outbound OneWay communications send a message asynchronously.

Stefano Pio Zingaro
stefanopio.zingaro@unibo.it
Case Study – Smart Home Example

```c
interface ThermostatInterface {
    OneWay: setTemp(int);
    RequestResponse: getTemp(void)(int);
}
```

/*
 * Lets define a common interface for the different equipped thermostats, using the same operations
 */
Case Study – Smart Home Example

```java
outputPort Coap_Thermostat {
    Location: "datagram://localhost:5683"
    Protocol: coap {
        .osc.getTemp << {
            .contentFormat = "text/plain",
            .alias = "/getTemperature",
            .messageCode = "GET"
        };
        .osc.setTemp << {
            .contentFormat = "text/plain",
            .alias = "/setTemperature",
            .messageCode = "POST"
        }
    }
}

Interfaces: ThermostatInterface
```
Case Study — Smart Home Example

```java
outputPort Mqtt_Thermostat {
    Location: “socket://iot.eclipse.org:1883”
    Protocol: mqtt {
        .osc.getTemp << {
            .alias = "getTemperature",
            .QoS = 2
        };
        .osc.setTemp << {
            .alias = "setTemperature",
            .QoS = 2
        }
    }
}

Interfaces: ThermostatInterface
```
A special thanks to the SPACES team ...

... so long, and thanks for all the fish ...