A gentle introduction to Jolie

Saverio Giallorenzo | sgiallor@cs.unibo.it
\[ j \in I \quad t_c = \text{eval}(e, t) \quad M(t_c) = (o_j, t') :: \bar{m} \]
\[
\sum_{i \in I} [o_i(x_i) \text{ from } e] \{ B_i \} \cdot t \cdot M \rightarrow B_j \cdot t \lt (x_j, t') \cdot M[t_c \mapsto \bar{m}] \]
\[
x = e; B \cdot t \cdot M \rightarrow B \cdot t \lt (x, t') \]
\[
P \rightarrow P' \]
\[
P = \text{cq}(x); B \cdot t \cdot M \]
\[
\langle B_s, P | \prod B_i \rangle \]
\[
P = \text{eval}(e_2, l) \quad \text{eval}(e_3, t) = t_c \quad eval(e_1, t_1) = l \quad Q = B \cdot t_\bot \lt (x, \text{eval}(e_2, t_1)) \cdot \emptyset \]
\[
\langle !(x); B, P \rangle_l | \langle B'_s, P_1 | P_2 \rangle_{\nu'} \rightarrow \langle !(x); B, Q | P \rangle_l | \langle B'_s, B_1 \cdot t_1 \cdot M_1 | P_2 \rangle_{\nu'} \]
What is Jolie?

A Service-Oriented Programming Language

Service-Oriented

- Service Instances
- Operations

Object-Oriented

- Objects
- Methods
Why SOC and Jolie?

Jolie is perfect for fast prototyping. In little time a small team of developers can build up a full-fledged distributed system.

But I already know Java! Why shall I use Jolie?
Why SOC and Jolie?

SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(new InetSocketAddress("http://someurl.com", 80));
Buffer buffer = ...; // byte buffer
while( buffer.hasRemaining() ) {
  channel.write( buffer );
}

Happy?

Ok, but you did not even close the channel or handled exceptions
A gentle introduction to Jolie

Why SOC and Jolie?

```java
SocketChannel socketChannel = SocketChannel.open();
try {
    socketChannel.connect(new InetSocketAddress("http://someurl.com", 80));
    Buffer buffer = ...; // byte buffer
    while (buffer.hasRemaining()) {
        channel.write(buffer);
    }
} catch (UnresolvedAddressException e) { ... }
catch (SecurityException e) { ... }
/* ... many catches later ... */
catch (IOException e) { ... }
finally { channel.close(); }
```

Happier now?

Yes, but what about the server?
A gentle introduction to Jolie

Why SOC and Jolie?

```java
Selector selector = Selector.open();
channel.configureBlocking(false);
SelectionKey key = channel.register(selector, SelectionKey.OP_READ);
while(true) {
    int readyChannels = selector.select();
    if(readyChannels == 0) continue;
    Set<SelectionKey> selectedKeys = selector.selectedKeys();
    Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
    while(keyIterator.hasNext()) {
        SelectionKey key = keyIterator.next();
        if(key.isAcceptable()) {
            // a connection was accepted by a ServerSocketChannel.
        } else if (key.isConnectable()) {
            // a connection was established with a remote server.
        } else if (key.isReadable()) {
            // a channel is ready for reading
        } else if (key.isWritable()) {
            // a channel is ready for writing
        }
        keyIterator.remove();
    }
    here you are
}
```

Here you are
Why SOC and Jolie?

Well, ok, but again, you are not handling exceptions. And what about if different operations use the same channel?

And if we wanted to use RMIs instead of Sockets?

In what format are you transmitting data? And if we need to change the format after we wrote the application? Do you check the type of data you receive/send?
Why SOC and Jolie?

Programming distributed systems is usually harder than programming non distributed ones.

Concerns of **concurrent** programming.

Plus (not exhaustive):
- handling **communications**;
- handling **heterogeneity**;
- handling **faults**;
- handling the **evolution** of systems.
Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```jolie
include "console.iol"

main
{
  println@Console("Hello, world!")
}
```

Include a service

program entry point

operation

service
Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```jolie
include "console.iol"

main
{
println@Console("Hello, world!")
}
```

`hello_world.ol`

```
$ jolie hello_world.ol
```

`hello_world.ol`
Let us see some Jolie in Action

Everything starts with a calculator...
Behaviours and Deployments

interface MyInterface {
  OneWay: sendNumber( int )
}

include "MyInterface.iol"
outputPort B {
  Location: "socket://localhost:8000"
  Protocol: sodep
  Interfaces: MyInterface
}
main {
  sendNumber @ B ( 5 )
}

include "MyInterface.iol"
inputPort B {
  Location: "socket://localhost:8000"
  Protocol: sodep
  Interfaces: MyInterface
}
main {
  sendNumber( x )
}
Deployments

Enabling Communication
Behaviours and Deployments

• Services communicate through **ports**.
• **Ports** give access to an **interface**.
• An interface is 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 - Locations

A location describes:

- the **communication medium**;
- the **parameters** to set the communication up.

In Jolie a **location** is a **Uniform Resource Identifier** (URI) with form: `medium[:parameters]`

<table>
<thead>
<tr>
<th>Medium</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
<td>socket://www.google.it:80</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>btl2cap://localhost:3B9FA89520078C303355AAA694238F07;name=Vision;encrypt=false;authenticate=false</td>
</tr>
<tr>
<td>Local</td>
<td>localsocket:/tmp/mysocket.socket</td>
</tr>
<tr>
<td>Java RMI</td>
<td>rmi://myRmiUrl.com/MyService</td>
</tr>
<tr>
<td>In-Memory</td>
<td>local</td>
</tr>
</tbody>
</table>
A gentle introduction to Jolie

A closer look on ports - Protocols

A protocol defines the format the data is sent (encoded) and received (encoded)

In Jolie protocols are names and possibly additional parameters:

- json/rpc
- sodep
- https
- soap
- http { .debug = true }
Behaviours
Composing Interactions
Interactions via Operations

**Input Operations**

```plaintext
oneWay( req )
reqRes( req )( res ){
    // code block
}
```

**Output Operations**

```plaintext
oneWay@Port( req )
reqRes@Port( req )( res )
```
Behaviour Composition

The sequence operator \( ; \) denotes that the **left operand** of the statement is executed **before** the one on the right.

```jolie
println@Console("A")();
println@Console("B")();
```

Prints \( A \) \( B \)
Behaviour Composition

The parallel operator \(|\)
states that both left and right operands execute concurrently.

```javascript
println@Console("A")()
println@Console("B")()
```

can print \(A\) \(B\)
but also \(B\) \(A\)
Behaviour Composition

The input choice implements input-guarded non-deterministic choice.

\[
\begin{align*}
&\text{oneWayOperation() } \{ \text{branch_code} \} \\
&\text{oneWayOperation2() } \{ \text{branch_code2} \} \\
&\text{requestResponseOperation() } \{ \text{rr_code} \} \\
&\{ \text{branch_code} \}
\end{align*}
\]
Behaviour Composition

The input choice implements input-guarded non-deterministic choice.

```java
main {
    [ buy( stock )( response ) {
        buy@Exchange( stock )( response )
    } ] { println@Console( "Buy order forwarded" )() }

    [ sell( stock )( response ) {
        sell@Exchange( stock )( response )
    } ] { println@Console( "Sell order forwarded" )() }
}
```
A gentle introduction to Jolie

Last stand - **that** ORC example

```java
include "net.inc"

val BingSpell = BingSpellFactoryPropertyFile("orc/orchard/orchard.properties")

println(y)
<y>
  ( Prompt("Input a string: ") > x >
    ( BingSpell(x) | (Rwait(250) >> x) ) )
```
Last stand - that ORC example

```iol
include "console.iol"
include "time.iol"

timeout = 250;
timeout.operation = "timeout";
txt = "Beutiful";
{
  spellCheck@BingSpell({ .text = txt, .location = myLoc })
  |
  setNextTimeout@Time( timeout )
};
[ spellCheckResponse( text )]
  { println@Console( text )() } 
[ timeout() ]
  { throw( TimeoutException ) } 
```
Before you take off
Jolie Website

http://www.jolie-lang.org

still working out the SEO...
The Jolie Interpreter

Last release

http://www.jolie-lang.org/downloads.html

- Requires JRE 1.6+
- Download jolie-installer.jar
- open a console and run

java -jar jolie-installer.jar
Jolie is an open source project with continuous updates and a well documented codebase

https://github.com/jolie/jolie

“This is the programming language you are looking for”
Documentation

Comprehensive and ever-growing documentation and Standard Library.

http://docs.jolie-lang.org
Editors

Sublime

Text but also Atom

Syntax highlight, online checking, etc.
Thanks for your time!