Jolie Microservices and Choreographies for the Web

Saverio Giallorenzo | sgiallor@cs.unibo.it
Service-Oriented Programming

3 Commandments:

• Everything is a service;

• A service is an application that offers operations;

• A service can invoke another service by calling one of its operations.
Service-Oriented Programming

3 Commandments

• Everything is a **service**;

• A service is an application that offers **operations**;

• A service can invoke another service by calling one of its operations.

Recalling the Object-Oriented creed

- **Services** ↔ **Objects**
- **Operations** ↔ **Methods**
A simple distributed system

Client

withdrawal

ATM

card_id

Bank

pin

Card Issuer

approval

validation

approval
Orchestration

1. withdrawal

bank SOA

2. card ID

ATM

3. pin request

4. pin

5. validation

6. approval

Card Issuer SOA

Card Validator

Jolie Microservices and Choreographies for the Web
Orchestration

Process-Oriented
Why SOC and Jolie?

Need for languages that express complex compositions of services.
Why SOC and Jolie?

Need for languages that express complex compositions of services.

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?
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
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) { ... }
catch (IOException e) { ... }
finally { channel.close(); }
```

Happier now?
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 (IOException e) { ... }
finally { channel.close(); }
```

Happier now?

Yes, but what about the server?
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
        } else {
            // a channel is not ready
        }
        keyIterator.remove();
    }
}
```

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 **RMI**s 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?

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 RMI 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.
Jolie Microservices and Choreographies for the Web
A Service-Oriented Orchestration Programming Language
Resources | Online

• Official Website:
  • http://www.jolie-lang.org

• Official Docs:
  • http://docs.jolie-lang.org

• Official Codebase:
  • https://github.com/jolie/jolie
Hello World! in Jolie

“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! in Jolie

“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
“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
Hello World! in Jolie

“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
Hello World! in Jolie

“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
A more interesting example

```java
interface MyInterface {
    OneWay: sendNumber( int )
}

A

include "MyInterface.iol"
outputPort B {
    Location: "socket://localhost:8000"
    Protocol: sodep
    Interfaces: MyInterface
}

main {
    sendNumber @ B ( 5 )
}

B

include "MyInterface.iol"
inputPort B {
    Location: "socket://localhost:8000"
    Protocol: sodep
    Interfaces: MyInterface
}

main {
    sendNumber( x )
}
```
A more interesting example

```
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 )
}
A more interesting example

```java
interface MyInterface {
    OneWay: sendNumber(int)
}

include "MyInterface.iol"
outputPort B {
    Location: "socket://localhost:8000"
    Protocol: sodep
    Interfaces: MyInterface
}

main {
    sendNumber @ B (5)
}
```

```java
include "MyInterface.iol"
inputPort B {
    Location: "socket://localhost:8000"
    Protocol: sodep
    Interfaces: MyInterface
}

main {
    sendNumber(x)
}
```
A more interesting example

```
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 )
}
```
A more interesting example

```plaintext
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 )
}
```
A more interesting example

```java
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 )
}
```
A more interesting example

```
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 )
}
```
A more interesting example

```
#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 )
}
```

interface MyInterface {
    OneWay: sendNumber( int )
}
Can service-orientation aid web programming?
SOC already snuck into web programming
Uses bookkeeping.

Good for simple sessions.

Cumbersome for complex behaviours.

SOC already snuck into web programming
SOC already snuck into web programming

Uses bookkeeping.
Good for simple sessions.
Cumbersome for complex behaviours.

Code that is poorly readable and hard to maintain.
SOC already snuck into web programming
SOC already snuck into web programming

Apache

PHP “pages” (JSP, Ruby, JS)

(Tomcat, Rails, Nginx, …)

BPEL Orchestrator

ESB

S1

S2

S3

S4

S5

S6

Multi-layered architecture
SOC already snuck into web programming

```java
startSession(req)(sid) { login@S1(req)(sid) };
{  
  userApproval(u)(ru){ userApproval@S3(u)(ru) }
  |  
  adminApproval(a)(ra){ adminApproval@S5(a)(ra) }
}
; 
...
```
SOC already snuck into web programming

```plaintext
startSession( req )( sid ) { login@S1( req )( sid ) } ;
{
    userApproval( u )( ru ){ userApproval@S3( u )( ru ) }
    \[ \]
    adminApproval( a )( ra ){ adminApproval@S5( a )( ra ) }
}
```

Apache

BPEL Orchestrator

BPEL Orchestrator

BPEL Orchestrator

PHP “pages”
(JSP, Ruby, JS)

(Tomcat, Rails, Nginx, …)
SOC already snuck into web programming

startSession( req )( sid ) { login@S1( req )( sid ) } ;
{ userApproval( u )( ru ){ userApproval@S3( u )( ru ) } |
adminApproval( a )( ra ){ adminApproval@S5( a )( ra ) } }
;
...

Apache

BPEL Orchestrator

PHP "pages"
(JSP, Ruby, JS)

(Tomcat, Rails, Nginx, ...)

ESB

S1

S2

S3

S4

S5

S6
SOC already snuck into web programming

```
startSession( req )( sid ) { login@S1( req )( sid ) } ;
{
  userApproval( u )( ru ){ userApproval@S3( u )( ru ) } |
  Parallel
  adminApproval( a )( ra ){ adminApproval@S5( a )( ra ) }
}
```

...
SOC already snuck into web programming

```plaintext
startSession( req )( sid ) { login@S1( req )( sid ) } ;
{
    userApproval( u )( ru ){   userApproval@S3( u )( ru ) }
    adminApproval( a )( ra ){   adminApproval@S5( a )( ra ) }
};
...
```

Same instance (session) subsequent requests

Apache

BPEL Orchestrator

BPEL Orchestrator

BPEL Orchestrator

BPEL Orchestrator

BPEL Orchestrator

ESB

S1

S2

S3

S4

S5

S6

Tomcat, Rails, Nginx, ...

PHP "pages" (JSP, Ruby, JS)
SOC already snuck into web programming

However:

• many “moving parts”;
• heterogeneous (specific know-how);
• hard to maintain;
• prone to breakage with modifications (upgrades, patches, etc).
Can Jolie aid web programming?
Jolie Microservices and Choreographies for the Web

Jolie Web Server (Leonardo)

Jolie Templates

Jolie Orchestrator (Behavioural Composition)

Jolie Orchestrator

Jolie Orchestrator

Jolie

Architectural Composition

S1

S2

S3

S4

S5

S6
Jolie Microservices and Choreographies for the Web

Jolie Web Server (Leonardo)

Jolie Templates

Jolie Orchestrator (Behavioural Composition)

Architectural Composition

Jolie

S1

S2

S3

S4

S5

S6
Jolie Microservices and Choreographies for the Web

Jolie Web Server (Leonardo)

- Jolie Templates

Jolie Orchestrator (Behavioural Composition)

Architectural Composition

S1
S2
S3
S4
S5
S6
Behavoural Composition

\[ B ::= η \quad \text{(input)} \]
\[ \overline{η} \quad \text{(output)} \]
\[ [ η_1 ] \{ B_1 \} \ldots [ η_n ] \{ B_n \} \quad \text{(input choice)} \]
\[ \textbf{if}(e) B_1 \textbf{else} B_2 \quad \text{(cond)} \]
\[ \textbf{while}(e) B \quad \text{(while)} \]
\[ B ; B' \quad \text{(seq)} \]
\[ B | B' \quad \text{(par)} \]
\[ \textbf{throw}(f) \quad \text{(throw)} \]
\[ x = e \quad \text{(assign)} \]
\[ x -> y \quad \text{(alias)} \]
\[ \text{nullProcess} \quad \text{(inact)} \]

\[ η ::= o(x) \quad \text{(one-way)} \]
\[ o(x) (e) \{ B \} \quad \text{(request-response)} \]

\[ \overline{η} ::= o@OP(e) \quad \text{(notification)} \]
\[ o@OP(e) (y) \quad \text{(solicit-response)} \]
Architectural Composition (Deployment)

\[ IP ::= \text{inputPort} \ Port \quad OP ::= \text{outputPort} \ Port \]

\[ Port ::= \text{id} \ { \text{Location: \ Loc} \quad \text{Protocol: \ Proto} \quad \text{Interfaces: \ iface}_1, \ldots, \text{iface}_n } \]

+ **Aggregation**

+ **Redirection**
Jolie can support web applications with the **http** protocol.
Jolie HTTP Protocol

```java
interface SumIface { RequestResponse: sum(SumT)(int) }

inputPort SumInput {
  Location: "socket://localhost:8000"
  Protocol: soap
  Interfaces: SumIface
}

main {
  sum( req )( resp ) {
    resp = req.x + req.y
  }
}
```

Jolie Microservices and Choreographies for the Web

Jolie HTTP Protocol

```java
interface SumIface { RequestResponse: sum(SumT)(int) }

inputPort SumInput {
Location: "socket://localhost:8000"
Protocol: soap
Interfaces: SumIface
}

main {
sum( req )( resp ) {
resp = req.x + req.y
}
}
```

Jolie HTTP Protocol

```java
interface SumIface { RequestResponse: sum(SumT)(int) }

inputPort SumInput {
  Location: "socket://localhost:8000"
  Protocol: soap
  Interfaces: SumIface
}

main {
  sum( req )( resp ) {
    resp = req.x + req.y
  }
}
```

Jolie HTTP Protocol

http://localhost:8000/sum?x=2&y=3
Jolie HTTP Protocol

Jolie Web Server (Leonardo)

Jolie Templates

Jolie Orchestrator (Behavioural Composition)

Architectural Composition

Jolie

S1
S2
S3
S4
S5
S6
Jolie HTTP Protocol

- Jolie Web Server (Leonardo)
- Jolie Templates
- Jolie Orchestrator (Behavioural Composition)

Architectural Composition

S1
S2
S3
S4
S5
S6
Jolie HTTP Protocol

type FetchBib: void { .dblpKey: string }

interface DBLPInterface {
RequestResponse: fetchBib( FetchBib )( string )
}

outputPort DBLP {
Location: "socket://dblp.uni-trier.de:80/"
Protocol: http {
  .osc.fetchBib.alias = "rec/bib2/%!{dblpKey}.bib";
  .format = "html" }
Interfaces: DBLPInterface
}

main {
  r.dblpKey = args[0];
  fetchBib@DBLP ( r ) ( bibtex );
  println@Console( bibtex )();
}

Jolie HTTP Protocol

```typescript
type FetchBib: void { .dblpKey: string }

interface DBLPIface {
    RequestResponse: fetchBib( FetchBib ) ( string )
}

outputPort DBLP {
    Location: "socket://dblp.uni-trier.de:80/"
    Protocol: http {
        .osc.fetchBib.alias = "rec/bib2/%!{dblpKey}.bib";
        .format = "html" }
    Interfaces: DBLPIface
}

main
{
    r.dblpKey = args[0];
    fetchBib@DBLP( r )( bibtex );
    println@Console( bibtex )()
}
```

Jolie HTTP Protocol

```java
type FetchBib: void { .dblpKey: string }

interface DBLPIface {
RequestResponse: fetchBib( FetchBib )( string )
}

outputPort DBLP {
Location: "socket://dblp.uni-trier.de:80/"
Protocol: http {
.osc.fetchBib.alias = "rec/bib2/%!{dblpKey}.bib";
.format = "html" }
Interfaces: DBLPIface
}

main
{
    r.dblpKey = args[0];
    fetchBib@DBLP( r )( bibtex );
    println@Console( bibtex )()
}
```

**Major Novelty:**
Operation-Specific Configurations (osc) bridge the gap between RESTful and Service-Oriented Architectures

Jolie HTTP Protocol

Jolie Web Server (Leonardo)

Jolie Templates

Jolie Orchestrator (Behavioural Composition)

S1
S2
S3
S4
S5
S6

Architectural Composition

Jolie
Jolie HTTP Protocol

Same instance (session) subsequent requests

Jolie Web Server (Leonardo)

Jolie Templates

Jolie Orchestrator (Behavioural Composition)

Architectural Composition

Jolie

S1

S2

S3

S4

S5

S6
Multiple identifiers for the same process. We are particularly interested in the second (i) compound session identifier based on multiple values, as in BPEL [12], and (ii) behaviours. However, correlation sets are more expressive when it comes to providing that the generation and sending of correlation variables is explicit programmed in Jolie in the behaviour of a service. We discuss the usage of hyperlinks to keep track of process execution in § 7.


```java
inputPort RISInput {
    /* ... */
}
Protocol: http {
    .cookies.userKeyCookie = "userKey"
}

outputPort Logger { /* ... */ }
outputPort Moderator { /* ... */ }

main {
    login( cred )( r ) {
        checkCredentials;
        r.userKey = csets.userKey = new
    }
    addPub( pub );
    noti.bibtex = pub.bibtex;
    noti.modKey = csets.modKey = new;
    { log@Logger( pub.bibtex )
        | notify@Moderator( noti )
    };
    [ approve() ] {
        log@Logger( "Accepted " + pub.bibtex );
        updateDB
    }
    [ reject() ] {
        log@Logger( "Rejected " + pub.bibtex )
    }
}
```
Jolie HTTP Protocol

**inputPort** RISInput {
/* ... */
}

**outputPort** Logger { /* ... */ }
**outputPort** Moderator { /* ... */ }

**cset** { **userKey**: addPub.userKey }
**cset** { **modKey**: approve.modKey reject.modKey }

**define** checkCredentials { /* ... */ }
**define** updateDB { /* ... */ }

```plaintext

{ cookies.userKeyCookie = "userKey" }

main
{
    login( cred )( r ) {
        checkCredentials;
        r.userKey = csets.userKey = new
    }
    addPub( pub );
    noti.bibtex = pub.bibtex;
    noti.modKey = csets.modKey = new;
    { log@Logger( pub.bibtex )
        | notify@Moderator( noti ) };
    [ approve() ] {
        log@Logger( "Accepted " + pub.bibtex );
        updateDB
    }
    [ reject() ] {
        log@Logger( "Rejected " + pub.bibtex )
    }
}
```

Jolie HTTP Protocol

```java
inputPort RISInput {
/* ... */
Protocol: http {
   .cookies.userKeyCookie = "userKey"
}
}

outputPort Logger { /* ... */ }
outputPort Moderator { /* ... */ }

cset { userKey: addPub.userKey }
cset { modKey: approve.modKey reject.modKey }

define checkCredentials { /* ... */ }
define updateDB { /* ... */ }

main
{
   login( cred )( r ) {
      checkCredentials;
      r.userKey = csets.userKey = new
   }
   addPub( pub );
   noti.bibtex = pub.bibtex;
   noti.modKey = csets.modKey = new;
   { log@Logger( pub.bibtex )
      | notify@Moderator( noti )
   }
   [ approve() ] {
      log@Logger( "Accepted " + pub.bibtex );
      updateDB
   }
   [ reject() ] {
      log@Logger( "Rejected " + pub.bibtex )
   }
}
```

Integration with cookies


Multiparty Session
Jolie HTTP Protocol

inputPort RISInput {
    /* ... */
}

Protocol: http { .cookies.userKeyCookie = "userKey" }

outputPort Logger { /* ... */ }
outputPort Moderator { /* ... */ }

Main

{ login( cred ) ( r ) {
    checkCredentials;
    r.userKey = csets.userKey = new
}

addPub( pub );
noti.bibtex = pub.bibtex;
noti.modKey = csets.modKey = new;
{ log@Logger( pub.bibtex )
    | notify@Moderator( noti ) ;
}

[ approve() ] {
    log@Logger( "Accepted " + pub.bibtex ) ;
    updateDB
}

[ reject() ] {
    log@Logger( "Rejected " + pub.bibtex )
}

Multiparty Session

Correlation Sets defined on types (of operations).

Jolie HTTP Protocol

Jolie Web Server (Leonardo)

Jolie Templates

Jolie Orchestrator (Behavioural Composition)

Architectural Composition

S1
S2
S3
S4
S5
S6
Jolie HTTP Protocol

Jolie Web Server (Leonardo)

Jolie Templates

Jolie Orchestrator (Behavioural Composition)

Architectural Composition

Jolie

S1

S2

S3

S4

S5

S6
Jolie HTTP Protocol

```plaintext
outputPort RIS {
  Location: "socket://www.ris-example.com:8090/"
  Protocol: sodep
  Interfaces: RISIface
}

inputPort WebServerInput {
  Location: "socket://www.webserver-example.com:80/"
  Protocol: http {
    .default.get = "get";
    .cookies.userKeyCookie = "userKey"
    /* ... */
  }
  Interfaces: GetIface
  Aggregates: RIS
}

main {
  get( req )( resp ) {
    /* ... */
    readFile@File( req.requestUri )( resp )
  }
}
```

Jolie HTTP Protocol

```java
outputPort RIS {
  Location: "socket://www.ris-example.com:8090/"
  Protocol: sodep
  Interfaces: RISIface
}

inputPort WebServerInput {
  Location: "socket://www.webserver-example.com:80/"
  Protocol: http {
    .default.get = "get";
    .cookies.userKeyCookie = "userKey"
    /* ... */
  }
  Interfaces: GetIface
  Aggregates: RIS
}

main {
  get( req )( resp ) {
    /* ... */
    readFile@File( req.requestUri )( resp )
  }
}
```

Jolie HTTP Protocol

```plaintext
outputPort RIS {
    Location: "socket://www.ris-example.com:8090/"
    Protocol: sodep
    Interfaces: RISIface
}

inputPort WebServerInput {
    Location: "socket://www.webserver-example.com:80/"
    Protocol: http {
        .default.get = "get";
        .cookies.userKeyCookie = "userKey"
        /* ... */
    }
    Interfaces: GetIface
    Aggregates: RIS
}

main {
    get( req )( resp ) {
        /* ... */
        readFile@File( req.requestUri )( resp )
    }
}
```

Jolie HTTP Protocol

outputPort RIS {
  Location: "socket://www.ris-example.com:8090/
  Protocol: sodep
  Interfaces: RISIface
}

inputPort WebServerInput {
  Location: "socket://www.webserver-example.com:80/
  Protocol: http {
    .default.get = "get"
    .cookies.userKeyCookie = "userKey"
    /* ... */
  }
  Interfaces: GetIface
  Aggregates: RIS
}

main {
  get( req )( resp ) {
    /* ... */
    readFile@File( req.requestUri )( resp )
  }
}

Introduction to Choreographies
Introduction to Choreographies
Introduction to **Choreographies**

![Diagram of a transaction process involving a Client, ATM, Bank, and Card Issuer. The diagram includes steps such as withdrawal, card_id, pin, validation, approval, and an electron symbol representing a reaction.]
Introduction to **Choreographies**

| Client → ATM | withdraw | |
| ATM → Bank | card.id(String) | |
| ATM → Bank | pin(String) | |
| Bank → ATM | | acc k : lc.r[Card Issuer];
| | | k : Bank → Card Issuer.validation(data);
| | | if r.match(data) {
| | | | k : r[Card Issuer] → Bank.approve();
| | | } else {
| | | | k : r[Card Issuer] → Bank.reject();
| | | }

---

**Global Types**

| acc k : lc.p[ATM], lb.q[Bank];
| k : Client → p[ATM].withdraw(data);
| k : l_c.p[ATM], l_b.q[Bank];
| k : Client → p[ATM].withdraw(data);
| k : p[ATM].data.card_id → q[Bank].card.id(validate.card);
| k : p[ATM].data.pin → q[Bank].pin(validate.pswd);
| req k' : q[Bank] ↔ l_c.Card Issuer;
| k' : q[Bank].validate → Card Issuer.validation;
| k' : Card Issuer → q[Bank] {
| | .approve();
| | k : q[Bank] → p[ATM].approve();
| | .reject();
| | k : q[Bank] → p[ATM].reject();
| |

---

Saverio Giallorenzo | sgiallor@cs.unibo.it | DISI@Unibo

Workshop on Web/Reactive Programming | 39
Introduction to Choreographies

Main features:

• Global View;
• Modular, models asynchrony;
• Projects to correct processes;
• Deadlock- and Race-free by construction.
Choreographies for the Web?

Open for discussion!