Microservices

scenarios of the near and far future

Saverio Giallorenzo
Howdy

Saverio

Post-doc at the Department of Computer Science and Engineering of University of Bologna.

Research topics:
- Concurrent and distributed programming;
- Choreographies, Session
- Types and Process Algebras.
- Microservices;
- Jolie;

saverio.giallorenzo@gmail.com  |  DISI at Unibo  |  Bologna  |  Meeting on Microservices 2016
Microservices

scenarios of near and far future

WARNING

MAY CONTAIN CHOREOGRAPHIES

Saverio Giallorenzo
Today’s Limits

There is no effort without error and shortcoming.

“Citizenship in a Republic”, Theodore Roosevelt, 1910
Today’s Limits

innovation

There is no effort without error and shortcoming.

“Citizenship in a Republic”, Theodore Roosevelt, 1910
What’s here inside (e.g., error tracing)?

Distributed Programming
What’s here inside (e.g., error tracing)?

Internal functionality? Does the deliverer provide it? Docs/APIs?

Distributed Programming
Distributed Programming

What’s here inside (e.g., error tracing)?

Internal functionality? Does the deliverer provide it? Docs/APIs?
Distributed Programming

What’s here inside (e.g., error tracing)?

Internal functionality? Does the deliverer provide it? Docs/APIs?

Sequential (in which order) or in parallel?
“Not my problem”
“Not my problem”
“Not my problem”
“Not my problem”
“Not my problem”
“Not my problem”
“Not my problem”
Scalable Architectures
Scalable Architectures
Scalable Architectures

Sales
- Web Server
- CRM
  - Order Manag.

Storage
- Hardware Manag.
  - Order Manag.
  - Delivery Manag.
  - Delivery Tracking

Delivery
- Delivery Manag.

rebind the arrows at each “scaling”
Scalable Architectures

rebind the arrows at each “scaling”
Scalable Architectures

rebind the arrows at each “scaling”
A look at the future

Choreographic Programming
order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects )
Architectural Vision

```plaintext
order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects )
```
Architectural Vision

order@Client = getInput("Insert products");
request_quote: Client(order) -> Sales(order);
confirm_avail: Sales(order) -> Storage(objects)
order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects )
include checkAvail from "socket://storage:8000"

order@Client = getInput( "Insert products" )
request_quote: Client( order ) -> Sales( order )
confirm_avail: Sales( order ) -> Storage( objects )
avail@Storage = checkAvail( objects )
include checkAvail from "socket://storage:8000"

order@Client = getInput( "Insert products" );
request_quote: Client(order) -> Sales(order);
confirm_avail: Sales(order) -> Storage(objects);
avail@Storage = checkAvail(objects)
include checkAvail from "socket://storage:8000"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
What’s here inside

include checkAvail from “socket://storage:8000"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
include checkAvail from "socket://storage:8000"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )
Architectural Vision

Client → Sales

- Sales
  - Web Server
  - CRM
  - Order Manag.

Storage

- Storage
  - Hardware Manag.
  - Order Manag.

Delivery

- Delivery
  - Delivery Manag.
  - Delivery Tracking
Architectural Vision

Function `checkAvail`

- **Client**
- **Sales**
  - Web Server
  - CRM
  - Order Manag.
- **Storage**
  - Hardware Manag.
  - Order Manag.
- **Delivery**
  - Delivery Manag.
  - Delivery Tracking
include checkAvail from "socket://storage:8000"
include calcQuote from "socket://sales:8001"

order@Client = getInput("Insert products");
request_quote: Client(order) -> Sales(order);
confirm_avail: Sales(order) -> Storage(objects);
avail@Storage = checkAvail(objects);
if (avail)@Storage {
  quote@Sales = calcQuote(order);
  send_quote: Sales(quote) -> Client(quote);
  ...
}
else {
  product_unavailable: Sales() -> Client()
}
include checkAvail from "socket://storage:8000"
include calcQuote from "socket://sales:8001"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )

if ( avail )@Storage {
    quote@Sales = calcQuote( order );
    send_quote: Sales( quote ) -> Client( quote );
    ...
} else {
    product_unavailable: Sales() -> Client()
}
include checkAvail from "socket://storage:8000"
include calcQuote from "socket://sales:8001"

order@Client = getInput( "Insert products" );
request_quote: Client( order ) -> Sales( order );
confirm_avail: Sales( order ) -> Storage( objects );
avail@Storage = checkAvail( objects )

if ( avail )@Storage {
    quote@Sales = calcQuote( order );
    send_quote: Sales( quote ) -> Client( quote );
    ...
} else {
    product_unavailable: Sales() -> Client()
}
Netflix

Why not peer to peer choreography?
Netflix

Why not peer to peer choreography?

We found it was harder to scale with growing business needs and complexities.

Some of the issues associated with the approach are:
Netflix

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.
Netflix

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple applications.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.
Netflix (cont’d)

Why not peer to peer choreography?

We found it was harder to scale with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.
Netflix (cont’d)

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.
Netflix (cont’d)

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.
- True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.
- On the contrary. Choreographies help to clarify public functions and their APIs (I/Os).
Netflix (cont’d)

Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

- True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.
- On the contrary. Choreographies help to clarify public functions and their APIs (I/Os).
- Choreographies written in **AIOCJ** are adaptable at runtime!
Why not peer to peer choreography?

We found it was **harder to scale** with growing business needs and complexities.

Some of the issues associated with the approach are:

- Process flows are “embedded” within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.

True if you leave the choreographic domain. It is like writing C code and trying to change the program by changing the compiled assembly code.

On the contrary. Choreographies help to clarify public functions and their APIs (I/Os).

Choreographies written in **AIOCG** are adaptable at runtime!
Architectural Vision (Part II)
Architectural Vision (Part III)
Today’s Limits

There is no effort without error and shortcoming.
There is no effort without error and shortcoming.
There is no effort without error and shortcoming.
There is no effort without error and shortcoming.
Tomorrow’s Standards

- Distributed programming becomes easier;

There is no effort without error and shortcoming.

innovation effort
Tomorrow’s Standards

- Distributed programming becomes easier;
- Accountability and formal APIs;

There is no effort without error and shortcoming.
Tomorrow’s Standards

- Distributed programming becomes easier;
- Accountability and formal APIs;
- Scalable and reliable architectures.

There is no effort without error and shortcoming.

innovation  effort  innovation  effort
Thanks for the attention

Questions: Saverio(?) -> MoM2016(!)