AIOCJ: a Choreographic Framework for Safe Adaptive Distributed Applications

Mila dalla Preda[1], **Saverio Giallorenzo**[2], Ivan Lanese[2], Jacopo Mauro[2], and Maurizio Gabbrielli[2]

[1]Department of Computer Science - Univ. of Verona
[2]Department of Computer Science and Engineering - Univ. of Bologna / INRIA

Conference on Software Language Engineering, 2014

Why Choreographic?

Bob	Alice
`msg = "Want to dance?";`	`sendMessage: msg from Bob;`
`sendMessage: msg to Alice;`	`response = show( msg );`
`ok: response from Alice`	`ok: response to Bob`

Why **Choreographic**?

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

Why **Choreographic**?

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

Projects to

Why **Choreographic**?

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

Projects to

Bob

```
msg = “Want to dance?”;
sendMessage: msg to Alice;
ok: response from Alice
```

Alice

```
sendMessage: msg from Bob;
response = show( msg );
ok: response to Bob
```

Why **Choreographic**?

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

Why **Choreographic**?

```
sendMessage: Bob( "Want to dance?" ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

What if we want to change (parts of it) at runtime?

Safe Adaptive Choreographies

Choreographies are suitable for programming safe distributed systems.

Can we make them suitable for programming **safe** and **adaptable** distributed systems?

AIOCJ is our attempt at giving a positive answer to this question.

Safe Adaptive Choreographies

We deem **AIOCJ** suitable because:

1. It gives a general and neat overview of the (interaction in the) whole system;
2. It injects "good" (desirable) properties on distributed systems;
3. It has proven to be a feasible implementation of formal results. (We ensure "good" properties to hold on the distributed system at runtime and after any step of adaptation).

1. Neat overview

Neat overview | The **AIOC** Language

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

Neat overview | The **AIOC** Language

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

- **Interactions** (synchronous)**;**

Neat overview | The **AIOC** Language

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

- Interactions (synchronous);
- **Local Computation**;

Neat overview | The **AIOC** Language

```
sendMessage: Bob( "Want to dance?" ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

- Interactions (synchronous);
- Local Computation;
- **Participants**;

Neat overview | The **AIOC** Language

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

- Interactions (synchronous);
- Local Computation;
- Participants;
- **Operations**;

Neat overview | The **AIOC** Language

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

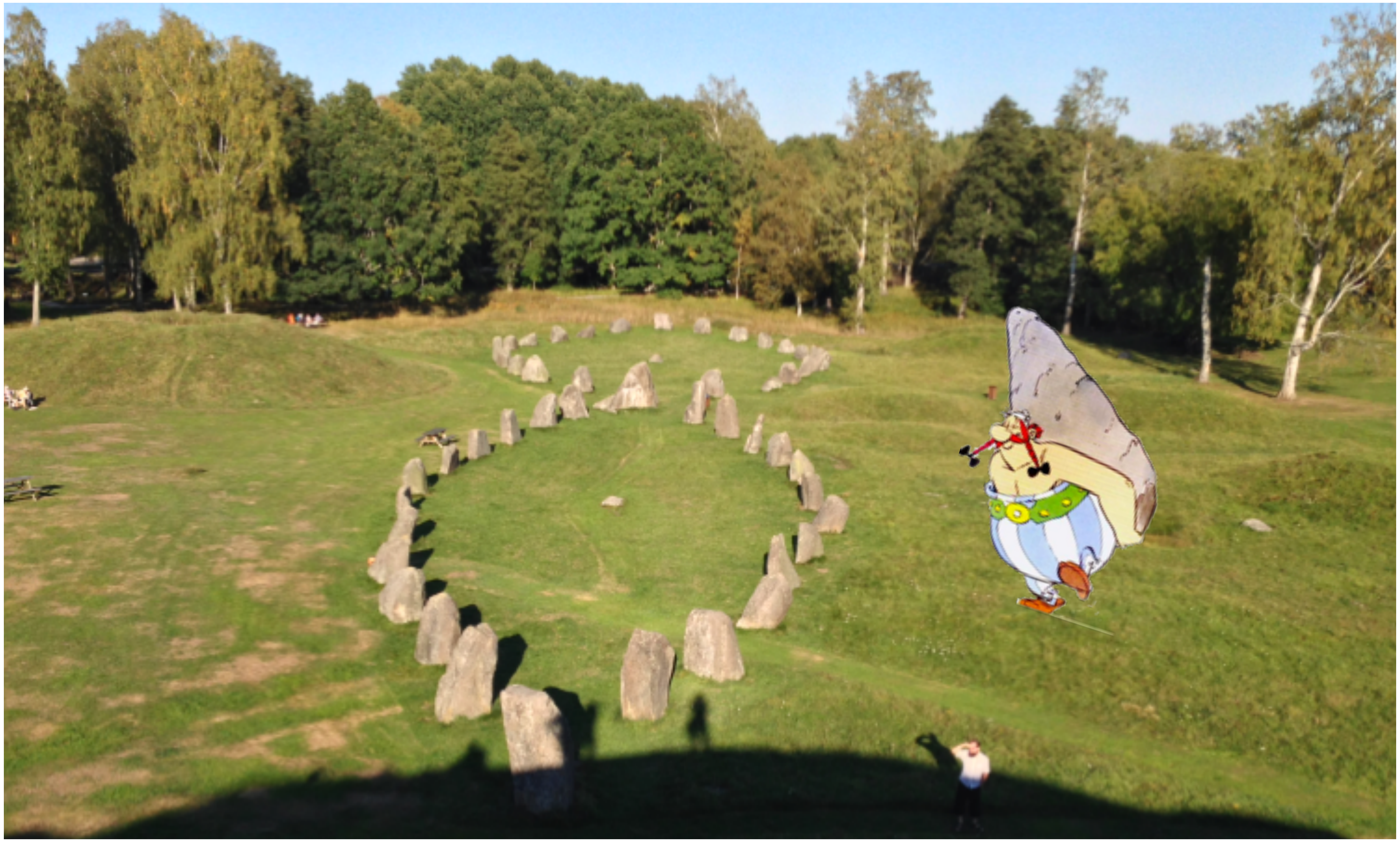
- Interactions (synchronous);
- Local Computation;
- Participants;
- Operations;
- **Functions**;

Neat overview | The **AIOC** Language

```
sendMessage: Bob( “Want to dance?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

- Interactions (synchronous);
- Local Computation;
- Participants;
- Operations;
- Functions;
- **Data**.

1. Neat overview,



1. Neat overview,

also when programming adaptation

Neat overview | The **AIOC** Language

Scopes

Neat overview | The **AIOC** Language

A **scope** defines a part of the interaction that can be replaced (adapted).

Neat overview | The **AIOC** Language

A **scope** defines a part of the interaction that can be replaced (adapted).

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” }
```

- **Scope Declaration**;

Neat overview | The **AIOC** Language

A **scope** defines a part of the interaction that can be replaced (adapted).

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” }
```

- Scope Declaration;
- **Scope Leader**;

Neat overview | The **AIOC** Language

A **scope** defines a part of the interaction that can be replaced (adapted).

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” }
```

- Scope Declaration;
- Scope Leader;
- **Sub-choreography**;

Neat overview | The **AIOC** Language

A **scope** defines a part of the interaction that can be replaced (adapted).

```
scope @Bob {
  sendMessage: Bob( "Want to dance?" ) -> Alice( msg )
} prop { N.scopename = "hangout" }
```

- Scope Declaration;
- Leader;
- Sub-choreography;
- **Scope properties**;

Neat overview | The **AIOC** Language

Neat overview | The **AIOC** Language

Rules

Neat overview | The **AIOC** Language

A **rule** defines a choreography that
can replace a scope.

Neat overview | The **AIOC** Language

A **rule** defines a choreography that can replace a scope.

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

- **Rule Declaration**;

Neat overview | The **AIOC** Language

A **rule** defines a choreography that can replace a scope.

```
rule {
 on { N.scopename == "hangout" }
 do {
  sendMessage: Bob( "What about Movies?" ) ->
   Alice( msg )
 }
}
```

prefixes
N. - properties of the scope;
E. - environmental variables;
non prefixed variables are local to the leader.

- Rule Declaration;
- **Applicability Condition**;

Neat overview | The **AIOC** Language

A **rule** defines a choreography that can replace a scope.

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

- Rule Declaration;
- Applicability Condition;
- **New Choreography**.

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )

rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )

rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )


rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
scope @Bob {
  sendMessage: Bob( “Want to dance?” ) -> Alice( msg )
} prop { N.scopename = “hangout” };
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )


rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
sendMessage: Bob( “What about Movies?” ) -> Alice( msg );
response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

```
rule {
 on { N.scopename == “hangout” }
 do {
  sendMessage: Bob( “What about Movies?” ) ->
   Alice( msg )
 }
}
```

Neat overview | The **AIOC** Language

What happens at runtime? Easy to figure out.

```
sendMessage: Bob( “What about Movies?” ) -> Alice( msg );

response@Alice = show( msg );
ok: Alice( response ) -> Bob( response )
```

2. “Good” properties

"Good" Properties

|

Deadlock- and Race-freedom by construction.

“Good” properties | **deadlock- and race-freedom**

Choreographies are deadlock- and race-free by construction.

1. **Interactions are atomic**
2. We enforce **well-formed choreographies** both in AIOCJ programs and rules.
3. **Correctness of projection**

“Good” Properties

||

Consistency of Adaptation

“Good” properties | **consistency of adaptation**

When a scope of an AIOCJ program adapts, the adaptation is **consistent** among the participants.

“Good” properties | **consistency of adaptation**

When a scope of an AIOCJ program adapts, the adaptation is **consistent** among the participants.

```
scope @Bob {
  sendMessage: Bob( a ) -> Alice( b )
} prop { N.scopename = “hangout” }
```

"Good" properties | **consistency of adaptation**

When a scope of an AIOCJ program adapts, the adaptation is **consistent** among the participants.

```
scope @Bob {
  sendMessage: Bob( a ) -> Alice( b )
} prop { N.scopename = "hangout" }
```

Bob is the **leader** of this scope of adaptation.

“Good” properties | consistency of adaptation

When a scope of an AIOCJ program adapts, the adaptation is **consistent** among the participants.

```
scope @Bob {
  sendMessage: Bob( a ) -> Alice( b )
} prop { N.scopename = “hangout” }
```

Bob is the **leader** of this scope of adaptation.
Only Bob can query the repositories of rules.

“Good” properties | **consistency of adaptation**

When a scope of an AIOCJ program adapts, the adaptation is **consistent** among the participants.

```
scope @Bob {
  sendMessage: Bob( a ) -> Alice( b )
} prop { N.scopename = “hangout” }
```

Bob is the **leader** of this scope of adaptation.
Only Bob can query the repositories of rules.
He decides whether to adapt and which rule applies.

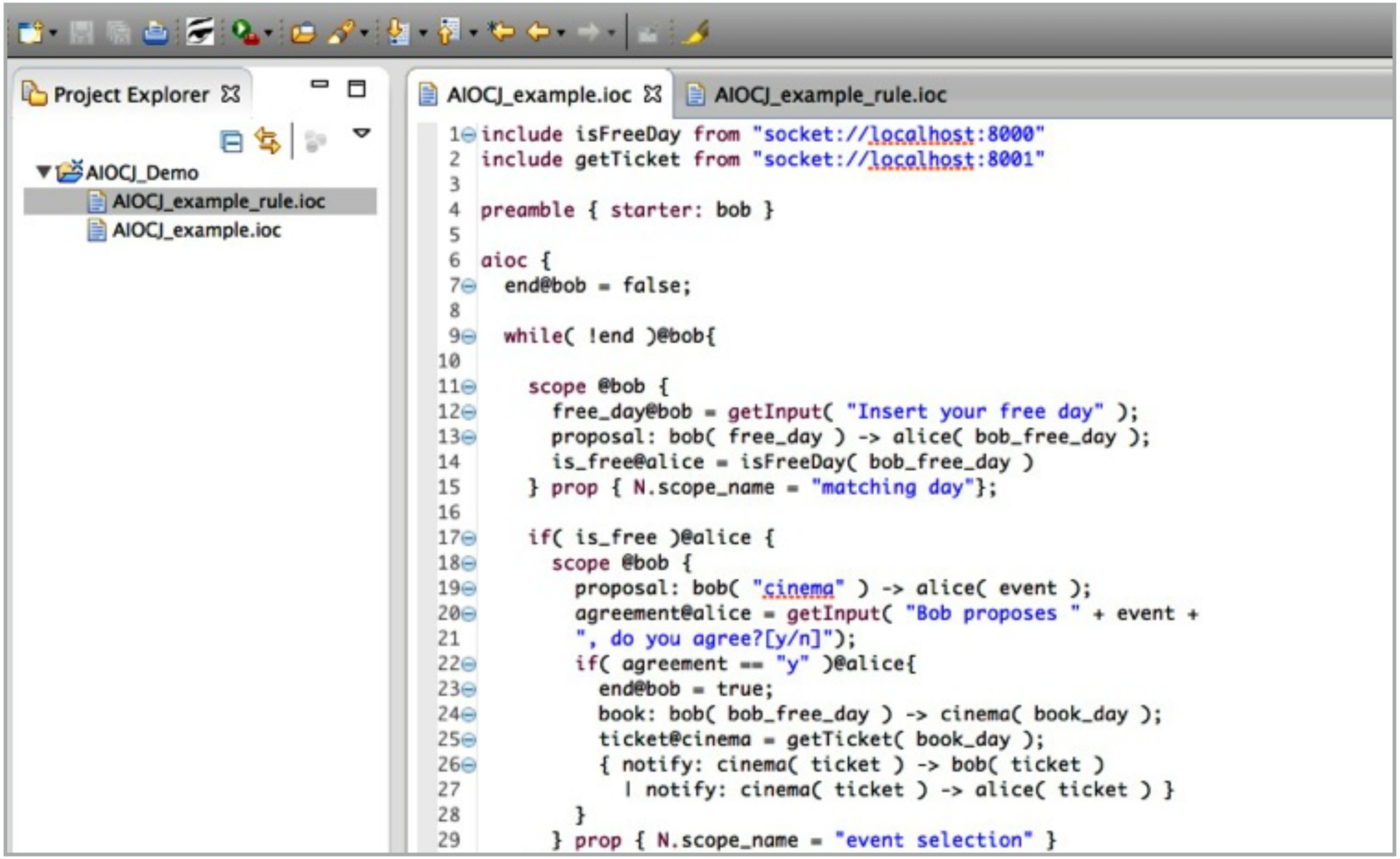
"Good" properties | **consistency of adaptation**

When a scope of an AIOCJ program adapts, the adaptation is **consistent** among the participants.

```
scope @Bob {
  sendMessage: Bob( a ) -> Alice( b )
} prop { N.scopename = “hangout” }
```

Bob is the **leader** of this scope of adaptation.
Only Bob can query the repositories of rules.
He decides whether to adapt and which rule applies.

This allows rules to change at runtime!



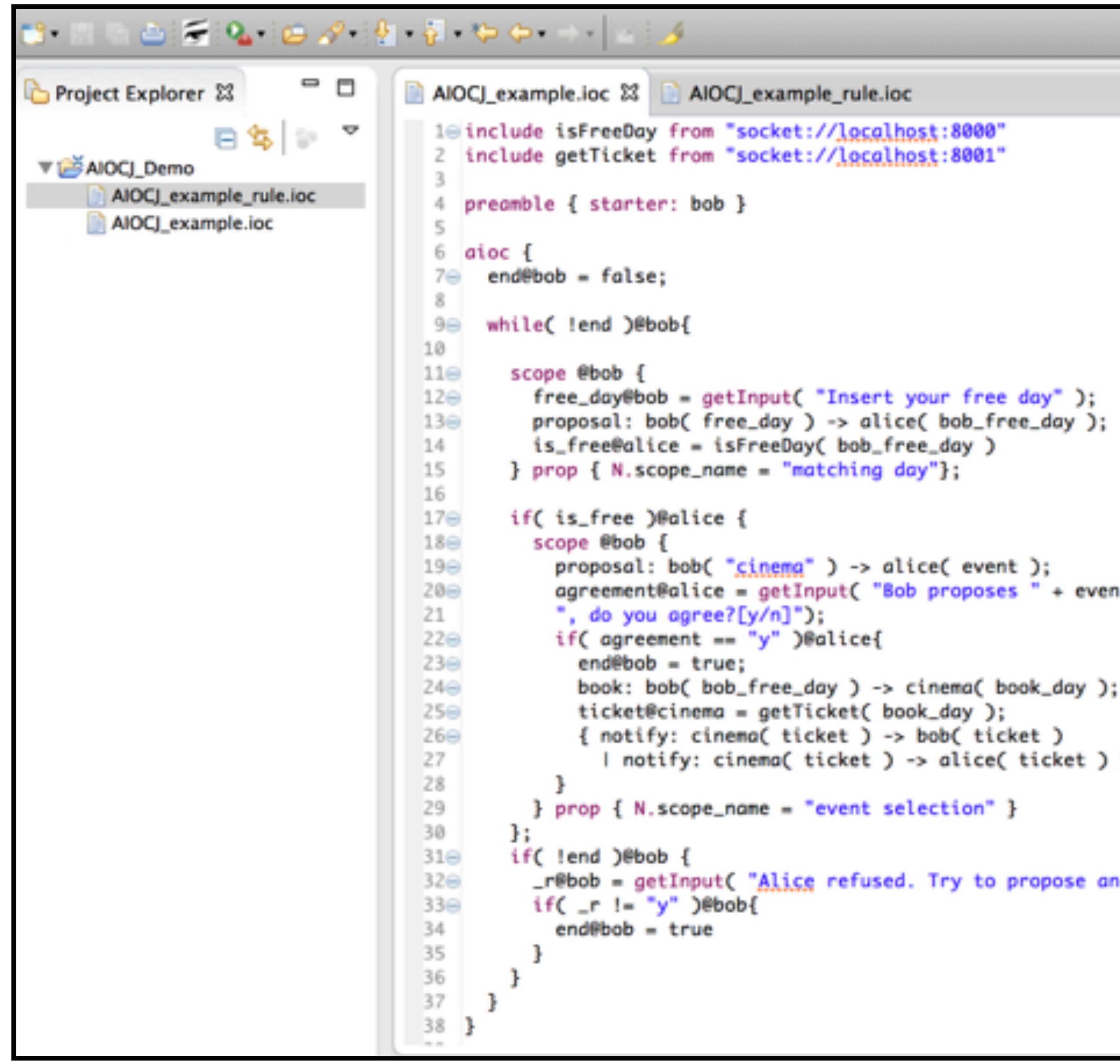
3. Feasible

Website: http://bit.do/**aiocj**

Feasible | **The AIOCJ Framework**

AIOCJ-ecl.

Plug-in for Eclipse.



Provides:

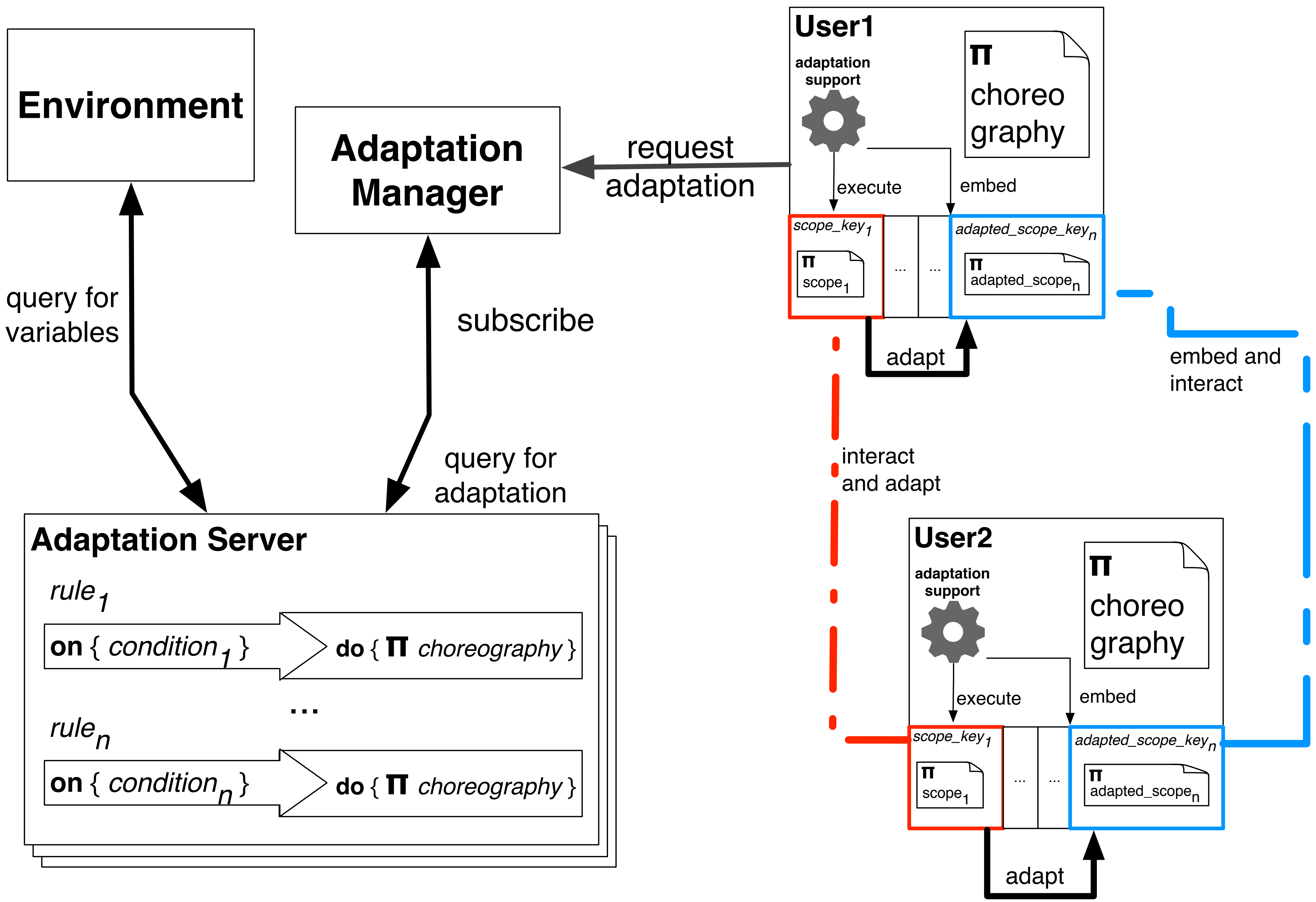
- syntax highlighting;
- syntax checking;
- online correctness checking;
- Projection to Jolie

www.jolie-lang.org

Website: http://bit.do/**aiocj**

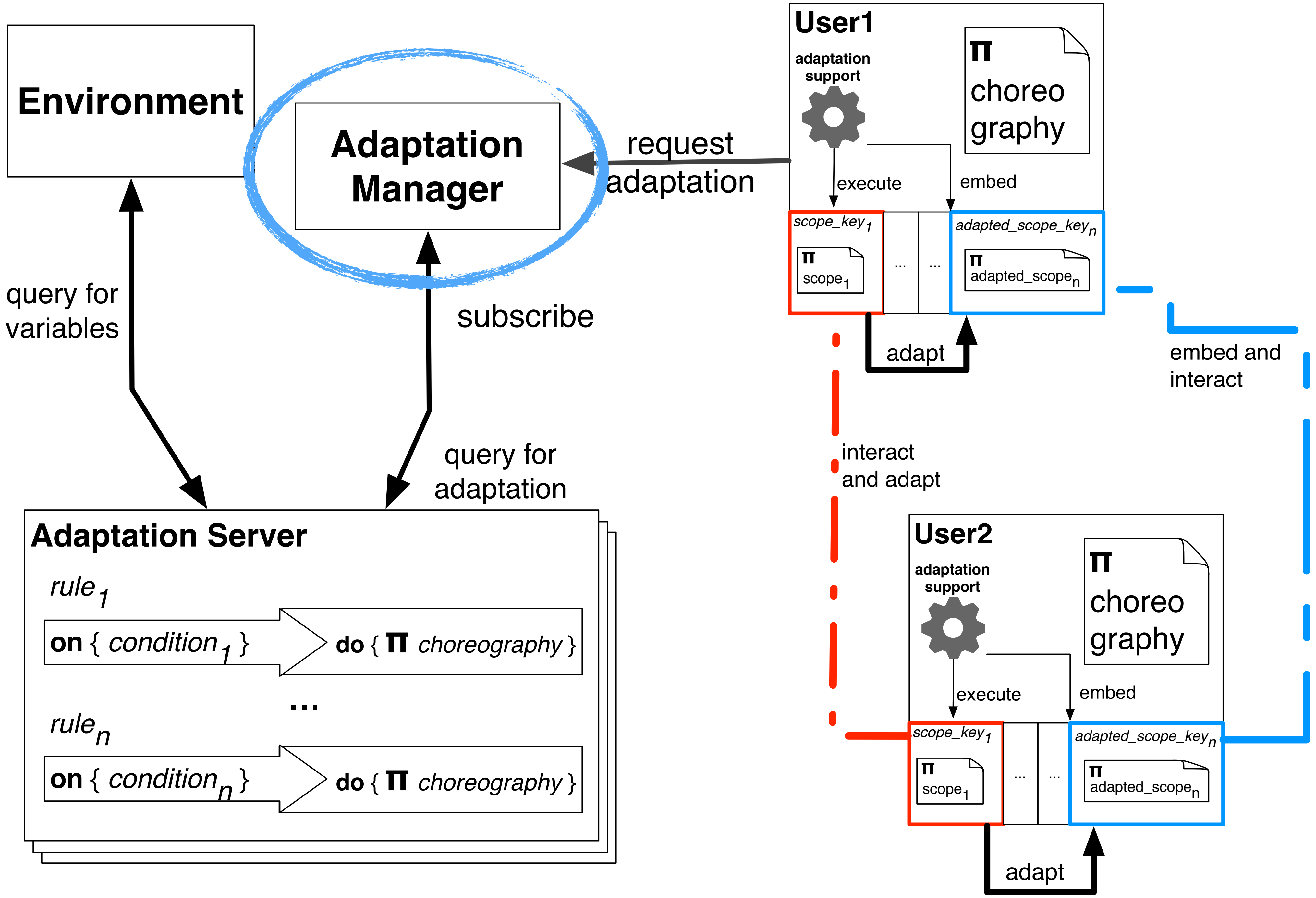
Feasible | **The AIOCJ Framework**

AIOCJ-mid



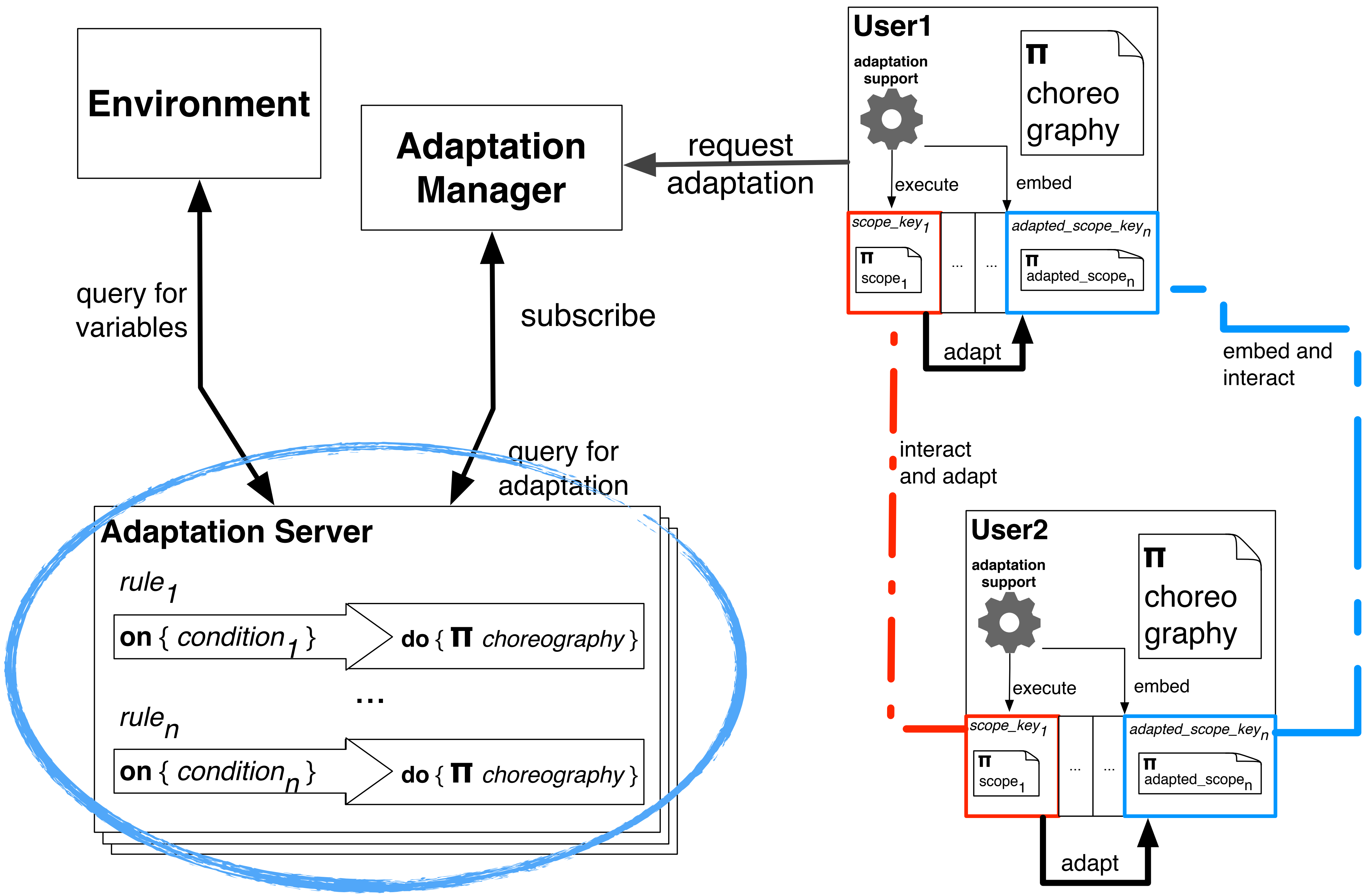
Feasible | **The AIOCJ Framework**

AIOCJ-mid



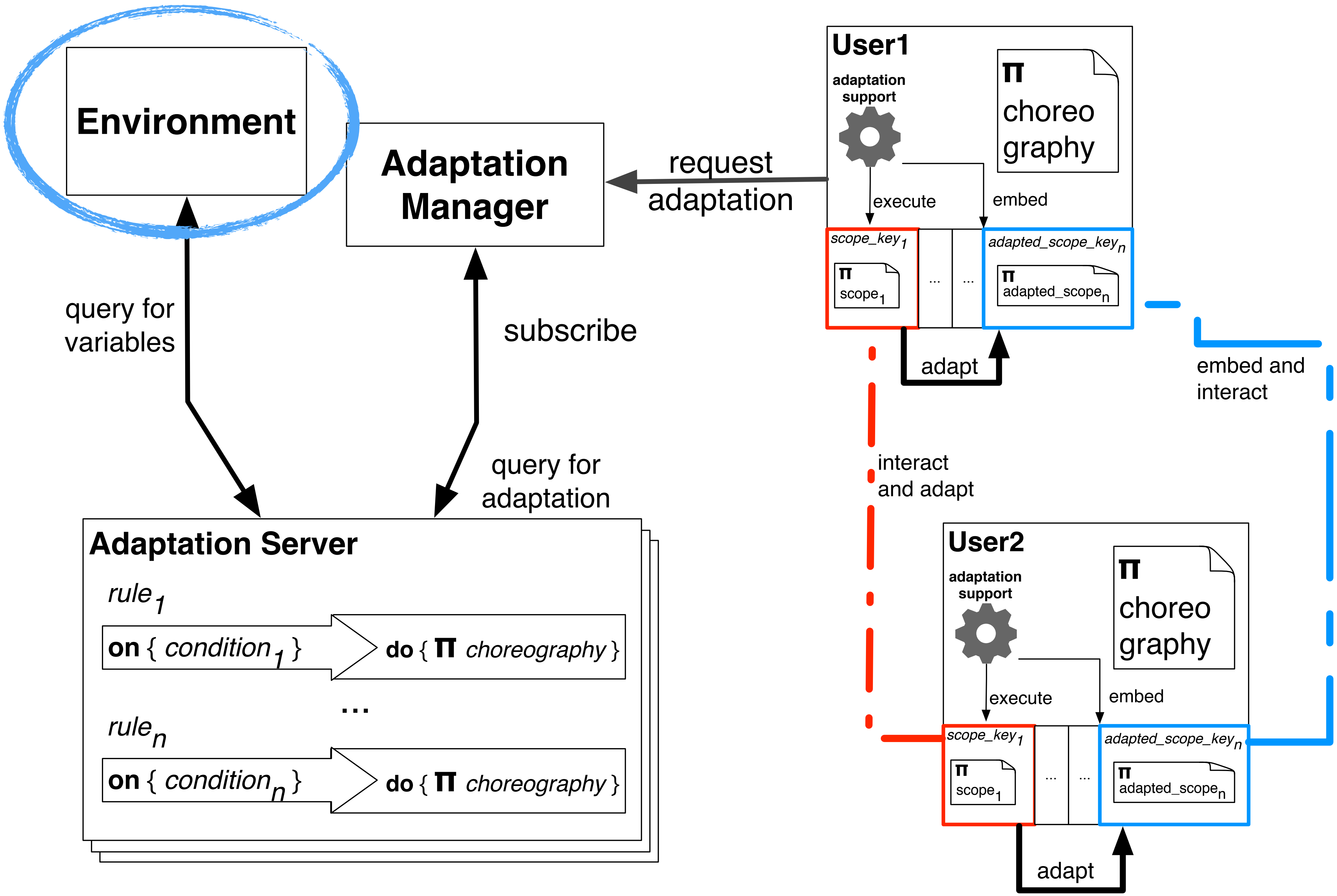
Feasible | **The AIOCJ Framework**

AIOCJ-mid



Feasible | **The AIOCJ Framework**

AIOCJ-mid



Demonstration

Safe Adaptive Choreographies | RECAP

Choreographies are suitable for programming safe distributed systems.

With **AIOCJ**, we made a first attempt at making them suitable for programming **safe** and **adaptable** distributed systems.

Website: http://bit.do/**aiocj**

Safe Adaptive Choreographies | RECAP

Main features of **AIOCJ**:

1. It gives a general and neat overview of the (interaction in the) whole system;

2. It injects "good" (desirable) properties on distributed systems;

3. It has proven to be a feasible implementation of formal results. (We ensure "good" properties to hold on the distributed system at runtime and after any step of adaptation).

Future Work

What is still missing?

- Communications in AIOCJ are synchronous. We are planning to include also **asynchronous** communications;
- Sessions;
- Injection of AIOCJ “good” properties in other adaptation mechanisms. E.g., Aspect-Oriented or Context-Oriented Programming, etc…

Thanks for your time

AIOCJ

Framework for Safe Adaptive Distributed Applications

Mila dalla Preda

[1]Department of Computer Science - Univ. of Verona
[2]Department of Computer Science and Engineering - Univ. of Bologna / INRIA

Conference on Software Language Engineering, 2014

Appendix

Connectedness Properties

“Good” properties | **deadlock- and race-freedom**

Choreographies are deadlock-free by construction.

Connectedness for sequence

```
op1: Bob( a ) -> Alice( b );
op2: Alice( b ) -> Bob( c );
```

“Good” properties | **deadlock- and race-freedom**

Choreographies are deadlock-free by construction.

Connectedness for sequence

```
op1: Bob( a ) -> Alice( b );
op2: Alice( b ) -> Bob( c );
op3: Carol( d ) -> Dave( e )
```

"Good" properties | **deadlock- and race-freedom**

Choreographies are deadlock-free by construction.

Connectedness for sequence

```
op1: Bob( a ) -> Alice( b );
op2: Alice( b ) -> Bob( c );
op3: Carol( d ) -> Dave( e )
```

No causality relation between either Alice, Bob, Carol or Dave

"Good" properties | **deadlock- and race-freedom**

Choreographies are deadlock-free by construction.

Connectedness for sequence

```
op1: Bob( a ) -> Alice( b );
op2: Alice( b ) -> Bob( c ) |
op3: Carol( d ) -> Dave( e )
```

A natural enforcement.
Probably the programmer wanted the last two instructions to run in parallel

"Good" properties | **deadlock- and race-freedom**

Choreographies are deadlock-free by construction.

Connectedness for parallel

```
op1: Bob( a ) -> Alice( c )|
op1: Bob( b ) -> Alice( d )
```

There might be interference between these interactions.

Interactions with the same signature (operation, sender, receiver) in parallel are forbidden.

AIOC Language Syntax

AIOC Program Syntax

$$
\begin{array}{rcl}
\mathcal{C} & ::= & \text{Include}^* \\
 & & \text{Preamble} \\
 & & \texttt{aioc}\ \ \{\ \ \mathcal{I}\ \ \} \\
\text{Include} & ::= & \texttt{include}\ \ f^+\ \ \texttt{from}\ \ \text{Location} \\
\text{Preamble} & ::= & \texttt{preamble}\ \ \{ \\
 & & \quad \texttt{starter}:\ \ r \\
 & & \quad \text{Deployment}^* \\
 & & \} \\
\text{Deployment} & ::= & \texttt{location@}r : \text{Location}
\end{array}
$$

AIOC Behaviour Syntax

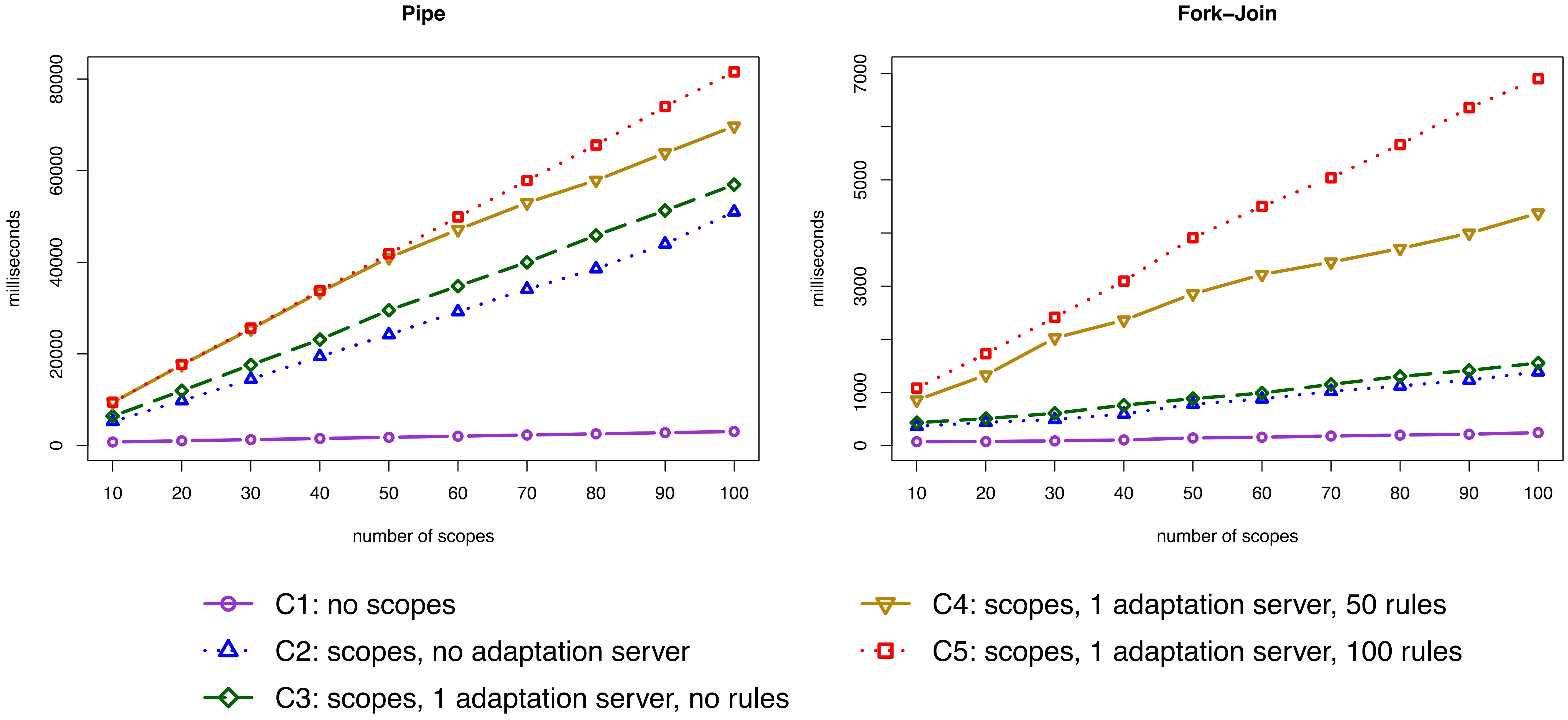
$$\begin{aligned}
\mathcal{I} \quad ::= \quad & o^? : r_1(e) \texttt{ -> } r_2(x) \quad | \quad \mathcal{I}\texttt{;}\mathcal{I}' \quad | \quad \mathcal{I}\texttt{|}\mathcal{I}' \\
| \quad & x\texttt{@}r = local \quad | \quad \texttt{skip} \quad | \quad \texttt{while} \quad b\texttt{@}r \quad \{\mathcal{I}\} \\
| \quad & \texttt{if} \quad b\texttt{@}r \quad \{\mathcal{I}\} \quad \texttt{else} \quad \{\mathcal{I}'\} \\
| \quad & \texttt{scope} \quad \texttt{@}r \quad \{\mathcal{I}\} \\
& [\, \texttt{prop} \quad \{\text{list of } \texttt{N}.x = e\} \,] \\
& [\, \texttt{roles} \quad \{r_i, \ldots, r_j\} \,]
\end{aligned}$$

$$local \quad ::= \quad e \quad | \quad f \quad | \quad \texttt{getInput}(x) \quad | \quad \texttt{show}(x)$$

Rules Syntax

$$
\begin{array}{ll}
\mathcal{R} ::= & \texttt{rule} \quad \{ \\
& \quad \text{Include}^* \\
& \quad \texttt{on} \quad \{ \quad \mathcal{B} \quad \} \\
& \quad \texttt{do} \quad \{ \quad \mathcal{I} \quad \} \\
& \}
\end{array}
$$

Performances


Pipe
Fork–Join
milliseconds
number of scopes
0
20000
40000
60000
80000
1000
3000
5000
7000
10
20
30
40
50
60
70
80
90
100
C1: no scopes
C2: scopes, no adaptation server
C3: scopes, 1 adaptation server, no rules
C4: scopes, 1 adaptation server, 50 rules
C5: scopes, 1 adaptation server, 100 rules