Type-safe Communication in Java with Session Types

Raymond Hu¹, Nobuko Yoshida¹ and Kohei Honda²

1. Imperial College London; 2. Queen Mary, Uni. of London
Overview (1)

1. Background and Related Work.
2. Worked Example: Session Programming.
3. Runtime Support for Session Communication.
4. Conclusion and Future Work.
Session Types

• Type systems for process calculi:
Session Types

- Type systems for process calculi:

- Session types for object calculi:
  - Dezani-Ciancaglini et al. Session Types for Object-oriented Languages. (2006)
  - Coppo et al. *Asynchronous Session Types and Progress for Object-oriented…* (2007)
Session Types

- Type systems for process calculi:

- Session types for object calculi:
  - Dezani-Ciancaglini et al. Session Types for Object-oriented Languages. (2006)
  - Coppo et al. *Asynchronous Session Types and Progress for Object-oriented*… (2007)

➤ Implementation of a practical, distributed language.
Implementation of Session Types

- Singularity OS:
  - Restricted features, not distributed.
Implementation of Session Types

- Singularity OS:
  - Restricted features, not distributed.
- Multithreaded functional language design [VGR06]:
  - Not distributed, no actual implementation.
Implementation of Session Types

• Singularity OS:
  – Restricted features, not distributed.

• Multithreaded functional language design [VGR06]:
  – Not distributed, no actual implementation.

• Encodings of sess. types (e.g. Haskell [NT04], C++):
  – Proof of concept, but not full implementations.
Implementation of Session Types

- Singularity OS [FAHHHLL06]:
  - Restricted features, not distributed.
- Multithreaded functional language design [VGR06]:
  - Not distributed, no actual implementation.
- Encodings of sess. types (e.g. Haskell [NT04], C++):
  - Proof of concept, but not full implementations.

- Why use untyped sockets if we can use sessions?
Overview (2)

1. Background and Related Work.
2. Worked Example: Language Design.
3. Runtime Support for Session Communication.
4. Conclusion and Future Work.
An Online Ticket Ordering System

Repeat until method of travel is decided

travDetails: String

cost: Double

ACCEPT

delivAddr: Address

disDate: Date

A requests session s with B
A sends v of type t to B
A selects LAB at B
A and B close s
A can choose to repeat this part of the session
Protocol Specification (1)

begin

\begin{center}
\begin{tikzpicture}[node distance=2cm, >=stealth, \figfont]
  \node (customer) [draw, shape=rectangle] {Customer};
  \node (agency) [draw, shape=rectangle, right of=customer] {Agency};
  \draw [->] (customer) -- node [near start] {\texttt{travDetails}: String} (agency);
  \draw [->] (agency) -- node [near end] {\texttt{cost}: Double} (customer);
  \draw [dashed] (customer) -- node [below] {Repeat until method of travel is decided} (agency);
  \draw [->] (customer) -- node [below] {$S_{ca}$} (agency);
\end{tikzpicture}
\end{center}
Protocol Specification (1)

begin.

!(String)
Protocol Specification (1)

begin.

!(String).

?(Double)
Protocol Specification (1)

begin.

![String).

?(Double)

]*.

...

Worked Example: Session Programming
Protocol Specification (1)

begin.

![](String).

?!(Double)

]*.

...

begin.

?[(String).

?!(Double)

]*.

...

Type-safe Communication in Java with Session Types

Worked Example: Session Programming
Protocol Specification (2)

```
... ?{
}
...
```

Worked Example: Session Programming
Protocol Specification (2)

...?

\{ 
    \textbf{ACCEPT}: \\
    \cdots, \\
    \textbf{REJECT}: \\
    \cdots 
\}.

\cdots
Protocol Specification (3)

```java
protocol p_as {
}
```

Worked Example: Session Programming
Protocol Specification (3)

```java
protocol p_as {
    begin.
    !(
    ).
    end
}
```
Protocol Specification (3)

```java
protocol p_as {
    begin.
    !(
        ?(Address).!(Date).end
    ).
    end
}
```
Protocol Implementation (1)

```java
protocol p_ca { begin.
  ![ (String).
  ?(Double)
]*.
...
```

// Customer
```
STSocket s_ca = STSocketImpl.create(p_ca);
```
Protocol Implementation (1)

```java
protocol p_ca {
    begin.
    ![ !(String) .
        ?(Double) ]*.
    ...
}

// Customer
STSocket s_ca =
    STSocketImpl.create(p_ca);

s_ca.request(Agency, port);
```
Protocol Implementation (1)

```
protocol p_ca {
  begin.
  ![ (String).
    !(String).
  (?Double)
    ?(Double)
  ]*. 
...

// Customer
STSocket s_ca =
  STSocketImpl.create(p_ca);

s_ca.request(Agency, port);

s_ca.outwhile(!decided) {

}
```
Protocol Implementation (1)

```java
protocol p_ca { begin.
  ![ (String).
  ?(Double)
] *

... // Customer

STSocket s_ca = STSocketImpl.create(p_ca);

s_ca.request(Agency, port);

s_ca.outwhile(!decided) {
  s_ca.send(travDetails);
}
```
Protocol Implementation (1)

```java
// Customer
STSocket s_ca = STSocketImpl.create(p_ca);

s_ca.request(Agency, port);

s_ca.outwhile(!decided) {
    s_ca.send(travDetails);
    cost = (Double)s_ca.receive();
    decided = ...;
}
```

```
protocol p_ca {
    begin.
    ![ (String).
    ?(Double) ]*
    ...
}
```
Protocol Implementation (1)

```java
protocol p_ca {
    begin.
    ![ (String).
    ?(Double)
}.*
... 
```

// Customer
```java
STSocket s_ca =
    STSocketImpl.create(p_ca);
s_ca.request(Agency, port);
s_ca.outwhile(!decided) {
    s_ca.send(travDetails);
    cost = s_ca.receive();
}
decided = ...;
```
Protocol Implementation (2)

...}

!{
  ACCEPT:
    ... , // 1

  REJECT:
    ... // 2
}

... // 2

// Customer

s_ca.select(ACCEPT) {
  ... // 1
}

... // 1
Protocol Implementation (2)

...  
!

    ACCEPT:  
    ...     // 1

    REJECT:  
    ...     // 2

}

// Customer
if (cost < 100.00) {
    s_ca.select(ACCEPT) {
        ...     // 1
    }
} else {

}
Protocol Implementation (2)

...  
!{  
  ACCEPT:  
    ... , // 1  
  REJECT:  
    ... // 2  
}  
...

// Customer
if (cost < 100.00) {
  s_ca.select(ACCEPT) {
    ... // 1  
  }
}
else {
  s_ca.select(REJECT) {
    ... // 2  
  }
}
Protocol Implementation (3)

...?

ACCEPT:
    ... , // 1'

REJECT:
    ... // 2'

}

... // Agency

s_ac.branch() {
Protocol Implementation (3)

```java
...?

  ACCEPT:  
    ...   // 1'

  REJECT:  
    ...   // 2'

}
...  

// Agency

...  

s_ac.branch() { 
  case ACCEPT: { 
    ...   // 1'
  } 
  case REJECT: { 
    ...   // 2'
  }
}
```
Putting Customer Together

```java
protocol p_ca { ... }
```
protocol p_ca { ... }

STSocket s_ca = STSocketImpl.create(p_ca);
Putting Customer Together

```java
protocol p_ca { ... }
STSocket s_ca = STSocketImpl.create(p_ca);

    s_ca.request(Agency, port);
    s_ca.outwhile(...) {
        ...
    }
    ...
```
Putting Customer Together

```java
protocol p_ca { ... }

STSocket s_ca = STSocketImpl.create(p_ca);

try {
    s_ca.request(Agency, port);
    s_ca.outwhile(...) {
      ...
      ...
    }
    ...
}
```
Putting Customer Together

```java
protocol p_ca { ... }
STSSocket s_ca = STSocketImpl.create(p_ca);
try {
    s_ca.request(Agency, port);
    s_ca.outwhile(...) {
        ...
    }
    ...
}

catch(IOException ioe) { ... }
```
Worked Example: Session Programming

```java
protocol p_ca { ... }
STSocket s_ca = STSocketImpl.create(p_ca);
try {
    s_ca.request(Agency, port);
    s_ca.outwhile(...) {
        ...
    }
    ...
} catch(STIncompatibleSessionException ise) { ... } catch(InterruptedException ise) { ... }
```
Putting Customer Together

```java
protocol p_ca { ... }

STSocket s_ca = STSocketImpl.create(p_ca);

try {
    s_ca.request(Agency, port);
    s_ca.outwhile(...) {
        ...
    }
    ...
}

catch(STIncompatibleSessionException ise) { ... }
catch(IOException ioe) { ... }
finally {
    s_ca.close();
}
```
Overview (3)

1. Background and Related Work.
2. Worked Example: Language Design.
3. Runtime Support for Session Communication.
4. Conclusion and Future Work.
Sessions over TCP

➢ Map session operations to `Socket` communications.
  – Preserve asynchrony.
  – Interesting case is delegation.
Sessions over TCP

- Map session operations to \textit{Socket} communications.
  - Preserve asynchrony.
  - Interesting case is delegation.

- Possible designs:
  - Proxy / Integrated runtime layer.
  - “Lost message” forwarding (cf. Mobile IP).
  - “Lost message” resending.
Session Initiation

```
s_ca.request(..., ...);  # s_ca
s_ac = ss_ac.accept();   # s_ac

s_ca = new Socket(..., ...);
C S_ca A
s_ac = ss_ca.accept();
```
Session Initiation

1. **C → A**: Type of intended session.
Session Initiation

1. **C → A**: Type of intended session.
2. **A → C**: “YES” if types compatible, else “NO”.

```
s_ca.request(..., ...);
s_ac = ss_ac.accept();
```

```
s_ca = new Socket(..., ...);
C  S_ca  A
  |    |
  |    |
  s_ac = ss_ca.accept();
```
Session Communication

```java
s_ca.send(travDetails);
travDetails: String
s_ac.receive();
os.writeObject(...);
Sca
C
A
ois.receiveObject();
```
Session Delegation (1)

```java
s_ca.send(delivAddr);

s_as.send(s_ac);  s_sc = s_sa.receive();
```
1. **S:** Create server socket on `port`. 
Session Delegation (2)

1. **S**: Create server socket on `port`.
2. **S → A**: `port`
Session Delegation (2)

1. **S**: Create server socket on \( \text{port} \).
2. **S → A**: \( \text{port} \)
3. **A → C**: \( \text{IP}_S, \text{port}, \text{ST}_A(s_{ca}) \)
Session Delegation (3)

4. C: Connect to IP_S: port.
4. **C**: Connect to IP$_S$: *port*.

5. **C → S**: “Lost messages” according to $\text{ST}_A(s_{ca}) - \text{ST}_C(s_{ca})$. 
Overview (4)

1. Background and Related Work.
2. Worked Example: Language Design.
3. Runtime Support for Session Communication.
4. Conclusion and Future Work.
Conclusion

http://www/~rh105/sessiondj.html
Conclusion

- http://www/~rh105/sessiondj.html
- Realised many session type principles in a concrete, practical object-oriented language.
Conclusion

- Realised many session type principles in a concrete, practical object-oriented language.
- Asynchrony and distribution.

http://www/~rh105/sessiondj.html
Conclusion

- http://www/~rh105/sessiondj.html
- Realised many session type principles in a concrete, practical object-oriented language.
- Asynchrony and distribution.
- Session interleaving, exceptions/failure.
Conclusion

- http://www/~rh105/sessiondj.html
  - Realised many session type principles in a concrete, practical object-oriented language.
  - Asynchrony and distribution.
  - Session interleaving, exceptions/failure.
  - Protocols for session initiation, delegation and close.
Conclusion

- [http://www/~rh105/sessiondj.html](http://www/~rh105/sessiondj.html)
  - Realised many session type principles in a concrete, practical object-oriented language.
  - Asynchrony and distribution.
  - Session interleaving, exceptions/failure.
  - Protocols for session initiation, delegation and close.
  - Delegation protocol uses session type information.
Future Work

- Usability improvements (syntax, features, ...).
- Communication optimisations.
Future Work

- Usability improvements (syntax, features, ...).
- Communication optimisations.
- Language evaluation (large example applications).
- Performance evaluation.
Future Work

- Usability improvements (syntax, features, ...).
- Communication optimisations.
- Language evaluation (large example applications).
- Performance evaluation.
- Associating session types to addresses.
Future Work

- Usability improvements (syntax, features, ...).
- Communication optimisations.
- Language evaluation (large example applications).
- Performance evaluation.
- Associating session types to addresses.
- Class downloading and verification.
Future Work

- Usability improvements (syntax, features, ...).
- Communication optimisations.
- Language evaluation (large example applications).
- Performance evaluation.
- Associating session types to addresses.
- Class downloading and verification.
- Session exceptions.
Future Work

- Usability improvements (syntax, features, ...).
- Communication optimisations.
- Language evaluation (large example applications).
- Performance evaluation.
- Associating session types to addresses.
- Class downloading and verification.
- Session exceptions.
- Alternative runtime designs.
- Sessions over alternative transports.
Session Exceptions (Poss. Design)

```java
protocol p { begin.....end | X:.....end | ... }
STChannel c = new STChannel(..., p);
STSocket s = STSocketImpl.create(c);
try {
  try {
    s.request();
    ...
  }
  catch(STIncompatibleSessionException ise) { }
  catch(s:X) { ... }
  catch(...) { ... }
}

catch(IOException ioe) { ... }
finally { s.close(); }
```
Preliminary Results (1)

• For simple benchmark experiment, implemented:

\[
\text{\texttt{begin . ![ ?( MyObject ) ]* . end}}
\]
Preliminary Results (1)

- For simple benchmark experiment, implemented:

  ```java
  begin . ![ ?( MyObject ) ]* . end
  ```

- Using:
  - STSocket and equivalent “Plain Socket”
  - STNIOSocket and equivalent “Plain SocketChannel”
  - RMI: `MyObject remotemeth(Boolean bool)`
Preliminary Results (2)

(1) Runtime Overhead: Boolean (47 bytes), MyObject (100 bytes)
Preliminary Results (3)

(2b) Message Exchange: Boolean (47 bytes), MyObject (10 kilobytes)

- STSocket
- STNIO
- RMI