Type-safe Communication in Java with Session Types

Raymond Hu[1], Nobuko Yoshida[1] and Kohei Honda[2]

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:
  - Takeuchi et al. *An Interaction-based Language and its Typing System*. (1994)
  - Honda et al. *Language Primitives and Type Discipline for Structured Comm*… (1998)
  - Gay and Hole. *Subtyping for Session Types in the Pi Calculus*. (1999)
  - Hole and Gay. *Bounded Polymorphism in Session Types.* (2003)

Session Types

- Type systems for process calculi:
  - Takeuchi et al. *An Interaction-based Language and its Typing System*. (1994)
  - Honda et al. *Language Primitives and Type Discipline for Structured Comm*… (1998)
  - Gay and Hole. *Subtyping for Session Types in the Pi Calculus*. (1999)
  - Hole and Gay. *Bounded Polymorphism in Session Types.* (2003)
- Session types for object calculi:
  - Dezani-Ciancaglini et al. *A distributed Object-oriented Language with Session...* (2005)
  - Dezani-Ciancaglini et al. Session Types for Object-oriented Languages. (2006)
  - Dezani-Ciancaglini et al. *Bounded Session Types for Object-oriented Languages.* (2007)
  - Coppo et al. *Asynchronous Session Types and Progress for Object-oriented...* (2007)

Session Types

- Type systems for process calculi:
  - Takeuchi et al. *An Interaction-based Language and its Typing System*. (1994)
  - Honda et al. *Language Primitives and Type Discipline for Structured Comm*… (1998)
  - Gay and Hole. *Subtyping for Session Types in the Pi Calculus*. (1999)
  - Hole and Gay. *Bounded Polymorphism in Session Types.* (2003)
- Session types for object calculi:
  - Dezani-Ciancaglini et al. *A distributed Object-oriented Language with Session...* (2005)
  - Dezani-Ciancaglini et al. Session Types for Object-oriented Languages. (2006)
  - Dezani-Ciancaglini et al. *Bounded Session Types for Object-oriented Languages.* (2007)
  - 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

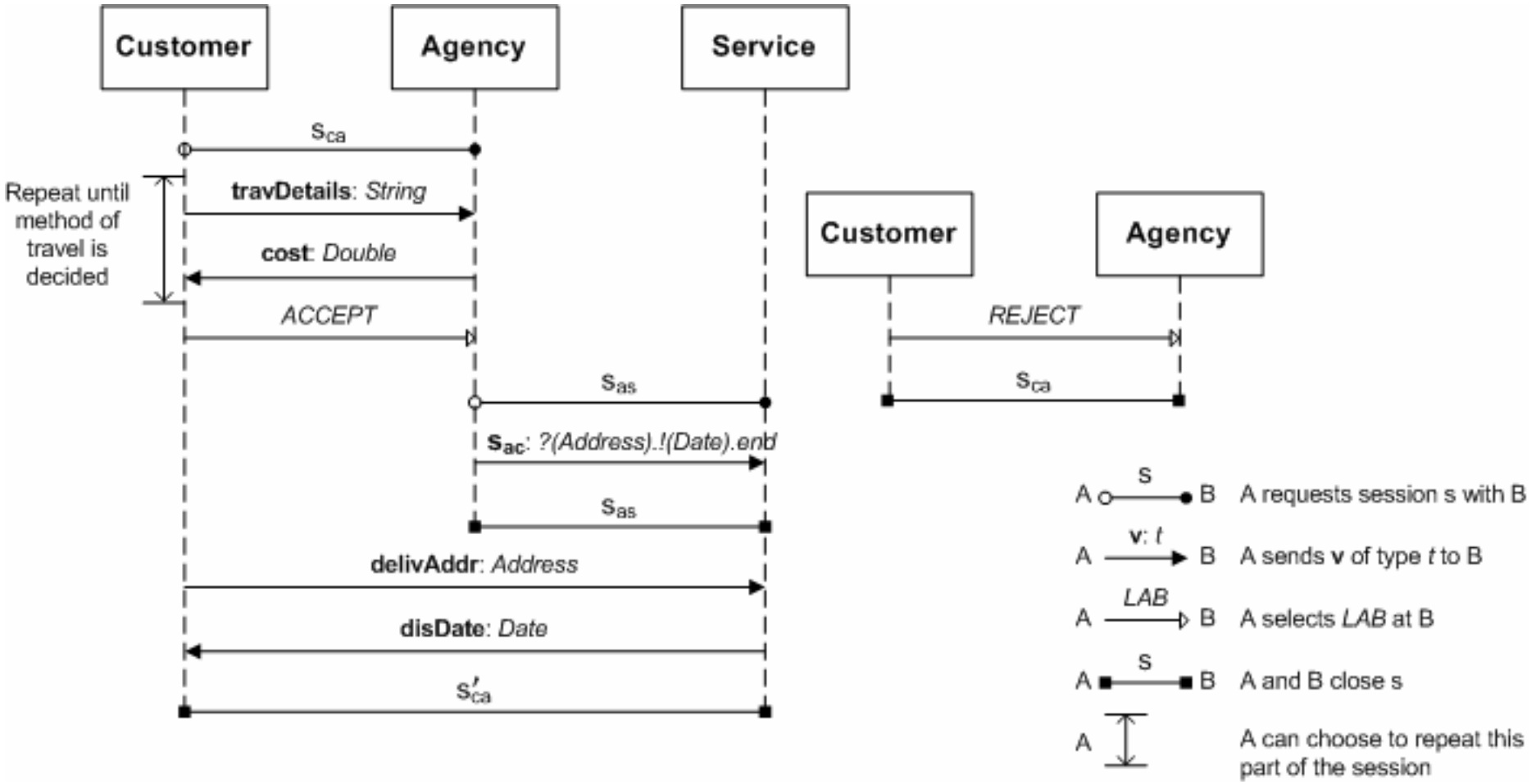
- Singularity OS [FAHHHLL06]:
  - Restricted features, not distributed.
- Multithreaded functional language design [VGR06]:
  - Not distributed, no actual implementation.
- Encodings of sess. types (e.g. Haskell [NT04], C++):
  - Proof of concept, but not full implementations.

➢ Why use untyped sockets if we can use sessions?

Overview (2)

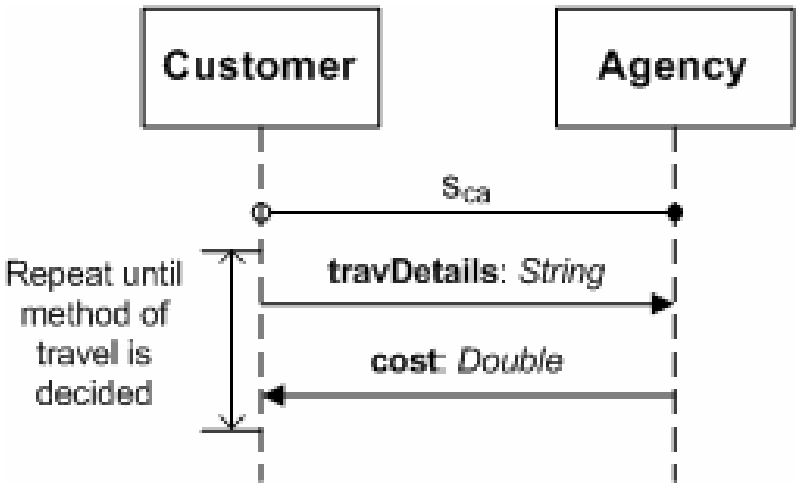
1. Background and Related Work.
2. Worked Example: Language Design.
3. Runtime Support for Session Communication.
4. Conclusion and Future Work.

An Online Ticket Ordering System



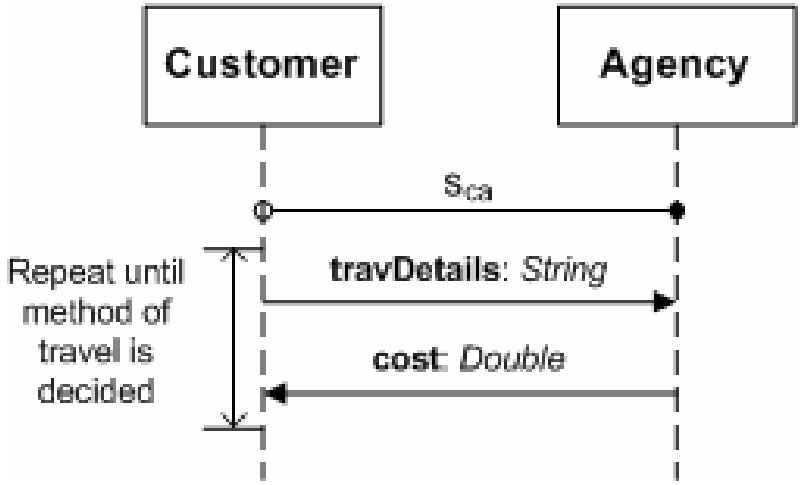
Protocol Specification (1)

```
begin
```



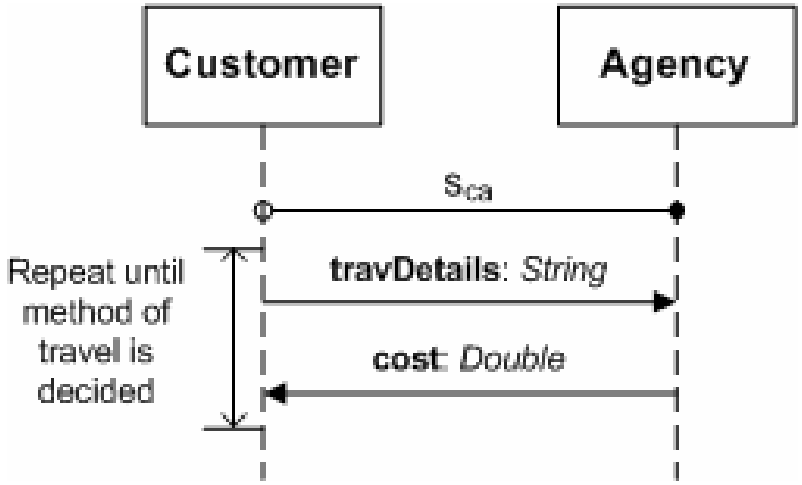
Protocol Specification (1)

```
begin.

  !(String)
```



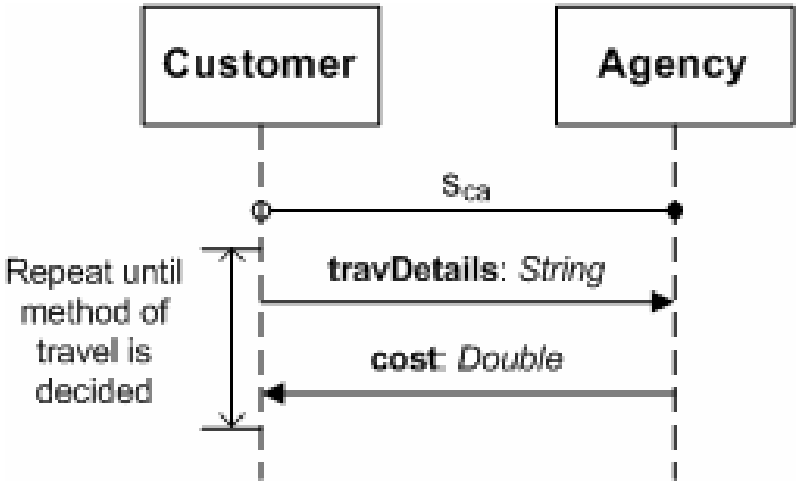
Protocol Specification (1)

```
begin.

  !(String).
  ?(Double)
```



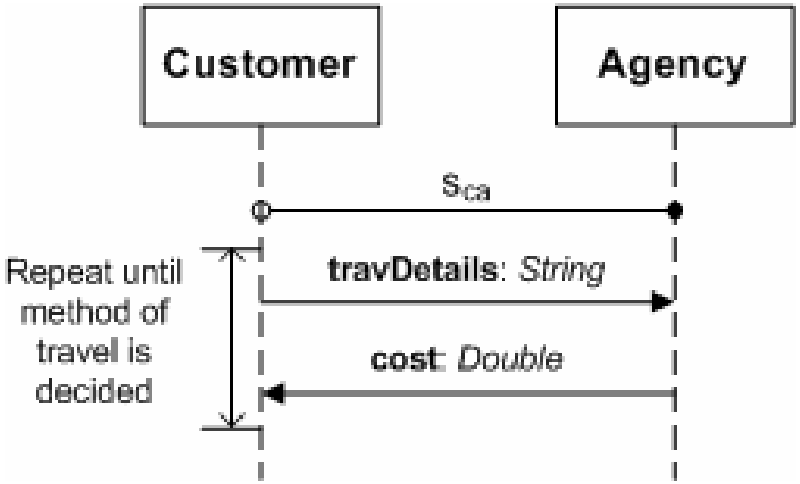
Protocol Specification (1)

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



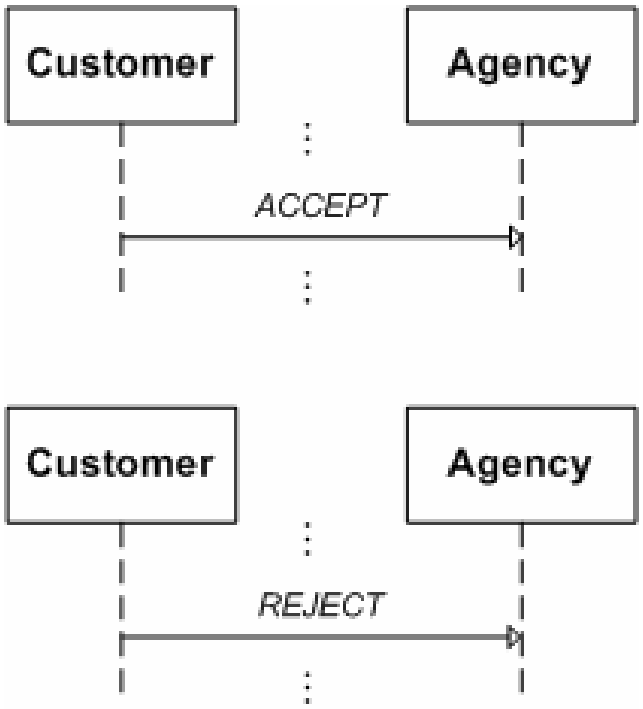
Protocol Specification (1)

```
begin.
![
  !(String).
  ?(Double)
]*.
…
```



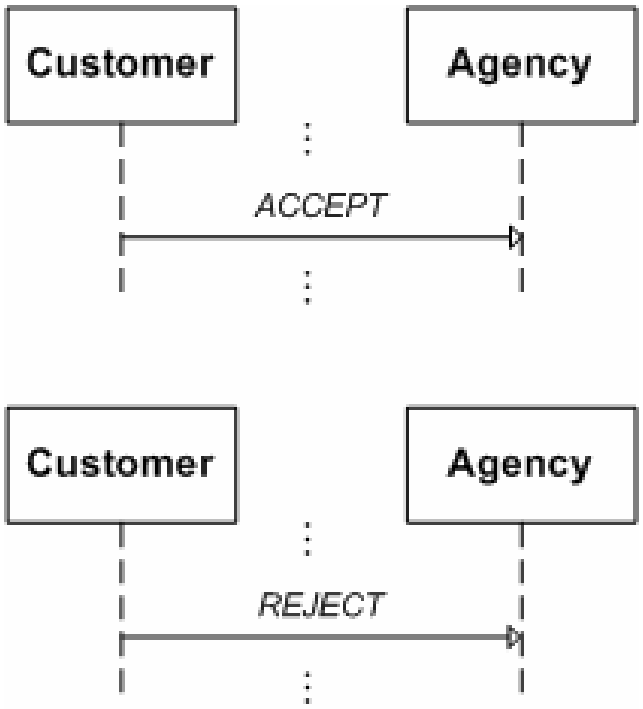

```
begin.
?[
  ?(String).
  !(Double)
]*.
…
```

Protocol Specification (2)




```
...
?{

}.
...
```

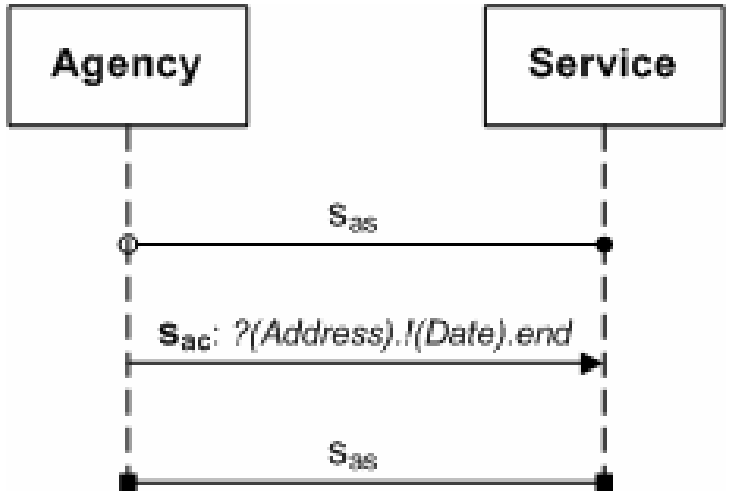
Protocol Specification (2)




```
…
?{
   ACCEPT:
         … ,

   REJECT:
         …
}.
…
```

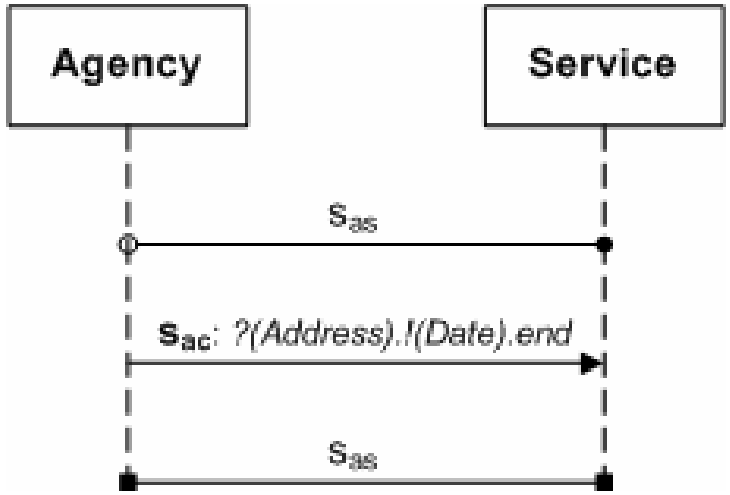
Protocol Specification (3)

```
protocol p_as {



}
```



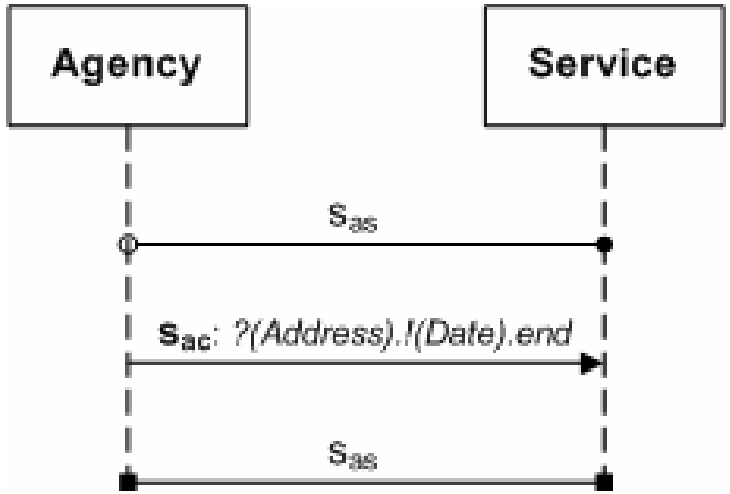
Protocol Specification (3)

```
protocol p_as {
  begin.
  !(


  ).
  end
}
```



Protocol Specification (3)

```
protocol p_as {
  begin.
  !(
    ?(Address).!(Date).end
  ).
  end
}
```



Protocol Implementation (1)

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

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

Protocol Implementation (1)

```
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).
     ?(Double)
   ]*.
   …
```

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

s_ca.request(Agency, port);
s_ca.outwhile(!decided) {




}
```

Protocol Implementation (1)

```
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)

```
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);
  cost =
     (Double)s_ca.receive();
  decided = …;
}
```

Protocol Implementation (1)

```
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);
  cost = s_ca.receive();
  decided = …;
}
```

Protocol Implementation (2)

```
…
!{
  ACCEPT:
      … , // 1

  REJECT:
      … // 2
}
…
```

```
// Customer

  s_ca.select(ACCEPT) {
      … // 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)

```
…
?{
  ACCEPT:
      … , // 1'

  REJECT:
      … // 2'
}
…
```

```
// Agency

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

Putting `Customer` Together

```
protocol p_ca { … }
```

Putting `Customer` Together

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

Putting `Customer` Together

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

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

Putting `Customer` Together

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

Putting `Customer` Together

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

catch(IOException ioe) { … }
```

Putting `Customer` Together

```
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) { … }
```

Putting `Customer` Together

```
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

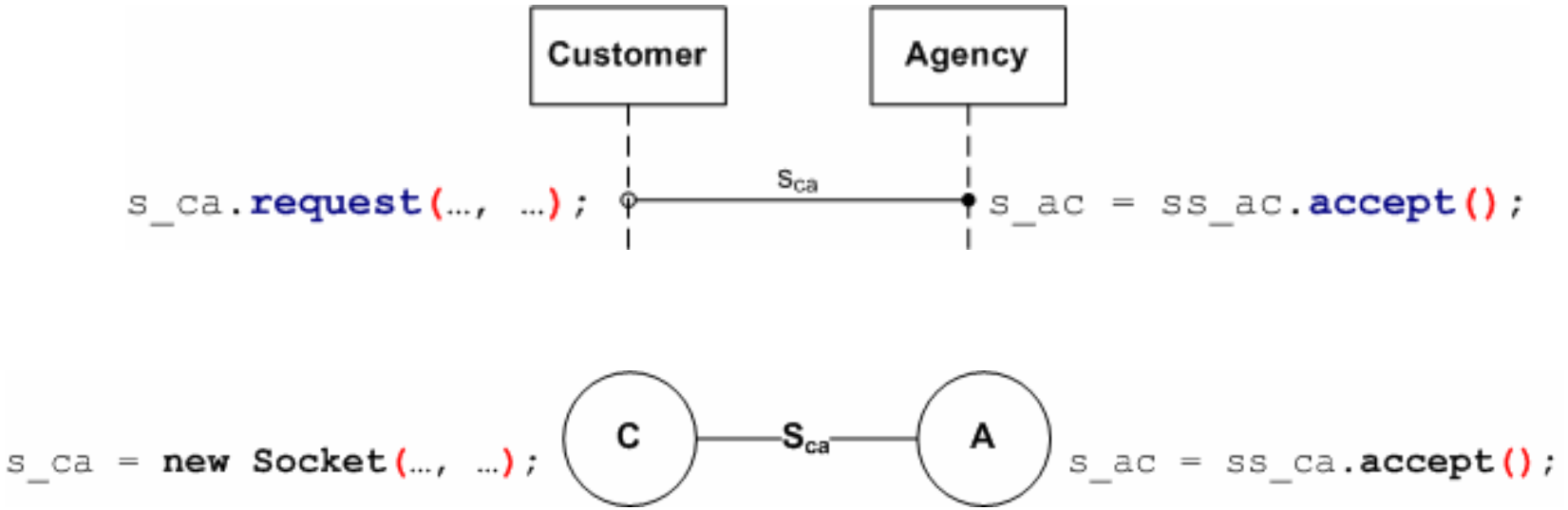
- Map session operations to `Socket` communications.
  - Preserve asynchrony.
  - Interesting case is delegation.

Sessions over TCP

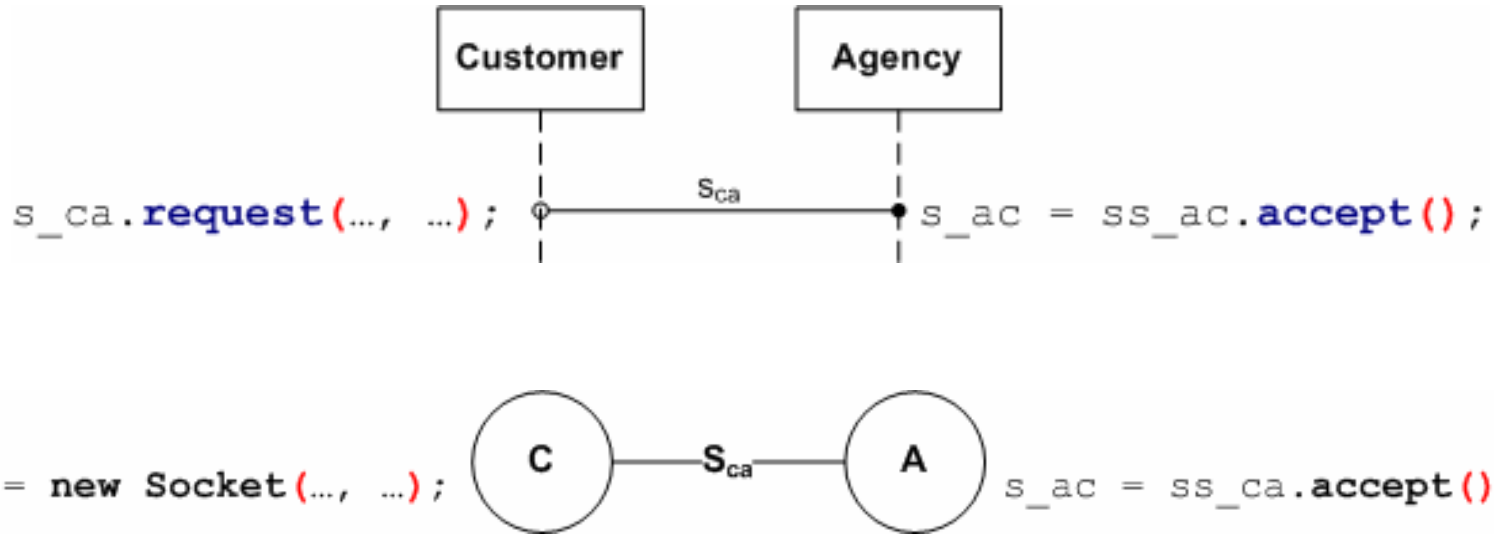
- Map session operations to `Socket` communications.
  - Preserve asynchrony.
  - Interesting case is delegation.

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

Session Initiation

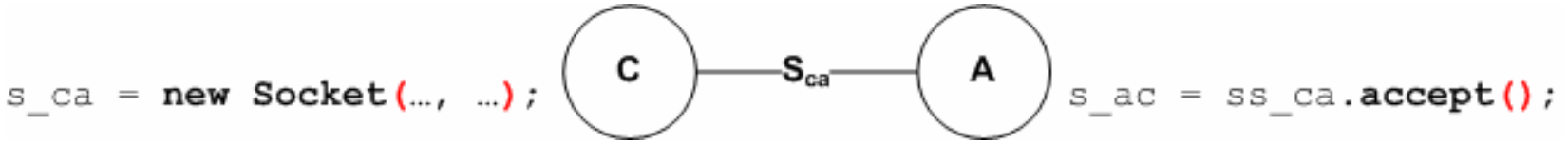


Session Initiation



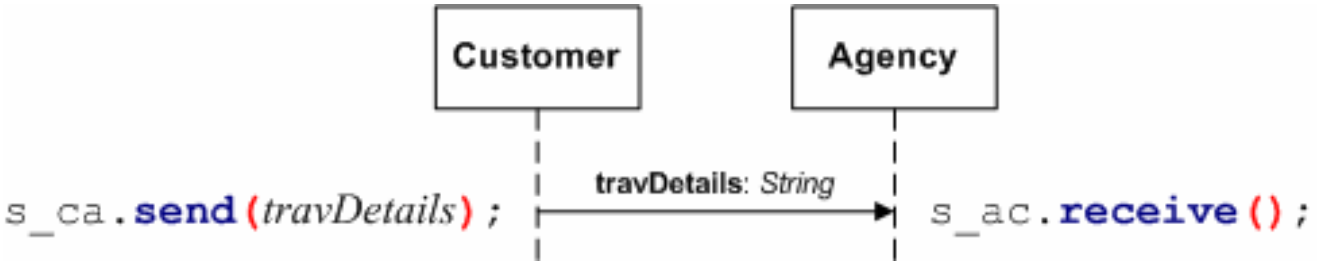
1. **C** → **A** : Type of intended session.

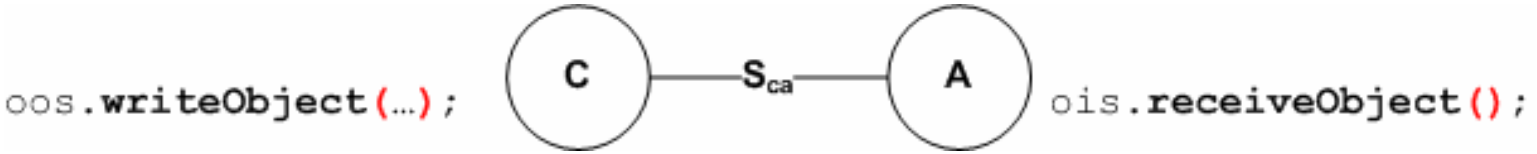
Session Initiation



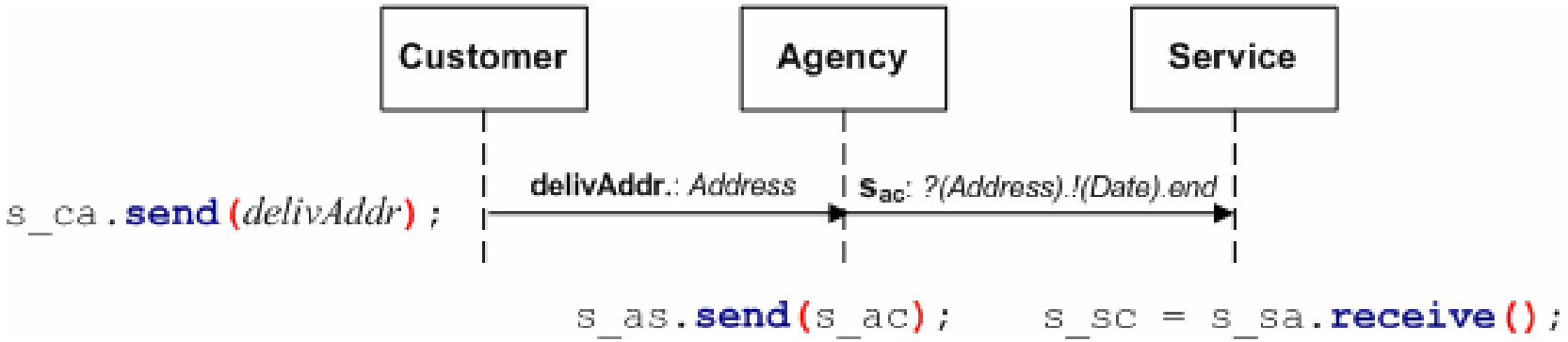
1. **C** → **A** : Type of intended session.
2. **A** → **C**: "YES" if types compatible, else "NO".

Session Communication

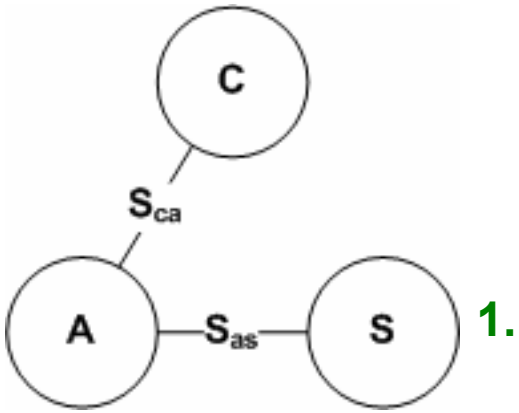




Session Delegation (1)

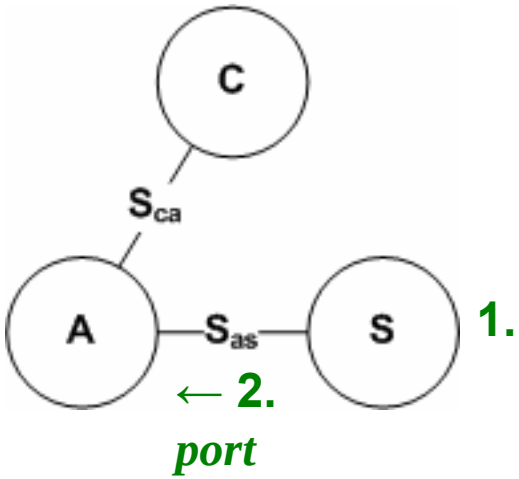


Session Delegation (2)



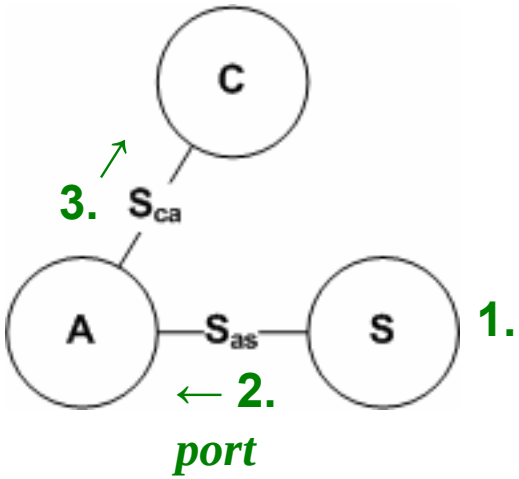
1. **S**: Create server socket on *port*.

Session Delegation (2)



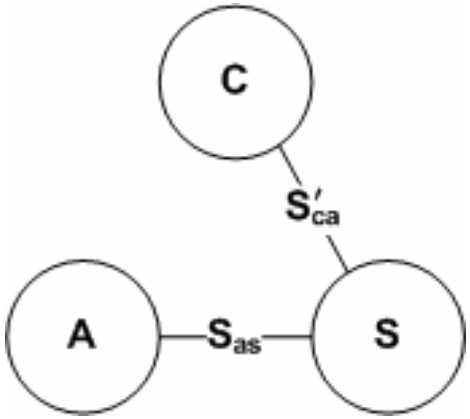
1. **S**: Create server socket on *port*.
2. **S** → **A**: *port*

Session Delegation (2)



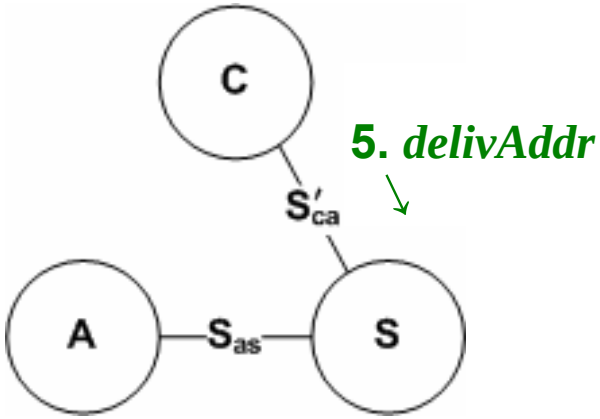
1.	**S**:	Create server socket on *port*.
2.	**S** → **A**:	*port*
3.	**A** → **C**:	IP_S, *port*, $\mathbf{ST_A(s_{ca})}$

Session Delegation (3)



4. **C**: Connect to IP_S: *port*.

Session Delegation (3)



4. **C**: Connect to IP_S: *port*.
5. **C** → **S**: "Lost messages" according to $ST_A(s_{ca})$ - $ST_C(s_{ca})$.

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

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

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
- 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)

```
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:

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

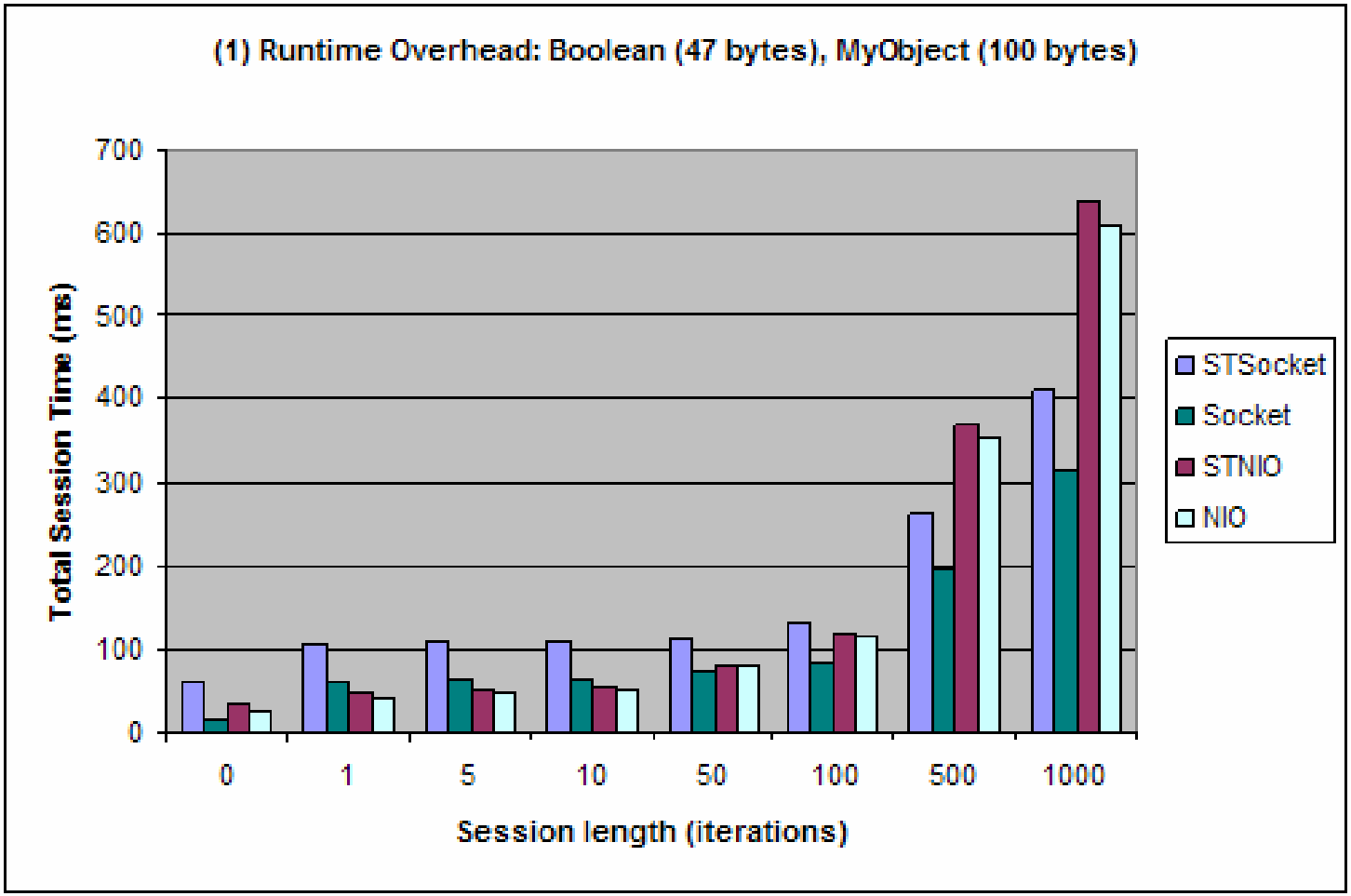
Preliminary Results (1)

- For simple benchmark experiment, implemented:

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

- Using:
  - STSocket and equivalent “Plain Socket”
  - STNIOSocket and equivalent “Plain SocketChannel”
  - RMI: `MyObject remoteMeth(Boolean bool)`

Preliminary Results (2)



Preliminary Results (3)