A concurrent programming language with refined session types

Vasco T. Vasconcelos

LaSIGE, University of Lisbon, Portugal

ABCD Project Meeting
January 10, 2014

Joint work with Juliana Franco and Dimitris Mostrous

Motivation

- Session types are by now a solid foundation for developing typed, message-passing concurrent applications
- Session types were originally proposed for the pi-calculus, then incorporated in functional and OO languages
- Currently, there is no implementation on which one may
  - exercise examples
  - test program idioms
  - experiment with type systems

SePi is **SE**ssions on **PI**

SePi is

- An exercise in the design and implementation of a concurrent programming language based on the monadic pi calculus, where process interaction is governed by linearly refined session types
- A basis for exploring the practical applicability of new (and old) works on session-based type systems
- A tool where new program idioms and type developments may be tested and eventually incorporated

SePi is not

- A language for developing industrial strength applications

SePi _ Processes

1. Monadic finite pi calculus:
   channel creation, input, output, parallel composition, conditional
2. Choice
3. Replication
4. Assume, assert

SePi _ Types

1. Input, output, **end**
2. Choice
3. Recursive types
4. Refinement types

SePi _ Formulae

1. Predicates over program values
2. Tensor
3. Unit

Essentially a multiset of predicates

SePi _ Derived constructs

1. Polyadic message passing
2. Process definitions
3. Session initiation
4. **dualof** type operator
5. Type abbreviations

Running example _ An online donation service

Four sorts of participants: bank, server, clients and benefactors

- **Clients** create donation campaigns and send the campaign link to benefactors
- **Benefactors** donate by providing a credit card number and an amount to be charged
- The **server** provides for the creation of campaigns and forwards the donations to the bank
- The **bank** charges the donations on credit cards

V1 _ Channel creation, input, output, parallel composition

```
new c s: !integer.end
c!2013 |
s?x. printIntegerLn!x
```

V2 _ Choice

```
new c s: +{setDate: !integer.end, commit: end}
c select setDate. c!2013 |
case s of
     setDate → s?x. printIntegerLn!x
     commit  → printStringLn!"done!"
```

V3 _ Recursive types and process definitions

```
type Donation = +{setDate: !integer.Donation, commit:
    end}

new c s: Donation
c select setDate. c!2013.
c select setDate. c!2014. c select commit |

def server s: dualof Donation =
    case s of
        setDate → s?x. printIntegerLn!x. server!s
        commit  → printStringLn!"done!"
server!s
```

V4 _ Linear channels that become unrestricted

```
type Donation = +lin{setDate: lin!integer.Donation,
    commit: Promotion}
type Promotion = un!(CreditCard, integer).Promotion
type CreditCard = string
new c s: Donation
c select setDate. c!2013.
c select setDate. c!2014. c select commit. {
    c!("1234", 500)  | c!("2434", 1000)
} |
def server s: dualof Donation = case s of
    setDate → s?x. printIntegerLn!x. server!s
    commit  → acceptDonation!s
def acceptDonation s: dualof Promotion =
    s?(card, amount).
    printStringLn!"Received " ++ amount ++ "euros on
        card " ++ card.
    acceptDonation!s
server!s
```

V5_ Multiple clients (I/II)

```
type Donation = +{setDate: !integer.Donation, commit:
    Promotion}
type Promotion = *!(CreditCard, integer)
type CreditCard = string

new client server: *?Donation

client?c.
c select setDate. c!2013. c select setDate. c!2014. c
    select commit. {
    c!("1234", 500)  | c!("2434", 1000)
} |

client?c.
c select setDate. c!2014. c select commit. {
    c!("9876", 5000)  | c!("8796", 10)
} |
```

V5_ Multiple clients (II/II)

```
def donationServer server: *!Donation =
    def setup s: dualof Donation =
        case s of
            setDate → s?x. setup!s
            commit  → acceptDonation!s

    def acceptDonation s: dualof Promotion =
        s?(card, amount).
        printStringLn!"Charging " ++ amount ++ " on
            card " ++ card.
        acceptDonation!s

    server!(new s: dualof Donation). // session
        initiation
    setup!s.
    donationServer!server

donationServer!server
```

V6_ The bank comes into play

```
def bank (ccard: CreditCard, amount: integer) =
    printStringLn!"Charging " ++ amount ++ " euros on
       card " ++ ccard

def donationServer server: *!Donation =
    def setup s: dualof Donation =
        case s of
            setDate → s?x. setup!s
            commit  → acceptDonation!s
    def acceptDonation s: dualof Promotion =
            s?(card, amount).
            bank!(card, amount + 10).
            acceptDonation!s

    server!(new s: dualof Donation). // s. initiation
    setup!s.
    donationServer!server

donationServer!server
```

V7_ Clients assume; server forwards; bank asserts (I/II)

```
type Donation = +{setDate: !integer.Donation, commit:
    Promotion}
type Promotion = *!(c:CreditCard, {x:integer|charge(c,x
    )})
type CreditCard = string

new client server: *?Donation

client?c.
c select setDate. c!2013. c select setDate. c!2014. c
    select commit. {
    assume charge("1234", 500) | c!("1234", 500) |
    assume charge("2434", 1000) | c!("2434", 1000)
} |
```

V7_ Clients assume; server forwards; bank asserts (II/II)

```
def bank (card: CreditCard, amount: {x:integer | charge
  (card,x)}) =
  assert charge(card,amount).
  printStringLn!"Charging " ++ amount ++ " euros on
    card " ++ card

def donationServer server: *!Donation =
  def setup s: dualof Donation =
    case s of
      setDate → s?x. setup!s
      commit  → acceptDonation!s
  def acceptDonation s: dualof Promotion =
      s?(card, amount).
      bank!(card, amount).
      acceptDonation!s

  server!(new s: dualof Donation).
  setup!s.
  donationServer!server
```

V8_ Fraudulent servers do not compile (I/II)

```
def donationServer server: *!Donation =
    def setup s: dualof Donation =
        case s of
            setDate → s?x. setup!s
            commit  → acceptDonation!s
    def acceptDonation s: dualof Promotion =
            s?(card, amount).
            bank!(card, amount).
            bank!(card, amount). // charge twice
// Type mismatch: expecting: {x:integer|charge(card,x)
    }; found: integer.
            acceptDonation!s

    server!(new s: dualof Donation).
    setup!s.
    donationServer!server
```

V8_ Fraudulent servers do not compile (II/II)

```
def donationServer server: *!Donation =
    def setup s: dualof Donation =
        case s of
            setDate → s?x. setup!s
            commit  → acceptDonation!s
    def acceptDonation s: dualof Promotion =
            s?(card, amount).
            bank!(card, amount+10). // charge tax
// Type mismatch: expecting: {x:integer|charge(card,x)
    }; found: integer.
            acceptDonation!s

    server!(new s: dualof Donation).
    setup!s.
    donationServer!server
```

Summing up

- SePi is a new concurrent programming language where
  - communication between processes is governed by session types,
  - refinement types allow the specification of properties about the values exchanged.
- SePi is based on the monadic pi-calculus; includes a few abbreviations and derived constructs, such as
  - the **dualof** operator,
  - input/output of multiples values,
  - session initiation
  - mutually recursive process definitions, channel creations, and type declarations.
- An Eclipse plugin for SePi facilitates code development. Try it at `http://gloss.di.fc.ul.pt/sepi`

Future work

- Predicates on expressions, using a SMT solver
- Persistent (exponential) formulae → affine logic
- What about your future work on top of SePi?
    - Subtyping
    - Type systems for progress
    - Polymorphism
    - ...