Multiparty Session Types for Safe Runtime Adaptation in an Actor Language

Paul Harvey¹  Simon Fowler²  Ornella Dardha³  Simon Gay³

¹Rakuten Institute of Technology, Japan
²School of Informatics, University of Edinburgh, UK
³School of Computing Science, University of Glasgow, UK

ABCD Meeting, December 2018
Overview

- **Adaptive** software is increasingly important for pervasive computing.

- Adaptation includes *discovering, replacing and communicating with* software components that are not part of the original system.

- **Ensemble** [Harvey 2015] is an actor-based language with support for adaptation.
We designed and implemented EnsembleS by adding session types to Ensemble.

Static type checking guarantees safe runtime adaptation.

We extended the StMungo tool to generate skeleton EnsembleS code from Scribble local types.
EnsembleS language features

- Imperative actor-based language.
- Channels instead of mailboxes.
- Support for adaptation.
A simple EnsembleS program

```plaintext
type Isnd is
  interface(out integer output)

type Ircv is
  interface(in integer input)

stage home{
  actor sender presents Isnd {
    value = 1;
    constructor() {}
    behaviour {
      send value on output;
      value := value + 1;
    }
  }
  actor receiver presents Ircv {
    constructor() {}
    behaviour {
      receive data from input;
      printString("received: ");
      printInt(data);
    }
  }
  boot{
    s = new sender();
    r = new receiver();
    establish topology(s, r);
  }
}
```

Paul Harvey, Simon Fowler, Ornela Dardha, Simon Gay

Multiparty Session Types for Safe Runtime Adaptation in an Actor Language
Session types in EnsembleS

- As well as presenting an interface, an actor can follow a session type.

- The session type is a Scribble local type.

- Typechecking checks the sequence of messages to and from other actors, and connect/disconnect actions.

- Individual messages are sent on standard Ensemble channels.
Buyer/Seller protocol in EnsembleS (1): global type

```
1 global protocol Bookstore (role Sell, role Buy1, role Buy2) {
2   book(string) from Buy1 to Sell;
3   book(int) from Sell to Buy1;
4   quote(int) from Buy1 to Buy2;
5   choice at Buy2 {
6     agree(string) from Buy2 to Buy1, Sell;
7     transfer(int) from Buy1 to Sell;
8     transfer(int) from Buy2 to Sell;
9   } or {
10     quit(string)
11     from Buy2 to Buy1, Sell;
12   }
13 }
14 }
```
Local types are generated by *projection*, as usual.

```plaintext
local protocol Buy1 (role Sell, self Buy1, role Buy2)
{
  book(string) to Sell;
  book(int) from Sell;
  quote(int) to Buy2;
  choice at Buy2 {
    agree(string) from Buy2;
    transfer(int) to Sell;
  } or {
    quit(string) from Buy2;
  }
}
```
The interface is generated from the local type: channels for each role and message type.

```text
1 type Buy1_interface is interface(
2   out {Seller, string} toSell_string,
3   in  {Seller, integer} fromSell_integer,
4   out {Buy2, integer} toBuy2_integer,
5   in  {Buy2, Choice0} fromBuy2_agreequit,
6   in  {Buy2, string} fromBuy2_string,
7   out {Sell, integer} toSell_integer,
8 )
```
Buyer/Seller protocol in EnsembleS (4): actor definition

- Skeleton actor definitions are generated from the local types.
- Actors are also typechecked.

```plaintext
stage home {
actor Buy1A presents Buy1_interface follows Buy1_session {
  constructor() {}
  behaviour {
    payload1 = "";
    send payload1 on toSell_string;
    receive payload2 from fromSell_integer;
    payload3 = 42;
    send payload3 on toBuy2_integer;
    //Choice from other actor
    receive payload4 from fromBuy2_agreequit;
  }
}
switch(payload4) {
  case Choice0_agree:
    receive payload5 from fromBuy2_string;
    payload6 = 42;
    send payload6 on toSell_integer;
    break;
  case Choice0_quit:
    receive payload7 from fromBuy2_string;
    break;
}
```

Paul Harvey, Simon Fowler, Ornela Dardha, Simon Gay
Multiparty Session Types for Safe Runtime Adaptation in an Actor Language
Adaptation in Ensemble

- **Discover**: locate an actor with a given interface and satisfying a given query.
- **Install**: spawn a new actor instance at a specified stage.
- **Migrate**: move an executing actor to a different stage.
- **Replace**: replace an executing actor with a new actor instance with the same interface.
- **Interact**: connect to another actor and communicate with it.
Adaptation in EnsembleS, with session types

- **Discover**: locate an actor with a given interface and satisfying a given query and a given session type.

- **Replace**: replace an executing actor with a new actor instance with the same interface and the same session type.

- **Interact**: connect to another actor and communicate with it, following its session type.
EnsembleS discovery / replacement with session types (1)

1 actor fastA presents accountingI follows accountingSession
2 {
3   constructor() {}
4   behaviour {
5     receive data on input;
6     quicksort(data);
7     send data on output;
8   }
9 }

Paul Harvey, Simon Fowler, Ornela Dardha, Simon Gay
Multiparty Session Types for Safe Runtime Adaptation in an Actor Language
actor slowA presents accountingI
follows accountingSession {
pS = new property[2] of property("",0);
constructor() {
pS[0] := new property("serial",823);
pS[1] := new property("version",2);
publish pS;
}
behaviour {
receive data on input;
bubblesort(data);
send data on output;
}
query alpha() {
$serial==823 && $version<4;
}
actor master presents masterI{
  constructor() { }
  behaviour {
    // find the slow actors with the query
    actor_s = findSessionActors(
      accountingI,
      accountingSession,
      alpha());
    // replace them with efficient versions
    if(actor_s[0].length > 1){
      replace actor_s[0]
      with fastA();
    }
  }
}
Formalisation

- We have formalised a core calculus for EnsembleS.

- Operational semantics, type system, type preservation.

- A well-typed configuration proceeds until all of its actors have terminated, except for runtime situations such as attempting to use a disconnected channel, or absence of an actor matching a `discover` query.
Session types for an existing, implemented (albeit experimental) actor language.

Session types for adaptive features: discovery, dynamic connection, replacement.

The formalisation matches the implementation; there are further possibilities for typechecking.