# *Session types* modulaires pour la programmation orientée-objet distribuée

Nils Gesbert

Université de Glasgow

Travail réalisé en collaboration avec : Simon Gay (Université de Glasgow),
António Ravara (Nouvelle université de Lisbonne),
Vasco Vasconcelos et Alexandre Caldeira (Université de Lisbonne).

Séminaire IRISA 68NQRT, 8 avril 2010

# Plan

# Exemple : interface d'accès à une messagerie

```
enum ErrorStatus { OK, ERR }

interface MailReader {
  ErrorStatus login (String user, String pass);
  int getNumberOfMessages();
  ErrorStatus fetchAndDelete (int index);
  String getMessageContent();
  void logout();
}
```

# Contraintes d'utilisation

- Impossible de télécharger des messages hors connexion
- Impossible de lire un message avant de l'avoir téléchargé
- Télécharger un nouveau message écrase le précédent : à éviter s'il n'a pas été lu ou sauvegardé

# Contraintes d'utilisation

- Impossible de télécharger des messages hors connexion
- Impossible de lire un message avant de l'avoir téléchargé
- Télécharger un nouveau message écrase le précédent : à éviter s'il n'a pas été lu ou sauvegardé

Toutes les méthodes ne sont pas disponibles à tout moment : l'interface (ensemble des méthodes disponibles) se modifie en fonction des actions effectuées.
Nous proposons de représenter ce protocole d'utilisation par un type.

# Des *session types* pour objets

- Plusieurs méthodes disponibles : choix externe
  `{fetchAndDelete: S, logout: S'}`

  Branchement côté objet / Sélection par le client via appel d'une méthode

# Des *session types* pour objets

- Plusieurs méthodes disponibles : choix externe
  `{fetchAndDelete: S, logout: S'}`

  Branchement côté objet / Sélection par le client via appel d'une méthode

- État indiqué par le résultat d'une méthode : choix interne
  `<OK: S, ERR: S'>`

  Branchement côté client / Sélection par l'objet via retour d'une étiquette

# Des *session types* pour objets

- Plusieurs méthodes disponibles : choix externe
  `{fetchAndDelete: S, logout: S'}`

  Branchement côté objet / Sélection par le client via appel d'une méthode

- État indiqué par le résultat d'une méthode : choix interne
  `<OK: S, ERR: S'>`

  Branchement côté client / Sélection par l'objet via retour d'une étiquette

Session `Init = {login: <OK: NoMsg, ERR: Init>}`

where `NoMsg = {fetchAndDelete: <OK: MsgRead, ERR: NoMsg>, getNumberOfMessages: NoMsg, logout: {} }`

and `MsgRead = {getMessageContent: NoMsg, getNumberOfMessages: MsgRead, logout: {getMessageContent: {}} }`

# Fonctionnement

- Chaque appel de méthode fait avancer le *session type* de l'objet
- Si la suite du protocole est un choix interne, le client analyse le résultat par un `switch`
- Types linéaires : une seule référence à chaque objet

# Fonctionnement

- Chaque appel de méthode fait avancer le *session type* de l'objet
- Si la suite du protocole est un choix interne, le client analyse le résultat par un `switch`
- Types linéaires : une seule référence à chaque objet

Jugements de typage avec environnement initial et final : $\Gamma \triangleright e : T \triangleleft \Gamma'$

# *Session Types* pour les canaux de communication

- Honda *et al.*, depuis 1993
- Types conçus au départ pour les canaux du $\pi$-calcul ; décrivent un protocole de communication

# *Session Types* pour les canaux de communication

- Honda *et al.*, depuis 1993
- Types conçus au départ pour les canaux du $\pi$-calcul ; décrivent un protocole de communication
  - Séquence, envoi, réception : `!String.?Bool...`

# *Session Types* pour les canaux de communication

- Honda *et al.*, depuis 1993
- Types conçus au départ pour les canaux du $\pi$-calcul ; décrivent un protocole de communication
  - Séquence, envoi, réception : `!String.?Bool...`
  - Choix : $\oplus$`{validate: Σ, cancel: Σ′}`
    Permet une sélection par l'envoi d'une des étiquettes

# *Session Types* pour les canaux de communication

- Honda *et al.*, depuis 1993
- Types conçus au départ pour les canaux du $\pi$-calcul ; décrivent un protocole de communication
  - Séquence, envoi, réception : `!String.?Bool...`
  - Choix : $\oplus$`{validate: Σ, cancel: Σ′}`
    Permet une sélection par l'envoi d'une des étiquettes
  - Branchement : `&{ok: Σ, error: Σ′}`
    Attend la réception de l'une quelconque des étiquettes

# *Session Types* pour les canaux de communication

- Honda *et al.*, depuis 1993
- Types conçus au départ pour les canaux du $\pi$-calcul ; décrivent un protocole de communication
  - Séquence, envoi, réception : `!String.?Bool...`
  - Choix : $\oplus${`validate:` $\Sigma$, `cancel:` $\Sigma'$}
    Permet une sélection par l'envoi d'une des étiquettes
  - Branchement : `&{ok:` $\Sigma$, `error:` $\Sigma'$}
    Attend la réception de l'une quelconque des étiquettes

Les canaux peuvent être considérés comme des objets

$$[\![?T\,.\,\Sigma]\!] = \{\texttt{receive}_T : [\![\Sigma]\!]\}$$
$$[\![!T\,.\,\Sigma]\!] = \{\texttt{send}_T : [\![\Sigma]\!]\}$$

# *Session Types* pour les canaux de communication

- Honda *et al.*, depuis 1993
- Types conçus au départ pour les canaux du $\pi$-calcul ; décrivent un protocole de communication
  - Séquence, envoi, réception : `!String.?Bool...`
  - Choix : $\oplus${`validate:` $\Sigma$, `cancel:` $\Sigma'$}
    Permet une sélection par l'envoi d'une des étiquettes
  - Branchement : `&{ok:` $\Sigma$, `error:` $\Sigma'$`}`
    Attend la réception de l'une quelconque des étiquettes

Les canaux peuvent être considérés comme des objets

$$[\![?T\,.\,\Sigma]\!] = \{\texttt{receive}_T : [\![\Sigma]\!]\}$$
$$[\![!T\,.\,\Sigma]\!] = \{\texttt{send}_T : [\![\Sigma]\!]\}$$
$$[\![\&\{l : \Sigma_l\}_{l\in E}]\!] = \{\texttt{receive}_E : \langle l : [\![\Sigma_l]\!]\rangle_{l\in E}\}$$

# *Session Types* pour les canaux de communication

- Honda *et al.*, depuis 1993
- Types conçus au départ pour les canaux du $\pi$-calcul ; décrivent un protocole de communication
  - Séquence, envoi, réception : `!String.?Bool...`
  - Choix : $\oplus${`validate:` $\Sigma$, `cancel:` $\Sigma'$}
    Permet une sélection par l'envoi d'une des étiquettes
  - Branchement : `&{ok:` $\Sigma$, `error:` $\Sigma'$`}`
    Attend la réception de l'une quelconque des étiquettes

Les canaux peuvent être considérés comme des objets

$$[\![?T\,.\,\Sigma]\!] = \{\texttt{receive}_T : [\![\Sigma]\!]\}$$
$$[\![!T\,.\,\Sigma]\!] = \{\texttt{send}_T : [\![\Sigma]\!]\}$$
$$[\![\&\{l : \Sigma_l\}_{l\in E}]\!] = \{\texttt{receive}_E : \langle l : [\![\Sigma_l]\!]\rangle_{l\in E}\}$$
$$[\![\oplus\{l : \Sigma_l\}_{l\in E}]\!] = \{\texttt{send}_l : [\![\Sigma_l]\!]\}_{l\in E}$$

# Exemple : protocole POP3 (simplifié)

```
Type POP3 = !String.!String.&{OK: Trans, ERR: POP3}

where Trans =
  ⊕{
    STAT: ?int.Trans,
    DELE: !int.Trans,
    QUIT: End,
    RETR: !int.&{OK: ?String.Trans, ERR: Trans}
  }
```

# Un client POP3 (implémente l'interface MailReader)

```
class POP3Client implements MailReader {
  Chan[POP3] c; Null n, m;                                  // fields
  ErrorStatus login (String user, String pass) {
    c.send (user); c.send (pass);
    switch (c.receive()) {
      case OK: c.send (STAT); n = c.receive(); return OK;
      case ERR: return ERR;
  }}
  ErrorStatus fetchAndDelete (int index) {
    c.send (RETR); c.send (index);
    switch (c.receive()) {
      case OK: m = c.receive(); c.send (DELE); return OK;
      case ERR: return ERR;
  }}
  int getNumberOfMessages() {return n;}
  String getMessageContent() {return m;}
  void logout() {c.send (QUIT);}
}
```

# Relation entre interface et implémentation

Session `Init = {login: <OK: NoMsg, ERR: Init>}`

where `NoMsg = {fetchAndDelete: <OK: MsgRead, ERR: NoMsg>, getNumberOfMessages: NoMsg, logout: {} }`

and `MsgRead = {getMessageContent: NoMsg, getNumberOfMessages: MsgRead, logout: {getMessageContent: {}} }`

- Type d'un objet vu de l'extérieur : *session type*, $S$

# Relation entre interface et implémentation

Session `Init = {login: <OK: NoMsg, ERR: Init>}`

where `NoMsg = {fetchAndDelete: <OK: MsgRead, ERR: NoMsg>, getNumberOfMessages: NoMsg, logout: {} }`

and `MsgRead = {getMessageContent: NoMsg, getNumberOfMessages: MsgRead, logout: {getMessageContent: {}} }`

- Type d'un objet vu de l'extérieur : *session type*, $S$
- État interne d'un objet : classe et type de ses champs, $C[F]$

# Relation entre interface et implémentation

Session `Init = {login: <OK: NoMsg, ERR: Init>}`

where `NoMsg = {fetchAndDelete: <OK: MsgRead, ERR: NoMsg>, getNumberOfMessages: NoMsg, logout: {} }`

and `MsgRead = {getMessageContent: NoMsg, getNumberOfMessages: MsgRead, logout: {getMessageContent: {}} }`

- Type d'un objet vu de l'extérieur : *session type*, $S$
- État interne d'un objet : classe et type de ses champs, $C[F]$

Jugements :

- Expressions : $\Gamma \rhd e : T \lhd \Gamma'$
- Pour le corps d'une méthode : $\texttt{this} : C[F] \rhd e : T \lhd \texttt{this} : C[F']$
- Compatibilité entre états interne et externe : $F \vdash C : S$
  Vérification des corps de toutes les méthodes dans l'ordre

# Relation entre états interne et externe

$$\texttt{\{Chan[POP3] c; Null n; Null m\}} \vdash \texttt{POP3Client : Init}$$

# Relation entre états interne et externe

```
{Chan[POP3] c; Null n; Null m} ⊢  POP3Client : Init


 <OK: {Chan[Trans] c; int n; Null m},
  ERR: {Chan[POP3] c; Null n; Null m}>
⊢
  POP3Client : <OK: NoMsg, ERR: Init>
```

# Relation entre états interne et externe

```
{Chan[POP3] c; Null n; Null m} ⊢  POP3Client : Init
```

```
 <OK: {Chan[Trans] c; int n; Null m},
  ERR: {Chan[POP3] c; Null n; Null m}>
⊢
  POP3Client : <OK: NoMsg, ERR: Init>
```

```
 {Chan[Trans] c; int n; Null m} ⊢  POP3Client :  NoMsg
```

etc.

# Sous-typage

- Un objet possédant davantage de méthodes peut remplacer de façon sûre un objet en possédant moins

  par ex. ajout de fetch(); ajout de getNumberOfMessages() à la fin

- Un objet faisant moins de choix internes (plus déterministe) peut remplacer de façon sûre un objet en faisant davantage

# Implémentation : Bica

Extension de Java, utilisant Polyglot

En plus du système formel :

- Objets partagés en plus des objets contrôlés par les sessions
- Boucles while
- Héritage

Sémantique standard de Java. Utilisation d'annotations pour les *session types*.

http://gloss.di.fc.ul.pt/bica/

# Conclusion

- Permet l'implémentation modulaire d'un protocole de communication
- Vérification statique par le typage

# Conclusion

- Permet l'implémentation modulaire d'un protocole de communication
- Vérification statique par le typage

Communication à POPL 2010, version étendue avec preuves disponible sur http://gloss.di.fc.ul.pt/bica/

- Sûreté du typage :
  - La suite des appels de méthode sur un objet suit toujours le protocole déclaré
  - Accord sur les communications dans un contexte distribué
- Algorithme de vérification du typage, implémentation d'un prototype

# Conclusion

- Permet l'implémentation modulaire d'un protocole de communication
- Vérification statique par le typage

Communication à POPL 2010, version étendue avec preuves disponible sur http://gloss.di.fc.ul.pt/bica/

- Sûreté du typage :
  - La suite des appels de méthode sur un objet suit toujours le protocole déclaré
  - Accord sur les communications dans un contexte distribué
- Algorithme de vérification du typage, implémentation d'un prototype

Principaux travaux liés :

- *Session types* dans des langages orientés-objet : Dezani-Ciancaglini *et al.*
- *Typestates* pour objets : Bierhoff et Aldrich

# Conclusion

- Permet l'implémentation modulaire d'un protocole de communication
- Vérification statique par le typage

Communication à POPL 2010, version étendue avec preuves disponible sur http://gloss.di.fc.ul.pt/bica/

- Sûreté du typage :
  - La suite des appels de méthode sur un objet suit toujours le protocole déclaré
  - Accord sur les communications dans un contexte distribué
- Algorithme de vérification du typage, implémentation d'un prototype

Principaux travaux liés :

- *Session types* dans des langages orientés-objet : Dezani-Ciancaglini *et al.*
- *Typestates* pour objets : Bierhoff et Aldrich

À explorer : contrôle de l'aliasing

- Peut-on utiliser des travaux existants ?
- Relation avec le sous-typage ?