Mutation Operators for Actor Systems

Vilas Jagannath, Milos Gligoric, Steven Lauterburg,
Darko Marinov, and Gul Agha

University of Illinois at Urbana-Champaign

April 6th, 2010
Mutation 2010, Paris, France

Why actors?

- Multicore computing is here to stay
- Shared memory multithreaded programs have problems
- Data races, deadlocks, atomicity violations...
- Promising alternative: message passing approaches like actors
- However, still need to test actor systems
- Mutation operators: towards applying mutation testing

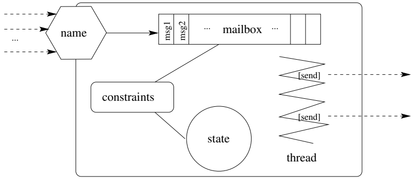
What is an actor?

Object with:

- own thread of control
- local/non-shared state
- mailbox
- unique name

That can:

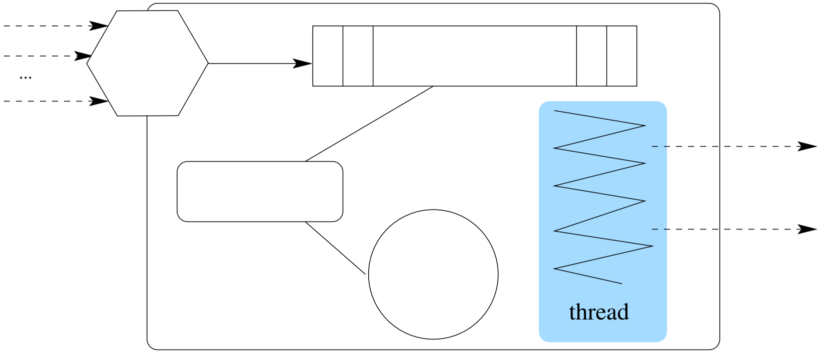
- send/receive messages to/from other actors
- create other actors, destroy actors



Own thread of control

Each actor runs in a separate thread

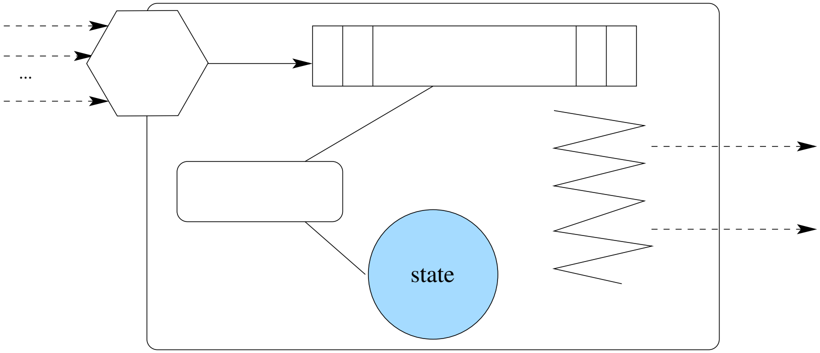
All actors run concurrently



No shared state

Actors can only access their own local state

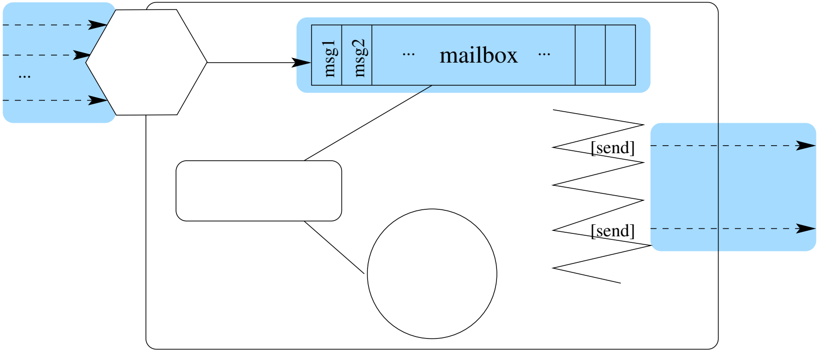
Communication with other actors is performed through messages



Communication

Actors can asynchronously send/receive messages

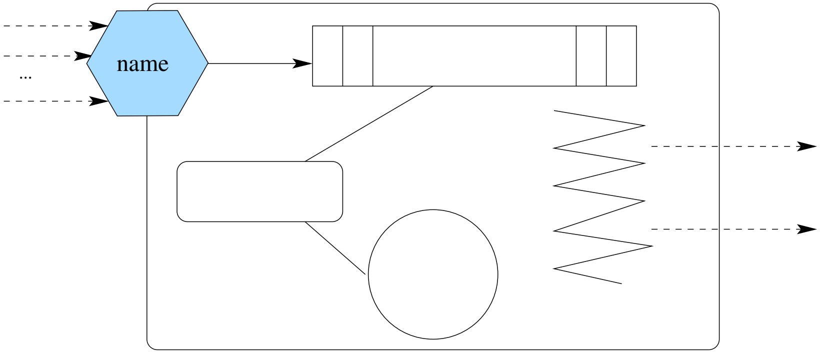
Messages are buffered in mailboxes until processed



Creation and identification

Actors can create other actors (also destroy)

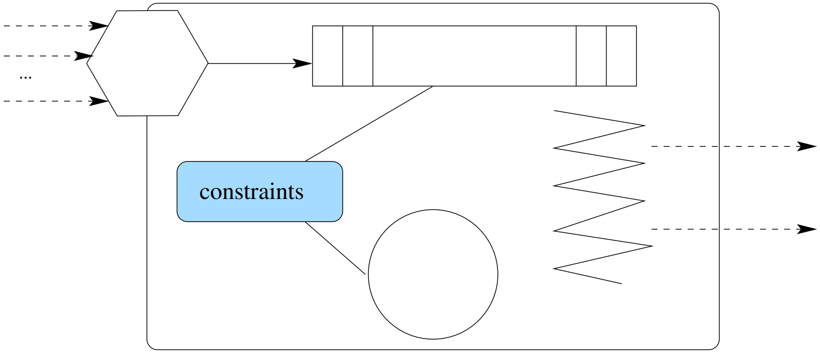
Creation returns a unique name that identifies the new actor



Constraining communication

Actors can have a set of messaging constraints

Constraints enable/disable receipt of messages based on local state



Actor languages/frameworks

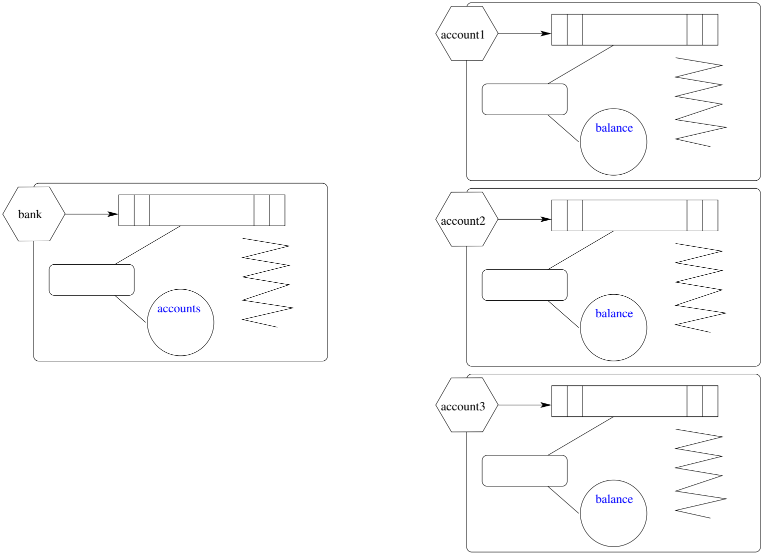
Languages:

- Act, Erlang, Io, Salsa, Scala, Thal...

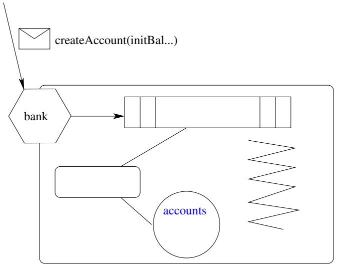
Frameworks:

- C++: Act++, Theron
- Java: ActorFoundry, Kilim
- .Net: Axum, Asynchronous Agents, Singularity
- Python: Parley, Stage
- Ruby: Revactor, Dramatis
- Smalltalk: Acttalk

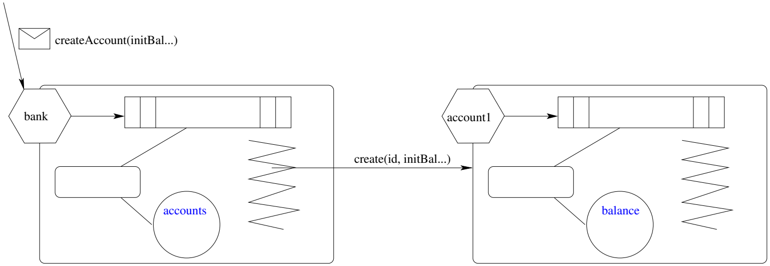
Banking actor system



Actor creation - Opening an account



Actor creation - Opening an account



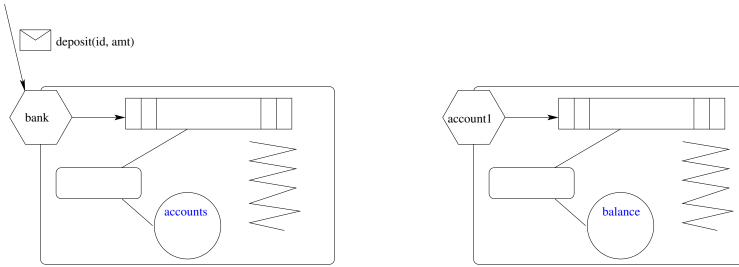
Actor creation - Opening an account

```
class BankActor extends Actor {
  Map<Integer, ActorName> accounts;
  ...
  @message
  int openAccount (String fstName, String lstName) {
    return openAccount(0, fstName, lstName);
  }
  @message
  int openAccount (double initBal, String fstName,
      String lstName) {
    ActorName acc = create(AccountActor.class, nextId,
        initBal, fstName, lstName);
    accounts.put(nextId, acc);
    return nextId++;
  }
  ...
}
```

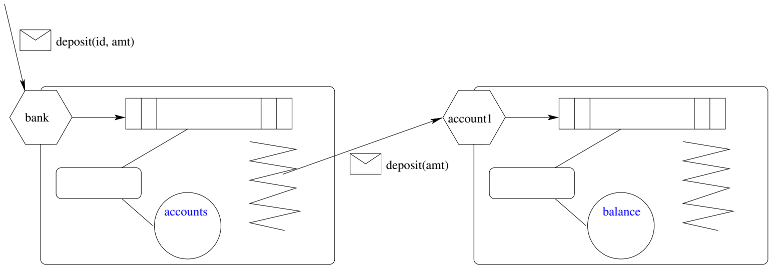
Actor creation - Opening an account

```
class AccountActor extends Actor {
  double balance = 0;
  ...
  AccountActor(int id, int balance, String fstName,
       String lstName) {
    this.id = id; this.balance = balance;
    this.fstName = fstName; this.lstName = lstName;
  }
  ...
}
```

Asynchronous Messaging - Deposit



Asynchronous Messaging - Deposit



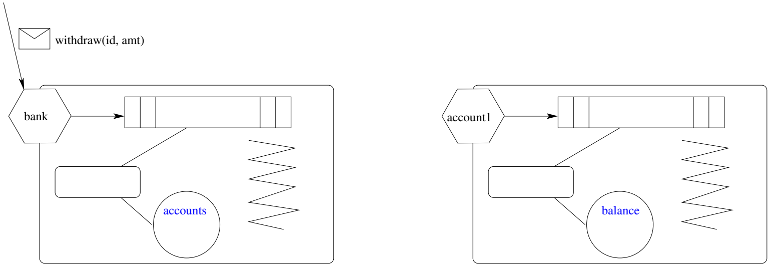
Asynchronous Messaging - Deposit

```
class BankActor extends Actor {
  Map<Integer, ActorName> accounts;
  ...
  @message
  void deposit (int accId, double amount) {
    ActorName acc = accounts.get(accId);
    send(acc, "deposit", amount);
  }
  ...
}
```

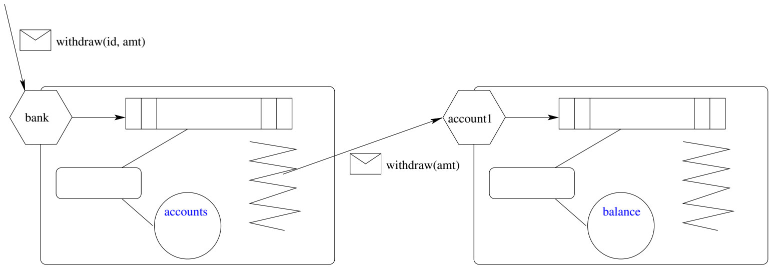
Asynchronous Messaging - Deposit

```
class AccountActor extends Actor {
  ...
  @message
  void deposit (double amount) {
    balance += amount;
  }
  ...
}
```

Messaging Constraints - Withdraw



Messaging Constraints - Withdraw



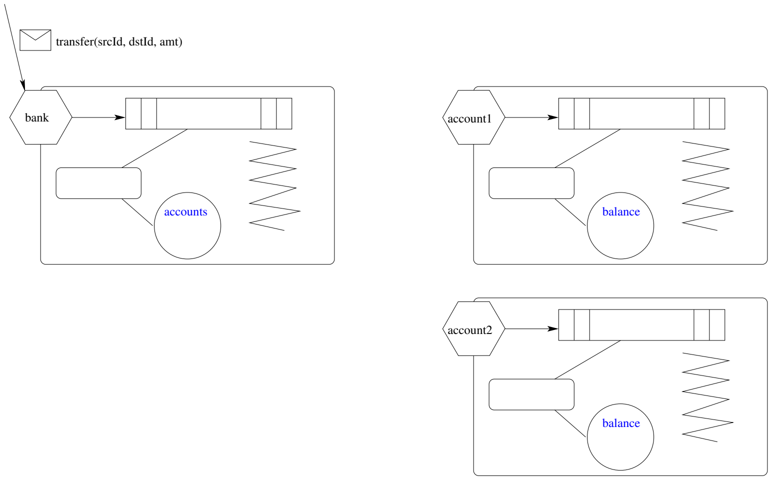
Messaging Constraints - Withdraw

```
class BankActor extends Actor {
  Map<Integer, ActorName> accounts;
  ...
  @message
  void withdraw (int accId, double amount) {
    ActorName acc = accounts.get(accId);
    send(acc, "withdraw", amount);
  }
  ...
}
```

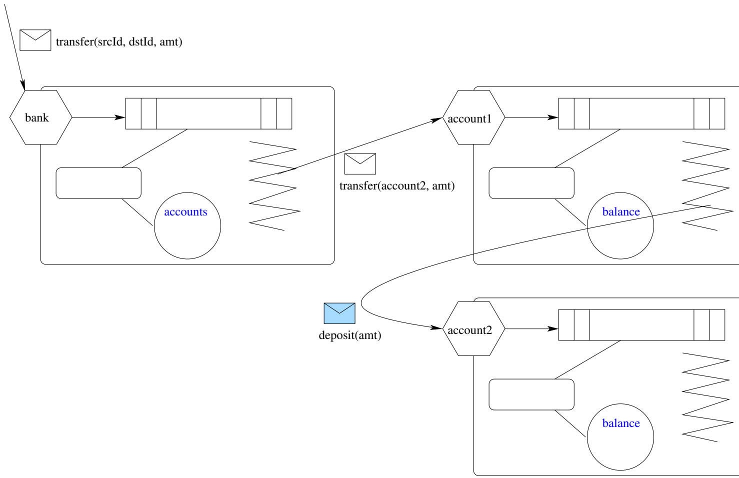
Messaging Constraints - Withdraw

```
class AccountActor extends Actor {
  ...
  @message
  void withdraw (double amount) {
    balance -= amount;
  }
  @disable(messageName = "withdraw")
  boolean withdrawDisabled (double amount) {
    return (amount > balance);
  }
  ...
}
```

Synchronous Messaging - Transfer



Synchronous Messaging - Transfer



Synchronous Messaging - Transfer

```
class BankActor extends Actor {
  Map<Integer, ActorName> accounts;
  ...
  @message
  void transfer (int accIdSrc, int accIdDst, double
      amount) {
    ActorName accSrc = accounts.get(accIdSrc);
    ActorName accDst = accounts.get(accIdDst);
    send(accSrc, "transfer", accDst, amount);
  }
  ...
}
```

Synchronous Messaging - Transfer

```
class AccountActor extends Actor {
  ...
  @message
  void transfer (ActorName accDst, double amount) {
    balance -= amount;
    call(accDst, "deposit", amount);
  }
  @disable(messageName = "transfer")
  boolean transferDisabled (ActorName accDst, double
      amount) {
    return (amount > balance);
  }
  ...
}
```

Communication constructs

- Categories:
  - Messaging: @message, send, call
  - Messaging constraints: @disable
  - Creation: create, destroy
- Common errors related to communication interface
- Operator categories match communication interface

Operators

Category	Actor Mutation Operators
Messaging	**RSR** - Remove Send/Receive
	MMP - Modify Message Parameter
	RMP - Reorder Message Parameters
	MMN - Modify Message Name
	MMR - Modify Message Recipient
	CRT - Change (message) Reference Type
	CST - Change (message) Synchronization Type
Constraint	**RC** - Remove Constraint
	MC - Modify Constraint
Creation/Deletion	**RCD** - Remove Creation/Deletion
	MCP - Modify Creation Parameter
	RCP - Reorder Creation Parameters

Message operators

Category	Actor Mutation Operators
Messaging	**RSR** - Remove Send/Receive
	MMP - Modify Message Parameter
	RMP - Reorder Message Parameters
	MMN - Modify Message Name
	MMR - Modify Message Recipient
	CRT - Change (message) Reference Type
	CST - Change (message) Synchronization Type
Constraint	**RC** - Remove Constraint
	MC - Modify Constraint
Creation/Deletion	**RCD** - Remove Creation/Deletion
	MCP - Modify Creation Parameter
	RCP - Reorder Creation Parameters

MMN - Modify Message Name

Original Code:

```
@message
void deposit (int accId, double amount) {
  ActorName acc = accounts.get(accId);
  send(account, "deposit", amount);
}
```

MMN Mutant:

```
@message
void deposit (int accId, double amount) {
  ActorName acc = accounts.get(accId);
  // deposit changed to withdraw
  send(account, "withdraw", amount);
}
```

CST - Change (message) Synchronization Type

Original Code:

```
@message
void deposit (int accId, double amount) {
  ActorName acc = accounts.get(accId);
  send(account, "deposit", amount);
}
```

CST Mutant:

```
@message
void deposit (int accId, double amount) {
  ActorName acc = accounts.get(accId);
  // send changed to call
  call(account, "deposit", amount);
}
```

Constraint operators

Category	Actor Mutation Operators
Messaging	**RSR** - Remove Send/Receive
	MMP - Modify Message Parameter
	RMP - Reorder Message Parameters
	MMN - Modify Message Name
	MMR - Modify Message Recipient
	CRT - Change (message) Reference Type
	CST - Change (message) Synchronization Type
Constraint	**RC** - Remove Constraint **MC** - Modify Constraint
Creation/Deletion	**RCD** - Remove Creation/Deletion
	MCP - Modify Creation Parameter
	RCP - Reorder Creation Parameters

RC - Remove Constraint

Original Code:

```
@disable(messageName = "withdraw")
boolean withdrawDisabled (double amount) {
  return (amount > balance);
}
```

RC Mutant:

```
// removed annotation mapping this constraint
// method to the withdraw message
boolean withdrawDisabled (double amount) {
  return (amount > balance);
}
```

MC - Modify Constraint

Original Code:

```
@disable(messageName = "transfer")
boolean transferDisabled (ActorName accDst, double
    amount) {
  return (amount > balance);
}
```

MC Mutant:

```
@disable(messageName = "transfer")
boolean transferDisabled (ActorName accDst, double
    amount) {
  // changed > to <
  return (amount < balance);
}
```

Creation/Deletion Related Mutation Operators

Category	Actor Mutation Operators
Messaging	**RSR** - Remove Send/Receive
	MMP - Modify Message Parameter
	RMP - Reorder Message Parameters
	MMN - Modify Message Name
	MMR - Modify Message Recipient
	CRT - Change (message) Reference Type
	CST - Change (message) Synchronization Type
Constraint	**RC** - Remove Constraint
	MC - Modify Constraint
Creation/Deletion	**RCD** - Remove Creation/Deletion
	MCP - Modify Creation Parameter
	RCP - Reorder Creation Parameters

MCP - Modify Creation Parameter

Original Code:

```
@message
int openAccount (double initBal, String fstName, String
    lstName) {
  ActorName acc = create(AccountActor.class, nextId,
     initBal, fstName, lstName);
  accounts.put(nextId, acc); return nextId++; }
```

MCP Mutant:

```
@message
int openAccount (double initBal, String fstName, String
    lstName) {
  // initBal parameter changed to 0
  ActorName acc = create(AccountActor.class, nextId, 0,
      fstName, lstName);
  accounts.put(nextId, acc); return nextId++; }
```

RCP - Reorder Creation Parameters

Original Code:

```
@message
int openAccount (double initBal, String fstName, String
    lstName) {
  ActorName acc = create(AccountActor.class, nextId,
     initBal, fstName, lstName);
  accounts.put(nextId, acc); return nextId++; }
```

RCP Mutant:

```
@message
int openAccount (double initBal, String fstName, String
    lstName) {
  // reordered fstName and lstName
  ActorName acc = create(AccountActor.class, nextId,
     initBal, lstName, fstName);
  accounts.put(nextId, acc); return nextId++; }
```

Related Work

- Mutation testing researched for couple of decades
- Operators for many languages/paradigms [Jia, Harman 2010]
- Closest work:
  - Interface mutation [Gosh, Mathur 2001]
  - Operators for specifications & models [Srivatanakul et al 2003 & Aichernig, Delgado 2006]
  - Fault injection based reliability testing [Arlat et al 1990 & Chandra et al 2004]

Conclusions

- Actor systems gaining popularity
- Identified mutation operators for actor systems
- Future work:
  - Classify/document common errors
  - Measure effectiveness of operators
  - Implement mutation testing system
  - Support multiple actor frameworks
  - Efficient exploration (related talk tomorrow - MuTMuT)