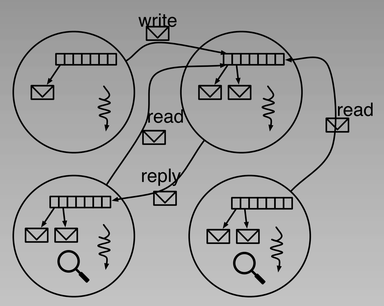
Coordination in Ambient Environments

Christophe Scholliers, Wolfgang De Meuter
Software Languages Lab - Vrije Universiteit Brussel - Belgium

PAM

Motivation

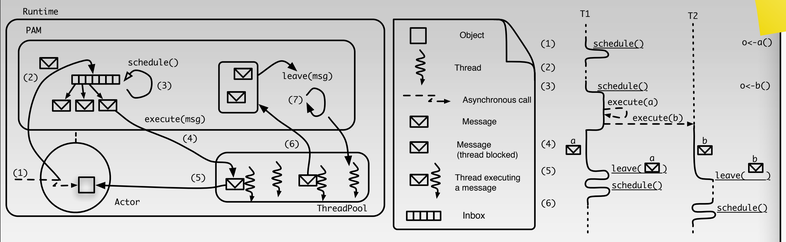
While the actor model of concurrency is well appreciated for its ease of use, its scalability is often criticized. The fact that execution **within** an actor is sequential prevents certain actor systems to take advantage of multicore architectures.



Example

```
def RWSched() { scheduler: {
    def R := Category();
    def W := Category();
    def writing := false;
    def readers := 0;
    def schedule() {
        if: !(writing) then: {
            def executing := super.executeAllOlderThan(R,W);
            readers := readers + executing;
            if: (readers == 0) then: {
            writing := super.executeOldest(W);
            };
        };
    };
    def leave(letter) {
      dispatchCategory: letter as:
      [[R, { readers := readers - 1 }],
       [W, { writing := false }]];
    };
};};
```

Model

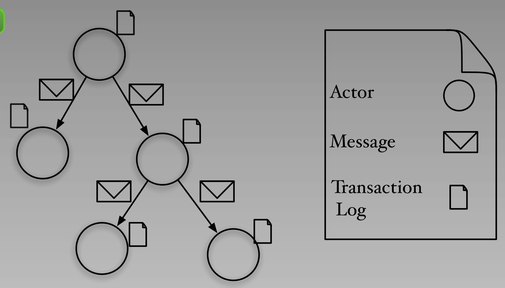


PAM relaxes event-loop-concurrency models by allowing the parallel execution of messages.

Transactors

Motivation

Model



Example

```
atomic: {
    def day := Find(Hotel,Train);
    Hotel<-Book(day);
    Train<-Book(day);
};
```

```
atomic: {
    SeatReservation<-Book(day);
    TicketReservation<-Book(day);
};
```

Efficiency

Atomicity

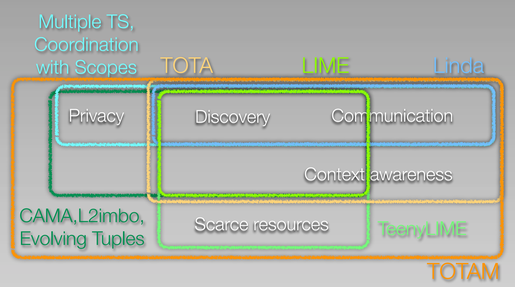
Transitivity

Synchronization

TOTAM

Motivation

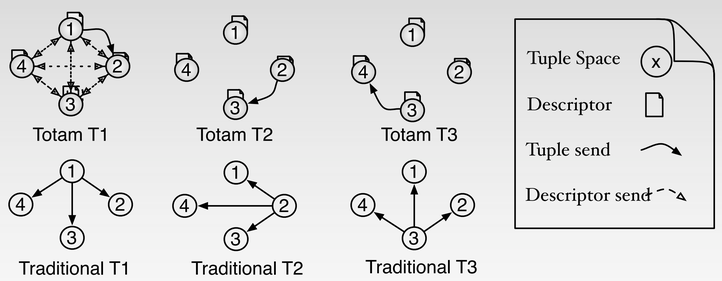
Coordination of mobile applications posses a number of issues. Devices should be able to communicate with each other without being connected with each other at the same time while maintaining privacy and limited network traffic. Current tuple based approaches solve these issues partially but none of them solves **all** of them.



Example

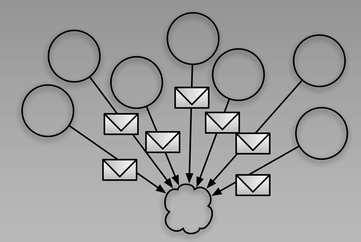
Each tuple space has a tuple space **descriptor**.
Tuples are propagated to neighbours **in the scope** of the tuple.

Model



Quatre Mains

Motivation



Coordinating concurrent invocations from the ambient.

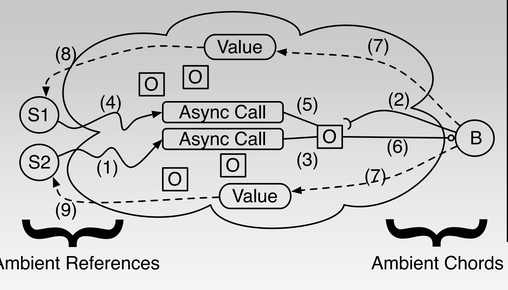
Example

```
defTypeTag searchObject;
// All request  items in the ambient
def ambientReference := ambient: searchObject;
//Seller One
ambientReference<-offer(new offer("Tv",150))@all;
//Seller Two
ambientReference<-offer(new offer("Radio",100))@all;

//Search protoype
def search := object: {
  offer(one:[category = "tv" ]),offer(two:[category="radio"])
    { |Senders|
      Senders.return("I am intrested");
  };
};

//Put search in the ambient
export: search as: searchObject;
```

Model



http://soft.vub.ac.be/~cfscholl