Vrije Universiteit Brussel

Loosely-coupled Distributed Reactive Programming in Mobile Ad Hoc Networks

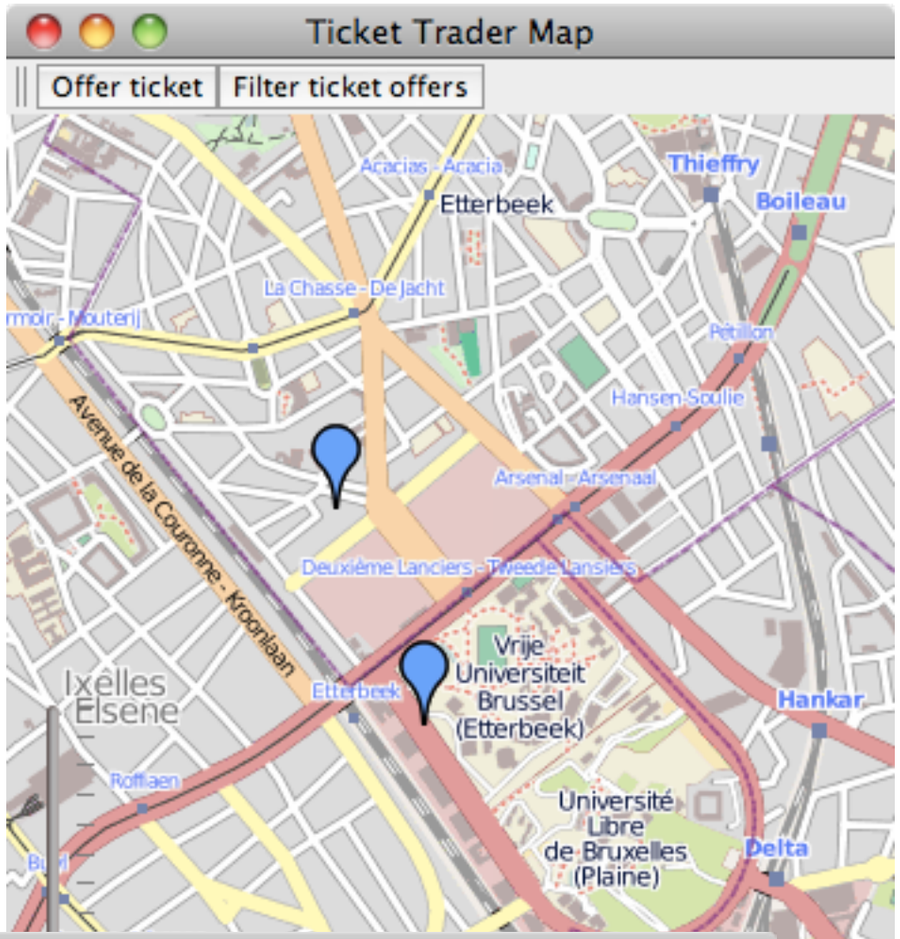
Andoni Lombide Carreton

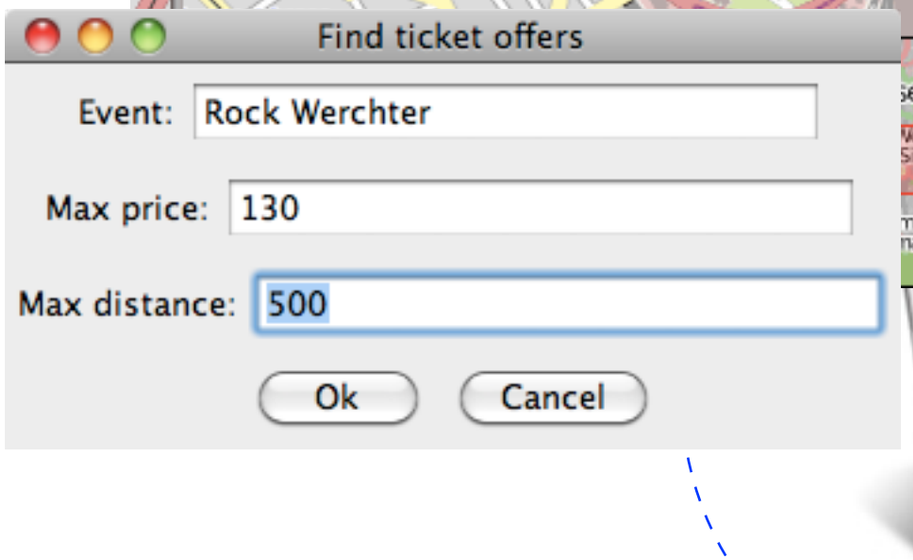
Software Languages.Lab

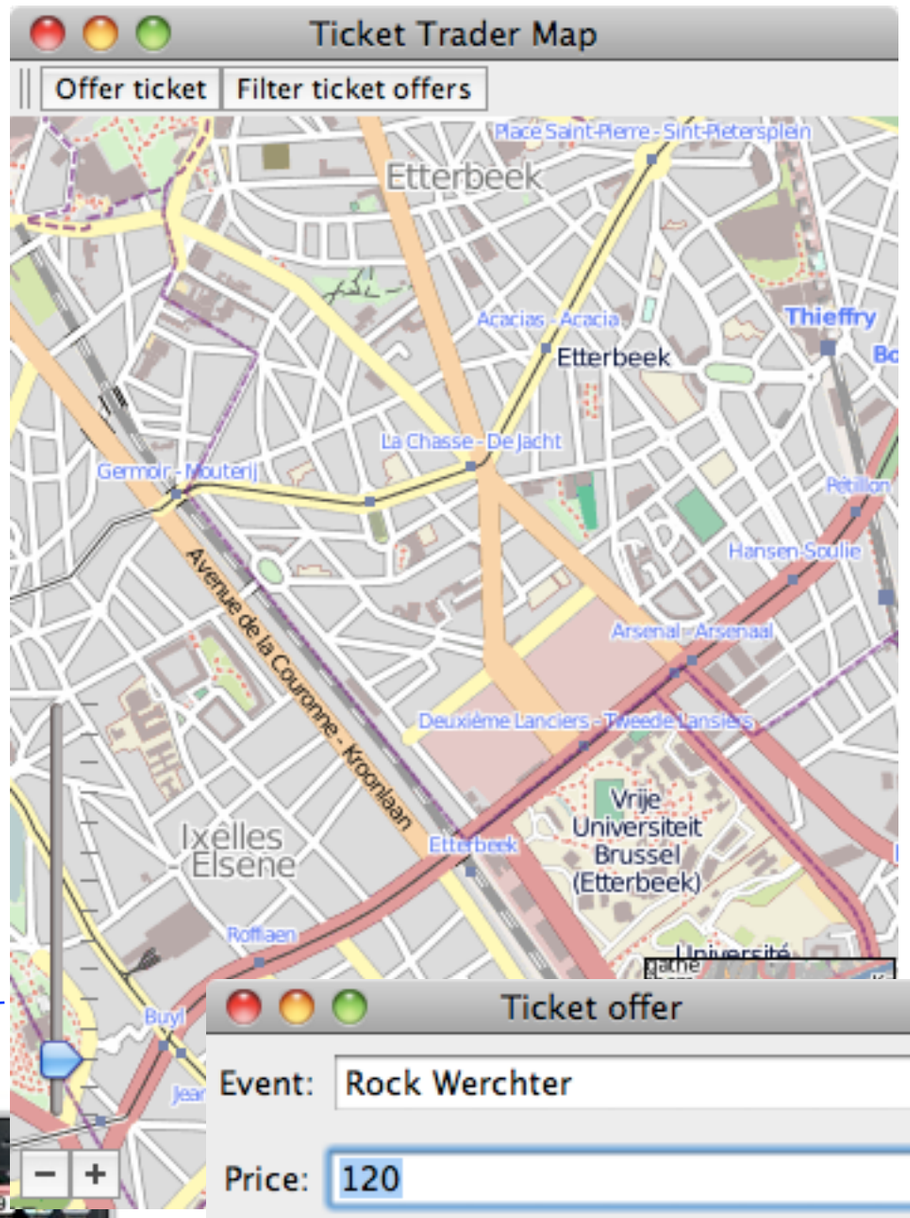
Department of Computer Science
Vrije Universiteit Brussel

December 17
Santiago, Chile

The mobile ticket trader application

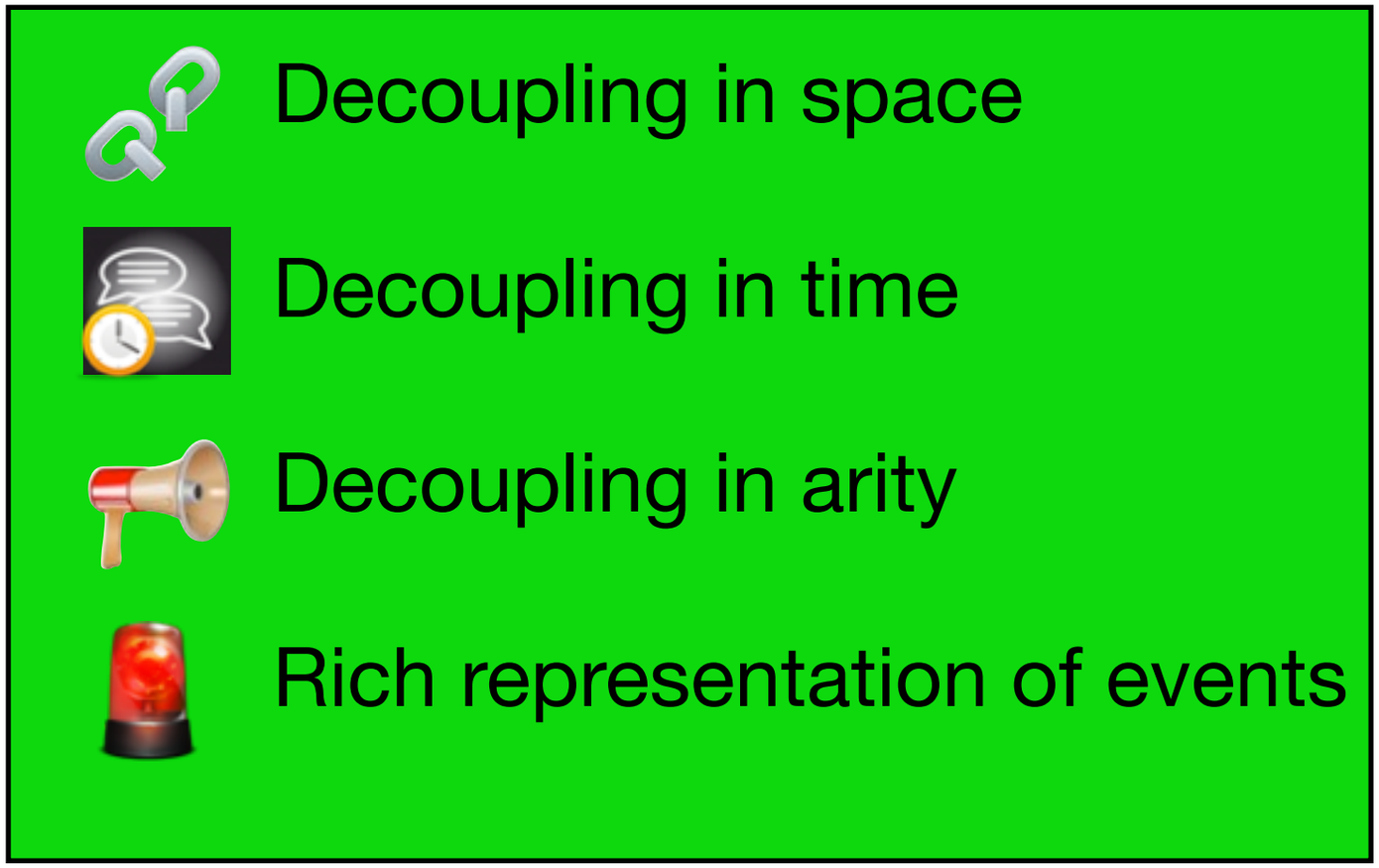




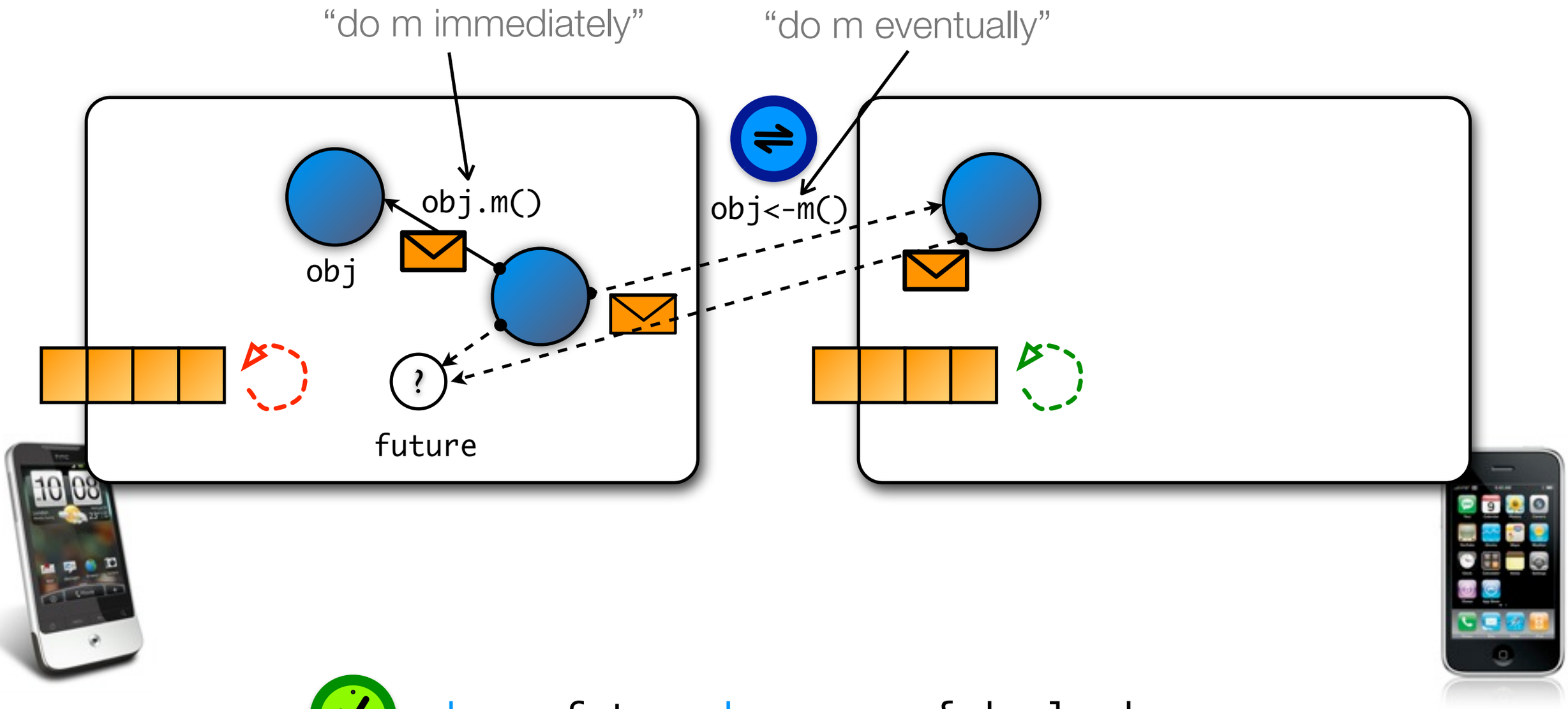


Pervasive Computing and Mobile Ad Hoc Networks

Ambient-oriented Programming in AmbientTalk

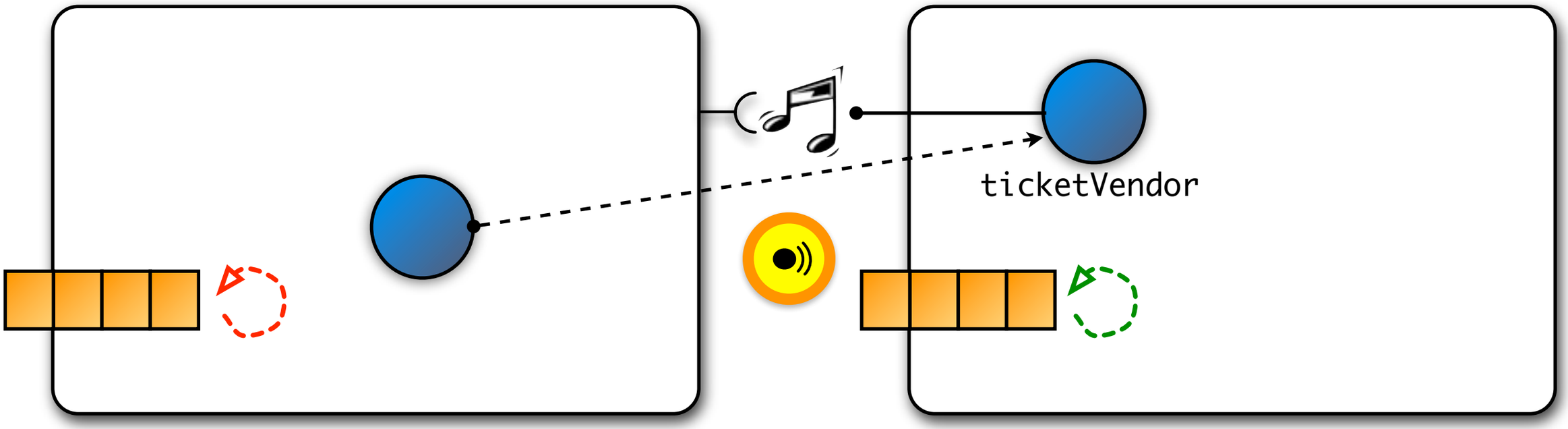


Event Loop Concurrency in AmbientTalk




```
when: future becomes: { |value|
    // process reply
}
```

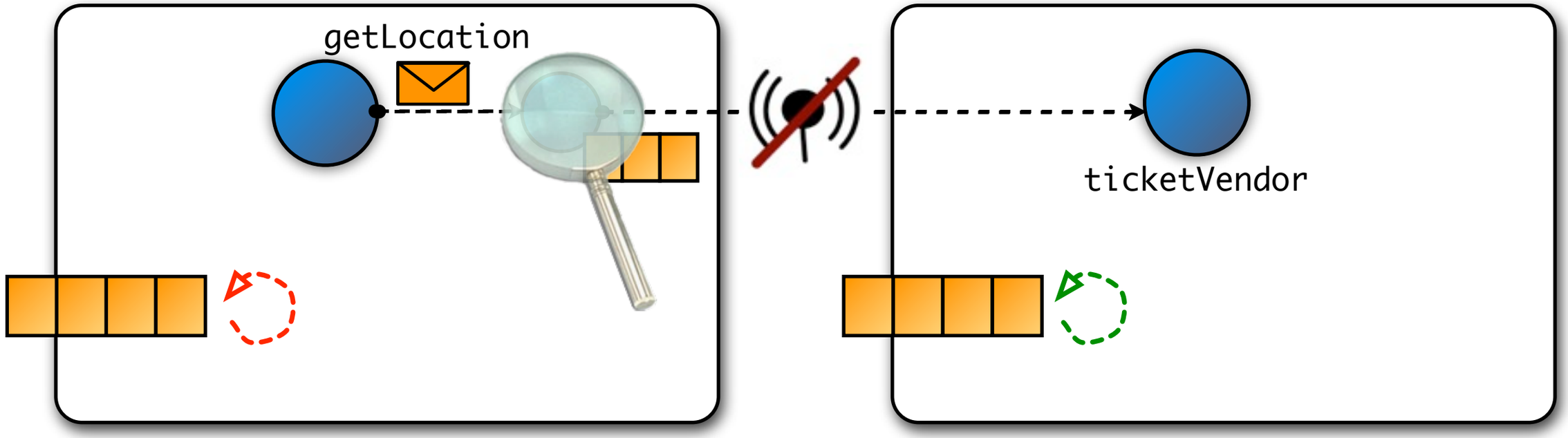
Publishing & discovering objects by topic




```
deftype TicketVendor
```

```
deftype TicketVendor
  export: ticketVendor as: TicketVendor
whenever: TicketVendor discovered: { |ticketVendor|
  // React on ticketVendor appearance
  when: ticketVendor disconnected: {
    // React on disappearance
  }
  when: ticketVendor reconnected: {
    // React on reappearance
  }}
```

Asynchronous communication




```
when: TicketVendor discovered: { |ticketVendor|
  when: ticketVendor<-getLocation()@Due(timeout) becomes: { |loc|
    // Update user interface with the location
  } catch: TimeoutException using: { |e|
    // Remove ticket offer from user interface
  }
}
```

The Ticket Trader Application

Event	Signaling	Reacting
New ticket trader connected	Automatic notification by AmbientTalk	`whenever:discovered:`
Ticket trader disconnected	Automatic notification by AmbientTalk	`when:disconnected:`
Ticket trader reconnected	Automatic notification by AmbientTalk	`when:reconnected:`
New ticket for sale	`notifyTicketForSale` asynchronous message	`notifyTicketForSale` callback
Price of ticket changed	`notifyPriceChanged` asynchronous message	`notifyPriceChanged` callback
Location of ticket trader changed	`notifyLocationChanged` asynchronous message	`notifyLocationChanged` callback
Own location changed	GPS abstraction invokes callback with new coordinates	Callback registered on GPS abstraction

*main.at

```
1
2
3 def seconds := jlobby.edu.vub.at.signals.natives.NATSeconds.instance();
4
5 system.println(seconds);
6
7
8
9
10
```

Problems | Javadoc | Declaration | Search | LogCat

Android

Save and Launch

Select resources to save:

main.at

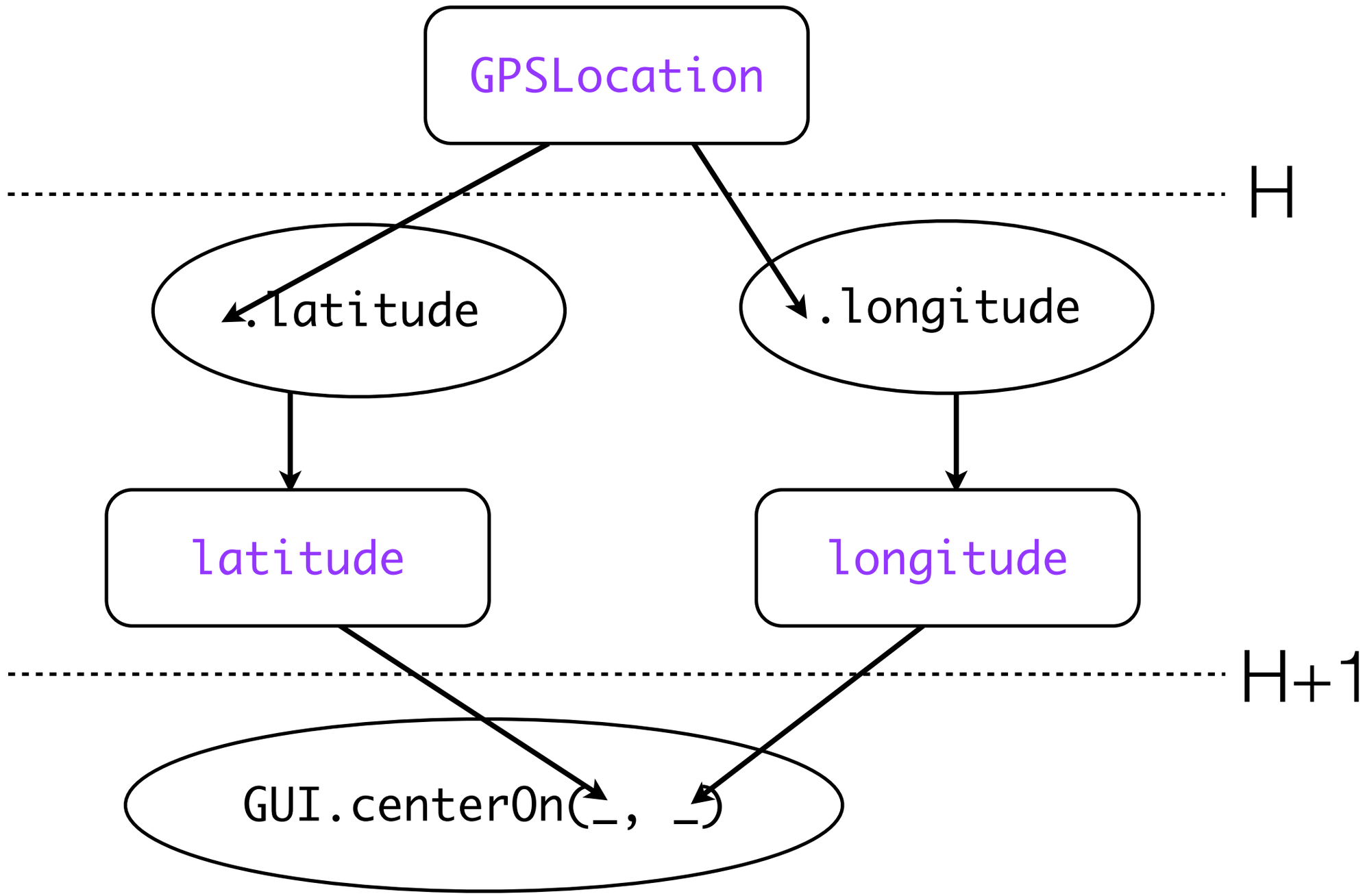
Select All | Deselect All

Always save resources before launching

Cancel | OK

Reactive Programming in AmbientTalk/R

```
GUI.centerOn(GPSLocation.latitude, GPSLocation.longitude);
```

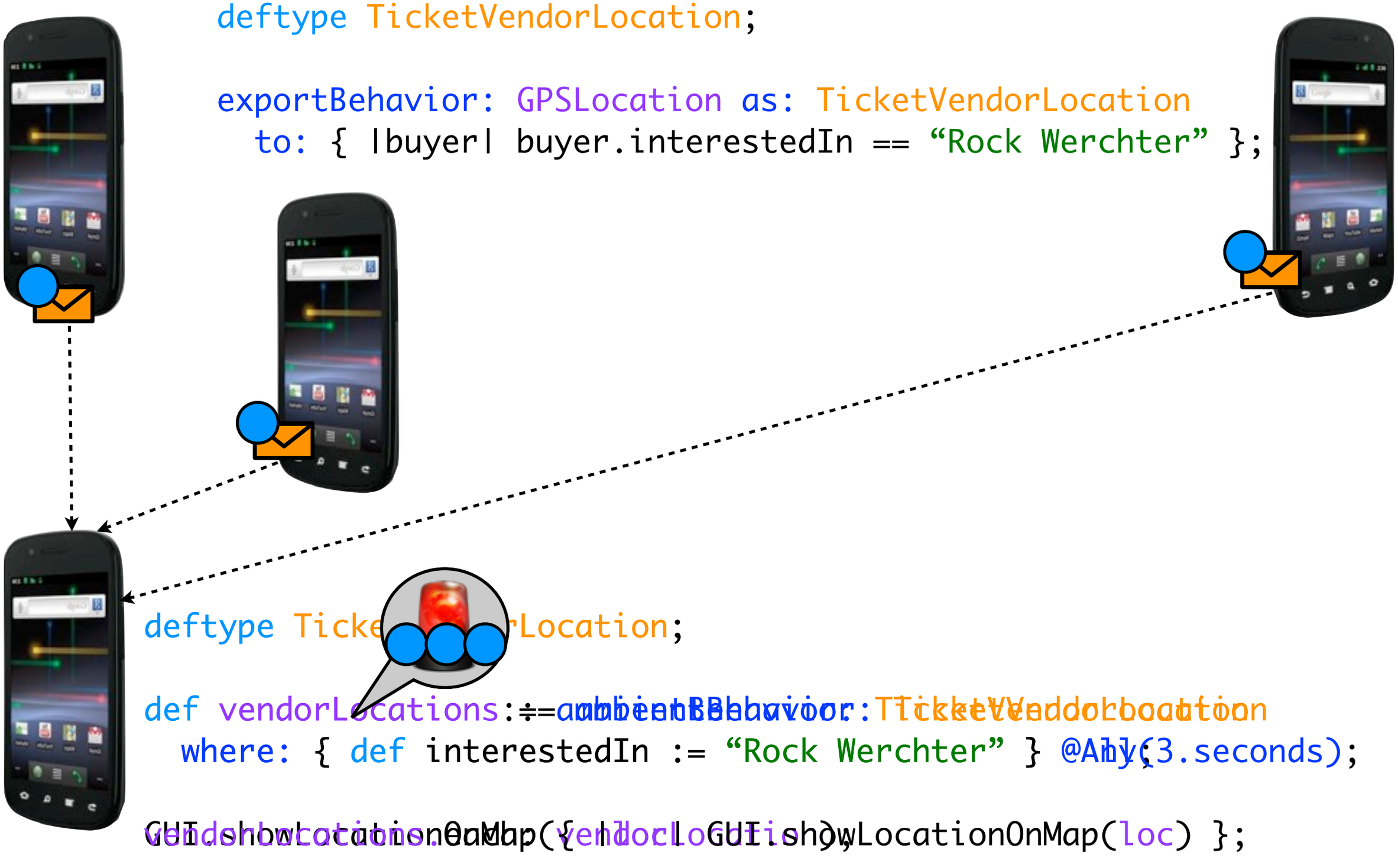


Reactive Objects 

```
def Coordinate := isolate: {
  def latitude := 0;
  def longitude := 0;

  def distanceTo(anotherCoordinate) {
    // Compute via Haversine formula
  };

  def @Mutator update(newLatitude, newLongitude) {
    latitude := newLatitude;
    longitude := newLongitude;
  };
};

def GPSLocation := makeReactive(Coordinate.new());

GPS.addLocationObserver: { |lat, lon|
  GPSLocation.update(lat, lon)
};
```

GPSLocation

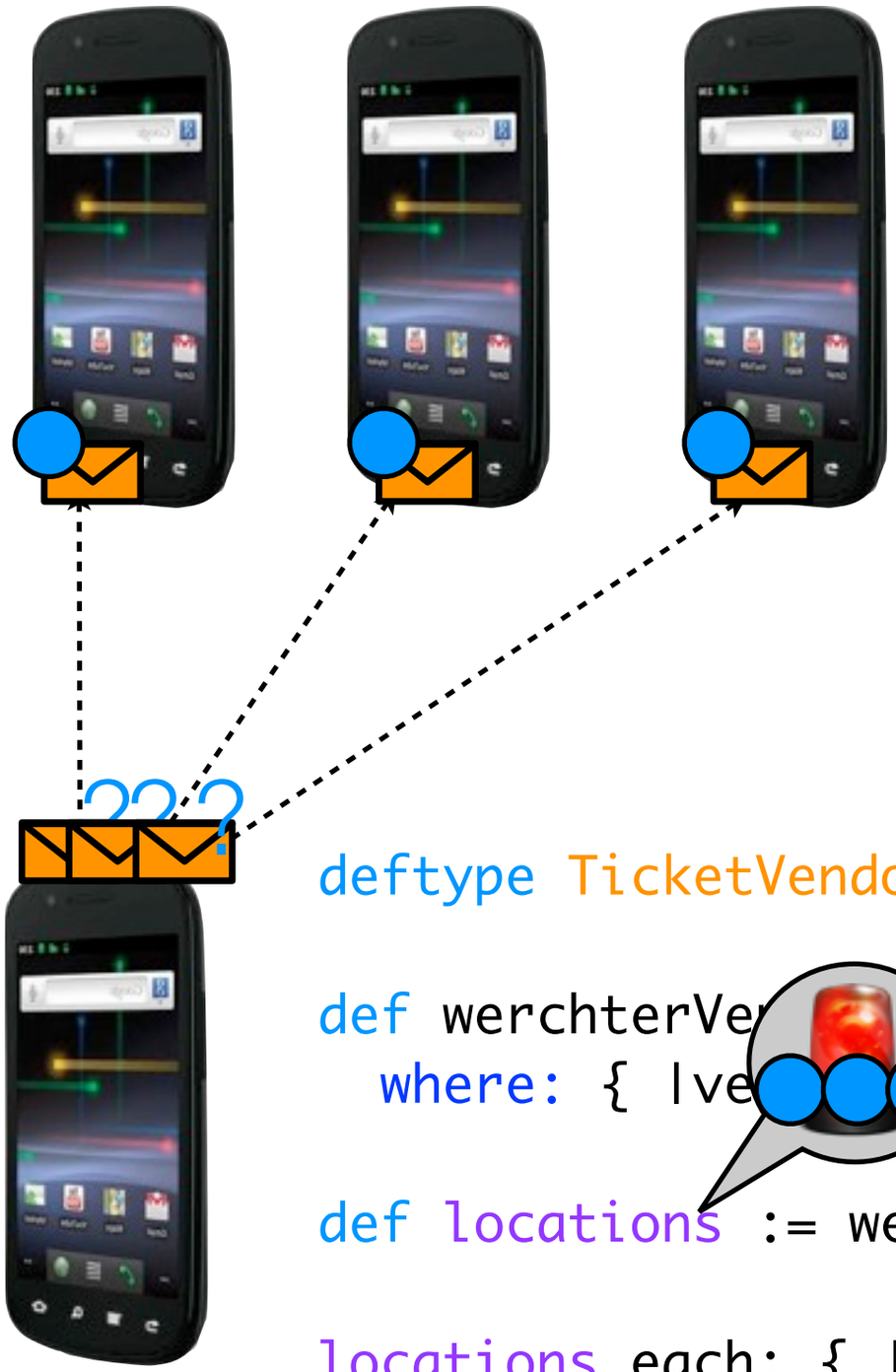
Ambient Behaviors



```
deftype TicketVendorLocation;

exportBehavior: GPSLocation as: TicketVendorLocation
  to: { |buyer| buyer.interestedIn == “Rock Werchter” };
```

```
deftype TicketVendorLocation;

def vendorLocations := ambientBehavior: TicketVendorLocation
  where: { def interestedIn := “Rock Werchter” } @Any(3.seconds);

vendorLocations.onEach: { |loc| GUI.showLocationOnMap(loc) };
```

Reactive Queries



```
deftype TicketVendor;

export: ticketVendor as: TicketVendor
  with: { def event := “Rock Werchter” };
```

```
deftype TicketVendor;

def werchterVe      := ambient: TicketVendor
  where: { |ve      vendor.event == “Rock Werchter” };

def locations := werchterVendors<-getLocation()@Refresh(3.seconds);

locations.each: { |loc| GUI.showLocationOnMap(loc) };
```

Evaluation

```
ef ticketVendor := object: {

  def clients := HashSet.new();
  def ticketOffers := HashSet.new();

  whenever: TicketClient discovered: { |client|
      clients.add(client);
      when: client disconnected: {
          clients.remove(client);
      };
      when: client reconnected: {
          clients.add(client);
      };
  };

  def offerTicket(ticketOffer) {
      clients.each: { |client| client<-notifyTicketForSale(ticketOffer) };
  };

  def changeTicketPrice(ticketOffer, newPrice) {
      ticketOffer.price := newPrice;
      clients.each: { |client| client<-notifyPriceChanged(ticketOffer) };
  };

      GPSLocation.addChangeListener({ |newLocation|
      ticketOffers.each: { |offer|
          offer.location := newLocation;
          clients.each: { |client| client<-notifyLocationChanged(offer) };
      };
  });

  def forceNotifyLocationChanged(client) {
      ticketOffers.each: { |offer| client<-notifyLocationChanged(offer) };
  };

xport: ticketVendor as: TicketVendor;
```

```
def interestingOffers := HashSet.new();

whenever: TicketVendor discovered: { |vendor|
    allVendors.add(vendor);
    when: vendor disconnected: {
        allVendors.remove(vendor);
    };
    when: vendor reconnected: {
        allVendors.add(vendor);
    };
};

def isInterestingOffer(ticketOffer) {
    ((ticketOffer.eventName == event).and:
        { ticketOffer.price <= maxPrice }).and:
            { myLocation.distanceTo(ticketOffer.location) }
        };

def notifyTicketForSale(ticketOffer) {
    if: isInterestingOffer(ticketOffer) then: {
        interestingOffers.add(ticketOffer);
        GUI.showTicketOffersOnMap(interestingTicketOffers);
    };
};

def notifyPriceChanged(ticketOffer) {
    if: isInterestingOffer(ticketOffer) then: {
        interestingOffers.add(ticketOffer);
    } else: {
        interestingOffers.remove(ticketOffer);
    };
    GUI.showTicketOffersOnMap(interestingTicketOffers);
};

def notifyLocationChanged(ticketOffer) {
    if: isInterestingOffer(ticketOffer) then: {
        interestingOffers.add(ticketOffer);
    } else: {
        interestingOffers.remove(ticketOffer);
    };
    GUI.showTicketOffersOnMap(interestingTicketOffers);
};

    GPSLocation.addChangeListener({ |newLocation|
    allVendors.each: { |vendor| vendor<-forceNotifyLocationChanged(sel
});
```

Limitations and Future Work

- AmbientTalk/R yields a higher computational overhead than plain AmbientTalk.

- Event message order preservation not guaranteed on very fine-grained levels over different communication partners.

- No timing guarantees.

- Event consumers can only create and cancel their distributed dependencies: they cannot limit the number of propagated events.