Actor as a Router

Tomohiro Suzuki

Outline

- Short introduction about AmbientTalk
- My work
  - I extended AmbientTalk to join many networks
  - I implemented Gateway Actors
  - Now trying to routing protocols in actor
- Pantaxou
  - Language approach for coordination in networks

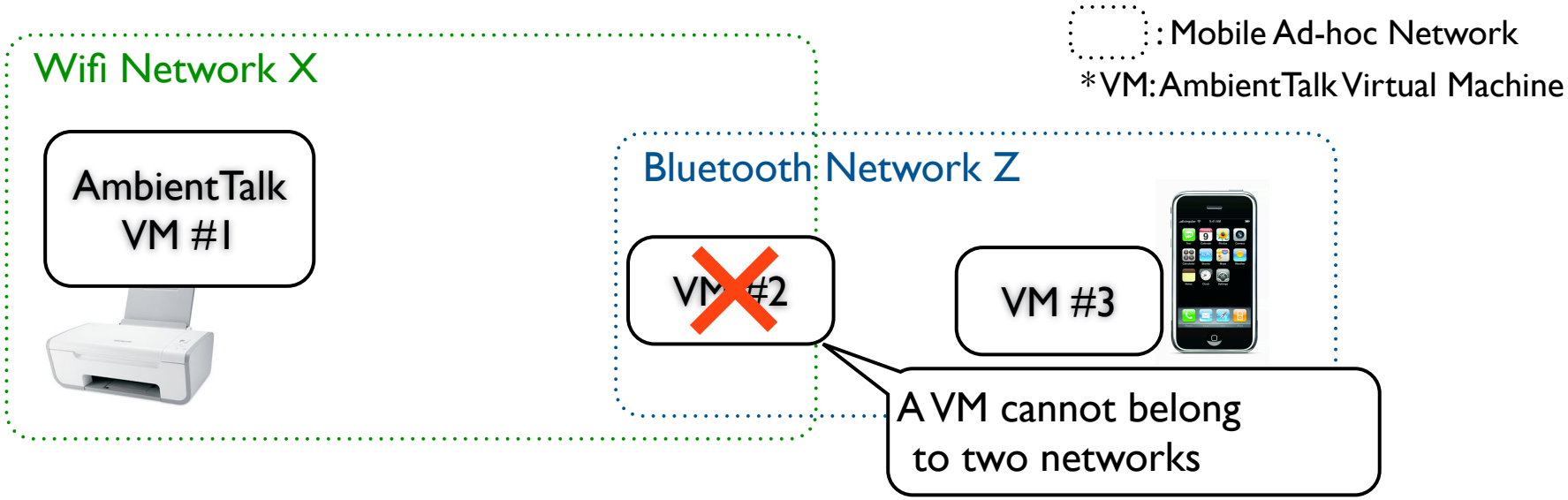
AmbientTalk Introduction

- <別すらいど>

修士論文中間発表より

Limitation of AmbientTalk

Original AmbientTalk cannot communicate beyond one MANET

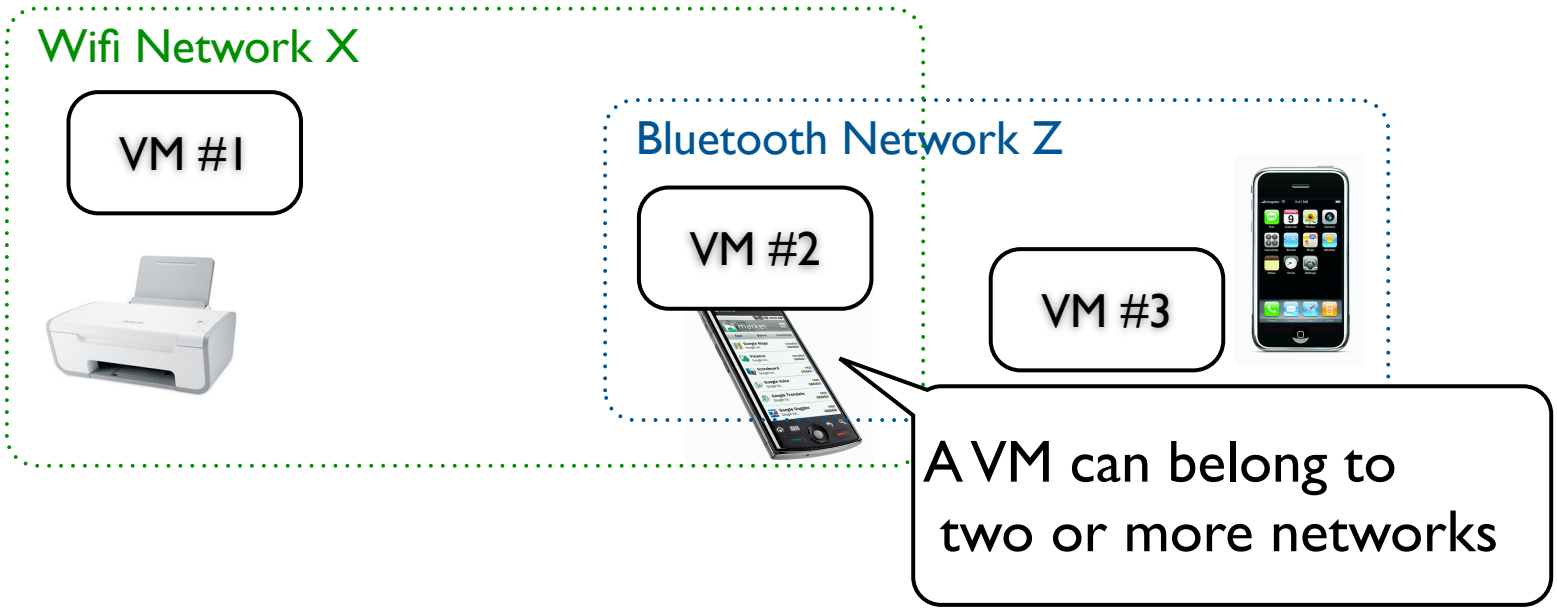


Our Approach to the limitation

AmbientTalk VM to join several MANETs

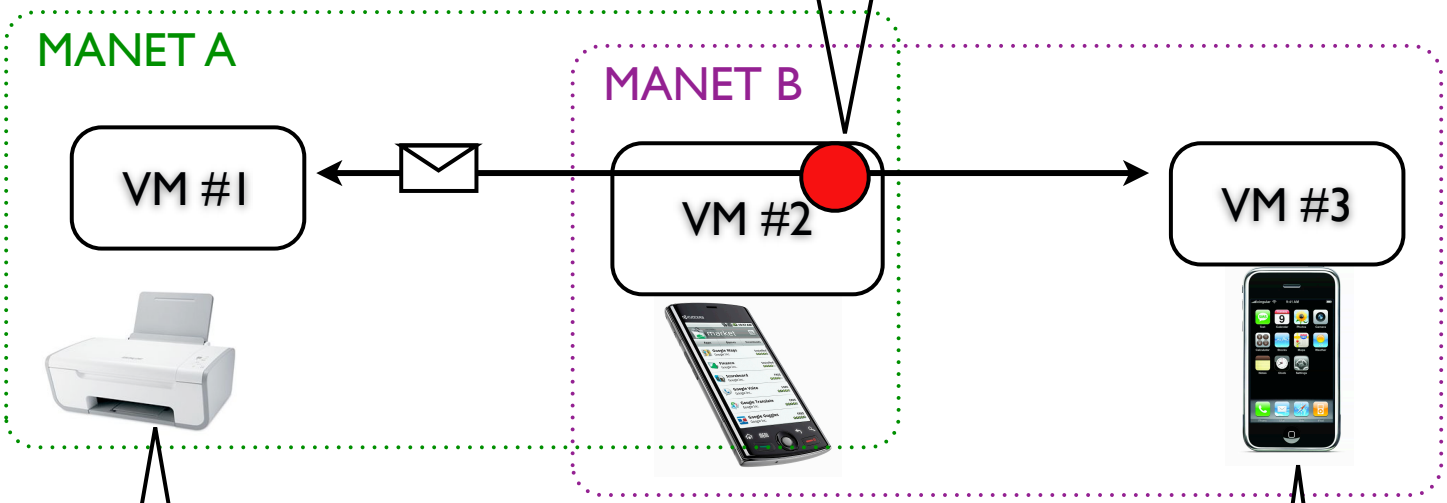
* MANET: Wifi, Ethernet, Bluetooth (not yet)

<u>VM #2 can receive messages from both VM #1 and VM #3</u>



Gateway Actors

that relays messages through different MANETs



* MANET:
Wifi, Ethernet,
Bluetooth (not yet)

```
def obj := object:{
  def print(s) {
    ...
  }
};
export: obj as: Printer;
```

```
when: Printer discovered:{|p|
  p<-print("Hello");
}
```

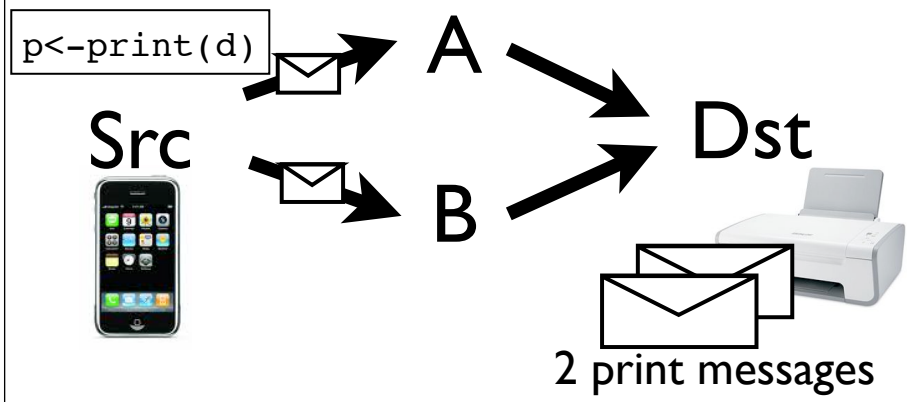
No need to change existing code

Approach by *Smart* Gateway Actors

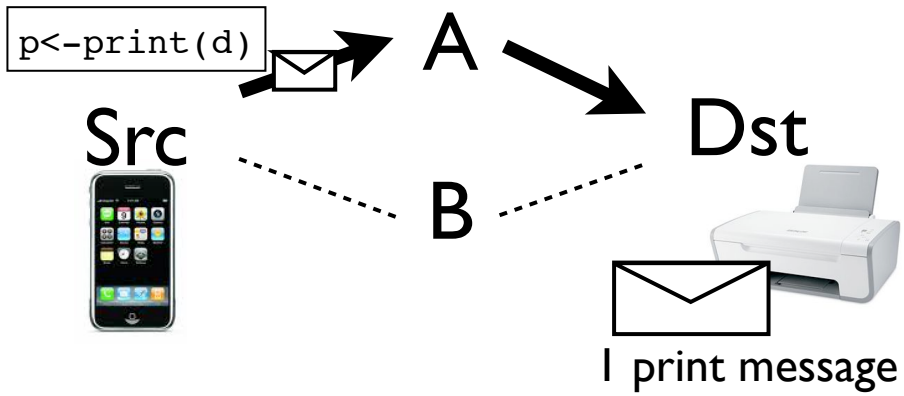
Smart Gateway Actors to avoid the problem, by

- managing service identity to avoid message duplications
- deciding which path(s) to use, for efficiency and robustness

(Naive) Gateway Actor: A, B



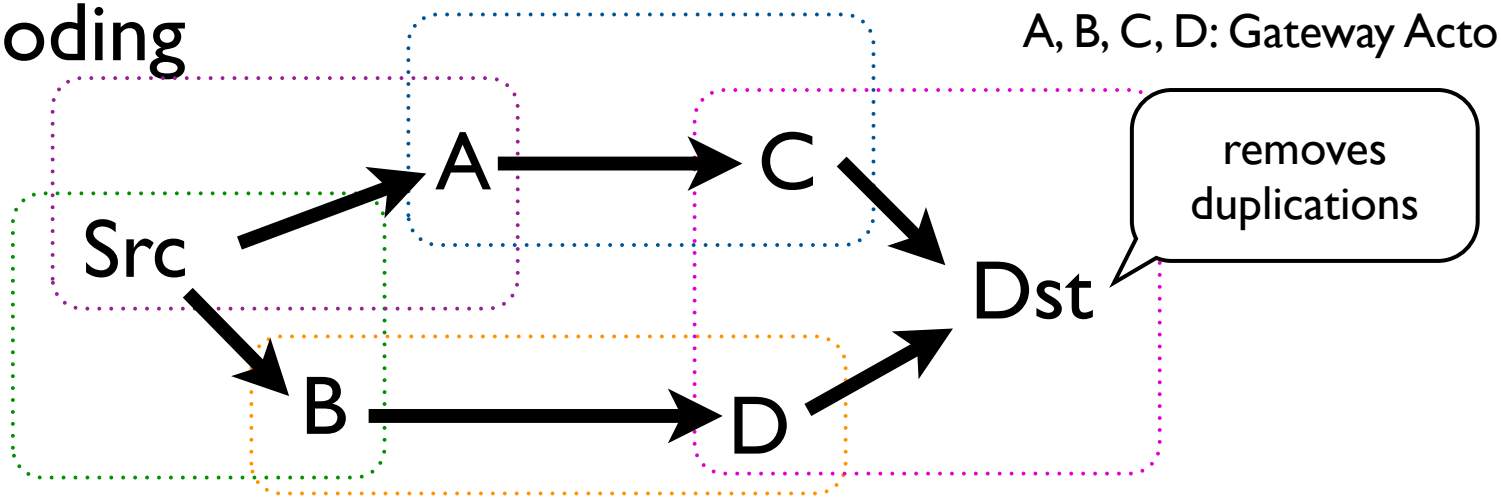
Smart Gateway Actor: A, B



Smart Gateway Actors Example

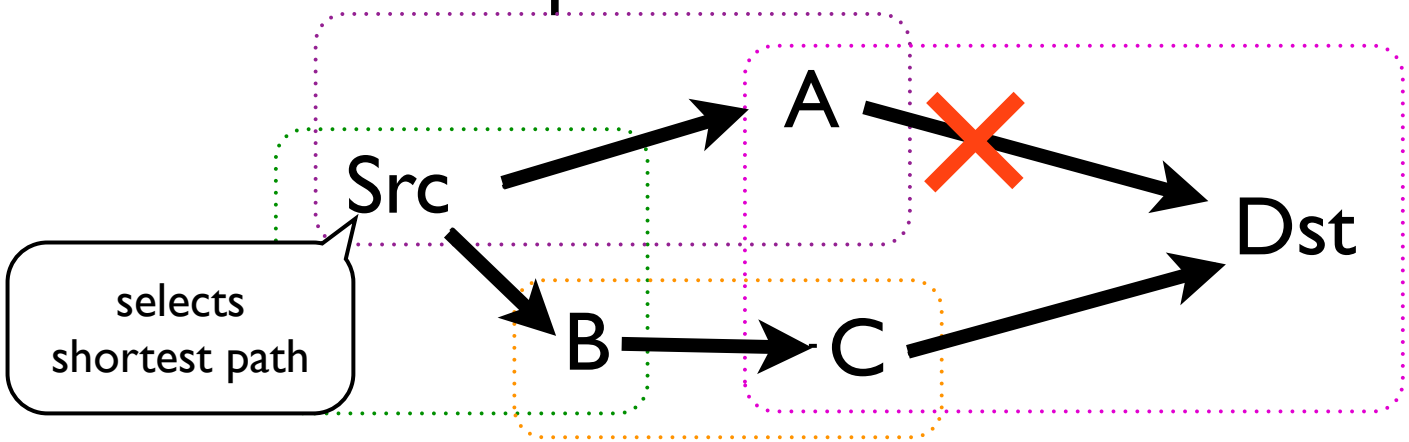
- Flooding

A, B, C, D: Gateway Actors in VMs

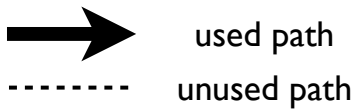


When Src sends a message,
it sends the message to all paths to Dst

- Smallest hops

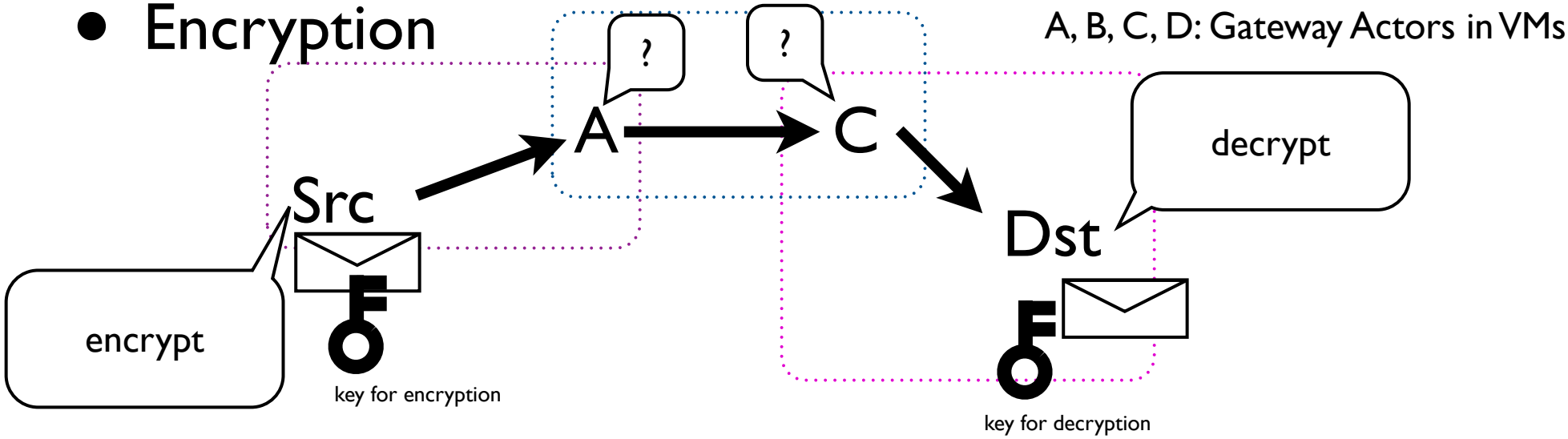


When Src sends a message,
it chooses a path that has smallest hops



Smart Gateway Actors Example cont.

- Encryption



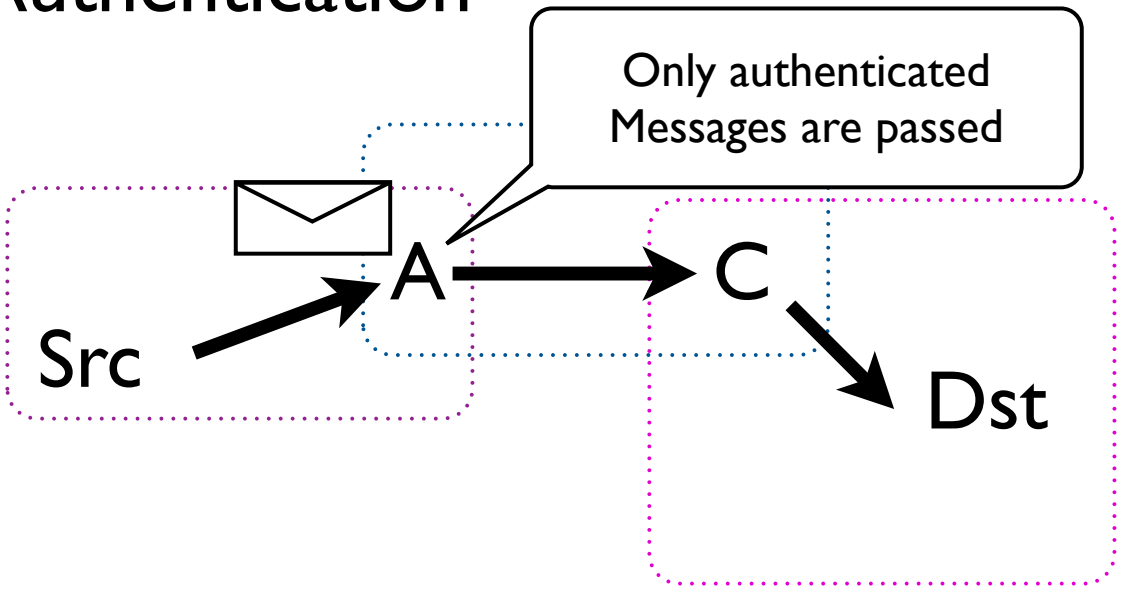

```
when: (src<-getMessage()@withEncryption) becomes:{|result|
  processResult(result);
}
```

Implementation: modifying message send and reception behaviors
(not implemented yet)

Smart Gateway Actors Example cont.

- Authentication

A, B, C, D: Gateway Actors in VMs




```
/* in
when: (src<-getMessage()@withAuth(key)) becomes:{|result|
  processResult(result);
}
```

Implementation: modifying gateway actor's reception behavior
(not implemented yet)

Port Object

- Objects that represent a network interface
- They are used in inter-VM communication

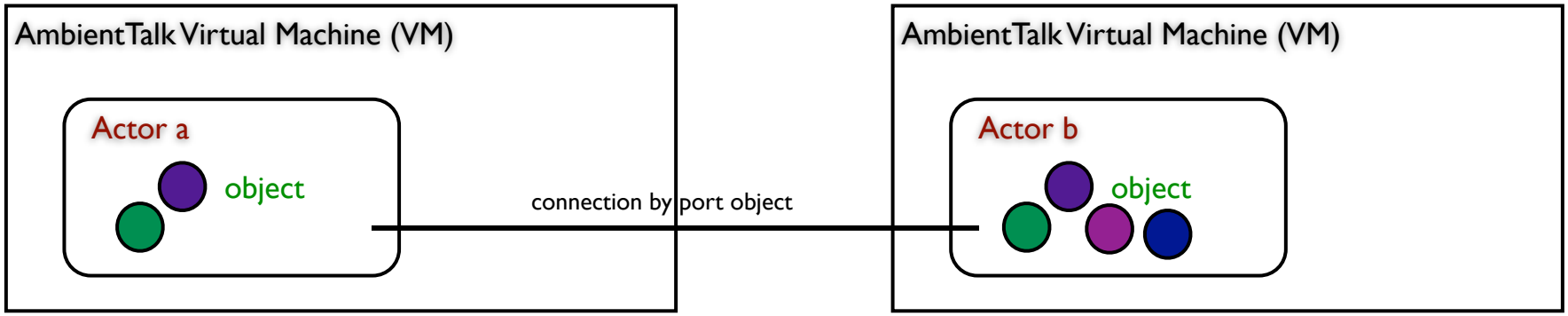
```
~ $ iat -m
Interactive AmbientTalk Shell
>networks.getAll()
>>[<native object: port:133.11.12.176>,
   <native object: port:133.11.12.224>]
```

VM1

```
def a := actor:{
  deftype Test;
  def svc := object:{
    def print(){
      ...
    }
  };
  def n := networks.getAll()[1];
  export: srv as: Test on: n;
}
```

VM2

```
def b:= actor:{
  deftype Test;
  def n := networks.getAll()[1];
  when: Test discovered: {|t|
    t<-print();
  } on: n;
}
```



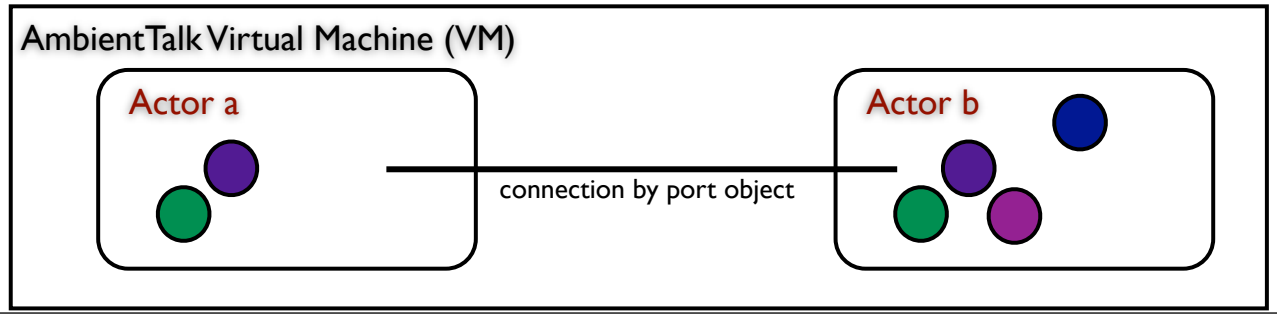
Virtual Port Object

- They are pseudo port objects used to VM-local communication
- Useful for testing gateway actors

```
~ $ iat -m
Interactive AmbientTalk Shell
>networks.createPort("foo")
>><native object: vport:foo>
```

VM1

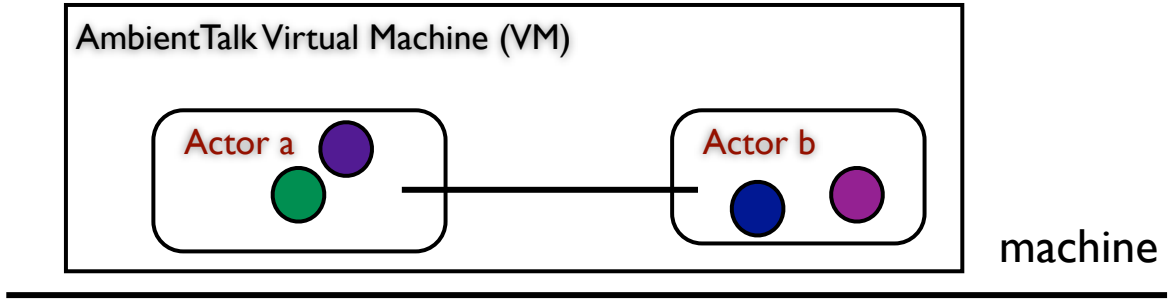
```
def a := actor:{
  deftype Test;
  def svc := object:{
    def print(){
      ...
    }
  };
  def n := networks.createPort("net1");
  export: srv as: Test on: n;
}

def b:= actor:{
  deftype Test;
  def n := networks.createPort("net1");
  when: Test discovered: {|t|
    t<-print();
  } on: n;
}
```



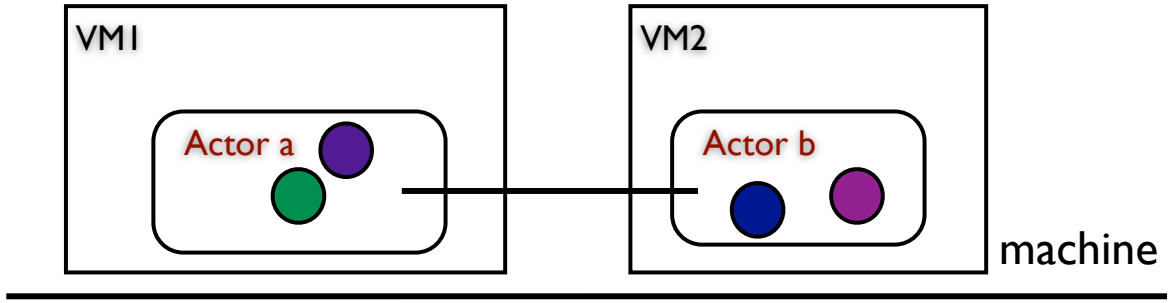
Transparency

Actors' connectivities are decoupled in VM and physical machine

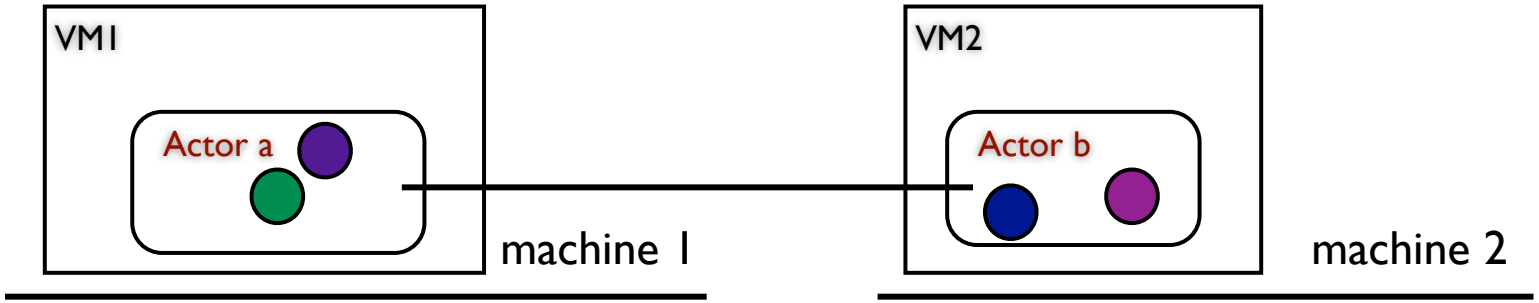
one VM in one Machine



two VMs in one Machine



two VMs in two Machines

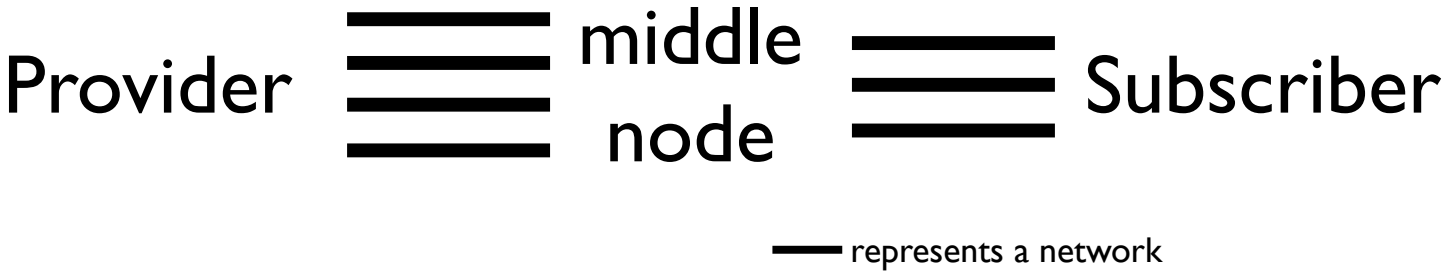


Example MultiPathTest

List. 1 in appendix

- 4 paths from provider to middle node
  and 3 paths from middle node to subscriber
- Results in 12 discoveries at subscriber
- Without virtual ports, it would require 3 machines and lots of network interfaces

概念図

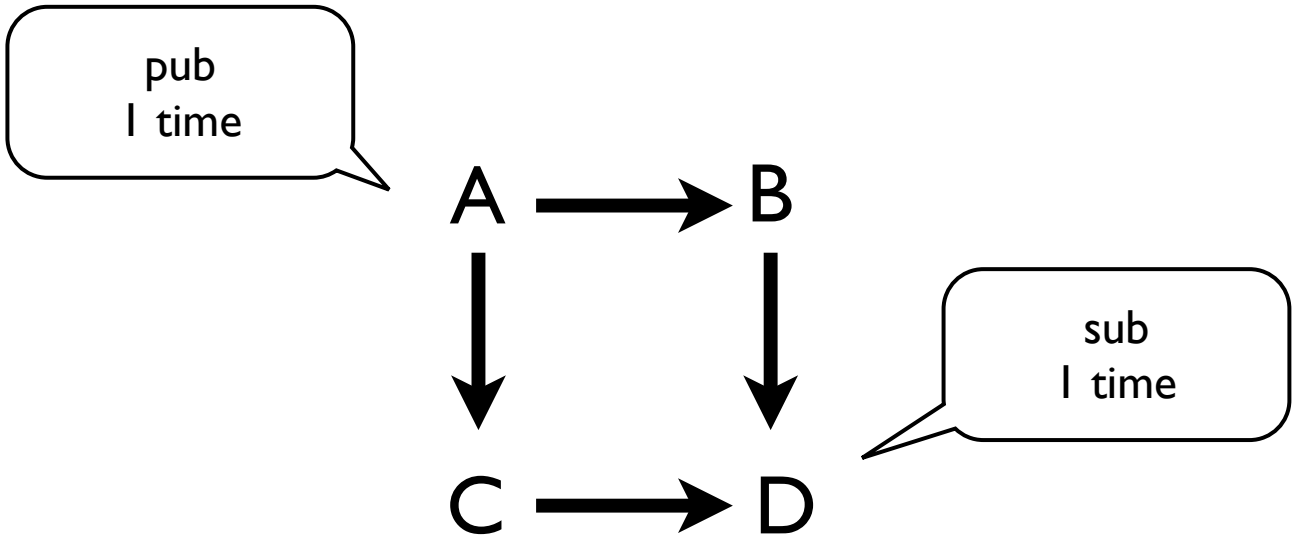


MultiwayReferences

developed by Kevin

- Example

  atlib_multinet/test/actorbasedport/multiwayReference.at



My Work Summary

- Easy programming in multi-networks
- Smart gateway actors
- Virtual Ports

Contribution

Easy Programming in Multi-network Environment

- Using actors as gateways, programmings in several network become easy
- Feature
  - ability to specify which network to use
  - No(little?) need to change existing code

Smart Gateway Actors

- Ability to modify routing behavior using AmbientTalk's reflective feature
- Example
  - Robustness, by Flooding algorithm
  - Less traffic, by selecting optimal path
  - Safety, by encryption or authentication in messaging

Virtual Port Object

- Virtual port objects decouple the gateway actors in physical machines
- Feature
  - Transparency: gateway actors that use virtual port work in real port objects

Pantaxou :

a Domain-Specific Language for Developing Safe Coordination Services

Julien Mercadal, Nicolas Palix, Charles Consel
INRIA / LaBRI, Talence, France

Julia L. Lawall
DIKU, University of Copenhagen, Copenhagen, Denmark

in Generative Programming and Component Engineering (GPCE'08)

Pantaxou

- Domain specific language for coordinating devices in networked environment
- Environment code + implementation code
- Static verification

Motivation

Existing framework, CORBA, is not sufficient

- It does not support in programming language
- It lacks hierarchy in service discovery;
  CORBA uses string in service discovery
- It lacks reliability; no static verification

Sample Application: Forwarding Phone Call

Forward a phone call to the phone in
the room where Alice is currently sitting

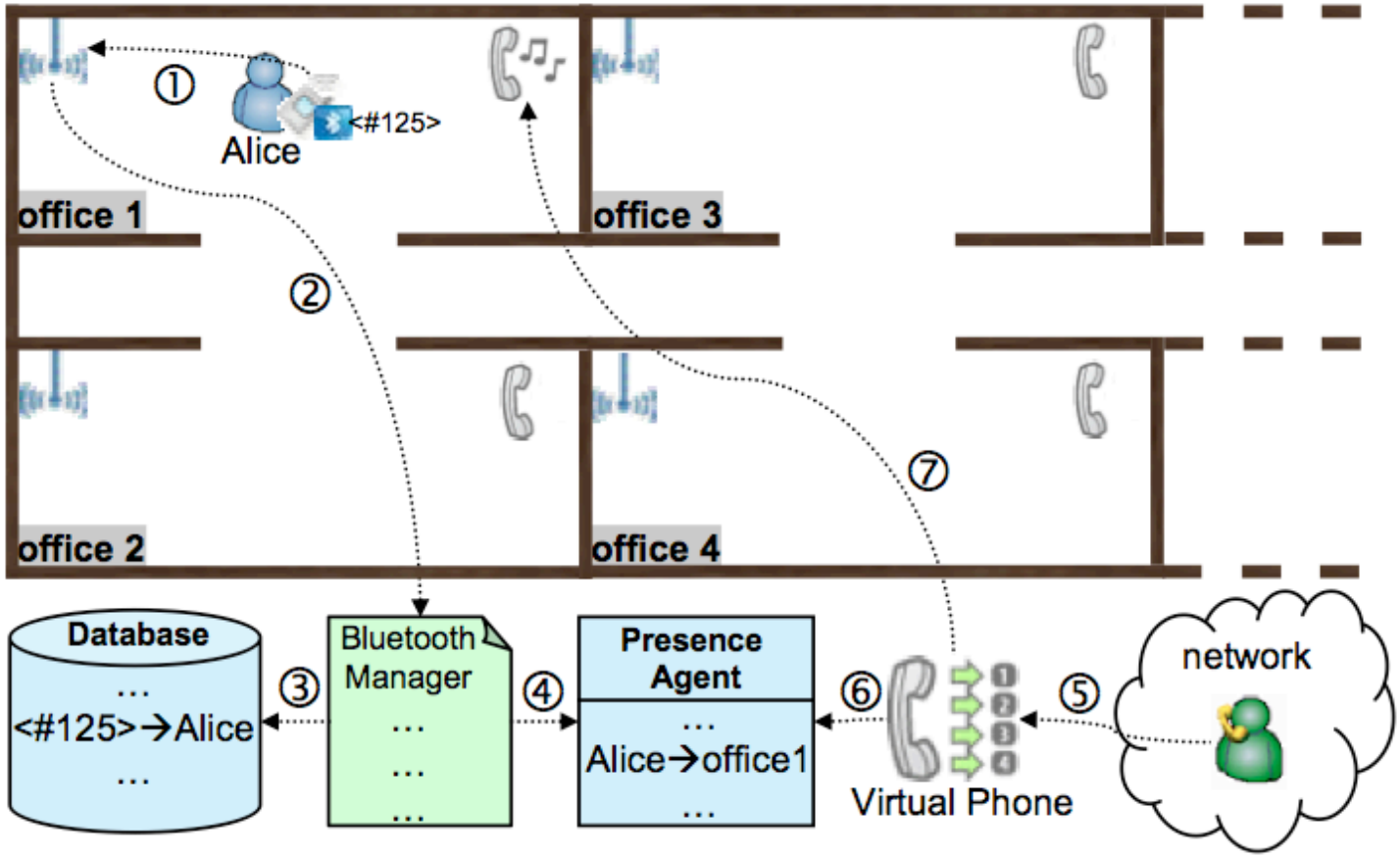


Fig. 8 Fig. 7(a) Fig. 7(b)

How it works

1. The Bluetooth detector detects that a Bluetooth device has just entered.
2. The Bluetooth detector publishes an event to signal the presence of the Bluetooth device. This event is received by the Bluetooth Manager.
3. The Bluetooth Manager queries the database to identify the owner of the tag ⟨#125⟩.
4. The Bluetooth Manager publishes an event to signal that Alice is in office 1. This event is received by the Presence Agent.
5. When Alice is called, an audio session request is received by her Virtual Phone.
6. The Virtual Phone queries the Presence Agent to locate Alice.
7. The Virtual Phone transfers the call to the phone of office 1.

Example code in CORBA(C++)

```
#include <omg/CosEventComm.idl>
...
interface GetRoom {
  room getRoom(in uri user);
};
interface PresencePusher {
  void push_presence(in uri entity, in boolean status,
    in room r);
};
interface PresenceAgent : PresencePusher,
  CosEventComm::PushConsumer, GetRoom { };
interface Phone : Device, MMDevice {
  Phone_A create_A(in Phone_StreamCtrl the_requester,
                   out v_Phone the_vdev,
                   inout streamQoS the_qos,
                   out boolean met_qos,
                   inout string named_vdev,
                   in flowSpec the_spec)
    raises(streamOpFailed, streamOpDenied, notSupported,
           QoSRequestFailed, noSuchFlow);

  Phone_B create_B(...)  // similar to Phone_A
    raises(streamOpFailed, streamOpDenied, notSupported,
           QoSRequestFailed, noSuchFlow);

  Phone_StreamCtrl bind(in Phone peer_device,
                        inout streamQoS the_qos,
                        out boolean is_met,
                        in flowSpec the_spec)
    raises (streamOpFailed, noSuchFlow, QoSRequestFailed);

  Phone_StreamCtrl bind_mcast(...)  // similar to bind
    raises (streamOpFailed, noSuchFlow, QoSRequestFailed);

  string add_fdev(in F_Audio the_fdev)
    raises(notSupported, streamOpFailed);
};
interface v_Phone : VDev { ... };
interface Phone_StreamCtrl : StreamCtrl { ... };
interface Phone_A : StreamEndPoint_A { ... };
interface Phone_B : StreamEndPoint_B {};

interface Audio_Producer : FlowProducer {};
interface Audio_Consumer : FlowConsumer { ... };
interface F_Audio : FDev { ... };
interface Audio_Connection : FlowConnection { ... };
 ...
```

Figure 2. Fragments of a CORBA-based interface declarations of the secretary scenario

CORBA provides only *commands* in their interaction

while Pantaxou provides event, commands and session

```
import ...
class PresenceAgent extends PresenceAgentPOA {
  ...
  public PresenceAgent(...) {
    ...
    ORB orb = ORB.init(...);
    Object obj = orb.resolve_initial_references("EventService");
    TypedEventChannelFactory m_factory = TypedEventChannelFactoryHelper.narrow(obj);
    IntHolder id = new IntHolder();
    TypedEventChannel tec = m_factory.create_typed_channel("TypedChannel", id);
    org.omg.CosTypedEventChannelAdmin.TypedConsumerAdmin tca = tec.for_consumers();
    org.omg.CosEventChannelAdmin.ProxyPushSupplier pps = tca.obtain_typed_push_supplier("IDL:Pr
    pps.connect_push_consumer(this);
  }
  Room getRoom(Uri user) { ... }
  void disconnect_push_consumer() { }
  void push_presence(Uri entity, boolean status, Room r) {
    ... // communication logic
  }
  void push(org.omg.CORBA.Any a) { }
}
```

Figure 3. Fragments of a CORBA-based implementation of the secretary scenario (PresenceAgent class has been omitted.

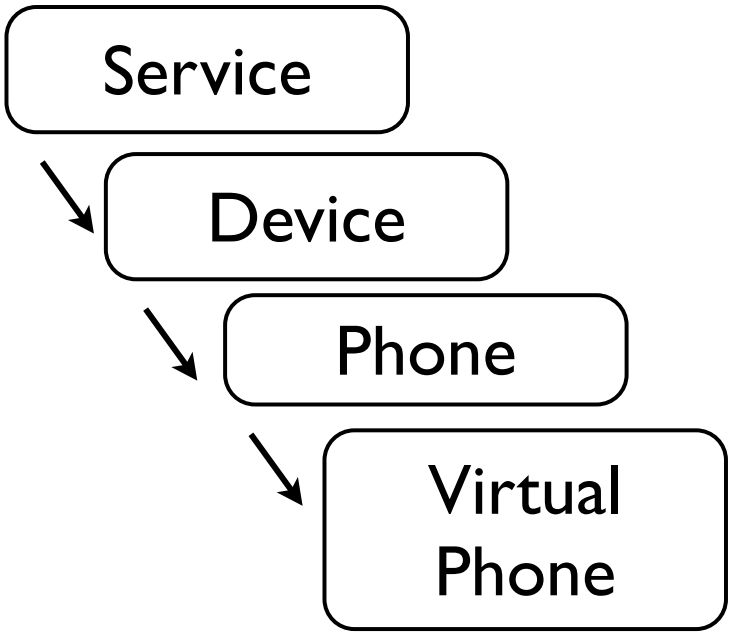
Pantaxou Sample Code

See appendix

- Environment Code (Fig. 6)
- Service Implementation Code (Fig. 7)

Service Hierarchy

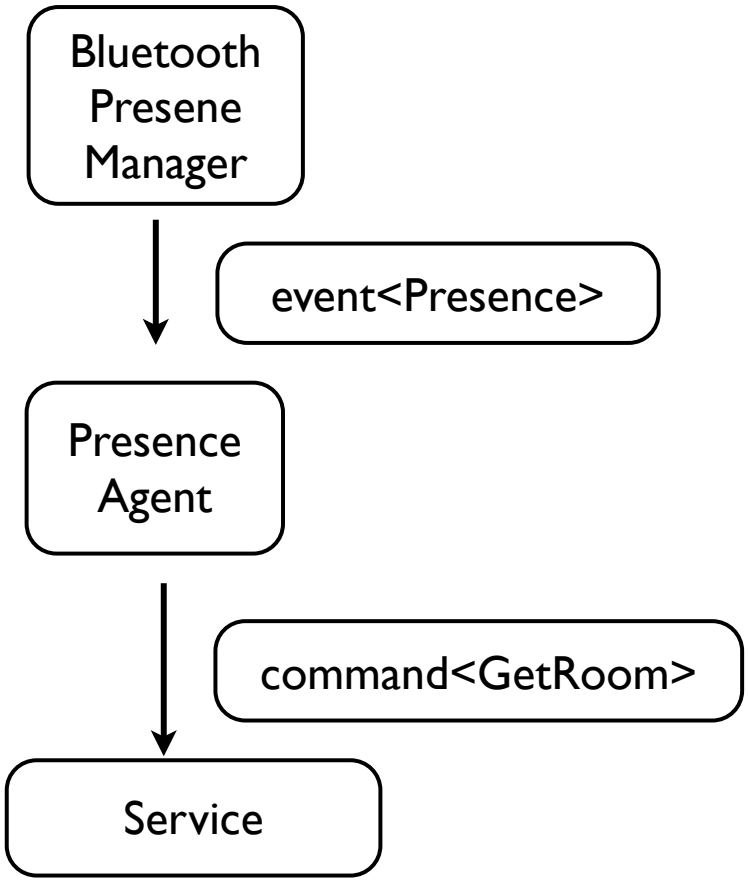
- Service discovery based on hierarchy (ontology)
  - Similar to AmbientTalk's type tag




```
(略)

service Device extends Device {
 Room room;
 ...
}
service Phone extends Devoce{
 ...
}
(略)
```

Environment Description

- Description of entity's relation (dependency)




```
(略)
service PresenceAgent extends Service {
  requires event<Presence>
    from BluetoothPresenceManager;
  provide command<GetRoom> to Service;
}
(略)
```

Service/Event Discovery

```
/* Fig.7(a) line 9 */
service<BluetoothPresenceManager> { } bm
event<Presence> from bm[*] { } presenceEvt;
```

- BluetoothPresenceManagerを見つける
    - Service Discoveryの結果は配列
- “bm[*]”はどのBluetoothPresenceManagerからのPresenceイベントも受け取るということ
- “{}”内はpropertyへの条件(次スライド)

Service/Event Discovery

```
/* Fig.7 line 19 */
service<Phone> {
   room = agent.getRoom('sip:me@enseirb.fr');
} myPhone;
```

- 特定のroomにあるPhoneを探す
    - PhoneはDeviceを継承しているのでroom propertyを持つ
- VirtualPhoneがPresenceAgentのgetRoomを呼ぶことはenvironment descriptionで記述されている(Fig.6)
- getRoomの引数はユーザ名を表すURI

Concurrency?

```
/* Fig.7(a) line 17 */
if (p.status) {
  htable.put(p.entity, p.room);
} else{
...
```

- Concurrencyへの言及はなし

  race conditionを防ぐためにはhtableにlockが必要では?

  frameworkが勝手にやってくれる?

Static Verification

- Require and provide relation
  - Any required event, command or session must be provided
  - Any provided event, command or session must be required

No check against changes

- At the deployment of a service, the framework checks real environment for required services
  - the dependency is in environment description
  - if the check fails, human operator may be notified

Compilation Flow

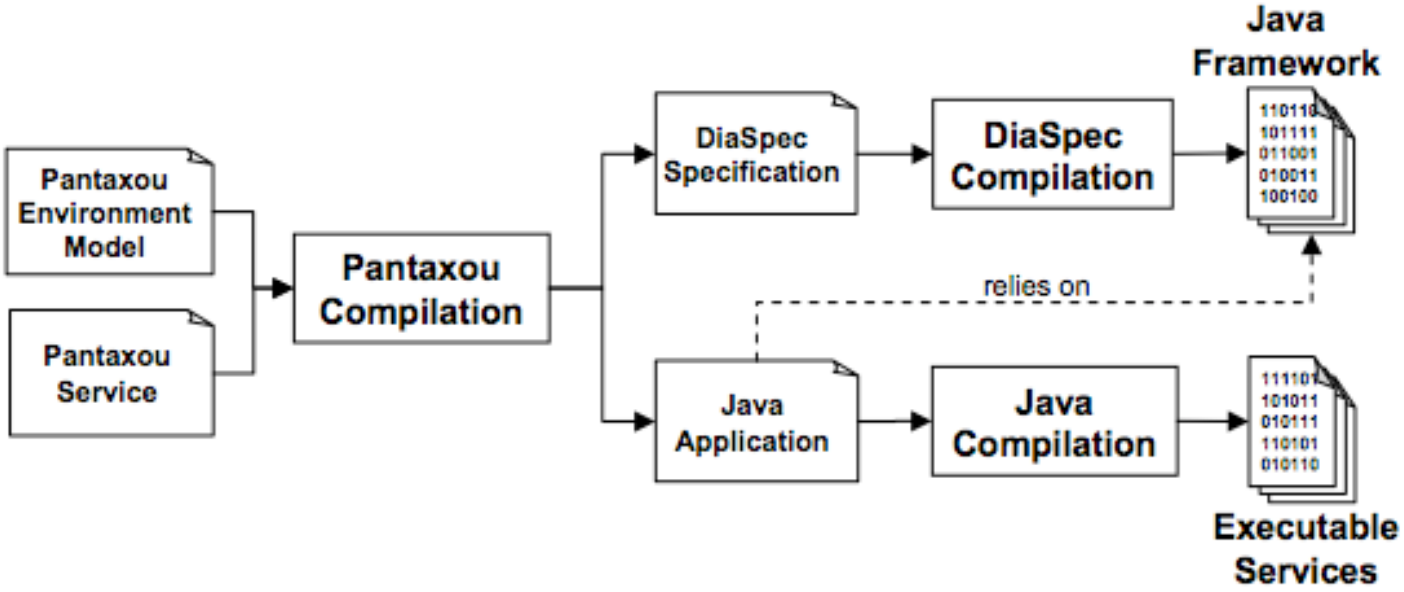


Figure 9. Complete compilation process

DiaSpec(developed in INRIA)

DiaSpec helps developers writing pervasive applications by generating dedicated programming frameworks from software architectures specified in an architecture description language, also named DiaSpec.

https://diaspec.bordeaux.inria.fr/

Evaluation

- Participants: graduate students
- They implemented call-forwarding program in both Pantaxou and Java
- Java's code is 5 times longer than that of Pantaxou
- They said Pantaxou is great because:
    - conciseness, high-level abstraction
    - static verification
    - service discovery

Pantaxou Summary

- Domain specific language for coordination in (stable) networked environment
- Service / Event discovery
- Environment description for entity's dependency relation
- Static Verification for the relation

おまけ

- DiaSim [Bruneau et.al., MobiQuitous 2009]
  A Parameterized Simulator for Pervasive Computing Applications
  https://diasim.bordeaux.inria.fr/