employ
① new kinds of actors
② ... new roles for actors

Programming Language Support for Routing in Pervasive Networks

Tomohiro Suzuki*, Kevin Pinte†, Tom Van Cutsem†, Wolfgang De Meuter† and Akinori Yonezawa*
* The University of Tokyo, Tokyo, Japan
Email: {tsuzuki, yonezawa}@yl.is.s.u-tokyo.ac.jp
† Software Languages Lab, Vrije Universiteit Brussel, Brussels, Belgium
Email: {kpinte, tvcutsem, wdmeuter}@vub.ac.be

attributes ? characteristics

Abstract—**Managing communication in pervasive environments is a difficult challenge because of characteristics such as: no central server and frequent disconnections. Furthermore, services to be composed for coordination are sometimes distributed in multiple networks. In that case, intermediate nodes linking these networks have to route communication data toward appropriate destinations. However, incorporating such routing protocols into applications significantly increases complexity of the code. In this paper we propose AmbientTalk/M, a concurrent distributed programming language for coordination of services among multiple networks. The language provides powerful support for creating routing frameworks to hook up two or more networks. With the language support, we can express how service information is propagated or how messages are routed using high-level abstraction over underlying network technology. We show the language is flexible enough to express a variety of routing frameworks with respect to robustness, traffic efficiency, and security. Because the frameworks are installed reflectively, they are completely separated from application code. To the best of our knowledge, this is the first attempt to integrate routing semantics among multiple networks into a programming language using reflection.**

relevant?

I. Introduction

Today, pervasive environments are common; many applications in mobile devices, including cellular phones, PDAs and laptop computers, are capable to communicate with each other. Sometimes they are directly connected through their own ad hoc networks, e.g. 802.11 or Bluetooth network, without requiring no central managing server. When there is no fixed access point, this kind of spontaneous communication is necessary to connect devices in a place. We call such networks *mobile ad hoc networks* or MANETs.

~~However,~~ coordination in MANETs is difficult, especially when services that are required to compose coordination are scattered in different ad hoc networks. In the first place, the mobility of the portable devices poses significant requirements for middleware or runtimes [1]. Furthermore, it ~~is~~ often that there are many separated networks in the same location; we often have a number of 802.11 wireless networks, Bluetooth networks and cabled Ethernet networks in university classroom. A service, e.g. printing service, held in a certain network may not be available in another one. To coordinate entities in such environment, we have to exchange service information and route data through intermediate nodes, if any, among them. Incorporating these routing semantics into applications causes a significant increase in complexity. Although a number of research activities for middleware [2], [3], [4], [5] , as well as new communication protocols [6], [7], have been developed to achieve better coordination, almost all these models assume that all entities are on a single network.

occurs

In this paper, we present AmbientTalk/M, a distributed concurrent programming language for coordination in multiple networks. From its ancestor AmbientTalk [8], it inherits service discovery based on publish/subscribe communication and messaging communication via asynchronous messages between *actors* [9]. These ~~attributes~~ make the language suitable for programming in MANETs. In addition to that, our new language AmbientTalk/M is able to utilize multiple networks and to coordinate entities among them.

Emph

The language employs new roles of actors located in intermediate devices between a device publishing a service and another one subscribing to the service. The roles of the actors are to propagate service information and transfer messages among them. The actors help to ~~create~~ *routing frameworks*, which ~~manage~~ how messages are routed in networks, and applications without routing semantics work on the frameworks. The frameworks in AmbientTalk/M have the following ~~attributes~~: those

Highly abstracted in that the routing is written as message passing and receiving behavior of AmbientTalk/M actors. Programmers think which services are available in their networks and what messages are to be sent to them, rather than thinking IP addresses of services. The routing frameworks are written in high-level operations in AmbientTalk/M, compared to manipulating binary data of some networking protocols like [10], [11].

the

Transparent in that they do not conflict with existing service discovery mechanism described in Section II; applications written for a single network will also work well in multiple networks in AmbientTalk/M. In other words, the routing frameworks are backward compatible with applications for a single network. Because they are installed *reflectively* into actors, they work independently from the application code and thus avoid unnecessary code complexity. adaptations.

Expressive in that AmbientTalk/M can describe a variety of routing frameworks required for coordination ~~in~~ across multiple networks. To show the expressiveness, this paper presents several applications of frameworks for specific tasks in Section III, including a printing service accessed from different kinds of networks, a chatting service using multihop messaging and message encryption in intermediate nodes.

Moreover. properties

about.

Far references are a resilient remote object designation technique. and thus

II. AMBIENTTALK/M

AmbientTalk/M is a programming language for coordination of services in two or more networks. Before erabolating on the language, we start with AmbientTalk [8], the ancestor of AmbientTalk/M.

A. AmbientTalk

AmbientTalk has been developed as an event-driven programming language for a single mobile ad hoc network. The language is dynamically-typed and object-oriented. As its virtual machine is written in Java, it is portable to many platforms where the JVM runs, such as Google Android [12]. ~~Note that as~~ AmbientTalk has a. powerful symbiosis between AmbientTalk objects and Java objects [13]. ~~For example~~, ~~we~~ W can write ~~networking communication of~~ an application partly in AmbientTalk/M and ~~other parts~~ in Java. ~~partially~~ partly

B. Actors

One of the ~~biggest~~ distinctive. features of AmbientTalk is actor-based communication. Actors are concurrent computational agents [9], each of which has its own message queue and independently and sequentially processes its messages. In AmbientTalk, distributed communication among devices is expressed as passing messages among actors. When an actor is to perform an action of an object residing in another actor, it sends a message to the actor. Then the message is added to the message queue of the actor, which sequentially processes messages from its message queue, usually resulting a method invocation of the object corresponding the message. These attributes of actors bring benefits to concurrency in that the actors do not cause race conditions or deadlocks against shared objects. We refer to [ ] for more details.

C. Service Discovery and Temporary Disconnection

Since mobile ad hoc networks cannot rely on any central server to provide the locations of services [2], [14], AmbientTalk employs the following procedure to discover services available in a network before exchanging messages. Suppose we have two devices: a device called Device A hosts a printing service and the other named Device B is to use the service by sending a message to the printing service.

Listing 1 and 2 depict typical communication in AmbientTalk. First an actor creates an object that implements necessary method for the service (line 2-6 in Listing 1) and publishes it as services to other AmbientTalk virtual machines in the network (line 7). A publication always carries a *type tag*, e.g. `Printer`. A type tag is ~~all about~~ service ~~information~~ in AmbientTalk. When another actor starts to subscribe to the service specifying a type tag (line 2 in Listing 2), the virtual machine ~~underneath~~ searches for services corresponding to the type tag. If the type tags match ~~to each other~~ the subscribing actor *discovers* the service, meaning that it gets a *far reference* to the service. A far reference is a stub for the service and any message sent to the far reference is sent to the actor holding the real service entity. Finally, through the far reference, the actor can send~~s~~ messages to the service (line 3 in Listing 2). The <– operator is a language construct for sending a message via a far reference. The message is sent to the recipient actor, which invokes the implemented method `print` of the object for the service. This procedure is abstract~~ed in that it~~ is simpler than that of focusing on low-level socket manipulations and other system calls.

~~Note that~~ When the virtual machines are physically disconnected, ~~the~~ messages are implicitly buffered in the interpreter and ~~they~~ are ~~sent again~~ flushed. when the peers are reconnected. This makes AmbientTalk suitable for MANETs where temporary failures are the norm rather than exception.

Listing 1. Service Publication in AmbientTalk

```
1 deftype Printer;
2 def obj := object: {
3   def print(s) {
4     ...
5   };
6 };
7 export: obj as: Printer;
```

Listing 2. Service Subscription in AmbientTalk

```
1 deftype Printer;
2 when: Printer discovered: {|p|
3   p<-print(somedoc);
4 }
```

D. Problem with Multiple Networks

Although AmbientTalk is suitable ~~for~~ a single mobile ad hoc network, it does not support programming in a multi-networked environment; an AmbientTalk virtual machine joins at most one TCP/IP network. However, it is often the case that there are a number of distributed entities to compose ~~coordination and not~~ without all of the entities belonging to the same network. In that case the intermediate nodes, if any, have to act as routers that transfer data from a network to another. This requires a means to specify which network is used for publishing service information or subscribing it. ~~Note that as~~ Since the routing ~~tactics~~ strategy ~~would~~ may vary ~~according to~~ depending on the applications and the status of the network environment, a system with flexibility of adopting various routing algorithms is ideal. AmbientTalk/M has been developed for that purpose.

E. AmbientTalk/M

AmbientTalk/M is capable of utilizing multiple networks and. ~~in~~ making coordination among services distributed in ~~the~~ those networks. The capability comes from *port objects* and ~~new functions on them.~~ the methods defined on them.

Listing 3 and 4 write the previous coordination in AmbientTalk/M. First, in addition to the type tags, each actor retrieves a port object, the first element of the return value of `getAllPorts` method (line 2 in the both listings). After defining an object for the printing service, it is published ~~by~~ using `export: as: on:` function (line 9). What is interesting about the code is its third argument; because AmbientTalk/M uses multiple networks, it is able to specify which network the service is to be published on by using a port object that represents a network interface, e.g. *eth0*.

On the remote side, the other actor of Listing 4 subscribes to the service. When the service is discovered by the actor,

identification used for the
underlying ~~logically~~

discovered.

as will be.

residing in the receiving actor

it invokes the closure of the second parameter of `when: discovered: on:` function (line 3-5). In the closure, the actor gets a far reference as the variable `p`. After getting the far reference, it sends a message `print` with its string parameter (line 4). The third parameter is a port object used to specify which network the service is to be ~~subscri~~bed from. Both `export: as: on:` and `when: discovered: on:` are new functions in AmbientTalk/M with the capability to specify which network to use in their communication.

C.R.

Listing 3. Service Publication in AmbientTalk/M

```
1 deftype Printer;
2 def port := networks.getAllPorts()[1];
3 def obj := object: {
4   def print(s) {
5     ...
6   };
7 };
8 export: obj as: Printer on: port;
```

bold/non-bold ??

the

Listing 4. Service Subscription in AmbientTalk/M

```
1 deftype Printer;
2 def port := networks.getAllPorts()[1];
3 when: Printer discovered: {|p|
4   p<-print("Hello, World");
5 } on: port;
```

scope

F. Port Objects

The third argument of the new functions is a port object (`port` in Listing 3 and 4). This is a new kind of object in AmbientTalk/M representing a physical network interface of the device. From an actor's point of view, it represents a *network*, which is used to specify the ~~direc~~tion of the publication and subscription. If the device, for example, has three network interfaces, the `getAllPorts` method returns an array of three objects that represent these interfaces respectively. The second line of Listing 3 retrieves the first element of the array. As in the previous example of Listing 3 and 4, these objects are used in publication and subscription of services ~~to~~ specify which network to export the service to and which network to ~~subscribe~~ to the service from. The `networks` objects ~~have~~ the control of port objects in the actors. Through the objects we can retrieve all port objects available ~~in~~ the actor.

all

has

to

discover.

G. Actors for Routing

Actors play an important role in coordination in AmbientTalk/M. Developing applications for mobile devices that use a service in another network increases the code complexity because they cannot reach the service directly and thus some routing algorithms are to be implemented in the application. To ~~ease the difficulty~~ AmbientTalk/M provides a means of ~~dealing with implementing~~ routing algorithms. This is done by creating a special actor in an intermediate node which is able to belong to both ~~of the~~ networks. We call the special actors *gateway actors*. A gateway actor acts as a router connected to other networks, through which service information is propagated and messages are transferred from ~~a~~ network to another. Propagating service information means that when a gateway actor discovers a service form one network it publishes a stub for the service to ~~another~~

One

the other

Solve this

hiding the

network. At this point, the gateway actor uses port objects to specify the direction of the propagation. Since gateway actors are implemented by the language's reflective features, ~~their~~ routing algorithms are dynamically customizable to a variety of routing frameworks explained in Section III.

By reflection, we mean the ability to intercept a language behavior from within the language [15], [16], [17]. Normally when a message is created and sent from an actor and ~~then~~ ~~it is~~ received by another actor, this results in invoking the corresponding method of an object. ~~However,~~ programmers can intercept such behaviors from a meta-level point of view using `reflectOnActor` function. AmbientTalk/M provides two methods to override the default behaviors ~~of actors~~: how the message is sent (by overriding `sendTo` method) and how an actor receives the message (by overriding `receive` method). Gateway actors are implemented using reflection; when they receive a message from a sender in a network, they forward the copy of the message toward the actual receiver in another network, instead of calling a method of an object. ~~inside them.~~

subsequently

the

the.

There may be another way to implement messaging frameworks ~~completely inside~~ the virtual machine of AmbientTalk/M in Java. However, it requires converting Java objects to AmbientTalk/M objects and vice versa. ~~Compared to implementing it in virtual machines,~~ It is thus far more concise to manipulate AmbientTalk/M object from within the language, especially when implementing routing algorithms based on AmbientTalk/M objects.

directly in

III. Coordination in AmbientTalk/M

AmbientTalk/M exhibits expressiveness and flexibility for creating a variety of coordination frameworks among different networks. This section shows how the actors in AmbientTalk/M are customized for specific tasks and how their reflective capability plays an important role in ~~such~~ coordination.

the

let us (no informal language!)

A. Messaging among Multiple Networks

Let's start with very simple coordination ~~in~~ two networks. Suppose we are going to print a document using a printer but it belongs to a different network than the network our mobile phone is using. Fig. 1 describes that situation where a printer logically belongs to *Network Y*, and a client to *Network X*. The client ~~has~~ the code in Listing 4 and the printer ~~has~~ the code in Listing 3. A device has two network interfaces to belong to ~~the~~ both networks, having the gateway actor to bridge them.

problem across

runs

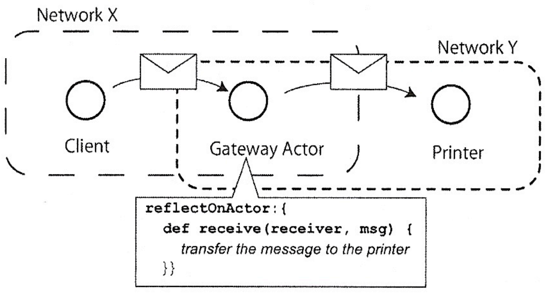
After the printer service information is propagated via the gateway actor, the client sends a message to the printer ~~to~~ the gateway actor. As the gateway actor's reception behavior is modified so that it transfers the message to the printer, the printer in *Network Y* receives the message from *Network X*. This simple communication can be extended to deal with many cases, where devices have to communicate beyond ~~one~~ network.

through

the

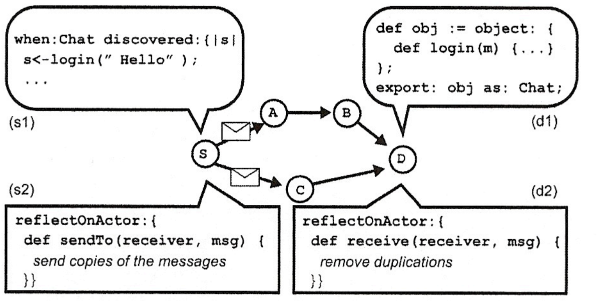
boundaries.

One possible situation is a heterogeneous network environment; *Network X* and *Y* in Fig. 1 can be IEEE 802.11



```
reflectOnActor:{
  def receive(receiver, msg) {
    transfer the message to the printer
  }}
```

Fig. 1. Coordination in Multiple Networks



```
when:Chat discovered:{|s|
 s<-login(" Hello");
 ...
```

```
def obj := object: {
  def login(m) {...}
};
export: obj as: Chat;
```

```
reflectOnActor:{
 def sendTo(receiver, msg) {
  send copies of the messages
 }}
```

```
reflectOnActor:{
 def receive(receiver, msg) {
  remove duplications
 }}
```

Fig. 2. Flooding Framework in Two Available Paths

[illegible]

and Ethernet. Bluetooth [18] is also a possible choice. Since there are already many portable devices that can belong to separated stationary networks and wireless networks, applications written in AmbientTalk/M would bridge the logically separated networks, where *logically separated* means these network form different subnets or networks on different protocols.

cool!

written in

Another target can be a number of networks linked together by gateway actors. Suppose a train of 8 cars and in each car at least one person is seated playing the same chatting application ~~upon~~ AmbientTalk/M. Even if the car is too long to bring persons in the first car and the last car into the same ad hoc network, the network can be logically linked together by ~~their own MANETs, given~~ intermediate devices ~~to do so~~, and ~~all of the~~ devices can exchange messages each other. In that case, gateway actors not only propagate service information one another and transfer messages ~~to all of them~~, but also receive the messages in themselves as normal behaviors for the users ~~playing~~ the application in the intermediate linking devices.

all

running

Note that AmbientTalk/M virtual machines depend packet routing in a single network on its underlying network stack and that routing frameworks in AmbientTalk/M complement the intra-network packet routing with the inter-network message routing.

?
(grammar)

enable

B. Routing Frameworks for Multiple Paths

the

Sometimes networks ~~have~~ two or more paths from source of a message to its destination. Fig. 2 depicts one ~~of~~ such situation~~s~~. A service *Chat* is published in *Actor D*. The actor is connected *Actor B* and *C*, but is separated from *Actor S*, which discovers the chat service through two paths: a path of D, B, A and S, and the other of D, C and S. After the service is discovered by *Actor S* through the two paths, the actor is to send a login message to the service, selecting which path to use according to its routing framework. Routing frameworks, reflectively installed into actors, control how the messages are transferred to the destination. Here, we introduce following two frameworks:

at the

Flooding uses all paths to deliver the messages as in Fig. 2. The message sending in *Actor S* is intercepted ~~in~~ meta-level ~~view~~ as in (s2) and the message is duplicated with the same identity number and these copies are sent to all of the available paths. When the service of *Actor D* receives the duplicated messages, its reception behavior is ~~also~~ intercepted to filter ~~with the~~ message identity, as in (d2), resulting one method invocation for ~~one~~ the original message. This framework has robustness against failures in intermediate nodes. Note that to add identities to services or messages we extend *multiway references* used in AmbientTalk RFID project [19]; a service that uses multiway references is published with its identity and actors recognize the references to the service from two or more paths as one reference based on the identity.

based on
ies
shown
at the meta level
is

Minimum Hop selects a shortest path. In ~~this~~ framework gateway actors propagate service information with its hop counts from its origin in addition to the service identity. This means, when an actor gets a far reference to a service, it also knows how far ~~the origin of~~ the service is. By using the hop count information, a message sender selects a reference which has the minimum hop count. Compared to the flooding framework, this framework has traffic efficiency by avoiding unnecessary duplicated messages flooded in networks.

away

Note that routing frameworks act transparently; the application code, (s1) and (d1) in Fig. 2, are written orthogonally from their routing frameworks, (s2) and (d2). This is the reason why applications written in AmbientTalk ~~[8]~~ for one network can be easily extended ~~for~~ environments with multiple networks, by installing these frameworks into actors. This separation avoids unnecessary code complexity.

the vanilla variant of

to

of concerns

C. Opportunistic Networks

Opportunistic networking [20] consists of mobile devices, some of which are frequently out of range from a network but are in the range of other networked devices. The network opportunistically uses the intermediate nodes' mobility to deliver data from one network and another as they move across networks. Suppose that, in Fig. 1, *Network X* and *Y* are physically separated and thus the intermediate node hosting the gateway actor cannot belong to both ~~of the~~ networks at the same time. If the intermediate node moves between the two networks and delivers necessary information among them, this situation is one example of opportunistic networks.

AmbientTalk/M can be easily used in such networks because the language regards moves of portable devices ~~from~~ one network to another one as temporary disconnections in MANETs. The messages which are sent during the period when far references are ~~not connected~~ ~~to~~ their senders are

disc from

(a property it inherits from plain AmbientTalk)

shown

graph

after/while

Use the word "relay" if you mean transfer / copy

~~that~~ requiring special attention.

implicitly buffered ~~in~~ by the virtual machine of the device and then automatically delivered to the destination when it is reconnected to the network of the destination as the device moves. This abstraction means that applications written in AmbientTalk/M work normally in opportunistic networks without ~~otherwise serious~~ cares for the mobility and the disconnections.

Note that, of course, AmbientTalk/M provides means of being notified when ~~the destination is~~ a communication partner. disconnected and of retrieving the buffered messages from the corresponding. virtual machines. They are necessary when we implement routing frameworks to select paths that are currently "alive."

D. Encryption in Routing

AmbientTalk/M is applicable to secure messaging. Sometimes we cannot trust intermediate nodes and do not want them to look inside our messages. In that case we can adopt the ~~following~~ encryption framework ~~as~~ shown in Listing 5 and 6 in the communicating endpoints and insert some additional annotations in the application code to modify messaging behavior.

When a service is discovered in the client of the service at the beginning of Listing 5, the client sends a `login` message with a secret password, which the client does not want to be seen in intermediate nodes. In this encryption framework, the client uses ~~a tag~~ the annotation `Encryption` (line 1) when sending a secret message, which triggers an additional encryption procedure (line 6-7) just before the message is sent. This encryption algorithm can be simple XOR encryption based on their pre-shared key. If so, only nodes which have the pre-shared key can decrypt the contents of the message. On the other hand, the receiving actor decrypts the message with their pre-shared key (line 4-6 in Listing 6) if the message is tagged as `Encrypted`; otherwise, the overriding function uses its default behavior (line 7). As all encryption and decryption procedures are performed in reflective parts of the code, the ~~additional~~ cost in the application code to act on this encryption framework ~~is~~ consists of. only the annotation.

Note that in Listing 5, we assume that some functions, e.g. `decrypt` and `encrypt`, are defined somewhere and, though we mention XOR encryption the example above, more sophisticated encryption algorithms such as asymmetric key cryptosystem can be installed into the. actors using reflection.

software engineering

Listing 5. Message Sending in Encryption Framework

```
1  service<-login(password)@Encryption
2  ...
3  reflectOnActor: {
4    sendTo(receiver, msg) {
5      if: (is:msg Tagged:Encryption) then: {
6         def key := getPreSharedKey();
7         sendTo(receiver, encrypt(msg, key));
8      }
9      super^sendTo(receiver, msg);
10     ...
```

Listing 6. Message Reception in Encryption Framework

```
1  reflectOnActor: {
2    receive(receiver, msg) {
3      if: (is:msg Tagged:Encrypted) then: {
4        def key := getPreSharedKey();
5        receive(receiver, decrypt(msg, key));
6      }
7      super^receive(receiver, msg);
8      ...
```

IV. Related Work

To achieve coordination among devices, a number of approaches have been presented. They can be classified as communication protocols, middleware, and languages. Communication protocols, such as JINI [6], or UPnP [7] are standards to connect devices in a easy and systemized ~~way~~ a. They step forward from conventional stable networks toward more dynamic ones, where clients dynamically discover services ~~of their interest~~ in an "plug-and-play" manner. As UPnP is supposed to work in stable network with plenty of computation resources, it requires each service to have an HTTP server to send their XML description. This characteristic makes the standard not suitable for being used in small portable devices that are common in MANETs. In contrast, JINI's *proxy* can play a substitute role for a service when the device of the service does not have enough machine power to hold a service.

There are packet routing protocols, such as AODV [10] and TORA [11], to transfer packets in dynamically changing ad hoc networks. There is generally a trade-off between them and high-level abstractions; these low-layer protocols ~~is~~ are relatively fast and can be used to support other systems than a specific programming language, while high-level abstraction presented in this paper allows programmers ~~to~~ a. adjust routing frameworks to their requirements in ~~the~~ high-level programming language. Evaluation ~~of~~ our approach efficiency remains future work. Sensor networks [21] are also active fields, where applications using them are being developed for surveillance and health monitoring. Compared to mobile ad hoc networks, they deploy ~~much more number~~ a much higher number of embedded sensor nodes and their focuses are also on relatively low layer, such as energy efficiency in digital circuit design, rather than programming support.

LIME [3] and TOTA [4] are middleware of tuple spaces for mobile environments. LIME's tuple spaces are processed by merging two tuple spaces when two nodes are close to each was other. ~~while~~ In TOTA, a tuple itself moves across ~~tuple~~ spaces according its own propagation rule. ~~By doing so~~, a host can send ~~tuples~~ to ~~other~~ another hosts that ~~are~~ was not initially connected to it. EMMA [5], middleware for epidemic messaging in ad hoc networks, also considers sending a message to a non-connected host as normal~~, which case is often in ad hoc networks~~. The middleware's *epidemic protocol* enables hosts to transfer a message to other ad hoc networks which do not have the message yet.

There are also object-oriented middlewares whose message transmission is performed in method invocation of objects. JINI [6], ~~though it is~~ (mentioned in communication protocols), and CORBA [22] fall into this category. JINI's proxy and Java's code mobility enable services to evolve implicitly without notifying its change to its clients. CORBA is a widely accepted middleware for its interface-based communication that hides details of service implementation and its implicit data conversion that occurs when a service receives a message.

As high-level programming languages, Pantaxou [23] pro-

I don't understand this sentence.

not English! to iterate over devices?

vides programming support~~s~~ especial~~ly~~ in audio and its verification of coordination consistency will help programming in distributed environment. SpatialViews [2] provides a simple way to iterate devices placed in a room. Each iteration ~~of loops~~ step of the is implicitly executed in each device to collect information in the space. YABS [24] is a programming language based on the behavior of social insects. Although the language needs no global knowledge, individual reactions against context change are enough powerful to produce robustness for changing environment. In session-based programming [25], the ability of passing a session through another session supported by underlying middleware is noteworthy abstraction. a.

As to reflection, DIL [26] employs techniques of reflection upon actors. The language provides simple yet expressive controls of actors' sending and receiving behaviors, enabling them to build up backup systems and distributed snapshots. Java RMI [27] uses reflection to translate calls on special references into remote calls to the server, keeping Java's security policy, compatibility with legacy code and object-oriented style. While these approaches also have achieved the separation between application and other semantics, it assumes a stable network~~s~~ to deploy the system rather than multiple MANETs; they have no abstraction over temporary disconnections and thus routing among several networks ~~are~~ is out of their scope.

? Note that almost all of high-level approaches for mobile ad hoc networks assume that the services or hosts to be composed are placed in a single network or they are not expressive enough to describe routing semantics in the system itself.

V. Conclusion

It is difficult to achieve coordination in pervasive environments where communicating entities are scattered over in ~~different kinds of~~ various networks. This paper presents AmbientTalk/M, a distributed concurrent programming language for coordination among devices in multiple networks. In addition to its inherited abstraction over difficult characteristics of MANETs, the language has the capability of composing entities distributed in several networks. Applications in AmbientTalk/M utilize intermediate devices ~~that are able to belong to the separated networks~~ other by installing routing frameworks into the gateway actors hosted ~~in the~~ devices. The special actors are responsible to propagate service information and to route messages to the services. Among many researches on MANETs, we say AmbientTalk/M distinguishes itself because it has powerful abstractions to cope with multiple mobile ad hoc networks and has the expressive functions to create a variety of routing frameworks with respect to robustness, traffic efficiency and security. As the language offers a means of overriding the actor's behaviors from within the language, it avoids ~~causing~~ unnecessary code complexity. Although an evaluation on the efficiency remains future work, we are ~~sure~~ that our programming model for network routing will bring benefits to the field of programming in pervasive environments.

residing

it

by those devices / by them.

confident.

at the application level.

References

[1] J. Collins and R. Bagrodia, "Programming in mobile ad hoc networks," *ACM International Conference Proceeding Series*, 2008.

[2] Y. Ni, *Programming ad-hoc networks*. Rutgers University New Brunswick, NJ, USA, 2006.

[3] G. Picco, A. Murphy, and G.-C. Roman, "LIME: Linda meets mobility," *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pp. 368–377.

[4] M. Mamei, F. Zambonelli, and L. Leonardi, *Tuples on the air: a middleware for context-aware computing in dynamic networks*. IEEE.

[5] M. Musolesi, C. Mascolo, and S. Hailes, "EMMA: Epidemic Messaging Middleware for Ad hoc networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 28–36, Aug. 2005.

[6] J. Waldo, "The Jini architecture for network-centric computing," *Communications of the ACM*, vol. 42, no. 7, p. 76, 1999.

[7] "Universal plug and play (upnp)," http://upnp.org.

[8] T. V. Cutsem, S. Mostinckx, E. G. Boix, J. Dedecker, and W. D. Meuter, "AmbientTalk:Object-oriented Event-driven Programming in Mobile Ad hoc Networks," *SCCC*, pp. 3–12, 2007.

[9] G. Agha, *Actors: A Model of Concurrent Computation*. MIT Press, 1986.

[10] C. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*. IEEE, 1999, pp. 90–100.

[11] V. Park and M. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2002, pp. 1405–1413.

[12] "Google android," http://www.android.com.

[13] T. V. Cutsem, S. Mostinckx, and W. D. Meuter, "Linguistic symbiosis between event loop actors and threads," *Computer Languages, Systems and Structures*, vol. 35, no. 1, pp. 80–98, 2009.

[14] R. E. McGrath, "Discovery and its discontents: Discovery protocols for ubiquitous computing," *University of Illinois at Urbana-Champaign, Champaign, IL*, 2000.

[15] B. C. Smith, "Reflection and semantics in LISP," *Annual Symposium on Principles of Programming Languages*, 1984.

[16] P. Maes, "Concepts and experiments in computational reflection," in *Conference proceedings on Object-oriented programming systems, languages and applications*. ACM, 1987, pp. 147–155.

[17] S. Mostinckx, T. Van Cutsem, S. Timbermont, E. Gonzalez Boix, E. Tanter, and W. De Meuter, "Mirror-based reflection in AmbientTalk," *Software: Practice and Experience*, pp. n/a–n/a, 2008.

[18] "The Official Bluetooth Technology Info Site."

[19] A. Lombide Carreton, K. Pinte, and W. De Meuter, "Distributed Object-Oriented Programming with RFID Technology," in *Distributed Applications and Interoperable Systems*. Springer, 2010, pp. 56–69.

[20] B. Garbinato, H. Miranda, L. Rodrigues, M. Conti, J. Crowcroft, S. Giordano, P. Hui, H. A. Nguyen, and A. Passarella, *Middleware for Network Eccentric and Mobile Applications*, B. Garbinato, H. Miranda, and L. Rodrigues, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[21] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE Personal Communications*, vol. 7, no. 5, pp. 16–27, 2000.

[22] S. Vinoski, "CORBA: integrating diverse applications within distributed heterogeneous environments," *IEEE Communications Magazine*, vol. 35, no. 2, pp. 46–55, 1997.

[23] J. Mercadal, N. Palix, C. Consel, and J. L. Lawall, "Pantaxou: a domain-specific language for developing safe coordination services," *Generative Programming And Component Engineering*, pp. 149–160, 2008.

[24] P. Barron and V. Cahill, "YABS:: a domain-specific language for pervasive computing based on stigmergy," *Generative Programming And Component Engineering*, p. 285, 2006.

[25] J. Vitek, R. Hu, N. Yoshida, and K. Honda, *Session-Based Distributed Programming in Java*, ecoop 2008 ed., ser. Lecture Notes in Computer Science, J. Vitek, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 5142.

[26] D. Sturman and G. Agha, *A protocol description language for customizing failure semantics*. IEEE Comput. Soc. Press, 1994.

[27] "Java remote method invocation specification," http://download.oracle.com/javase/7/docs/technotes/guides/rmi/.