



Secure P2P programming on top of tuple spaces

WETICE/CoMA 2008

Lange, Benigni, Brogi, Buchholz, Jacquet, Popescu.

Faculty of Computer Science - FUNDP

June, 2008



Outline

- 1 Introduction
- 2 SMEPP
- 3 SecureLime
- 4 Implementation
- 5 Conclusion



Introduction

- P2P systems more and more used
- ... in a wide range of applications

P2P in MANETs raises new challenges due to:

- Heterogeneity
- Lack of reliability
- Security issues
- Transient connection



Introduction

Middlewares have been developed in order to tackle this issues. One of such middleware is developed in the SMEPP Project.

- **Secure**
- **Middleware for**
- **Embedded**
- **Peer-To-Peer**
- **Systems**

Objectives

Objective

Study the effectiveness of using a tuple space based coordination language as a way to implement such a middleware, in this paper: SMEPP.

Why on top of tuple spaces (LINDA-like coordination language)?

- Abstract coordination model (ease distributed/heterogeneous systems specification)
- Basic high-level blocks of coordination
- Enhancement of other data-centric mechanisms (eg: DHT)



Outline

- 1 Introduction
- 2 SMEPP**
- 3 SecureLime
- 4 Implementation
- 5 Conclusion



SMEPP

Summary:

- Provide a high-level service model to ease P2P application development thanks to different (lgge dependent) APIs implementing a set of **primitives**.
- Introduce group-wise security,
- Service-oriented,
- Messages/event -based communication.



Key concepts

- **Peers:** service containers, program specified using SMEPP primitives...
- **Groups:** logical association of peers, secured, scope for services...
- **Services:** contract (descriptive information) and implementation, two ways of invocation...
- **Communication:** messages (service invocation), event...



SMEPP Primitives

// Peer Management

```
pId newPeer(creds)
pId getPeerId(id?)
pId[] getPeers(gId)
```

// Group Management

```
gId createGroup(grDescr) gId[] getGroups(grDescr?)
grDescr getGroupDescription(gId)
void joinGroup(gId, creds)
void leaveGroup(gId)
```

// Service Management

```
<gSId, pSId> publish(gId, contract)
void unpublish(pSId)
<gId, gSId, pSId>[] getServices(gId?, pId?, sContract?, maxRes?, creds)
sContract getServiceContract(id)
sessId startSession(sId)
```



SMEPP Primitives

// Message Handling

```
out? invoke(eld, opName, in?)  
<cld, in?> receiveMessage(gld?, opName)  
void reply(cld, opName, out?, fName?)
```

// Event Handling

```
void subscribe(evName?, gld?)  
void unsubscribe(evName?, gld?)  
void event(gld?, evName, in?)  
<cld, in?> receiveEvent(gld?, evName)
```



Outline

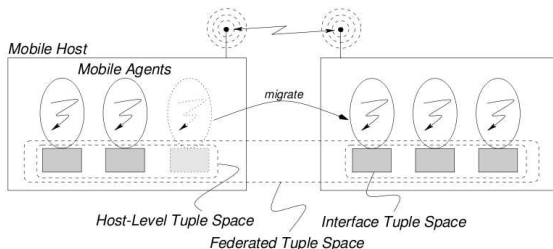
- 1 Introduction
- 2 SMEPP
- 3 SecureLime**
- 4 Implementation
- 5 Conclusion



Summary

SecureLime (Handorean and Roman) extends Lime (Murphy, Picco and Roman).

- Lime extends Linda with transiently shared TS,
- SecureLime extends Lime by adding security properties (access control on TS and tuple level).





Why SecureLime?

SecureLime meets the main requirements

- P2P orientation (decentralised architecture: every peer responsible for its own data)
- Available implementation (OpenSource, documented,...)
- Security: two levels (tuple space and tuple $\langle field_1, \dots, field_n, pwd_{read}, pwd_{write} \rangle$)
- Java-integrability (feedback to final SMEPP implementation)



Outline

- 1 Introduction
- 2 SMEPP
- 3 SecureLime
- 4 Implementation**
- 5 Conclusion



High-level design

- 2 tuple spaces for group and service discovery
- Peer mapped to a LimeAgent + 2 discovery tuple spaces
- Groups mapped to a collection of peers sharing a TS
- Service mapped to a tuple in the service discovery tuple space, implementation uses API, invocation using tuple
- Event mapped to a tuple publication



Security design

Design follows SMEPP guidelines:

- Every peer has an *AppKey* password granting access to a SMEPP application (ie: key of directory TS's) [tuple space level security],
- Accesses to group are protected using the key corresponding to the group (*GKey*) [tuple space level security],
- Groups and services are protected in directory using *GKey* as read-password [tuple level security].
- A peer owns a credential formed by an *AppKey* and a list of *GKeys*.



Scenario - code (1)

```
// TempReaderPeer  
myCredentials = opaque;  
newPeer(myCredentials);  
mySecurityInfo = opaque;  
myGroupDescription = <"TempReaderGroup",mySecurityInfo>;  
tempGroupId = createGroup(myGroupDescription);  
tempReaderServiceContract = opaque;  
publish(tempGroupId, tempReaderServiceContract); ...  
  
//TempReaderService  
<callerId> = receiveMessage("getTemp");  
temp = opaque;  
reply(callerId, "getTemp", temp);
```

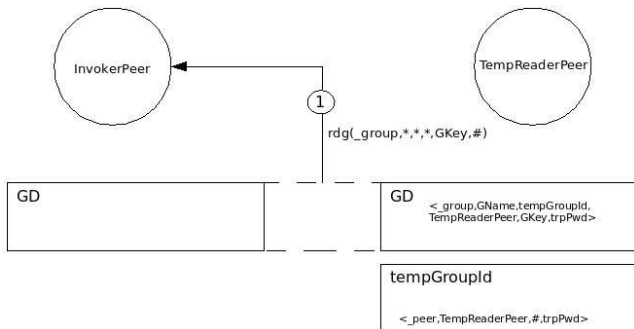


Scenario - code (2)

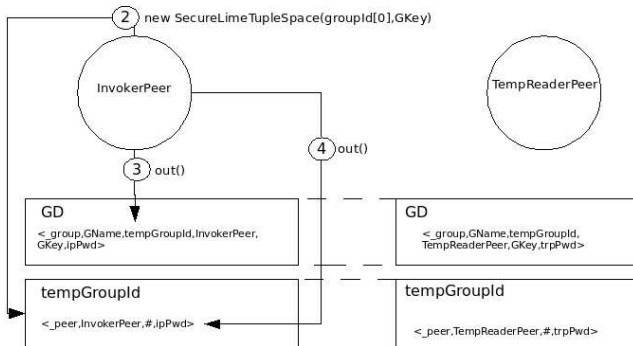
```
//InvokerPeer  
myCredentials = opaque;  
newPeer(myCredentials);  
desiredGroupDescription = <"TempReaderGroup">;  
gid[] = getGroups(desiredGroupDescription);  
tempReaderServiceContract = opaque;  
<groupId,tempReaderServiceGroupId, tempReaderServicePeerServiceId>[] =  
getServices(gid[0], tempReaderServiceContract, myCredentials);  
joinGroup(groupId[0], myCredentials);  
temp = invoke(tempReaderServicePeerServiceId[0], "getTemp");
```



Scenario - schema (1)

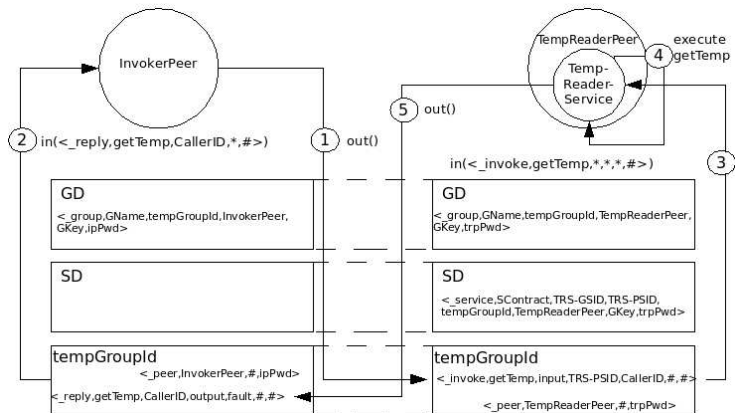


Scenario - schema (1)





Scenario - schema (2)





Outline

- 1 Introduction
- 2 SMEPP
- 3 SecureLime
- 4 Implementation
- 5 Conclusion**



Contributions:

- A simple high-level API supporting
 - Peer/service code
 - Peer groups
 - Group-wise security
 - (a)synchronous message patterns
- Proof-of-concept of the SMEPP model

Future work:

- Overcome security limitations
- ... using other coordination language,
- ... for instance *Bach* (I. Linden and J.M. Jacquet),
- ... currently being implemented on the Nokia N800 platform.



Thank you!

Question(s)?