A Practical Modelling Notation for Secure Distributed Computation

Yih-Jiun Lee Peter Henderson
DSSE Research Group, School of Electronics and Computer Science
University of Southampton, SO17 1BJ, United Kingdom
y.lee@ecs.soton.ac.uk p.henderson@ecs.soton.ac.uk

Abstract

Mobile code computation has lead to a new paradigm of distributed computation. A mobile process can move from site to site and interact with the resources as a local process. To prevent the misuse resources, authentication and authorization need to be dealt with. Many modeling languages have been proposed to model distributed computation; Ambient Calculus [1] is one of them. Ambient Calculus, a type of process calculus, can be used to model boundary crossing activities in a mathematical notation.
SJAN, a modeling language, extends the basic idea of Ambient Calculus, but it has a simple notation, design and representation. SJAN can be verified by J-Ambient, a JAVA implementation, to certify the correctness of the model.
In this paper, we will introduce SJAN notation and J-Ambient. SJAN can be also used to model grid computation and provides the security consideration. Thus, finally, we will address several security scenarios in SJAN to show its functionality.

1. Introduction

Distributed computing establishes an environment to allow processes in different places around the network to share distributed resources. Traditional distributed computing was achieved by code-on-demand or remote invocation. They can provide stateless mobile computation. The newest technologies of distributed computing, web-services based computation and grid computing, enable stateful computation for reliable and reusable geographically distributed resource. However, precisely because the resource providers and requesters are geographically distributed, so the security risks are the significantly more important.
To model the security issues and distributed computing, many abstract model notations have been proposed. Some of them concern the functional expressions and some of them focus on network process or dynamic connections. We give an overview in [2].
In this paper, we will introduce a modeling notation called SJAN. SJAN inherits the idea of ambient from Ambient Calculus [1] which will be addressed in Section 2. Thus, it can be used to model distributed computation which crossed enterprise boundaries. It can be also used to model stateful distributed computation. This notation is based on our experiment, J-Ambient, with JAVA and its associated technologies. The purpose of SJAN is to model secure distributed computation, including grid computing.
This paper has been organized as follows. In Section 2 we briefly describe the idea of Distributed Computation and the security risk. In Section 3 we introduce the SJAN notation and its functionality. Then the security discussion and solution based on grid computing will be addressed in Section 4. In Section 5 we conclude our work and describe our future plan.

2. Distributed Computation

From the computing's point of view, computation mobility can be classified into two different groups: weak mobility which refers to the stateless movable code and strong mobility which concerns both computation and the running state moving together [3]. Stateful computation, shown in Fig 1, is the capability that a submitted process can run part of it on one server and the remains on another. From the

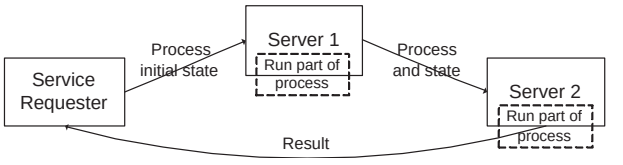

Figure 1. Stateful Distributed Computation

abstract models' viewpoint, some abstract models, such as CSP [4] and CCS [5], concern static connection and functional, concurrent and distributed computing and some of them, such as process calculi can be used to model pro-

cesses with dynamic connection, such as pi-calculus [6]. Ambient Calculus [1] is another case of processes calculi. In Ambient Calculus, the mobile objects can be restricted within the domains' boundaries [6]. A formal idea, "ambient", was used to represent a movable and executable environment. The inner processes, gremlins, can move with the ambient or be blocked until a condition is satisfied.
Since a process can carry computation and move around, security becomes a critical issue. Traditional security solution is cryptography which uses symmetric or asymmetric key to encrypt the message to ensure the integrity and privacy. However, in the Distributed computation system, "access-control" system is also important. Access-control is used to ensure the crossing boundary processes only happen if the processes are authenticated and authorized. In Ambient Calculus, both of them can be achieved by the use of ambient names.
Modeling distributed computation and its security solution is difficult because the model has to represent the security and stateful status. Secure J-Ambient Notation(SJAN) is one of the solutions which will be discussed in the next section 3.

3. Secure J-Ambient Notation(SJAN)

Secure J-Ambient Notation(SJAN) follows the executable movable environment idea from Ambient Calculus and inherits the syntax from J-Ambient, an implementation in JAVA and its associated technologies. The detailed introduction of SJAN and J-Ambient can be found on our WEB site [2].
Supposed, there are ambients, A, B, and C shown as following.

```
ambient A{
  ambient B{
    ambient C {
      1. out B; 2. open B;
    } || 3. in C;  ||  4. out A;
  } || 5. be D;
}
```

There are 5 actions in the above example. Action 1 and 2 should run in sequence, because they are connected with ";". Ambient C, action 3 and 4 can run in parallel, because they are separated by "||".
Consider all the preconditions, only action 1, 4, and 5 can be executed. If action 1 is chosen, **ambient** C moves out from **ambient** B and becomes a child of **ambient** A. Then, action 2 will be blocked because "C:open B" has the precondition, B must be a child ambient of C, to be satisfied. Now, only action 3, 4, 5 satisfy all the preconditions. If action 3 is chosen, **ambient** B will move into **ambient** C, so the precondition of action 2 is satisfied and can be fired. However, action 4 will be blocked because the **ambient** B's parent is not **ambient** A anymore.
Considering "C:open B", every gremlin in B will be moved into ambient C with its state. Once opened, **ambient** B no longer exists and action 4 becomes process in **ambient** C and can be unblocked.
Now, only action 4 and 5 are remaining. They have a sequence competition. Supposed, action 5 runs before action 4. **Ambient** A renames itself to **ambient** D. "C: out A" will be never awoken. The system will end with ambient D{ ambient C{ out A } }.
If action 4 was chosen, **ambient** C becomes ambient A's sibling. Because action "be" has no precondition, "A:be D" can be processed at any time. The final state of this example will be ambient A{} || ambient C{}.

4. Security in Grid Computing

4.1. Security risks on grid computation

According to [7], information security refers to protection of information systems against unauthorized access to storage, processing or transit. Besides, [7] also points out that two services, access control services and communication security services, are crucial for a secure Internet infrastructure. In this paper, only access control service will be considered, because Secure Sockets Layer(SSL) is a popular utility to implement the communication security and can be easily applied.
Considering two issues: authentication and authorization, they are important functions in distributed computation. Authentication proposes a service that a server can restrict the specific operations to be available for a certain client only. After authentication, the server can assign variant access right to the client. It is called authorization [8, 9].

4.2. Authenticate a mobile process

The purpose of the "open" reduction is "upgrading" the inner process which wants to be executed as a "first-level" process that can be seen as a local process. The precondition of open, which has been discussed earlier, leads to the idea of authentication, because an ambient can be opened only if it has been known. Once an ambient is opened, the internal processes can be authorized to execute as the local processes. See [1] for more detailed how ambients' names are used for authentication.

4.3. Authorization with a security token

The following scenario is shown in Fig 2. In this system, there is a user Joe on ServerA. Joe is authorized to

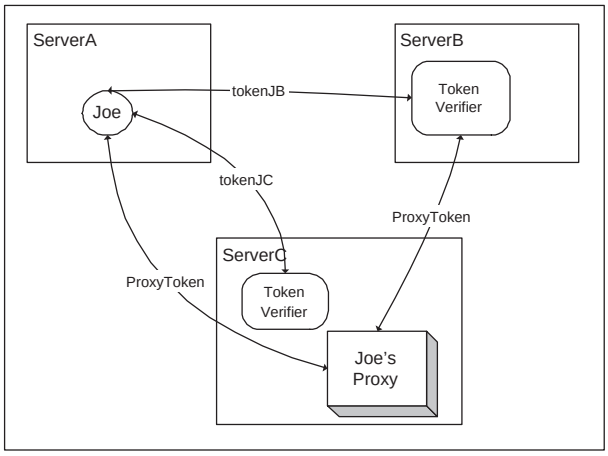

Figure 2. The scenario of authorization

run his jobs on ServerB by providing a security token, tokenJB. The idea of security token is broadly used in distributed computing. It is usually achieved by X.509 Public key certificate or a pair of username and password. Now, back to our scenario, Joe is on ServerA and wants to execute a job, P, on ServerB. The first thing he needs to do is transferring the process P from his domain to ServerB with his security token, tokenJB. The SJAN model is as below.

```
ambient ServerA{
  ambient Joe{
    ambient tokenJB{
      out Joe;    out ServerA;
      in ServerB; P;
  . . .
}
ambient ServerB{
  repeat{ open tokenJB;   }
  ||. . .
}
```

The behavior of tokenJB is moving itself from ServerB to ServerA. Then there is a gremlin on ServerB. It is always waiting to open tokenJB. The "open" operation can execute until there is an ambient tokenJB inside ServerB. Once tokenJB has been opened, process P can run as a local process.
Joe may have other computations running in parallel, but any two running threads will not mutual affect.
The tokenJB can be seen as box with a lock and the open statement is the mapping key. Joe is authenticated by virtue of the fact, but he has a valid token.

4.4. Stateful computing within three servers

The basic idea of stateful computation is that a job can run its first part on one server and another part on another server, but the job has to be authenticated and authorized on two servers. Thus, Joe has two security tokens: tokenJB for ServerB and tokenJC for ServerC and a submitted process which should run the first part, process P, on ServerB and then execute the remains on ServerC.
Following shows the SJAN model.

```
ambient ServerA{
  ambient Joe{
    ambient tokenJB{
      {out Joe;     out ServerA;
       in ServerB;  P;
       new Signal{in WaitForSignal;}
      }||
      ambient WaitForSignal{
        open Signal;    out ServerB;
        in ServerC;     be tokenJC;
        Q;
      }
    }
  }||  .  .  .
}
ambient ServerB{
  repeat{  open tokenJB;  } || . . .
}
ambient ServerC{
  repeat{  open tokenJC;  } || . . .
}
```

"TokenJB" firstly moves to ServerB and waits to be opened. Because its next stop is ServerC and the running result should go with it, **ambient** WaitForSignal can be used to wait until a signal, ambient Signal, arrives. After execution, **ambient** Signal will be generated by process P. The only job of Signal is to be opened by **WaitForSignal**, so that the following operations, move to ServerC, can be executed. Similar design can be used again if ServerC is not its final destination.

4.5. Proxy delegation

The idea of proxy comes from Grid computing [10]. Because an execution might take a long time, waiting on-line is not necessary. Thus, a user can create a temporary identity and delegate a subset of his rights to it. This temporary identity is called proxy. The proxy can run on the behalf of delegator.
In this scenario, Joe has created a proxy which is running on ServerC. The proxy will send a job to ServerB, so Joe needs to delegate his right, executing a job on ServerB, to the proxy. Firstly, Joe has to inform ServerB that someone

who holds the ProxyToken is allowed to execute in Joe's directory.

```
ambient Joe{
  ambient tokenJB{
    out Joe;        out ServerA;
    in ServerB;     open ProxyToken;
  }
  || . . .
}
```

The ServerB's job is similar to the earlier example. Once tokenJB has been opened, open ProxyToken became a run-once gremlin on ServerB.

```
ambient ServerB{
  open ProxyToken;
  || . . .
}
```

After the inform message has been sent, Joe sends the ProxyToken to Proxy via a message.

```
ambient Message{
  out Joe;        out ServerA;
  in ServerC;     in Proxy;
  output ProxyToken;
}
```

ProxyToken can output as a message in Proxy, so Proxy can consume it. When "**be** ProxyToken" is executed, the name and identity of **ambient** temp will be renamed to **ProxyToken**.

```
ambient Proxy{
  open Message;
  ambient temp{
    input ProxyToken;   be ProxyToken;
  }
. . .
}
```

Changed to . .

```
ambient ProxyToken{
  out Proxy;        out ServerC;
  in ServerB;       P;
}
```

Now the ProxyToken can move to ServerB and can be opened by the operation of "open ProxyToken;"
In this design, because Joe only send once "open ProxyToken" to ServerB, ProxyToken can be only used(opened) once to avoid malicious attack.

5. Conclusion

SJAN is a modeling language which can be used to model both stateless and stateful distributed computing system. To provide security modeling, SJAN models its security representation by the simple reductions. Once a scenario has been modeled, it can be easily implemented in the real implementation language. For instance, an opened name can be presented as a session key, so the content in the opened ambient is encrypted. By this practical notation, a distributed application can be modelled and validated before the real implementation. SJAN is also suitable for modeling web-services based computation and grid computing which has been shown in the earlier section. For formal model verification, SJAN has been implemented in a modeling language, ARC [11] for further validation. Nevertheless, we still need to give SJAN a more methodical definition and rules for the newest web-services based distributed computing because the naming principle for web-services has not been supplied in SJAN. Notwithstanding this problem, SJAN has proved that it can be used to model many the distributed computing cases and the models will be very legible and tiny. This is particular important because modelling security model is often difficult to get right.

References

[1] L. Cardelli, "Abstractions for Mobile Computation," Microsoft Research Technical Report MSR-TR-98-34," Technical Report, July 1998.

[2] Y. Lee and P. Henderson. (2004, Oct.) A practical modelling notation for secure distributed computation-full version. Internet. [Online]. Available: http://yihjiun.ecs.soton.ac.uk/paper.html

[3] P. W. L. Fong, "Viewer's discretion: Host security in mobile code systems," School of Computing Science, Simon Fraser University, Burnaby," SFU CMPT TR 1998-19, Nov. 1998.

[4] C. Hoare, *Communicating Sequential Processes*. Prentice Hall International, 1985.

[5] R. Milner, *A Calculus of Communicating Systems*. Springer Verlag.

[6] Raja and Shyamasundar, "Mobile Computation: Calculus and Languages (a tutorial)," in *CSC: Asian Computing Science Conference*, LNCS, 1998.

[7] J. Joshi, W. Aref, A. Ghafoor, and E. Spafford, "Security models for web-based applications," *Communications of the ACM*, vol. 4, no. 2, Feb. 2001.

[8] WebLogic. Introduction to weblogic security: Security fundamentals. [Online]. Available: http://e-docs.bea.com/wls/docs81/secintro/concepts.html

[9] G. Gabor. (2001) Evaluation of distributed authentication, authorization and directory services. [Online]. Available: http://www.caesar.elte.hu/eltenet/projects/demogrid/demogrid-report-1/dg-rep-1-sec-eval.pdf

[10] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, "A national-scale authentication infrastructure," *IEEE Computer*, vol. 33, no. 12, pp. 60–66.

[11] P. Henderson. Arc: A tool for system level architecture modelling. [Online]. Available: http://www.ecs.soton.ac.uk/˜ph/arc.htm