Summary of:
Pays as You Go – Associating Costs with Jini Leases

Nathan Balon
Advanced Operating Systems - CIS 578
October 14, 2004

Introduction

The paper "Pay as You Go – Associating Costs with Jini Leases" was written by Peer Hasselmeyer and Markus Schumacher. What the paper is focusing on is using the Jini lease framework to provide billable services to clients. Currently, Jini does not support the notion of a billable lease, but it does offer a leasing mechanism, so the authors are proposing an extension to Jini leases. A number of problems arise when charging a fee for the use of lease. The main problems deal with additional security requirements that are need for this environment. The authors propose various security mechanisms that can be used with their architecture. The main focus of the paper is the creation of a non-repudiation service and the components what are needed to implement it.

Jini Leases

A Jini lease is a contract between two parties. One party grants the use of a resource to another party. In Jini, leases are granted for a specific period of time. When a lease expires access to the resource is terminated, although it is possible for a client to renew a lease. In the negotiation of a lease, the lease grantor always has the final decision on what the length of time for lease will be. There is a Java interface `Lease` that lease granting objects implement. The methods of Lease interface are: `getExpiration, cancel, renew`, and etc. Using this `Lease` interface allows for services to be leased to clients.

Since Jini provides this basic leasing facility, it possible to extend this model to allow companies to provide services through a lease and to charge clients a fee for this use of a service. The authors point out that a different model is needed for the charging of services with Jini. In the Jini federation communication takes place between devices, so it is not possible to give a service away for free and then rely on generating revenue through advertising as is done on the web. The authors determine that two basic models can be used to purchase services which are unit-based and time-based. The authors give two examples of the possible services that can be provided for a fee which are: a storage service and the use of a software component. For instance, a user may pay for the use of a software module that is infrequently used, such as a module to compute income tax. The possibilities are virtually endless to the type of services that can be leased using Jini.

While providing distributed billable services with Jini offers many benefits there are also some problems that need to be addressed. Specifically, there are number security requirements that are needed in order to provide commercial services. Jini leases must provide the typical security services dealing with issues such as integrity, confidentially and denial of service. Along with these, repudiation must also be accounted for. Repudiation is the refusal to acknowledge or pay a debt or honor a contract. Non-repudiation is the concept of ensuring that a contract, especially one agreed to via the Internet, cannot later be denied by one of the parties involved (Wikipedia.com). The paper specifically explores ways to enforce non-repudiation.

The authors give the following reasons why it is essential that a non-repudiation service is used. It should be impossible for a service provider to charge for a service that has

been not provided. Next, it should be impossible for one to use a service and not pay for it. Last, a service should only be granted that was requested. To accomplish these goals it is essential to authenticate both the clients and servers involved in the lease. A non-repudiation service is needed to settle the disputes between parties when they arise.

Proposed Architecture

The authors' solution to combating the problem of repudiation is to create an extension to the Jini architecture. What the authors are proposing is adding two new interfaces and three run-time services as solution to the problem.

The two interfaces proposed are: `BillableLease` and `AccessCharge`. The interface `BillableLease` extends the Jini `Lease` interface. The interface is defined as:

```
public interface BillableLease extends Lease {
    Contract getContract() throws RemoteException,
                          NoSuchContractException;
    long getCost() throws RemoteException;
    Evidence renew(long duration, Evidence proofOfOrigin)
throws
                          LeaseDeniedException,
UnknownLeaseException,
                          IllegalEvidenceException,
RemoteException;
    Evidence cancel(Evidence proofOfOrigin) throws
                               UnknownLeaseException,
                               IllegalEvidenceException,
RemoteException;
}
```

This interface allows client get the cost of a lease, renew a lease, cancel a lease and get the contract associated with the lease. The contract is a remotely accessible object. Each lease in the architecture is linked to a specific contract. The contract contains information such as the type of service provided, the quality of the service and a price of the service. The authors envision using signed xml documents for contracts. The authors do not go into any detail of how contracts created between parties. The Evidence object is an object that is generated by the non-repudiation service.

The next interface that the authors defined for their architecture is the `AccessCharge` interface. The `AccessCharge` interface is defined as:

```
public interface AccessCharge {
    AccessGrant start(Contract termsAndConditions,
                      Evidence proofOfOrigin, long
duration)
                  throws NoSuchContractException,
IllegalEvidenceException,
                       RemoteException;
}
```

The interface only defines one method that must be implemented, which is `start`. The parameters of `start` are the contract to be applied, a proof of origin of the client and the length of time for the lease. The time length is specified in milliseconds. The `AccessGrant` object returned from `start` includes a proof of receipt and a `BillableLease`. Once the `AccessGrant` object has been returned the client can then use the lease.

Along with these two interfaces three additional run-time services are needed: a non-repudiation service, a time service and an accounting service. First, the non-repudiation service creates digital evidence. In the case of the architecture proposed proofs are generated. The two proofs created by this service are a proof or origin and a proof of receipt. The proof of origin is used to prove that a sender has actually sent a message. The proof or receipt it used to prove that the receiver has correctly received a message that was sent by the sender. The proofs are then persistently stored, so in the case of a dispute a settlement can be reached between the involved parties. Second, the time service is used to provide secure time stamps. A time service is need because of lack of synchronized time in a distributed system. Third, the accounting service stores and retrieves billing information. Below in figure 4, from the paper, shows how these three services fit together in the architecture.

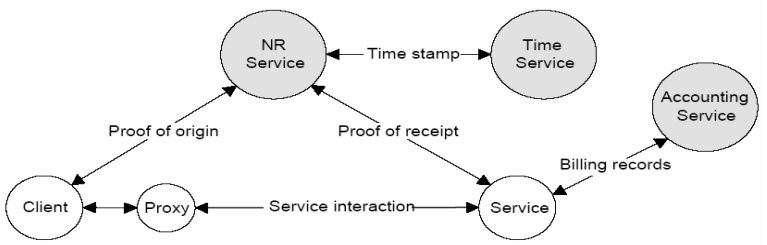

Figure 4. Architecture components

Another aspect of the architecture that must be considered is how a lease is actually started and canceled. To start a lease a number of messages must are sent between the various components in the system. The client initiates the process by sending a message to the non-repudiation service to generate a proof of origin. Anytime that a proof is to be generated by the non-repudiation service it sends a message to the time service to get a time stamp. Once the time stamp is returned to the non-repudiation service, the non-repudiation services then generates a proof of origin which is the sent back to the client. At this point the client calls the start method sending the proof of origin, the contract to be applied and a time length to the service provider. Next, the service provider must generate a proof of receipt, so it sends a message to the non-repudiation service to generate the proof. Upon receiving the proof of receipt from the non-repudiation service, the service provider sends back to the client the proof of receipt along with a billable lease. The client can then use leases at this time. When the lease expires or the lease is canceled, a billing record is sent to the accounting service so the client can be charged for the use of the lease. A similar sequence of messages is also sent throughout the system

anytime that a lease is either canceled or renewed. Figure 5 from the authors' paper show the interaction of component and the messages sent when a leasing session begins.

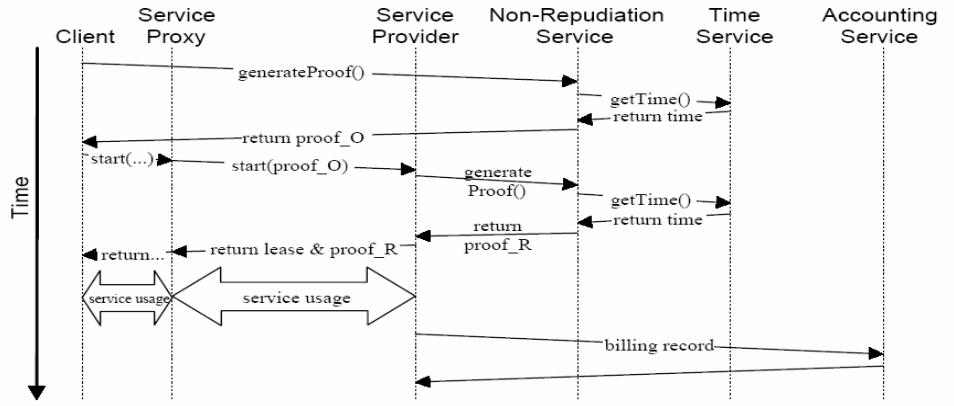

Figure 5. A leasing session

By using the service provided by the authors it would be possible to prove the events that take place in the system. If a client is billed for a service they claim to have never used it is possible to see if a proof of origin was generated by the non-repudiation service for a particular service. If no proof of origin is found the client would not be liable for paying for the services, since it can be proved that they never asked for it. The opposite case is also possible; where it can be proved that a client did actually initiate the service and should be held responsible for paying for a service.

Problems with Architecture

There are a few problems with the proposed architecture. The first problem is paper doesn't address if any fault tolerance or replication is used for the non-repudiation service. Another problem with the architecture is that client and service must trust all components that are provided in the architecture. When a client uses a service they must also trust the proxy that they download. In any security system some level of trust must be put into the other components. For instance, a great deal of trust in put in a PKI. A further downside is the architecture adds extra overhead to the system. Anytime that a client requests or renews a service a number of messages need to be sent back and forth to generate the proofs and time-stamps needed. There really is no way around this though. The messages to the time service are necessary. The extra messages come as an expense to the added functionality that is provided. Furthermore, the architecture does nothing to prove that quality of the service provided by the lease. A client may gain access to a resource, but there is no mechanism to ensure that the service meets the requirements of the client. The architecture favors the service provider. It seems more of a way to protect the interest of the service provider, rather than to protect the interest of the client. Last, for this extension to Jini to be effective it needs to be standardized and incorporated into Jini. If the architecture is not standardized there would be no way for components to lease service from each other on a large scale. At the present time there is

no standardized non-repudiation service such as the one proposed in the paper, so it seems unlike that this will become standardized.

Application of the Architecture in the Real-World

On application where a Jini billable lease could be used is with cable television, specifically the pay on demand service that is offered by most cable companies. The client, in this case cable box, could use the look up service to locate the movies that are currently available to watch. Then a lease could be established allowing users to view a movie for a specific time period, such as a day but basically whatever time period was agreed on. Once a lease has been established the signal for the movie would be sent to the cable receiver. It would be possible for the client to renew the lease by sending a message to the movie service. In cases such as this, it would be important to have a non-repudiation service such as the one proposed. If a non-repudiation service was not present, user of the service would be able to claim that they never ordered a movie and the cable company would have no way proving that they did. Even if the person thinks that they never ordered a movie it may be possible that their children or spouse did. If a dispute arises it would be possible to prove that a customer actually ordered the movie by examining the proofs that were generated by the non-repudiation service. If proof of origin is found that indicates that the client did actually order the movie they would be responsible for paying for it. The authors' architecture effectively provides a way of proving the use of a service in this case the use of a pay on demand movie service.

Lessons Learned

From this paper I learned a great deal about how Jini actually works. Jini uses many of the principles of RMI and extends RMI's functionality. One thing I noticed is I don't really see Jini being promoted the way some other technologies are such as web services. Jini provides an excellent solution that allows for devices share the services. I think in the future a technology such as this will have a greater impact. Whether it is Jini or a similar protocol that is used for the implementation, the use of billable services such as the author is proposing will have a large impact on what can be done with technology. The increased presence of cell phones provides one possible market for this technology.

Another thing that I noticed is that the Jini lookup service seems to be a large improvement over the RMI registry. The Jini lookup service is much more transparent. A client doesn't need to know the actual name of a service, all the client has to do is look for a service that implements a specific interface and they can locate a service.

As far as security, it is almost impossible to have a perfectly secure system. In a situation like the one proposed by the author the user of a service has to put trust into a number of external components, such as the non-repudiation service. A system is only as secure as its weakest component. If the security of any service is compromised all the gains that come from the authors' proposed architecture are lost.

Implementing security in a distributed environment is much more difficult than implementing it on a standalone computer or a client/server environment. For example, on a single computer the issue confidentiality is extremely important, such as access control of files. In a distributed environment access control must be addressed, along with the other issues that author has pointed out, such as the need for a secure time service. A number of external services are needed, which are not need in the traditional computing. Developing security for a distributed system becomes much more complex.

Conclusion

The paper addresses the problem of repudiation of billable Jini lease. Creating a non-repudiation service that generates proofs of the granting and terminating of a lease provides a way to prove that a client and service has actually completed there part in the lease. In the future methods such as those proposed will become even more important as it becomes more common for different devices to communicate with and uses the service of other devices.