MODELING A SERVICE DISCOVERY BRIDGE USING RAPIDE ADL

Ahmed Sameh
Dept. of Computer Engineering
The George Washington University
Washington, DC 20052
Email: sameh@gwu.edu

Rehab El-Kharboutly
Dept. of Computer Science,
The American University in Cairo
P.O.Box 2511, Cairo, Egypt
Email: sameh@aucegypt.edu

KEYWORDS
SDP, ADL, Jini, UpnP, Rapide

ABSTRACT

The exploding deployment of network enabled mobile devices, along with the expansion of networked services have created the need for users to easily manage these devices and services and also to coordinate with one another. Service Discovery Protocol (SDP) enables networked devices, applications, and services to seek out and find other complementary networked devices, applications, and services needed to properly complete specified tasks. A variety of Service Discovery Protocols have been proposed by the market and academia, including Jini, UPnP, SLP, Salutation and Bluetooth. For these protocols to co-exist, they should exhibit interoperability features. A number of bridging techniques have been proposed and implemented. Efforts have been on going to analyze these bridges from an architectural point of view. A most suitable means for such purpose is Architecture Descriptive Languages (ADLs). ADLs, like Rapide, enable the simulation of distributed systems such as Service Discovery Protocols. In this paper we propose a one directional bridging system (Jini-UPnP Bridge). To validate the proposed system, we model and simulate the bridge using Rapide ADL simulation and analysis toolset. We perform a number of simulation tests and use the Rapide Poset viewer to analyze the simulator's output Poset tree of events. The bridge overhead, compared to a non-bridged native Jini service was found to be about 93.5%. The bridge performance was measured under both light and heavy network loads. Under light loads the bridge achieved 0.071% improvement, while its performance has degraded 0.034% under heavy load. The bridge performance was also measured when bridging multiple services. The results fall in reasonable ranges from 1.00079s to 1.00143s for the overall bridging time. To further validate our model, we performed a set of experiments to test communication failures.

INTRODUCTION

The number of networked services is expected to increase enormously in the incoming era. Other than traditional services (e.g. printing, scanning and faxing), new networked-services for business purposes, such as network based computational systems, or light weight services, such as restaurant directories and translators, are becoming available and highly important. For an effective use of these services, users should have means for direct and easy access to these services. Service Discovery Protocol (SDP) presents an attractive solution for services discovery and coordination (Bettstetter and Renner 2000).

One of the main factors of judging the efficiency of a given SDP is its ability to interoperate with other SDPs. Interoperability is a vital issue since it would enable services and clients with different service discovery protocols to communicate and interact with one another. Some of the SDPs use a proxy or bridge as a solution to enable services that don't support their SDP to nevertheless have role in their federations.

In this paper, we present a new approach for bridging between Jini and UPnP. We use architectural modeling to develop a Jini-UPnP Bridge. We validate our work by carrying out a series of simulation tests and experiments on the executable architectural model. Initially, we set a hypothetical topology of Jini and UPnP clients and services in addition to our proposed Jini-UPnP bridge. This setup is used to verify that the Jini-UPnP Bridge is capable of registering a UPnP Service that offers a JiniFactory, with the Jini Lookup service. The basic functionalities of The Jini-UPnP Bridge are tested and verified. We assess the performance of the Jini-UPnP Bridge through a number experiments including: 1- measuring the overhead of bridging a UPnP service versus direct registration of a Jini native service, 2- measuring the performance of the bridge under both light and heavy network loads, 3- deducing the performance of the bridge on bridging multiple UPnP JiniFactory services. Moreover, we performed the set of experiments conducted by Dabrowski and Mills in (Dabrowski and Mills 2001) to test the behavior of our hybrid-bridging environment in cases of communication failures. We compared their results to ours to validate the correctness of our model (El-Karboutly 2002).

This remainder of this paper is organized as follows: In Section 2, we describe the proposed Jini-UPnP bridging technique. First we give a high level design view and then we present some implementation details. Our tests and experimental work is discussed in Section 3. We conclude in Section 4.

Proceedings 18th European Simulation Multiconference
Graham Horton (c) SCS Europe, 2004
ISBN 3-936150-35-4 (book) / ISBN 3-936150-36-2 (CD)

THE PROPOSED Jini-UPnP BRIDGE

One of the main factors of evaluating and judging any of the available SDP protocols is the extent to which it allows for interoperability. A bridge between UPnP and Jini has not been investigated before; though it has been mentioned as possibility in a number of references (IBM 1999) (Richard 2000) (ADL 1997).

Both Jini and UPnP introduce the concept of bridging a foreign network device as part of their specifications. Jini refers to it as a network proxy (Luckham 2001). While UPnP refers to it explicitly as a UPnP Bridge (Wang 2003). In both SDPs, the bridging concept is based on introducing a foreign device to the SDP environment through the use of a representing entity that speaks on its behalf (a bridge).

The choice of bridging Jini and UPnP is based upon the fact that both protocols, though similar at the core functionality level, have dissimilar points of strength. Both Jini and UPnP support the same set of basic SDP operation, including service advertisement and service discovery. They both support the concept of leasing for registered services and support eventing and notification mechanisms for updating service information. Jini a centric protocol, based on the presence of a central cache manager, is an example of three-party protocols, which cannot function without a Lookup Service. On the other hand, UPnP is decentralized and is more of a peer-to-peer communication model. Compared to Jini, UPnP is a lightweight protocol. This is due to the fact that Jini requires the presence of a JVM for all its entities. Bridging between Jini and UPnP will enable thin services that don't have a JVM to announce their services to Jini clients. Jini's most attractive feature is the ability of downloading services driver's or proxy, which enables easy and direct usage of the service.

Our work is built on the concept of a Jini network proxy described in Jini Device Architecture and is based on the efforts of Eric Guttman in (Guttman and Kempf 1999). A Jini-UPnP Bridge is an entity that enables services that support UPnP protocol to be reachable by Jini clients. For Jini clients, Jini-UPnP is a transparent layer that they are unaware of. The UPnP services that are advertised via the bridge are treated as native Jini services.

The proposed Jini-UPnP Bridge is modeled as a special network node that can communicate with other network nodes in both Jini and UPnP protocols. It mainly acts as a *Service User* (i.e. Control Point) in UPnP environment and a *Service Manager* (Service) in Jini environment. It waits for announcements made by UPnP devices and services that are willing to advertise their presence to the Jini clients and acts as a representative, almost a mirror for them in the Jini environment.

The first order of business of the proposed bridge is to prepare an appropriate entry for UPnP services, in the Jini Lookup Service. This involves primarily setting the appropriate attributes required and creating a service object as part of Jini service's registration.

UPnP services that are willing to advertise their presence to Jini clients are not required to have a JVM installed. They are mainly required to have a ***Jini driver Factory*** (Guttman and Kempf 1999). A Jini driver factory is a (*.jar) file that bares a manifest for the advertised service. A Java Archive File (*.jar) file is used to bundle multiple files into a single archive file. Typically a JAR file contains the class files and auxiliary resources associated with applications.

The proposed bridging process is done through the following steps:

The Jini-UPnP bridge searches the UPnP reachable entities to find devices and services that have Jini driver Factory or waits till it receives announcements made by Jini driver Factory services.
Once a Jini driver Factory service is found, the Jini-UPnP bridge obtains a complete description of the service including attributes, GUI URL and control URL.

The URL of the Jini driver factory is composed by extending the control URL with a unique identifier. The Jini driver factory is downloaded using GET method over HTTP.

The Jini-UPnP bridge performs attributes transformation from UPnP format to Jini format to prepare for service registration. Upon successfully translating the entire service attributes and obtaining the Jini driver factory, the Jini-UPnP bridge registers the discovered service with **Jini Lookup Service**. Using the Jini driver factory, the bridge creates a service object that is used for registration. Registration is done by sending a join request with all necessary attributes to **Jini Lookup Service** that adds the new service to its cache.

Whenever a Jini client needs our bridging service, it contacts **Jini Lookup Service** and downloads the instantiated object that is used to drive the service. Like any typical Jini service, the Jini-UPnP bridge should be equipped with JVM to be able to participate in the Jini SDP.

The first step in modeling our bridge is to set a hybrid Service Discovery environment, where different services and clients speak different service discovery protocols. This means that we would have n Jini services, m Jini clients, e Jini lookup services , p UPnP services and q UPnP clients, where n,m,e,x,p,q are natural numbers > 0 and by setting them we define our topology. This topology would be ADL modeled such that entities are able to perform normal service discovery operations with no conflicts.

Having the two NIST Rapide models for Jini and UpnP (Dabrowski and Mills 2001) , we merged the two models into one model with both Jini and UPnP interfaces and main modules in preparation to build our proposed bridge. The proposed Jini UPnP bridge is basically a network node that acts as a *UPnP SM* in UPnP environment and a *Jini SU* in Jini environment. It's basic sub modules are the basic components of UPnP SM and Jini SU models, in addition to sub modules that perform bridging.

The main sub modules of Jini-UPnP Bridge architecture are:

UPnP Service User (UPnP SM): is a modified implementation of the UPnP SM entity that also includes ***UPnP Local Cache Manager*** and the ***UPnP SU Filter***. The ***UPnP Local Cache Manager*** is modified such that it handles attribute translation from UPnP to Jini and also Jini driver factory download.

Jini Service Manager (Jini SM) : is a modified implementation of the Jini SU that communicates directly with the UPnP SM module of the bridge to receive bridged services.

In normal UPnP SU, the local cache Manager module is an interface for the internal cache of the SU. It handles UPnP discovered service records, notifications and events. In our bridged model it also handles the functionality of managing a cache for the Jini driver factory of the discovered Jini Driver Factory services. It implements the interface MANAGED_RESOURCE_JAR which exposes two methods: SUGetJar that requests downloading a jar file for a given Jini Factory service, and SMJarResponse which is the response to a SUGetJar request. MANAGED_RESOURCE_JAR is represented in Rapide ADL as follows:

```
TYPE MANAGED_RESOURCE_JAR IS INTERFACE
ACTION
 OUT
    SUGetJar
     (SU_ID, SM_ID : IP_Address;    -- Source SU, target SM
      QueryIssueTime : TimeUnit;    -- time query issued
      URLField : Integer);          -- This should be a URL or a  Device ID for  identification purposes
 IN
    SMJarResponse
     (SM_ID, SU_ID : IP_Address;     -- Sending SM, Receiving SU
      UniqueID     : Integer;        -- Unique Identifier for SD
      Jar  : String;                 -- a dummy string representing the  downloaded file
      TimeStamp : TimeUnit);
END;
```

Upon discovering the presence of a UPnP Service that provide a Jini Driver Factory, the Jini UPnP Bridge; first retrieves its complete description and downloads its jar file and then advertises its presence to the Jini Lookup Service. To perform the last functionality, Jini UPnP Bridge uses the interface ADVERTISE_SERVICE. ADVERTISE_SERVICE is responsible for propagating discovery of new service, change of a currently discovered service and deletion of a service to the JINI SM sub modules of the bridge. It is called by the Bridge Local Cache Manager sub module and implemented by the Jini Service Repository sub module.

ADVERTISE_SERVICE interface is presented as follows in Rapide ADL:

```
TYPE ADVERTISE_SERVICE IS INTERFACE
ACTION
OUT AddNewService(?Service_ID : Integer; -- ID of the service
ServiceType,          -- service type /name
ServiceAttributes,    -- service attributes
ServiceAPI,           -- service Proxy and APIs
ServiceGUI : String; -- service GUI
NLeaseTime,
NDuration : TimeUnit
- lease duration),
    ChangeServiceEv (?Service_ID : Integer; -- ID of the service
   ServiceAttributes:String -– new service
Attributes ),
    DeleteServiceEv   (?Service_ID : Ind_Service_ID;   -- Service ID
    ExpireOption : String       --Expire Option);

END; --ADVERTISE_SERVICE
```

A UPnP service that wishes to be used by Jini clients through our Jini UPnP bridge, should provide a Jini driver factory. The Jini UPnP Bridge issues an HTTP Get command to download the Jini driver factory file. A change was necessary to the UPnP SM Rapide Model for providing this functionality. The MANAGED_RESOURCE_JAR interface, introduced in the last section, is added to the UPnP Service Manager Model to be implemented by the UPnP SM_Repository sub module.

The overall Rapide model for a hybrid SDP environment with Jini UPnP Bridge consists basically of six different types of network entities: Jini SM, Jini SU, Jini SCM, UPnP SU, UPNP SM and Jini UPnP Bridge. Each of these modules implements the basic functionally of UPnP and Jini SDP Protocols. The Jini UPnP Bridge modules implements protocols of Jini SM and UPnP SU in addition to bridging functionality.

On the **network level**, the Jini-UPnP bridging environment consists of network nodes that are connected through communication links. Communication links are

mainly TCP/IP and UDP connections that are used for multicasting and unicasting messages. These communication links are modeled in our Rapide ADL as separate entities representing different multicasting and unicasting functionality.

The six network nodes: Jini SM, Jini SU, Jini SCM, UPnP SU, UPNP SM and Jini UPnP Bridge consist of **major functional components.** These are shown on the **Entity Major Functions** layer or the third layer from top. For Example the ***Jini Service Manager*** entity consists of ***a Service Repository*** and ***SCM discovery*** modules.

The lower level in the architecture shows the **main functional subcomponents**. These are the main components that carry out the main functionalities in the system. Some of these subcomponents are modeled as a Rapide interface and are implemented by different higher level models, while the rest are implemented as independent low level functionality modules. The main functional subcomponents of the ***SCM Discovery*** module, which is a basic module required in all Jini entities, is divided into three groups: Direct Discovery Protocol subcomponents, Aggressive Discovery subcomponents and Lazy Discovery subcomponents. Subcomponents that implement **Lazy Discovery Protocol** are: the ***Announcement Responder,*** which listens passively for announcements from entities that the SCM may wish to discover, the ***Announcer*** subcomponent, whose role is to send announcements to entities that may wish to discover the SCM to which it belongs, the ***SCM API Server,*** which provides service interfaces (APIs) to discovering entities after the initial response by the discovering entity to the SCM announcement and the ***Executive*** subcomponent whose main task is to control switching between aggressive, lazy and directed discovery.

Jini-UPnP BRIDGE TESTING and PERFORMANCE MEASURES

The next step after modeling the bridging between Jini and UPnP is to verify that the basic functionality of the bridge is correct through simulation tests. The Rapide toolset provides a set of compilation and runtime execution tools whose output is a simulation of the Rapide architectural model. The output of the simulation could be analyzed in various ways, including constraint checking, analysis for surprises and depiction of behavior. We chose to analyze the output of our simulation using the **P**artial **O**rder **S**et (Poset) browser. Poset browser enables us to view how a given architectural design behaves. It represents casual event simulations in a DAG form, nodes representing events and directed arcs representing causality.

In each of our tests, we first establish initial conditions by constructing a topology of Jini and UPnP basic entities in addition to the Jini-UPnP Bridge. The following tests have been conducted and proven successful: 1- testing to validate that initial discovery and advertisement activities in our hybrid environment of both Jini and UPnP entities, function correctly, 2- testing a complete scenario of bridging a Jini Service to examine the correctness of the bridging process, 3- testing that the proposed UPnP Jini Bridge successfully propagates changes that occur in the JiniFactory service to the SU Jini clients that have previously discover it, 4- testing to confirm that the JiniFactory service shutdown is propagated successfully to Jini SCM through Jini-UPnP Bridge.

We have conducted five experiments to measure the performance of the proposed Jini-UPnP bridge. In the following we discuss and report only four of them, naming: 1- measuring the overhead of bridging a UPnP service verses direct registration of Jini native service, 2- measuring the performance of the bridge under both light and heavy network loads, 3- deducing the performance of the bridge on bridging multiple UPnP JiniFactory services. Moreover, we performed the experiments conducted by Dabrowski and Mills in (Dabrowski and Mills 2002) to test the behavior of our hybrid-bridging environment in cases of communication failures. We compared their results to ours to validate the correctness of our model (El-Karboutly 2002).

The usage of a bridge in a hybrid system implies the presence of an overhead in time and resources. We are interested in measuring the overhead of bridging a UPnP service compared to having that same service as a native Jini service. The overhead is measured in terms of time and the number of messages exchange.

The following table shows the most relevant parameters and values for our experiment.

Table 1 Jini-UPnP Rapide Model Input Parameters

	Parameter	**Value**
General Parameters	Simulation overall time	3600s
	Node Startup Delay	1-15 s uniform
Behavior in both Jini and UPnP architectures	Polling interval	180s
	Registration TTL	1800s
UPnP specific behavior	Announcement interval	1800s
	Msearch query interval	120s
	SU purges SD	At TTL expiration
Jini specific behavior	Probe interval	5s (7 times)
	Announce interval	120s
	SM or SU purges SD	After 540s with only REX
Jini UPnP Bridge specific behavior	Jar file size	11Kb

Transmission and processing delays	UDP transmission delay	10 µs constant
	TCP transmission delay	10-100 µs uniform
	Per item processing delay	10 µs for cache items 10 µs for other items

First, we ran the Jini Rapide model with a topology of one Jini Service Cache Manager (SCM), two Jini Service Users (Jini SUs) and one Jini Service manager (Jini SM), where one of the Jini SUs requests a service of the same type as that offered by the Jini SM. We measure the time taken and the number of messages exchanged since the Jini SM starts up and until the Jini SU receives the service description. Next, we run our Jini-UPnP Bridged model with a topology of one Jini SCM, two Jini SU, one Jini SM, one Jini-UPnP Bridge, one UPnP SU and two UPnP SM. The time taken by a Jini SU to discover a requested UPnP service is measured. This time value is the sum of the time taken for Jini UPnP Bridge to discover the services; the time the bridge registers this service with the Jini SCM and the time the Jini SCM forwards the service description to the interested Jini SU.

Measurements for Jini are done on two stages; first we measure the time taken for Jini SM to register with SCM and the number of messages needed. We assume that SCM discovery has already taken place. The time taken for this operation, as shown in the results is **TIME TAKEN 1 = 0.064s** , and the number of messages exchanged is four messages (NUM MSGs 1 :4). The second stage is where the SCM starts matching the newly added service description to the available SU requests. Two messages are exchanged for this operation to complete and the total time needed is **TIME TAKEN 2 = 0.00081s**. Thus the total time for the whole operation starting with SM registration to SU discovery takes **TOTAL TIME = 0.06481s** on average.

Bridging a UPnP SM service to be reachable for Jini SUs is done in three stages. First the Service SM is discovered by the Jini-UPnP bridge, then the bridge registers the service with Jini SCM. The time taken for a Jini-UPnP bridge to discovery and obtain the complete description of Jini Factory service is **TIME TAKEN 1:1.00132s** where five messages are exchanged in this operation. Secondly, the bridge registers the newly discovered service with the SCM by exchanging two messages in **TIME TAKEN 2:.00022**. The last stage is where the SCM matches the added service to the notification for services that SUs have registered with the SCM earlier. This operation exhausts about **TIME TAKEN 3: 0.00061s**. The total time consumed in the process of bridging **TOTAL TIME = 1.00215s**

Comparing the results for a native Jini service to that of bridging the service through Jini-UPnP Bridge, it is clear that the bridging process has an overhead of about **0.93734s** or a 93.5% overhead.

Network Bandwidth is a main factor in the behavior of any distributed system. The performance of different entities in a SDP is very much affected by network delays as a main parameter. In our model for Jini-UPnP Bridge, we simulate network bandwidth by having network delay as one of the main model input parameters. Parameters are defined for unicast and multicast delays between any pair of nodes and also for the network as a whole. The following tests record the effect of varying network delays on the performance of UPnP-Jini Bridge.

In the pervious experiment we were interested in measuring the overhead of bridging a service in terms of time and number of messages. We fixed the TCP/IP network delay to a typical network delay value of 10-100 µs uniform. To measure the performance of the Jini-UPnP Bridge in a light loaded network, we repeat the experiment done in the previous section with the same input parameters, yet changing the TCP/IP network delay to **10-30 µs uniform.** The results would be compared to those obtain in the pervious section. We repeated the experiment ten times to compute the average overall time taken by the bridge.

Compared to the results obtained in the previous experiment, the bridge performance increases about 0.071 % with a less loaded network (i.e. higher bandwidth) of 10-30 µs uniform delay. The results show an improved value for the time of registration with the bridge from 1.00132 s in normal network to 1.000617 in a less loaded network. We are more interested in the last time value (**Overall Time)** since the time taken to download the Jini driver factory is a factor of it. The results are up to our expectations since an overall improvement in time delay is noticed.

To measure the performance of Jini-UPnP Bridge in a congested network, we apply the same experiment with a higher network load with the same input parameters, yet changing the TCP/IP network delay to **80-100 µs uniform.** The results would be compared to those obtain in case of typical network delays. We repeated the experiment ten times to compute the average overall time taken by the bridge.

Compared to the results in normal network condition that are obtained in the previous experiment, the bridge performance degraded about 0.034 % with a congested network (i.e. low bandwidth) of 80-100 µs uniform delay. The result is as expected since the effect of having a low bandwidth is of direct effect on the time taken to transfer messages and to download Jini driver factory. The overhead in time is more obvious in the time taken for registration with the bridge, as downloading the Jini driver factory file is a factor in it.

A UPnP client (UPnP SU), in a pure UPnP environment, is capable of discovering and communicating with multiple UPnP Services at the same time. Also, a Jini Service Manager (Jini SM) could advertise and register the availability of more than one service. Our UPnP-Jini Bridge is primarily composed of both a UPnP SU and Jini SM. Thus a UPnP-Jini Bridge is capable of bridging more than one UPnP service and registering it with the Jini SCM at the same time. We are interested in testing this capability of our modeled Jini-UPnP Bridge to bridge successfully multiple services at the same time and also to depict the effect of multi-service bridging on the Bridge performance.

In the previous experiment, we've chosen a topology with one UPnP Service (UPnP SM) that offered a Jini Factory and is bridged using the UPnP Jini Bridge. In this experiment, we conduct a topology of five UPnP SMs to be bridged, one UPnP Jini Bridge, one UPnP Service User, one Jini SCM, two Jini SUs and one Jini SM. We assume the same input delays and parameters presented above. We record the time taken for a Jini SU to discover a requested UPnP service. This time value is the sum of the time taken for Jini UPnP Bridge to discover the services; the time the bridge registers this service with Jini SCM and the time the Jini SCM forwards the service description to the interested Jini SU.

The results obtained are not uniform, yet they fall in a certain time range, for example the overall time taken by UPnP-Jini bridge to bridge a given service ranges between 1.00079 s and 1.00143 s. These results are expected since the behavior of the bridge is a function of the number of events it receives at the same time and the way it schedules the incoming events. The results fall in reasonable ranges and are close to the results obtained in case of bridging one service. These results are also dependent on the time each node starts announcing its service. Nodes that announce their services consecutively with a small time variant (e.g Nodes 2, 3), cause high frequency of events on the bridge, which results in degradation in the bridge performance and higher delay values.

CONCLUSION

The problem we addressed in this research is enabling thin servers and lightweight devices to offer their services to Jini clients through passive and indirect registration using our proposed Jini-UPnP Bridge. This problem has been addressed before by using SLP instead of Jini (Guttman and Kempf 1999), yet the bridging between Jini and UPnP has not been investigated before in SDP research literature.

We modeled and simulated our solution using Rapide ADL toolkit. Modeling is an approach for designing quickly, efficiently and correctly. It allowed us to control the quality and performance. We've chosen Rapide ADL to benefit from the set of modeling and simulation tools it offers. We used architectural models of Jini and UPnP as a basis to create hybrid discovery environment including both Jini and UPnP and to design and model our proposed bridge. For testing and simulating the bridge, we created a hypothetical topology of Jini and UPnP clients and services in addition to our proposed Jini-UPnP bridge. We simulated the topology to verify that the Jini-UPnP Bridge is capable of registering a UPnP Service that offers a JiniFactory, with the Jini Lookup service. The Jini-UPnP Bridge is tested for cases where the bridged service is updated or deleted. A number of performance experiments have been done on the bridge.

REFERENCES

ADL 1997, "Using Architecture Description Languages (ADLs) to Improve Software Quality and Correctness in Dynamic Distributed Systems" http://www.itl.nist.gov/div897/ctg/adl/sdp_projectpage.html

Bettstetter, C. and C. Renner, 2000, "A Comparison of Service Discovery Protocols and implementation of the Service Location Protocol", *In Proceedings of EUNICE 2000, Sixth EUNICE Open European Summer School, Twente, Netherlands*.

Dabrowski, C. and K. Mills, 2001, "Analyzing Properties and Behavior of Service Discovery Protocols using an Architecture-based Approach", *Proceedings of Working Conference on Complex and Dynamic Systems Architecture*.

Dabrowski, C. and K. Mills, 2002 "Understanding Self-healing in Service-Discovery Systems," *ACM Workshop on Self-Healing Systems*, Charleston.

El-Kharboutly, R. 2002, "Modeling Jini-UpnP Bridge Using Rapide ADL", M.Sc. thesis in Computer Science, The American University in Cairo.

Guttman, E. and J. Kempf, 1999, "Automatic Discovery of Thin Servers: SLP, Jini and the SLP-Jini Bridge," *Proc. 25th Ann. Conf. IEEE Industrial Electronics Soc. (IECON 99), IEEE*, Press, Piscataway, N.J.

IBM 1999, white paper, "Discovering Devices and Services In Home Networks".

Luckham, D. 2001, "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide .

Richard, G. 2000, "Service Advertisement and Discovery: Enabling Universal Device Cooperation," *IEEE Internet Computing*.

Wang, O. 2003, "Interoperability of COM/DCOM objects with CORBA objects by using DCOM/CORBA Bridge and their performance analysis", http://www.engr.sjsu.edu/fatoohi/wang-report/abstract.html