A Scalable Location Service Supporting Overload Situations

Luís Bernardo
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
INESC, R. Alves Redol 9 P-1000 Lisboa, Portugal
lflb@inesc.pt

Paulo Pinto
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
P-2825 Monte da Caparica, Portugal
pfp@uninova.pt

Abstract

This paper addresses the issue of defining a location service suitable for very dynamic and highly populated networks (millions of users), where services might experience highly correlated peaks of traffic or synchronized access to specific servers. A cooperative mobile agent solution is proposed to solve the major problems, allowing the dynamic deployment of new application servers when needed. But it requires an adequate location service to route the clients to the application servers, which scales to a large number of clients and still allows a high number of updates. This paper presents a very dynamic location service, which adapts to overload situations by modifying the routing information distribution and (possibly) its servers' internal structure. The responsive and fast route update results in a load re-distribution, which allows it to scale to a broader range compared to other alternative approaches.

I. Introduction

Traffic in future multi-service networks will be much more dependent on the location of the applications, or the way clients find servers, than the present situation of sparse servers at the edges of the network. There is today a growing widespread requirement for applications with guaranteed quality of service, used by an unpredictable and large number of clients, and which must be deployed in an affordable way. If the network routing capability is not accounted during application deployment then, server overload and network congestion can easily happen. Under these conditions, the routing of communications traffic is no longer simply a strict network problem (it needs to be taken into account at the application level).

A common characteristic of some applications can be the possibility of generating highly correlated peaks of traffic due to client interaction with the servers. Examples are easy to envision: applications based on interactive TV interfaces, where contests, promotional prices announcements or audience queries may synchronize the sending of requests to particular servers; real-time sport brokering; teleshopping; etc. The application technology will most likely rely on a highly variable server group to adapt to client load peaks, producing a non-static environment.

The scalability of such applications to a large number of users depends on the number of application servers available, and also on the bandwidth and on the system features on the overall server support. The location service is one of these features. It matches clients with server objects and might suffer from overload or become a critical point in the system.

This paper extends a previous work on a dynamic location service [1] by defining an adjusting mechanism of its structure in response to overload conditions. The following section provides the necessary background and section III presents the location service main characteristics. The location service architecture is described in section IV and section V describes the location service adjusting mechanism.

II. Background

When the world-wide network structure is analyzed, the tendency is to have very high bandwidths on local networks with their nodes densely connected, but interconnected more sparsely by core networks (with growing bandwidth available, supported by technologies like WDM and optical switches). On such networks, the bandwidth limitations and network delay will probably be originated on the core networks, in result of traffic correlation between local networks.

The availability of the applications on the network will depend on the number and distribution of the application servers, and on the "routing" of the application clients requests, which must interact with the nearest application server, using the minimum bandwidth channels at the core networks. Moreover, the application servers must be deployed near the clients, and their number must be able to support the client's load. Present solutions rely mainly on sets of static processing servers. Client requests are balanced using transaction processing monitors with load balancing facilities, or routers of IP anycast group addresses (used for instance, to implement replicated WWW servers [2]), etc. The common AI approach to the problem is to assign tasks to processors (ex. market oriented techniques [3], advanced contract net based co-ordination protocols [4], etc). However, new technologies such as mobile agent systems [5] and active networks [6] bring the possibility of creating and destroying servers in real-time, at any enabled node on the network. The mobile application servers are able to collect information about the network state, and adapt to machine and network load dynamically. New co-operation techniques are then needed, to select where and when to create or destroy application servers.

A new algorithm for controlling the deployment of application servers was proposed on [7][8] to deal with connectionless clients, which have atomic interactions with the servers. The application servers are able to decide when and where they create new servers, or when they destroy them by measuring their load, by storing a statistic map of the client's origins, and by running a co-operation algorithm with their neighbors. Simulation results proved the algorithm scalability, with bounded client service delay. The algorithm was extended in [9] to handle session oriented client-server interactions, where the allocation of connection links is also taken into account. The quality of a connection becomes the fourth aspect to take into consideration to control the deployment of application servers. Moreover, clients might be relocated during a session in result of a connection quality of service violation, originating the migration or replication of an application server.

However, the algorithm proposed in [7][8][9] introduces new requirements to the location service, which are not met by most of today's name services, directory services and routing services. Three main difficulties are:

(a) high variability introduced by the dynamic deployment of application servers;
(b) responsiveness required during server's overload (the client must know the availability of new servers quickly); and
(c) the global scale involving billion of nodes, where almost every gadget might be connected to the network and be involved in the applications (e.g. Jini technology [10]).

The application name resolution must depend on the application servers available, and evolve according to its distribution, balancing the load on the network according to its distribution. A new location service architecture was introduced in [1]. It proposed a new information structure, with enhanced features compared to the traditional one (hierarchical structure). However, the dynamic mechanism of its structure was disabled to focus on other aspects. This paper enhances the architecture by defining a dynamic location service control algorithm, which co-ordinates the location server's deployment and the dissemination of routing information.

III. Location Service Characteristics

From the application point of view, the location service acts as an interface between clients and servers, and between both clients and servers and the other network services (e.g. for selecting agent virtual machines). Some characteristics where added to the basic "name resolution" functionality.

Servers register their application specifying the range in the network where it must be known. The ranges are specified using a metric provided by the location service, based on geographical proximity and possibly on network state measurements. For instance, a car parking information application, used by the vehicles' computer to search for free spaces in car parks, would probably restrict its range to the neighborhood of the servers. Clients performing a lookup would catch the nearest servers. The main advantage of restricting the application routing information is the scalability of the location service, which deals with less unnecessary global information [1].

When more than one server registers a name, the location service balances the resolution of the name using two different approaches, depending on the distance to the servers. Near the server, the location service keeps server processing capacity and load information provided by the servers (e.g. 30% of occupancy) and uses it when resolving the name. For higher distances, the location service does the selection based exclusively on the "distance" criteria, or using a round-robin order, for similar distances. Therefore, it promotes the usage of local resources and reduces the core network traffic, improving the application scalability.

Clients may use two alternative lookup modes: a full lookup, which returns a server interface (if available), or a next step lookup, which returns a reference to an intermediate location service interface or to the server, depending on the distance. The second mode supports mobile agent clients, which may want to migrate to an intermediate node before running the application [8].

IV. Location Service Architecture

The location service was implemented using a distributed set of location servers (L-servers), which form a location network. Connectionless approaches (e.g. based on roaming scout agents [11]) do not guarantee a limited name resolution time for a new name.

The location service was designed to be fault resistant. It adapts automatically to the network structure, to network partitions and link's congestion. A dynamic structure was adopted, to allow the adaptation of the L-server's hierarchy to the state of the network. L-servers are implemented using mobile agents, and run on the Agent Virtual Machines (AVM) in parallel with the other application agents. The mobile agent semantics allows the dynamic creation, migration and destruction of L-servers. Each AVM runs a local object, the Local Location service Proxy (LLP), which provides an uniform local interface to the location service. The location network is structured on top of a LLP network, which defines the neighborhood relations between AVMs (see fig. 1). This LLP network is modified only by system administrator configuration, network faults or by the adding or removing of AVMs. Each LLP is associated with a first hierarchical level L-server, for publicizing the local servers and for searching for external ones. The LLP monitors the links to its neighbors and to the L-server, to detect faults. If a connection to the L-server is lost (for instance, due to a network partition), a voting algorithm is used to select an LLP, which will create a new L-server. Failures are also detect by the L-servers, resulting on an automatic reconstruction of the location network on each of the network's partitions.

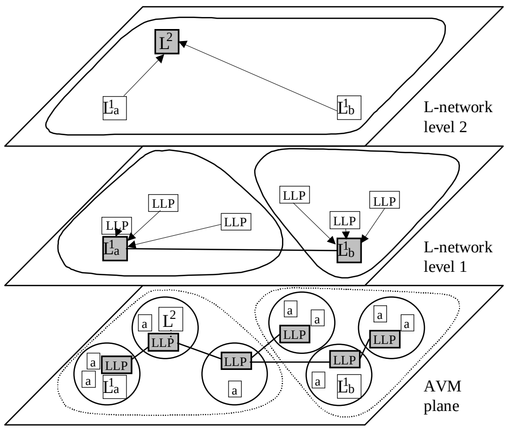

Figure 1: Network Model

Application names are resolved by performing lookups through a sequence of L-servers. The routing information for the search path is based on service hints. Hints are either the full application name (including the server interface reference) or incomplete information about the application (a pointer to an intermediate L-server, a hash value of the name) just to direct the search to another L-server. The further away from the server the more incomplete the hint becomes. Servers register their interfaces and names on the local LLP, which forwards it to the local L-server. The L-server will then disseminate service hints to other L-servers, possibly at various hierarchical levels, to create paths from every L-server on the requested range. A large range implies the use of higher levels to aggregate the necessary L-servers. The rationale is similar to other packet routing algorithms. However, a coordination protocol was introduced, to dynamically control the number of paths available between the L-servers.

The location service must provide the information about a new server very quickly, to allow clients to start using it. Consequently, no caches can be used. The alternative of using network-wide cache invalidation would be too complex and costly. This requirement (of not using caches) is not specific to our system, but generally applies to systems that balance load between replicated servers. For instance, for the CISCO DistributedDirector [2] operating on DNS mode, it is recommended that TTL must be set to zero. An alternative solution was also proposed at the GLOBE project [12]: to cache pointers to L-server's interface references. However, it may hide a new server, which appears nearer the client, reachable from an L-server on the short-circuited search path.

The location network does not use a pure hierarchical structure, where each L-server is only connected to lower-level L-servers or LLPs, and possibly to an upper-level L-server. On previous experiments [1] we showed that, if no caches are used, a large percentage of the requests would reach the L-servers at the top hierarchical levels, creating a processing bottleneck that would limit the maximum number of lookups (e.g. DNS with TTL=0). Even the solution of having differentiated root L-servers specialized on a subset of the names [12] may fail because the overloading resulting from a single application used by concurrent millions of users might be enough to overload the higher-level L-servers.

The location network is structured as a mixture of a meshed and a hierarchical structure where L-servers at each hierarchical level interact with some of the others at that level and (possibly) with one above. Higher hierarchical levels always have incomplete information about the available services, to reduce the update rate needed. Routing information (service hints) is disseminated between L-servers horizontally, at the same hierarchical level (possibly at more than one hierarchical level) and vertically to higher hierarchical levels. Horizontal dissemination involves L-servers sending information packets to their neighbors. The receivers process the packet, discard the elements within the offer-list, which are out of range, and forward it to the next L-server further away from the origin. Simulation results [1] showed that the cost of horizontal dissemination (in number of messages exchanged) is proportional to the number of L-servers existing on the requested range, and very high compared to the vertical dissemination. However, it can support many more client requests, due to the creation of multiple paths using different L-servers, whereas the alternative (vertical dissemination) creates a single path through a root L-server. A threshold for the maximum number of L-servers was defined to limit the dissemination costs of horizontal hint dissemination. This value delimits the maximum range, which can be supported by each hierarchical level (the distance to the furthest L-server). In consequence, for each application, the maximum possible hierarchical level required is defined by the requested application range.

V. Location Network Control Algorithm

The dissemination of service hints and the location network structure are dynamic and change due to three factors: the network-state, the names registered, and the lookup and registration load. An inter-L-server coordination algorithm is used to guarantee that the L-servers are enough to respond to the requests and to keep the routing information coherent during the internal modifications. The algorithm controls four main location service parameters:

- the horizontal hint dissemination (called the "Spread");
- the range where local service hints are replicated on neighbor L-servers (called the "Core" range);
- the number of L-servers at each hierarchical level;
- the number of hierarchical levels.

Spread. The horizontal dissemination of service hints on L-servers at lower hierarchical levels reduces the lookup load on the L-servers at higher hierarchical levels. For instance, in fig. 2, if the L^1_c´s spread for the illustrated server's application name includes L^1_b and L^1_d, then L^2_a only answers to searches coming from L^1_a or from outside L^2_a domain. If no horizontal dissemination were used, then every client searching for the application name from outside L^1_c domain would use L^2_a.

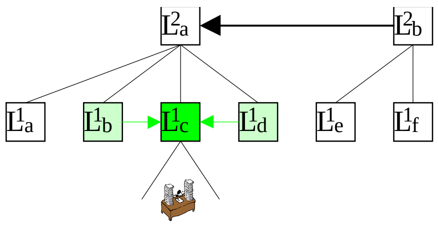

Figure 2: Spread parameter

Each L-server controls the horizontal dissemination for the immediately lower-level L-servers, using Spread Control Messages: an L-server may send a request to any of its lower-level L-servers to increment or reduce the spread on a set of names. Each receiver will test its maximum range and local load, and may refuse an increase if they are higher than the maximum values allowed. When a service hint is first disseminated, no horizontal dissemination is used, except for the maximum hierarchical level L-server required by the service hint. At that level, the service hint is disseminated horizontally on the server's requested range. The rationale is to deploy the structure with the lowest update overhead, yet adapted to the search load.

Core. The replication of a service hint on other L-servers reduces the lookup load on the original L-server. As fig. 3 shows, the lookups will be balanced between L^1_b, L^1_c and L^1_d. However, it also increments the service update overhead. All L-servers on the core range will disseminate the replicated service hints as their own. Another side effect is that the server processing capacity information of each individual service hint has to be updated on the L-servers, to avoid compromising the application client's load balancing.

Each L-server controls its core ranges, but co-ordinates the modifications with the neighbors. When an L-server receives a core update message, it may refuse to create a local service hint replica (because it could already be overloaded). Hence, the operation may fail.

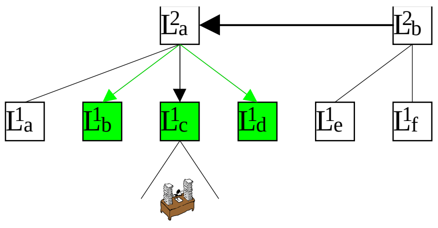

Figure 3: Core parameter

L-server replication. The creation of extra L-servers at the same hierarchical level reduces the lookup load at the neighbor L-servers. However, it will also increment the service hint update cost on the location network, reduce the maximum range supported at that hierarchical level, and in result, it may possibly create a new hierarchical level. This is so because there will be less low level L-servers per L-server. In consequence, it is used as a last resort to deal with lookup overload situations.

Top hierarchical level. The number of hierarchical levels on a location network depends on the ranges requested by the application servers. The maximum requested range must always be supported by some of the top hierarchical level L-servers. A new hierarchical level is created when a not supported range is requested (or when a range becomes not supported due to L-server replication). The downsizing is performed in the following way: a root L-server ceases to exist when it is inactive for more than a threshold time and does not have any service hints. New hierarchic levels allow the reduction of actualization costs for a restricted set of names. However, they produce longer resolution paths and support lower lookup load peaks.

Load adaptation algorithm. The first three location service parameters are controlled by a distributed co-operation algorithm run by all L-servers. L-servers monitor their local lookup load (for each name), determining if the lookups were routed from lower level or "near" neighbor (compared to the average distance of the neighbors) L-servers (FromDown), or if they were routed from higher level or "distant" L-servers (FromUp). They react when the average Load is outside an allowed variation range.

The adaptation reaction speed depends on the load measurement. If the algorithm responds too fast, it may create an unstable behavior when the load pattern has some kind of periodic variation, origination a high re-configuration overhead. Tests were made measuring the average load on fixed length intervals, and applying formula (1) (a modified discrete first order filter to attenuate the variation of the load measured). The coefficient α_i depends on the last measurement ($\mathtt{load}_n$) being higher (α_{up}) or lower (α_{down}) than the previous average ($\mathtt{Load}_{n-1}$). The oscillations were avoided by setting α_{up} greater than α_{down}, thus making the system respond slowly to load reductions.

(1) $\mathtt{Load}_n = \alpha_i . \mathtt{load}_n + (1-\alpha_i) . \mathtt{Load}_{n-1}$

When $Load_n$ is above the MaximumLoadThreshold an L-server tries to:

1. Increase the Spread on the lower-level L-servers (if FromDown);
2. Increase the Core (if FromUp);
3. Create an L-server replica if 1 and 2 failed and ($Load_n$ > "Minimum Replica Creation Threshold").

When $Load_n$ is below the MinimumLoadThreshold an L-server tries to:

1. Reduce Spread on the lower-level L-servers;
2. Reduce Core (if the resulting load (proportional to the modification on the number of replicas) is within the constraints;
3. Self-destroy if 1 and 2 failed and ($Load_n$ < Minimum Existence Threshold).

Before self-destroying, an L-server selects a neighbor, which will receive its lower-level L-servers or LLPs. The dying L-server (contractor) sends the neighbors a request for bids, and selects the L-server, which sent the best bid (less loaded and nearer). If any of the neighbors is overloaded, it cancels the self-destruction procedure. As a final stage, it sends a control message to the lower-level L-servers/LLP for modifying their upper-level L-server. The lookup load originated by the lower-level L-servers is accounted on further interactions of this algorithm by the contracted L-server, which increments its local load variables.

The location service parameters are also influenced by application server updates. If a server changes its location frequently, the spread and core parameters will be reset frequently to zero, and in consequence, the service hint dissemination is almost restricted to the vertical dimension.

Server migration or destruction handling. After server migration or destruction, L-servers disseminate service hint update packets to correct the routing information, creating a cancellation wave. If an alternative server is known, a temporary forward-chained pointer is created on the server's local L-server. The cancellation wave can then be completed as a low priority task, during low load periods, except on the region of the location service where the server's load and capacity is accounted. The location service clients must store the path of L-servers looked up. When a name lookup reaches a dead end (due to crossing a cancellation wave, for instance), the client must return to a previous looked up L-server.

Location Service Performance

The location service implementation was optimized for applications based on replicated servers, where peaks of concurrent client requests may occur. It achieves the best performance when the number of server replicas is high and servers are uniformly distributed on the network. Under these circumstances, service hints are available on the lower hierarchical levels, resulting on fast local searches and fast updates.

Nevertheless, the application adaptation algorithm requires also that the location system support peaks of requests concentrated on a network region. When an unpredictable large number of clients start to use the application and the number of initial servers is low (for instance one), all the clients from the initial peak will concentrate their name resolutions on a single L-server (where the initial application server registered its interface). A previous paper [8] showed that for a static hierarchy location server, if the bandwidth is large enough and the application server deployment algorithm is used, the application bottleneck would be the location service. The increment of the core parameter and the replication of L-servers allow the location service to respond to a peak of requests concentrated on a single L-server on the network.

The proposed location service implementation is less effective if static servers (which do not move) use it or if the name is searched by a diminutive number of clients (which do not require load balancing). For instance, when the location service is used for tracking the location of a mobile object. The dissemination and the name resolution overheads are higher under these conditions. Better alternative approaches are the caching of the previous resolutions, or the assignment of "home" L-servers for the resolution of names (e.g. DNS round robin assignment [13], uniform resource names [14]). However, these solutions are not reliable in the case of a server failure (because of the cached interface references) or in the case of the "home" L-server failure.

Simulation Results. A simulator was developed using the "Discrete Event" model on the Ptolemy system [15], which implements the location service protocol. The simulation presented in this paper compare the name resolution delay of a static hierarchical location service (without caches) and of the dynamic location service proposed. All simulations were conducted with a network of 625 AVMs, where each AVM has an average of 3 connections to its neighbors, with a maximum distance of 24 (AVM) hops. Three hierarchical levels were used for the static service with 125 L-server at the first level, five L-servers at the second level and a single root L-server at the third level. The dynamic network was initialized with the same configuration at the beginning of the simulations. The average time to process location service lookups at each L-server is 40 tics and the transmission time between nodes was set to one tic. Client load was constant (0.0625 new name searches per tic) and symmetrical. Three initial servers where registered on three AVMs in near symmetrical regions of the network. On the dynamic network case, L-servers measured the processor utilization time during intervals of 10000 and 1000 tics, and tested the average load after each measuring interval (using 75% for α_{up} weight and 50% for α_{down}). The Maximum and Minimum Load Thresholds were respectively 95% and 10%. The graphic of figure 4 shows the evolution during the simulations of the average name resolution time measured in intervals of 5000 tics.

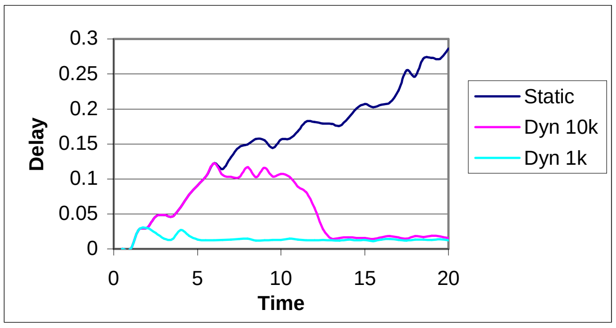

Figure 4: Average Name Resolution Time for a static location network and for a dynamic location network with load measurement periods of 10000 and 1000 tics.

On the conditions simulated, the first level L-servers where the servers registered the name can deliver the requested load (they support 0.025 queries per tic and the load is distributed by three L-servers). However, the static version is overloaded (the root L-server is overloaded). Both dynamic location services reconfigured by setting a spread of six AVM hops on three second level L-servers, plus a temporary core update at the local servers' location server (in result of a peak of requests which followed the spread modification and some asymmetry in the distribution). The dynamic network with the lowest load measurement interval reacts faster, in result of the higher measured load update frequency.

Conclusions

The scalability of applications on large network is strongly related to the responsiveness of the location service and its ability to distribute the load between all the server replicas according to the network state. This paper presents a scalable location service architecture, suitable to very large systems with mobility and the possibility of handling overload situations. We show that a dynamic structure based on intelligent L-servers, allows a faster and more scalable response, compared to a static structure (the simulation proves it). The trade-off is the need to exchange information inside the service. However, this is not so critical because application servers adapt to client load and most of the relevant lookups tend to be local.

On-going work includes (a) a thorough study about the dynamics of the interaction between the application dynamic deployment algorithm and the location network adaptation algorithm, and (b) the trade-off between application server clone creation and the cost of maintaining consistency of application data ("low-cost" inter-replica state synchronization techniques).

Acknowledgments

This research has been partially supported by the PRAXIS XXI program, under contract 2/2.1/TIT/1633/95.

References

[1] Bernardo, L.; and Pinto, P. 1998. A Scalable Location Service with Fast Update Responses. In Proceedings of IEEE Globecom'98, 2876-2881. Sydney, Australia: IEEE Press.

[2] Delgadillo, K. 1999. Cisco DistributedDirector. Cisco White Paper. http://www-europe.cisco.com/warp/public/751/distdir/dd_wp.htm

[3] Mullen, T.; Wellman, M.P. 1995. A simple Computational Market for Network Information Services. In Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS'95) 283-289. San Francisco.

[4] Chavez, A.; Moukas, A.; Maes, P. 1997. Challenger: A Multi-agent System for Distributed Resource Allocation. In: Proceedings of the International Conference on Autonomous Agents. Marina Del Ray, California.

[5] Rothermel, K.; and Hohl, F. eds. 1998. Mobile Agents. LNCS 1477: Springer.

[6] Tannenhouse, D.; and Wetherall, D. 1996. Towards an Active Network Architecture. ACM Computer Communication Review 26(2):5-18.

[7] Bernardo, L.; and Pinto, P. 1998. Scalable Service Deployment on Highly Populated Networks. In Proceedings of the 2nd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'98), 29-44. Paris, France: Springer LNAI Vol.1437.

[8] Bernardo, L.; and Pinto, P. 1998. Scalable Service Deployment using Mobile Agents. In Proceedings of the 2nd International Workshop on Mobile Agents (MA'98), 261-272. Stuttgart, Germany: Springer LNCS Vol. 1477.

[9] Bernardo, L.; Crespo, D.; Marques, A.; and Pinto, P. 1999. Scalability Issues in Telecommunication Services. In Proceedings of 2nd Conference on Telecommunications (ConfTele'99), 409-413. Sesimbra, Portugal. Available in: http://mariel.inesc.pt/~lflb/ conftele99.pdf

[10] Waldo, J. 1998. Jini^{TM} Architecture Overview. White Paper. http://java.sun.com/products/jini/whitepapers/architectureoverview.pdf

[11] Bonabeau, E.; Henaux, F.; Guérin, S.; Snyers, D.; Kuntz, P.; Theraulaz, G. 1998. Routing in Telecommunications Networks with Ant-Like Agents. Proceedings of the 2nd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'98), 60-71. Paris, France: Springer LNAI Vol.1437.

[12] Steen, M. van; Hauck, F.; Tannenbaum, A. 1996. A Model for Worldwide Tracking of Distributed Objects. Proceedings TINA '96 Conference, 203-212. Heidelberg, Germany.

[13] Gulbrandsen, A.; and Vixie, P. 1996. DNS RR for specifying the location of services (DNS SRV). IETF RFC 2052.

[14] Sollins, K. 1998. Architectural Principles of Uniform Resource Name Resolution. IETF RFC 2276.

[15] Ptolemy home page. http://ptolemy.eecs.berkeley.edu