

A Taxonomy for Resource Discovery

Koen Vanthournout, Geert Deconinck, Ronnie Belmans

Katholieke Universiteit Leuven, Belgium,
Department Electrical Engineering (ESAT),
e-mail: Koen.Vanthournout@esat.kuleuven.ac.be,
<http://www.esat.kuleuven.ac.be>

The date of receipt and acceptance will be inserted by the editor

Abstract Resource discovery systems become more and more important as distributed systems grow and as their pool of resources becomes more variable. As such, an increasing amount of networked systems provide a discovery service. This paper provides a taxonomy for resource discovery systems by defining their design aspects. This allows comparison of the designs of the deployed discovery services and is intended as an aid to system designers when selecting an appropriate mechanism. The surveyed systems are divided into four classes that are separately described. Finally, we identify a hiatus in the design space and point out genuinely distributed resource discovery systems that support dynamic and mobile resources and use attribute-based naming as a main direction for future research in this area.

1 Introduction

A large and growing number of modern computing systems are composed of a diverse multitude of networked components, that cooperate to accomplish the application's targets. Examples are: peer-to-peer file sharing networks, GRID computing, the ambient environment [13], LAN plug-and-play, etc., or: Gnutella [14], CAN [25], Globus [9], CORBA [15], UPnP [8], Jini[4], etc¹.

The enabling technology for all these systems is resource sharing. But before resources can be shared, potential users must have the ability to locate them. This can be accomplished by either manual configuration or discovery at-runtime of the resources (from here on named *resource discovery* (RD)). The larger the system grows, the more cumbersome manual configurations are.

¹ A complete list of all systems that were surveyed for this paper can be found in Table 2 (legend in Table 3). Mind that this list is not exhaustive and only includes IP-based resource discovery systems.

And if the resource pool is variable and not static, manual configuration is ruled out completely. Since the current trend is towards such larger, complex and more variable systems, resource discovery becomes an ever more important service, worth special attention.

Because of its indispensable nature, all current networked systems that contain some level of self-configuration, already provide a RD service. For instance, a GRID-client's computational task is automatically distributed to a varying pool of available processing units, in a plug-and-play environment an arriving laptop automatically finds the local printer, a file is located in a peer-to-peer network where nodes continuously enter and leave, etc. Although the resource discovery systems used in these examples pursue comparable tasks, the diversity of the applied strategies is as large as the different types of applications that require it.

The aim of this paper is to provide a taxonomy for the different strategies that are used to tackle IP-based resource discovery. This is accomplished by defining the design space of RD systems. After clarifying the terminology, used in this paper, the relevant design aspects and the values they take are discussed (Sect. 3). This is followed in Sect. 4 with an overview of the large clusters of existing RD systems and the indication of the hiatuses in the design space.

2 Terminology

To understand resource discovery, one must first clearly understand what is meant by 'resources'. The dictionary gives us following definition:

A source of supply, support, or aid, esp. one that can be readily drawn upon when needed.

We discuss resources in the context of resource discovery as an enabling step for networked resource sharing. Thus, we can transform above definition to:

Definition 1 A resource is any source of supply, support, or aid a component in a networked environment can readily draw upon when needed.

Examples are: files, measurements, CPU-cycles, memory, printing, control-devices, forums, online shops, etc. This definition requires further specification, since different systems support different types of resources. A classification of resources is presented in Sect. 3.7.

Definition 2 Resource discovery is the ability to locate resources that comply to a set of requirements given in a query.

Sect. 3 elaborates on the available mechanisms for resource discovery.

The process of resource discovery involves three main actors: resource providers, resource users and the RD service itself.

- **Definition 3** A resource provider is any networked entity that allows sharing of its resources.
- **Definition 4** A resource user is any networked entity that uses shared resources.
- **Definition 5** A resource discovery service is the service that returns the location of matching resources in response to a query with requirements.

Multiple types of actors can be exerted by one single entity in a system, e.g, a resource provider can at the same time be resource user or a discovery service can be implemented by the resource users and providers themselves, without third party interference (see Sect. 3.1). It always remains possible, however, to logically divide a single entity into the actors it's composed of.

The term *node* is used throughout this paper to indicate a network-enabled device. Such a device can contain any of the above actors.

3 Relevant Design Aspects

Resource discovery systems can be categorized by different design aspects. Each of these aspects represents a design choice for which several solutions are possible. None of these solutions can be entitled as 'the best' solution. Depending on the envisioned target application and environment, different choices may prove optimal. Therefore, a classification into a design space will not allow an absolute ranking, but rather permits the identification and comparison of clusters of RD systems with equal targets. It is also an aid to the designers of new systems and helps identifying hiatuses and potential new approaches. A list of the design aspects can be found in Table 1.

3.1 The Service Provider

A designer is presented with two options when selecting how to provide a RD service. First choice is to implement

Table 1 resource discovery design aspects

| design aspect |
|---------------------|
| service provider |
| construction |
| foreknowledge |
| architecture |
| registration |
| discovery |
| supported resources |
| naming and queries |

it as a *third party* service, i.e., as an entity, distinctly separate from the resource providers and users. This usually takes the form of a server or collection of servers that gather information on the available resources and use this information to respond to queries of users. The third party type of RD system is the most common one. It is the traditional well-established way to provide services and allows deterministic and centrally controllable systems. Examples of systems like these are: DNS [19], CORBA [15], Napster [22], etc. (see Table 2 for a full list and Table 3 for the legend of Table 2). It does occur however, that there is no organization willing or allowed to take the responsibility of providing a centralized RD service. An infamous example is mp3 file sharing, but also GRIDs with resources owned by various administrative organizations [18] and ambient intelligence applications [13] fall into this category. The alternative to a third party service is a *genuinely distributed* system, i.e., the RD service is distributed across all involved resource providers and users, without any central or coordinating components. Examples of genuinely distributed systems are: Gnutella [14], Freenet [6], Tapestry [31], CAN [25], etc. Mind though, that also third party systems can be distributed, although these systems act as a single entity to the outside world. DNS, LDAP [17] and CORBA are only a few examples.

A final note must be made on systems like Salutation [7] and Jini [4], since these are systems that support both models: in the presence of a RD server, this server provides a third party service, but in absence of such a server, the system automatically switches to genuinely distributed operation (more on this subject in Sect. 3.2).

3.2 Construction

A networked distributed system constructs an overlay network on top of the actual communication layer. This overlay network is a directed graph: vertices represent network nodes and an edge represents the knowledge of a node about another node. Two alternatives exist for the construction of such an overlay network: they can be manually configured or they can grow by means of self-organization.

- **Manually Configured Networks:** Third party systems that extend beyond the LAN-level are usually manually configured, i.e., a human administrator is responsible for providing each server with the location and function of the other components and users and providers must know the address of a server. Or, put differently, a human administrator must design the overlay network graph: the different components must be informed of the edges that leave them. While manual configuration allows more control and deterministic behavior, it is cumbersome to scale. The large organizational and human resources required to maintain the DNS system can serve as an example.
- **Self-organizing Networks:** Self-organizing systems become interesting when the size of the network is large, if human configurations are too expensive or if no central authority is available to coordinate configuration. Drawback is the increased network traffic, required for the self-organization and maintenance of the overlay net and the complexity of the algorithms. Examples are Gnutella, Freenet, CAN, etc.
- **Hybrids:** Also hybrid solutions are possible. This can be a means to add redundancy to a system and to increase its robustness. Examples are Salutation and Jini, where the absence of a (manually configured) RD server doesn't imply system failure, since the resource providers and users will automatically self-organize into a complete graph (see Sect. 3.4), though this increases the network load significantly. Another class of hybrid systems are those that use multicast or broadcast to self-organize on LAN-level and manual configuration to interconnect these self-organized subgraphs at WAN level (e.g., Ninja SSDS [10] and Uniframe [27]).

3.3 Foreknowledge

Most systems, even self-organizing, require some configuration per node, prior to its entrance in the system. The information contained in this node specific configuration is referred to as *foreknowledge* and is closely related to the method of overlay network construction:

- Nodes in manually configured systems require a list of all components they need to interact with. The foreknowledge composes of a list of *well-known addresses*.
- Self-organizing systems that operate at WAN level, usually require at least the address of one *random active node*. Using the knowledge of the overlay network edges leaving this node, the new node can find its location. Many require on top of this a *unique identifier*, though this can often be avoided by using a hash function on the node's network-address.
- Self-organizing systems that employ multicast or broadcast usually need *no foreknowledge*, but are limited to operate at LAN level.

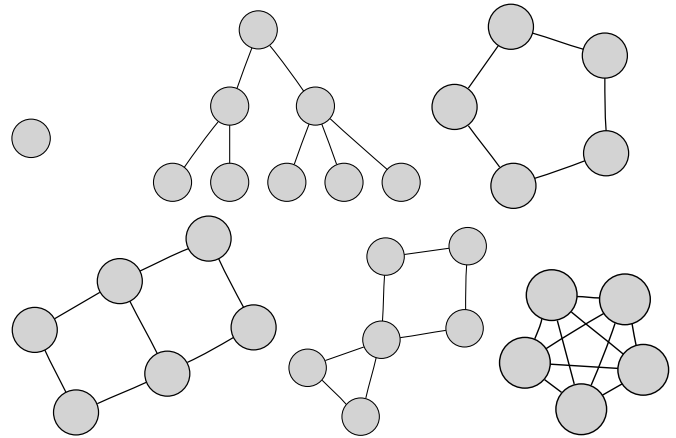


Fig. 1 Examples of the different overlay network architectures (from left to right, top to bottom): A trivial graph, a tree graph, a ring-shaped regular graph, a 2-dimensional Cartesian regular graph, a random graph and a complete graph.

3.4 Architecture

The nodes of a distributed RD system organize into an overlay network, as mentioned before. The architecture of these overlay structures can be visualized by graphs and thus can be categorized by using graph theory [12, 2] (see Fig. 3.4 for graphical representations):

- **Trivial Graph:** A graph of order 0 or 1 (0 or 1 vertices), is a trivial graph. Order 0 graphs are logically of no interest to us. Trivial graphs of order 1, on the contrary, are the purely centralized systems. The RD system consists of a single server, registering all resources and answering all queries. These systems may have scaling difficulties and are prone to single points of failures. Mind that trivial graph servers may still be internally duplicated. Trivial means here that the server has a single point of access.
- **Tree Graph:** Systems organized as trees scale well and allow for backtracking as a search algorithm, but high level node failures affect large portions of the system. They are especially suited for directory systems, e.g., DNS, LDAP, etc. P-Grid [1] is an example of a self-organizing system that constructs a tree graph. To overcome the problems associated with high level failures, P-Grid uses node redundancy.
- **Regular Graph:** If nodes are structured in an ordered lattice and if adjacent nodes are interconnected, the graph is regular. (See [2] for how to measure regularity by using clustering coefficients). Self-organizing networks that grow into a regular structure allow

for optimized search algorithms²(e.g., Plaxton routing [23], used by Tapestry).

- **Random Graph:** Systems exhibiting random graph behavior include the manually configured systems, where administrators can interconnect nodes as they please, e.g., CORBA, and self-organizing networks where the number and nature of links is not predefined, e.g., Gnutella and Freenet. It should be noted, though, that many networks that seem completely random at first sight, do have an emergent structure: they are scale free networks [2]. The distribution of the number of links per node of scale free networks follows a power law. Both Gnutella and Freenet exhibit this behavior. Main consequences are an increased resilience to random failures and shorter path lengths, but also a larger susceptibility to directed attacks.
- **Complete Graph:** In this case all nodes know each other. Multicast and broadcast systems without central server (e.g., Jini) and replicated server RD systems (e.g., NetSolve [5]) are examples.

3.5 Resource Registration

Before requests for resources can be made, a reference to them must be stored on a place where these references can be predictably accessed. The different techniques used are:

- **Local registration only:** Only the resource providers are aware of the resources they share. This implies that a resource can only be located by immediate interrogation of the provider. Clearly, this is an inefficient method. If only replicable resources are offered (see Sect. 3.7), improvements can be made by replication and caching (e.g., Freenet).
- **References:** References to the location of a resource are stored on a predictable place in a regular graph structure. Typically this is realized by using hash identifiers for both nodes and resources. References are then stored at the node whose identifier matches the resource's identifier closest.
- **Registration at the local server:** Resource providers announce their resources to the local server (in case of trivial graphs to the only server). This information may then be stored only locally or locally and at replicated servers.
- **Manual registration:** Resources are registered at RD servers by means of configuration files. This is only useful for long-term stable resources, e.g., URL names stored at DNS servers.

² Notable also is 'small world' behavior [30,2]: if nodes in a highly regular structure maintain few links to far parts of the lattice, the average path lengths drop strongly, while the regularity is still preserved. Chord, for instance, uses this technique to optimize search.

3.6 Query Routing

If a resource user wants to access a resource and thus first needs to locate it, a query with the user's requirements is composed and sent to that user's local contact point to the RD system. The latter is the local server or, in case of a genuinely distributed system, the RD module on the node itself. This query must then be routed in the graph structure towards the node that knows the location of a matching resource. The strategies are:

- **Central server or local replicated server:** In case of a single local server or a replicated server infrastructure, routing is limited to the single query sent from resource users to the local server, where the information is readily available.
 - **Query forwarding:** if a node is located in a graph and it fails to match a query, it must forward the query to one or more nodes, based on some metric or rule. For instance, Freenet uses hash identifiers to select the next node a query will be forwarded to (caches enlarge the probability of this being a successful strategy). The most popular query techniques are:
 - **Flooding:** Used by some systems that deploy a random graph structure, flooding burdens the communication network heavily and requires limited hop-distance settings and special precautions for loops. Yet flooding guarantees that the node with the requested knowledge will be reached by the shortest path. The best-known system with flooding is probably Gnutella, but also the CORBA traders use it. Multicast or broadcast LAN-discovery systems that operate without server, can also be included in this category.
 - **Backtracking:** The single biggest advantage of tree graph structures is that they allow backtracking search algorithms.
 - **Regular structure routing:** The effort of implementing the complex algorithms that allow nodes to self-organize into a regular lattice has but one goal: predictable locations of resource references and a method to route towards those locations. A multitude of techniques are used for this purpose (see the footnotes to Table 2).
- Almost all used query forwarding strategies fall into one of the above mentioned sub-categories. An overview of alternative query forwarding algorithms for RD systems can be found in [18].

3.7 Supported Resources

As defined in Sect. 2, resources include everything that could be useful for an entity within a networked system that requires resource sharing and thus resource discovery. This includes both services and information. But

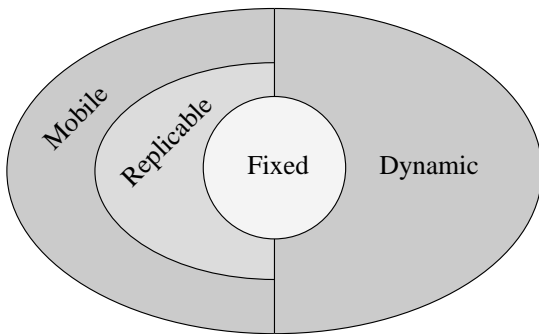


Fig. 2 The classes of resources

services and information are not clearly delineated categories of resources, since they often overlap. E.g., a measurement device provides a service that delivers information. Better is to sort them in classes by rate of change (see Figure 2):

- **Fixed resources:** These are resources that have both a fixed location and fixed properties. Examples are network printers with fixed properties (color, resolution, etc.), weather stations with a static set of measurement devices, etc.
- **Replicable resources:** Basically, replicable resources are fixed resources combined with replicated files. In the case of replicated files, each file will have a single set of identifiers and the resource discovery service will return the location of one of the copies.
- **Mobile resources:** The class of mobile resources contains resources with a fixed location, replicable resources and resources with a variable location. Examples of the latter are laptops, wireless devices, etc.
- **Dynamic resources:** Dynamic resources are resources that have a fixed location, but whose identifiers can vary. Servers or PC's that offer their (idle) computational resources are an example: the current load and the available amount of CPU time, memory and disk space vary.
- **Mobile & dynamic resources:** This class embraces all resources, delineated above.

Some of the surveyed systems target a specific set of resources, but could include other types with few or even no adaptations. Nevertheless, the values in Table 2 refer to the resources the system was specifically designed for and not to what could be included.

3.8 Resource Naming and Queries

References to resources are composed of two parts: the location and the name of the resource. Mind that 'name'

could be as large as an XML file. Queries for resources are matched to these names. Consequently, the used naming mechanism defines largely the types of resources that can be served by the RD system and the ease by which those resources can be found. Compare the cumbersome task of locating resources (e.g., web pages) by means of clear text boolean queries to the deterministic alternative of assigning values to a selection of predefined attributes.

- **Unique identifiers and hashes:** If a resource has a unique name, it is easier to locate. And, as shown before, such an identifier can be used to define and retrieve a fixed node where a reference to the resource's location can be stored. Identifiers like these are usually obtained by using a hash function on the clear text name of the resource. Its main problem is that this system does not allow for changing properties to be reflected in the naming system and thus excludes dynamic resources. Or at least, dynamic properties of resources are not reflected in the RD system.
- **String naming:** While hashes allow no search on specific terms in the resource name/description, clear text naming does allow more complex queries: boolean expressions, 'sound as' queries, etc.
- **Directories:** Directory naming systems build on scalable hierarchical name spaces, with as most famous examples DNS and LDAP. Most RD systems with directory naming employ the DNS or LDAP syntax or a variant on one of these two.
- **Attributes:** The most powerful naming system is the use of attributes: resources are described by means of a number of predefined attributes that take a value. Attributes allow for extensive queries with predictable results, as opposed to ad-hoc queries used with clear text naming. Attributes can be used for the full range of resources, even for descriptions of dynamic resources. Note that the most commonly used attribute system is XML (used by UPnP, Ninja SSDS, UniFrame, UDDI, etc.).

4 The Main Classes of RD Systems

Table 2 gives an overview of the RD systems that were surveyed for this paper and lists the values they take for the several design aspects. The legend to Table 2 can be found in Table 3. Table 3 also serves as an overview of the design aspects and their values. This list of RD

Table 2 Resource discovery systems and their properties (legend in Table 3), sorted by class (from top to bottom): P2P RD systems, multicast discovery RD systems, distributed third party RD systems and centralized RD systems

| <i>Name</i> | <i>Prov</i> | <i>Constr</i> | <i>Forekn</i> | <i>Arch</i> | <i>Res Reg</i> | <i>Routing</i> | <i>Sup Res</i> | <i>Naming</i> |
|--------------------------------|-------------|---------------|---------------|------------------|----------------|----------------|----------------|---------------|
| <i>Chord</i> [11] | GD | SO | ID+RN | Reg ³ | Ref@N | Route | Repl | Hash |
| <i>Tapestry</i> [31] | GD | SO | ID+RN | Reg ⁴ | Ref@N | Route | Repl | Hash |
| <i>Pastry</i> [26] | GD | SO | ID+RN | Reg ⁵ | Ref@N | Route | Repl | Hash |
| <i>CAN</i> [25] | GD | SO | RN | Reg ⁶ | Ref@N | Route | Repl | Hash |
| <i>P-Grid</i> [1] | GD | SO | RN | Tree | Ref@N | BT | Repl | Hash |
| <i>Freenet</i> [6] | GD | SO | RN | Rand | Res@N | QF | Repl | Hash |
| <i>Gnutella</i> [14] | GD | SO | RN | Rand | None | Flooding | Repl | String |
| <i>Salutation</i> [7] | Both | SO | None | Compl | Node/Serv | Flood/Serv | Mob | Attr |
| <i>Jini</i> [4] | Both | SO | None | Compl/Triv | Node/Serv | Flood/Serv | Mob | Attr |
| <i>UPnP</i> [8] | 3P | SO | None | Triv | Serv | Serv | M/D | Attr |
| <i>SLP</i> [16] | 3P | SO | None | Triv | Serv | Serv | Mob | Dir |
| <i>Ninja SSDS</i> [10] | 3P | SO/Man | No/WKAddr | Tree | Loc Serv | BT | Mob | Attr |
| <i>UniFrame</i> [27] | 3P | SO/Man | No/WKAddr | Rand | Loc Serv | Flooding | Fix | Attr |
| <i>NetSolve</i> [5] | 3P | Man | WKAddr | Compl | Loc Serv | Repl Serv | Dyn | Attr |
| <i>UDDI</i> [28] | 3P | Man | WKAddr | Compl | Loc Serv | Repl Serv | Fix | Attr |
| <i>RCDS</i> [20] | 3P | Man | WKAddr | Compl | Loc Serv | Repl Serv | Repl | Dir |
| <i>CORBA</i> ⁷ [15] | 3P | Man | WKAddr | Rand | Loc Serv | Flooding | Dyn | Attr |
| <i>Globe</i> [29] | 3P | Man | WKAddr | Tree | Loc Serv | BT | Mob | Hash |
| <i>DNS</i> [19] | 3P | Man | WKAddr | Tree | Man | BT | Fix | Dir |
| <i>LDAP</i> [17] | 3P | Man | WKAddr | Tree | Man | BT | Fix | Dir |
| <i>Matchmaking</i> [24] | 3P | Man | WKAddr | Triv | Serv | Serv | Dyn | Attr |
| <i>Napster</i> [22] | 3P | Man | WKAddr | Triv | Serv | Serv | Repl | String |
| <i>SuperWeb</i> [3] | 3P | Man | WKAddr | Triv | Serv | Serv | Dyn | Attr |
| <i>Ninf</i> [21] | 3P | Man | WKAddr | Triv | Man | Serv | Dyn | Attr |
| <i>Globus</i> ⁸ [9] | 3P | Man | WKAddr | Triv | Man | Serv | Fix | Attr |

systems in Table 2 is not exhaustive, but contains the four classes presented here.

4.1 Centralized Third Party RD Systems

Centralized RD systems are the systems that have a unique contact point for their users. They are *manually configured* systems with a *trivial graph*. The distinct advantage of centralized RD systems is that resource registrations, updates and queries require no (time consuming) routing, which is reflected in the resources they support: this class represents the largest portion of the

³ The regular graph for Chord is built from a circular identifier space.

⁴ The regular graph for Tapestry is built from a Plaxton Mesh [23].

⁵ The regular graph for Pastry is built from a circular identifier space.

⁶ The regular graph for CAN is built from a d-dimensional Cartesian identifier space.

⁷ The CORBA object implementing its resource discovery is the Trading Object Service.

⁸ The component of GLOBUS that provides the RD service is the MDS (Metacomputing Directory Service). The MDS builds on LDAP and enhances it with attribute-based search.

attributes using systems that support dynamic resources. Immediate drawback is scalability and the single point of failure created by the trivial graph.

4.2 Distributed Third Party RD Systems

The single point of failure in centralized RD systems can be overcome by replication and multiple contact points to the system. Scalability can be improved by spreading the resource references on several servers. Together, these systems make the class of distributed third party systems (identified by requiring *manual configuration* and by a *graph of larger complexity than trivial*). Long established and offering the most straight forward billing, these systems are well-developed. This is reflected in their diversity. Indeed, many combinations of techniques are available and together they cover all types of resources.

4.3 Multicast RD Systems

The use of multicast (or broadcast) offers a clean and straightforward method to implement resource discovery with minimal administration effort (*no foreknowl-*

Table 3 Legend to Table 2

| | | | |
|----------------|-----------------------------|---|--|
| <i>Prov</i> | Service Provider | <i>GD</i> <i>3P</i> <i>Both</i> | Genuinely Distributed Third Party <i>3P</i> , if absent <i>GD</i> |
| <i>Constr</i> | Construction | <i>SO</i> <i>Man</i> <i>SO/Man</i> | Self-Organizing Manual configuration @LAN level: <i>SO</i> , @WAN level: <i>Man</i> |
| <i>Forekn</i> | Foreknowledge | <i>RN</i> <i>ID+RN</i> <i>WKAddr</i> <i>No/WKAddr</i> | Random Node Unique Identifier and <i>RN</i> Well-known Addresses @LAN level: <i>None</i> , @WAN level: <i>WKAddr</i> |
| <i>Arch</i> | Architecture | <i>Triv</i> <i>Rand</i> <i>Tree</i> <i>Reg</i> <i>Compl</i> | Trivial graph Random graph Tree graph Regular graph Complete graph |
| <i>Res Reg</i> | Resource Registration | <i>None</i> <i>Ref@N</i> <i>Res@N</i> <i>Serv</i> <i>Loc Serv</i> <i>Man</i> | Local registration only Reference at node with closest ID Resource cached at nodes with a closer ID Registration at unique server Registration at local server Manual configuration |
| <i>Routing</i> | Query Routing | <i>Flooding</i> <i>BT</i> <i>Route</i> <i>QF</i> <i>Serv</i> <i>Repl Serv</i> <i>Flood/Serv</i> | Flood query Backtracking Route to node with closest ID Query forwarding Query central server Query local replicated server if GD: <i>Flooding</i> , if 3P: <i>Serv</i> |
| <i>Sup Res</i> | Supported Resources | <i>Fix</i> <i>Repl</i> <i>Mob</i> <i>Dyn</i> <i>M/D</i> | Fixed resources Replicated resources Mobile resources Dynamic resources Mobile & Dynamic resources |
| <i>Naming</i> | Resource Naming and Queries | <i>Hash</i> <i>String</i> <i>Dir</i> <i>Attr</i> | Hash ID String naming and queries Directories Attributes |

edge is needed) and is especially useful if mobile resources are to be handled. Its use, however, is limited by its very enabling technology. The system's size is restricted to multicast supporting LANs as internet-wide multicast is no option yet. Ninja and UniFrame overcome this problem by combining it with a distributed third party approach. However, the drawback of this solution is the manual configurations that are associated with third party RD systems. Combination of a genuinely distributed approach (see Sect. 4.4) with multicast would be an interesting approach to overcome this, but, as far as the authors know, no such system exists.

4.4 P2P Systems

Genuinely distributed systems are commonly referred to as peer-to-peer systems (P2P)⁹, though this might be a confusing term. Nevertheless, P2P is widely accepted and recognized and thus will be used in the remaining of this paper. Two classes of P2P are available:

- **Random P2P RD Systems:** The first class of P2P systems contains systems like Gnutella and Freenet. Using no fixed rules and varying metrics, nodes build a list of other nodes, starting from the location

⁹ Although Napster is commonly accepted as a P2P system, it is not included here. Napster does use peer-to-peer file transfer, but the RD mechanism is a centralized one.

of a random active node. Result is a random overlay network. Efficient and/or deterministic search strategies are not feasible on this architecture and they are limited to replicable resources only.

- **Regular P2P RD Systems:** By using a regular structure to store (hash-based) references to resources at predictable spots in this structure, regular P2P RD Systems allow for more efficient and deterministic search algorithms, hereby overcoming the main problem of random P2P RD systems. They too, are limited to replicable resources, though.

Hiatuses The immediate advantage of P2P systems is the absence of a need for a coordinating authority, a minimal required configuration effort, improved scalability and no dependency on third parties (and their servers). But unfortunately, P2P systems are limited to replicable resources; they focus on file sharing. The lack of genuinely distributed systems that support mobile and dynamic resources and use attribute naming is a true hiatus in the design space. The algorithms for self-organization and query forwarding, required for their construction, should become subject of intensive research. Notable is the initial work of Iamnitchi et al. [18] in this area.

5 Conclusions

The presented taxonomy for resource discovery systems is based on their design space that consists of eight design aspects, which allows categorization and comparison of RD systems. It is intended as an aid to the designers of new RD systems and as a means to facilitate the selection of a suited RD system for new distributed applications: all too often, RD systems with almost identical properties have been re-designed and -implemented.

Four main classes of RD systems have been identified: centralized, distributed third party, multicast discovery and P2P RD systems. While centralized and distributed third party systems are well established and cover all types of resources and naming, multicast discovery systems require additional development to operate at WAN level without loss of their strongest advantage: the lack of any required foreknowledge. Potentially, the combination with P2P technology could offer a solution.

The largest gap in the design space, however, is the limited types of resources that are supported by P2P systems. Genuinely distributed RD systems that support dynamic and mobile resources and use attribute-based naming require increased attention from the research community.

Acknowledgements

This work is partially supported by the K.U.Leuven Research Council (GOA/ 2001/04) and the Fund for Scientific Research - Flanders through FWO Krediet aan Navorsers 1.5.148.02.

References

1. K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):858–869, Jul 2004.
2. Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, Jan 2002.
3. A. Alexandrov, M. Ibel, K. Schausser, et al. SuperWeb: Towards a global web-based parallel computing infrastructure. In *Proc. of the 11th Int'l Parallel Processing Symposium (IPPS'97)*, pages 100–106, apr 1997.
4. K. Arnold, B. O'Sullivan, et al. The jini specification, 1999. See also <http://www.sun.com/jini>.
5. Henri Casanova and Jack Dongarra. NetSolve: A network-enabled server for solving computational science problems. *The Int'l Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.
6. I. Clarke, O. Sandberg, et al. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46–66, 2001.
7. Salutation Consortium. Salutation architecture specification. Technical report, salutation.org, 1999.
8. Microsoft Corporation. Universal plug and play device architecture. http://www.upnp.org/download/UPnPDA10_20000613.htm, 2000.
9. K. Czajkowski, I. Foster, N. Karonis, et al. A resource management architecture for metacomputing systems. In *Proc. of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.
10. Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, et al. An architecture for a secure service discovery service. In *Mobile Computing and Networking*, pages 24–35, 1999.
11. F. Dabek, E. Brunskill, M. F. Kaashoek, et al. Building peer-to-peer systems with Chord, a distributed lookup service. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 81–86, May 2001.
12. Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag New York, second edition, 2000.
13. K. Ducatel, M. Bogdanowicz, et al. Scenarios for ambient intelligence in 2010. <ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf>, Feb 2001.
14. Gnutella. The gnutella protocol specification. <http://rfc-gnutella.sourceforge.net>.
15. Object Management Group. Corbaservices: Common object services specification. <ftp://ftp.omg.org/pub/.docs/formal/98-07-05.pdf>, 1999.
16. Erik Guttman. Service location protocol: Automatic discovery of IP network services. *IEEE Internet Computing*, 3(4):71–80, 1999.
17. T. Howes and M. Smith. Rfc 1823: The ldap application program interface. <http://www.faqs.org/rfcs/rfc1823.html>, Aug 1995.
18. A. Iamnitchi, I. Foster, and D. Nurmi. A peer-to-peer approach to resource location in grid environments. In *11th Symposium on High Performance Distributed Computing*, Edinburgh, UK, Aug 2002.

19. Paul V. Mockapetris and Kevin J. Dunlap. Development of the domain name system. In *SIGCOMM*, pages 123–133, 1988.
20. Keith Moore, Shirley Browne, Jason Cox, et al. Resource cataloging and distribution system. Technical Report UT-CS-97-346, University of Tennessee, Jan 1997.
21. Hidemoto Nakada, Mitsuhsisa Sato, and Satoshi Sekiguchi. Design and implementations of ninf: towards a global computing infrastructure. *Future Generation Computing Systems*, 15:649–658, 1999.
22. Napster. <http://www.napster.com>.
23. C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.
24. Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *7th IEEE Int'l Symposium on High-Performance Distributed Computing*, 1998.
25. S. Ratnasamy, P. Francis, et al. A scalable content addressable network. In *Proc. of ACM SIGCOMM*, pages 161–172, San Diego, USA, Aug 2001.
26. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM Int'l Conference on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany, 2001.
27. N.N. Siram, R.R. Raje, A. M. Olson, et al. An architecture for the uniframe resource discovery service. In *Proc. of the 3rd Int'l Workshop of Software Engineering and Middleware*, 2002.
28. uddi.org. Uddi technical white paper. Technical report, <http://www.uddi.org>, sep 2000.
29. Maarten van Steen, Franz J. Hauck, Philip Homburg, et al. Locating objects in wide-area systems. *IEEE Communications Magazine*, 36(1):104–109, January 1998.
30. Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.
31. B. Zhao, J. Kubiawicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, Apr 2001.