On Service Discovery Process Types

Peer Hasselmeyer

C&C Research Laboratories, NEC Europe Ltd.,
53757 Sankt Augustin, Germany
hasselmeyer@ccrl-nece.de

Abstract. With the growing adoption of service-oriented computing, locating services becomes increasingly commonplace. Accordingly, a large number of systems for service discovery have been developed. Although all these systems perform the same function, they do it in lots of different ways. Finding commonalities of and differences between these systems can be hard due to the lack of criteria to compare and classify various discovery schemes.

This paper identifies the processes of registration and look-up as a distinguishing feature of the various discovery systems. It describes the possible types of processes, shows how they are distributed across the lifecycles of the involved entities and classifies existing service discovery systems according to these criteria. Some hints are given on how the process-based view can help guide the selection of a particular discovery style for a problem at hand.

1 Introduction

The use of service-oriented computing and service-oriented architectures becomes increasingly prevalent. The main principle of service-oriented architectures is the loose coupling between service providers and service users. Service providers can be internal to an organization as well as third parties external to the service user's organization. In any case, the service user needs to know the service's location and communication protocol before he can access it. Accordingly, all service-oriented architectures offer some facility for locating services. Although the functionality of these facilities is always the same (i.e. finding an access point to a desired service), its realizations vary significantly among the available architectures. The systems have different methods for describing services, they offer different query possibilities, they use different transport protocols, and they have different registration and query interfaces. The multitude of these features can make comparing different architectures hard. Hidden below these features are the processes for service registration and look-up. These processes are usually not mentioned explicitly, although they make comparing and classifying different architectures possible.

This paper analyzes the processes used for registration and look-up. It is discovered that both processes have a static and a dynamic form. These forms are distinguished by how the processes are spread across the lifecycles of the

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 144–156, 2005.

components performing registration or look-up. It is described how the different processes affect certain aspects of systems using service discovery, e.g. application development. Existing discovery systems are classified according to their types of processes. In addition, it is shown how a process-based view can guide the selection of a discovery architecture.

The paper starts in section 2 with definitions of the terms used throughout the paper. It continues in section 3 with a detailed description of the various types of processes commonly found in service discovery systems. Section 4 describes a number of systems providing a discovery facility and classifies them according to the process types. Some advice on selecting a particular service discovery system using the proposed process-based view is given in section 5. Related work is discussed in section 6 and some conclusions are presented in section 7.

2 Definitions

There are many different understandings of the terms involved in service-oriented architectures. For example, the term "service" means different things to different people. It is therefore necessary to first define the terms used throughout this paper to avoid ambiguity.

Service. A service is a component that provides a certain set of functions to other entities over a communications network. *Service instance* is a synonym for service.

Service Type. A service type names the functionality of a service. Services are of a certain service type if they provide at least the set of functions referred to by this type. The actual description of these functions is called *service type description.* It is assumed that this is a syntactic (interface) description.

Service Description. Description of non-functional service attributes. Service descriptions can be used in registrations as well as in queries. In registrations, they provide information about actual service properties, while in queries, they state desired service properties.

Service Metadata. The service metadata includes all the information a service is registered with. It consists of the service type, the service description, and the service's endpoint.

Service Provider. An entity that operates one or more services.

Client. An entity acting in the role of a service consumer in a specific service interaction. A client can be a "real" client, i.e., it only consumes services, or a service that happens to be in the client role in this particular interaction but provides services in other interactions.

Service-Oriented Architecture. A service-oriented architecture (SOA) is based on services as the main entities. Services provide certain functions to other entities. These entities can be pure clients or services themselves. Services and their functions are discovered using a service registry.

Service Discovery. The process of finding services and their endpoints. This includes registration and look-up.

Endpoint. A communication port at which a service can be contacted. Services might offer multiple endpoints.

Service Registry (often just called *registry*). A service that provides references to other services. It accepts requests for registration from services (or other entities that act on behalf of a service) and relays registration information on demand to clients.

Registration. Registries keep records of available services. The resources that such a record consumes at a registry are called a registration. Also, the process of having registries store information about a service is called registration.

Look-up. The process of mapping queries to service endpoints. Clients looking for certain services send queries to service registries. The queries contain some description of what kind of service clients are looking for. Registries return a set of available services matching the query. Services are represented by their endpoints.

3 Discovery Processes

Service discovery consists of two main processes: the registration process and the look-up process. Registration is used by services or their operators to announce service availability. Look-up is used by clients to find the endpoints of needed services.

Different service discovery architectures employ different processes for registration and look-up. These will be described in the following sections. The method for distinguishing them involves looking at how the individual steps in the process are allocated to the stages in the lifecycles of the entities executing the process. The registration process is executed by services, the relevant lifecycles therefore are the ones of the services. The look-up process is executed by clients, the relevant lifecycles therefore are the ones of the clients. As both services and clients are software components, they follow a software lifecycle. The stages of that lifecycle that are relevant to the registration and look-up processes are the development, the deployment, the operation, and the undeployment phases.

3.1 Registration Process

The basic lifecycle of a registration is rather simple. The registry creates a registration when a service is registered. From that point on, the registry includes the registered data in responses to matching queries. The registration is deleted when a service is removed from a registry.

As registrations belong to registries, their lifecycles can be observed at registries. The process for performing registration is executed by services (or their providers), though. For the registration process, only the lifecycle of the associated service is relevant. This is the lifecycle considered here. Two different types of service registration processes can be identified: a static and a dynamic type. Figure 1 presents an overview of these two types. Interactions with the registry are shown as rounded rectangles while supplied information is shown as bubbles.

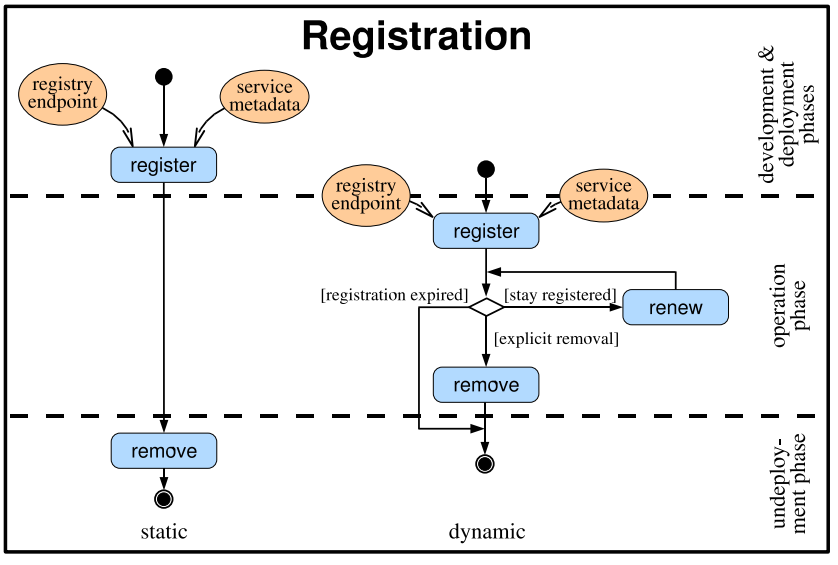

Fig. 1. Registration Process Types

Static Registration Process Type. In systems using static registrations, services are registered once and stay registered for an extended period of time. Registration and removal are usually initiated by human staff members of the service's provider. Registrations are rarely updated. Different parts of the service's metadata are supplied at different stages of the service's lifecycle. The service type is usually encoded in the implementation, i.e., it is supplied during the development phase. The service endpoint is only known at deployment time and is therefore supplied in the deployment phase. Some default values for the service description and the service registry's endpoint can be supplied in the development phase, but they will likely be modified during the deployment phase.

The actual registration of a service following a static process type happens when it is initially deployed. The registration is removed when the service is discontinued. Between registration and removal no communication is going on between the service (or its provider) and the registry. The exception to this rule is a possible maintenance phase in which the service's metadata is changed. Obviously, the new metadata needs to be supplied to the registry which involves communication between the service (provider) and the registry. A consequence of the absence of communication is that static registrations do not contain any current information, e.g. about the availability of services.

Dynamic Registration Process Type. Systems supporting a dynamic registration process type usually employ automated methods for service registration. Services or their supporting middleware initiate service registration upon startup. Human intervention is not needed. Registries use soft-state (lease-based) registrations [3]. Such registrations are valid for only a limited amount of time,

most commonly minutes to hours. To stay registered, services must periodically extend the lifetime of their registrations. Upon expiration, registrations are automatically removed from the registry. The system is therefore automatically cleaned from stale registrations and can be considered self-managing. The manual removal of services is nevertheless possible as well.

Service metadata is again supplied at different stages in the lifecycle. Service type information is handled the same way as in the static case – it is supplied during the development phase. The service description can be supplied in the development, the deployment, and, contrary to the static case, in the operation phase. Depending on the middleware used, the service endpoint is either supplied at deployment time or run-time. With technologies like CORBA (Common Object Request Broker Architecture) or Java RMI (Remote Method Invocation), the endpoint will be supplied at run-time as it can change across restarts. Using web service technologies, the endpoint may be supplied during the deployment phase as it rarely changes. The registry location can, as in the static case, be supplied during the deployment phase. Alternatively, as the actual registration happens during the operation phase, supplying the registry location can be postponed to that phase. One popular solution for this is to use a multicast scheme to discover the registry location at run-time.

3.2 Look-Up Process

Just as service registrations can be static or dynamic, so can be service look-up process types. The two different types are shown in figure 2. Although the figure

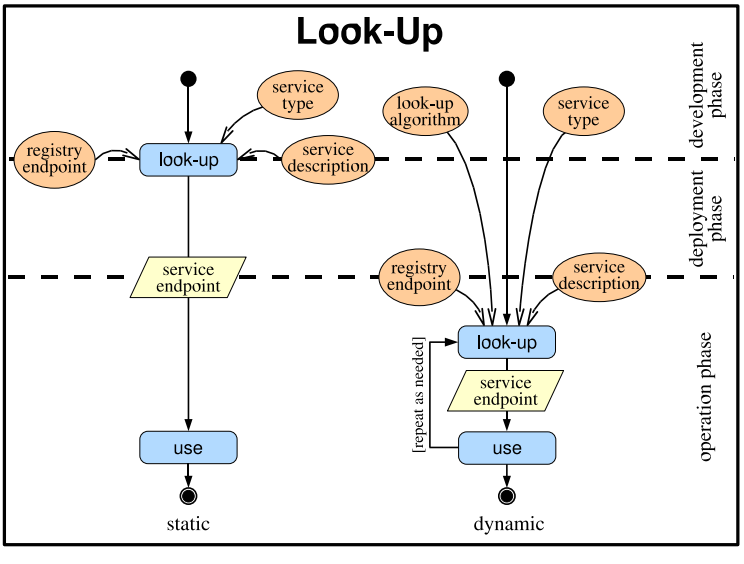

Fig. 2. Look-up Processes

shows look-up as a single step, it might actually consist of multiple steps, e.g. if some iterative method to slowly narrow down the selection is used. The look-up method is independent of the use of a static or a dynamic look-up process type.

Static Look-Up Process Type. In a static environment it is common practice to look up services during the application development phase and put their locations in the code. Alternatively, a configuration file can be used and the look-up process can be moved to the deployment phase. In this case, only the service type is supplied during the development stage. The service description (i.e. the query) as well as the registry's endpoint only need to be known during the deployment phase. In both cases, the registry is not accessed at application run-time.

Dynamic Look-Up Process Type. In a dynamic system, services are assumed to be volatile and those available during the development or deployment phases might not exist during the operation phase. Applications in dynamic environments therefore usually perform look-up at run-time to find currently available services and their endpoints. Information supplied during the development phase are the service type and the look-up algorithm. This algorithm is rather simple in "traditional" discovery systems, but is an important part of the discovery process in content addressable networks [8].

The registry endpoint and the service description can be supplied during the deployment or the operation phase. During deployment, this information goes into some configuration file. If the endpoint is supplied at run-time, it can be retrieved via some multicast scheme or calculated with the help of the look-up algorithm. The service description might be affected by user input and might therefore only be available at run-time.

The choice of a static or dynamic look-up process type is not directly related to the choice of a particular registration process type. A dynamic look-up style can be used in conjunction with a static registry. A static look-up style is ill-suited for a dynamic registry, though, as service information is expected to change during the lifetime of the service client. It is therefore common to bundle static look-up with static registration and dynamic look-up with dynamic registration.

3.3 Foreknowledge

To perform service registration or look-up, the endpoint of the registry needs to be known. This information is called "foreknowledge" in [10]. Such information is needed by all entities using the service registry, i.e. services as well as clients. It is important to note that the information does not need to be available explicitly. In content addressable networks the location of the registry can be derived from registration metadata and look-up queries at run-time. The foreknowledge in this case is embedded in the queries or the service metadata and the algorithm mapping that information to the registry location. Another important observation is the fact that the foreknowledge does not need to exist in all phases of the registration and look-up lifecycles. The knowledge needed and the lifecycle phase

in which it is supplied are shown in Figures 1 and 2 in the form of bubbles. The foreknowledge needed does not depend on whether a process' type is static or dynamic. Only the lifecycle phase in which it is used varies with the process type.

In a system using static registrations information about the registry is only needed at development or deployment time. It is therefore sufficient that the human developer or deployer has access to the registry's endpoint information. As that data is not needed at run-time, it is not kept anywhere in a running client.

In a dynamic system, information on the registry's endpoint must be available at run-time. The required information can be supplied in different ways. It could for example be read from some configuration file. A multicast scheme can be employed when a local registry is used thereby reducing configuration effort to a minimum.

3.4 Process Type Implications

The adoption of a particular type of process influences the functionality of registries, the development of entities using discovery, and the performance of a service-oriented system as a whole.

Functionality. The functionality of a registry depends on the registration style. In the dynamic case, it must support soft-state registrations and the associated functionality which is not needed in the static case. Furthermore, lifetime extensions happen quite frequently and those changes to the registration database need to be accommodated. In systems using a static registration process type, registrations have to be made persistent as they occur only once. For systems using a dynamic registration process type it is okay to forget all the registrations every once in a while as services will eventually re-register.

Implementation. Entities that are using service discovery, i.e., services performing registration and clients using look-up, are implemented in different ways depending on the type of the registration and look-up processes. When using static registrations, services do not need to care about the actual registration as this is supposed to be done by human operators before the service's operation phase. In systems using dynamic registrations, services must register at run-time and need to take care of their registrations. This job can be delegated to the infrastructure, though. In that case, services do not need to care about registrations and their renewal. In fact, they do not even need to be aware of what kind of registration process is used.

The situation is a bit different for look-ups. There is a difference in the program logic depending on when look-up is performed. If a static look-up process type is adopted, the client code does not need to contact the registry at run-time. With a dynamic look-up process type, though, some communication with the registry has to occur at run-time. The program code therefore needs to be different depending on the look-up type. It is nevertheless possible to always use code that assumes a dynamic look-up style and let the infrastructure map this to a static environment.

Performance. The registration and look-up process types also affect the performance of systems using service discovery. Obviously, a static system has a better run-time performance as no communication between services or clients and the registry has to occur during the operation phase. Depending on what kind of discovery technique is used, the run-time penalty of dynamic look-up can be significant. On the other hand, using dynamic registration and look-up improves a system's failure resilience. If look-ups follow a static process type, the disappearance of an endpoint to a particular service renders all clients unusable. A dynamic look-up process type allows this failure to be masked by being able to switch over to other instances/endpoints of services.

4 Classifying Discovery Systems

This section briefly describes a number of well-known service discovery systems. The main part of each description deals with registration and look-up properties. Each system is classified according to its use of static or dynamic registration and look-up process types. The results of the analysis are shown in Figure 3.

4.1 UDDI

The Universal Description, Discovery and Integration (UDDI) specification [7] is the most well-known service discovery standard for the web services world. It defines a centralized registry service, the interface to access it, and a data model to describe services. The UDDI registry is modeled after the telephone book. It lists companies ("white pages") and the services that they provide ("yellow pages"). Sticking to the telephone book analogy, data in UDDI registries is stored

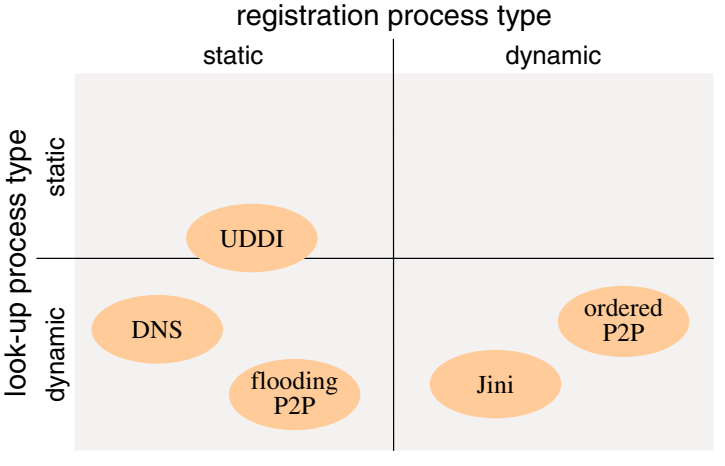

Fig. 3. Classification of Popular Service Discovery Systems

once and stays in them until it is actively removed. UDDI registries obviously employ a static registration lifecycle and therefore follow a static registration process type.

Look-up in UDDI registries usually follows a static process type as well. The structure of the data in the directory makes drill-down operations a convenient way of finding a needed service. A common approach to finding services in UDDI registries is looking at the list of companies, finding a few that are trusted and presumably offer the needed service, have a look at their service offers, and then choose one of them. As trust is a complicated concept for machines, the described process is always done by human beings. Usually, the developer of a service client is looking for an appropriate service, not the deployer. The reason for this is the lack of standardized interfaces that would make exchange of services a simple configuration option. Currently, more often than not, the program code needs to be modified to accommodate a different service provider.

Although dynamically exchanging a service provider in a web-service-based world is currently not widely practiced, performing look-up at run-time is still a sensible option. In this case, it is not used to make the choice of a service provider configurable or dynamic, it is used to automate locating the current endpoint of a well-known service of a well-known provider. A service client would perform a look-up with a predefined query that returns the current endpoint of the needed service. With this scheme, a provider can change the location of his services without explicitly telling his clients. Dynamic look-up can increase the looseness of the coupling between service provider and service client.

Because UDDI-based systems can work with both static and dynamic look-up, the system appears in both sections with a tendency towards the static side.

4.2 Jini

Jini [9] is a Java-based infrastructure for handling service discovery. Registries can be statically configured, or found at run-time via a multicast-based protocol. Registrations contain service descriptions and, instead of the endpoint of a service, a Java object that enables remote service access. Such proxy objects are stored at the registry and copied to the service user's address space at run-time. There, they adapt local function calls to whatever communications protocol the associated service uses. Information about the endpoint of a service is therefore stored inside the proxy object and kept hidden from the service user.

Registrations in Jini are lease-based. Clients wanting to register a service negotiate the registration lifetime with the registry. Leases need to be renewed in time for services to stay registered. Registrations therefore follow a strictly dynamic process type. Due to the use of proxy objects, this method is the only viable option. Most of the Jini services use the Java RMI mechanisms for remote procedure calls. Proxy objects contain the remote stubs for performing invocations. These in turn contain references to the associated server object. As the server object might change with every restart of the server, previously handed out references become invalid. Registration data therefore needs to be updated at the registry. With an automated update system (as used with dynamic reg-

istrations), this is no problem. Performing updates by hand is rather tedious, though. A static model is therefore not an optimal solution.

Look-ups also follow a dynamic process type, basically for the same reason why registrations are dynamic. Remote object references may change frequently. Embedding fixed, static references in client code is no option. In addition to that, Jini transfers around proxy objects. These consist of data (e.g. object references) and code. Just as the data changes, the code might change as well. As the code is supplied by the service provider, it is impractical to statically embed that code in service clients.

4.3 DNS

The Domain Name System (DNS) [6] is the most commonly used method for resolving human-readable internet host names. The DNS system translates those names to machine-understandable Internet Protocol (IP) addresses. DNS can store mappings for different types of services, e.g. for mail relays. It can therefore be considered a service discovery system.

The assumption for the DNS is that internet host names and the corresponding addresses do not change frequently. As a result, registrations follow a static process type. New internet host and domain names are stored at designated name servers. The mapping stays at the name server until it is removed.

DNS look-ups follow a dynamic process type. To find a host, clients send a query to the DNS sub-system on the local host. Queries that cannot be answered locally are passed on to DNS servers one level up in the hierarchy. The path followed by a query is statically configured by the administrators of the involved DNS servers. The location information of the registry (the next DNS server) is therefore configured for a host, not an individual client.

4.4 Peer-to-Peer Systems

Peer-to-peer (P2P) systems are dynamic networks of entities that cooperate to provide certain services to participants. At present, P2P system are aimed mostly at locating content, usually media files, but the same concepts can be used to locate arbitrary services. As the set of participants in a P2P system is dynamic, the set of offered services (or content) is dynamic as well. Searching (i.e. look-up) in P2P systems therefore always follows a dynamic process type.

Regarding registrations, two different kinds of peer-to-peer systems can be distinguished. Most of the P2P systems popular today employ a flooding search approach. They send queries to all (or at least a large subset of) the service providers. Service providers also act as registries. Content files are registered locally and they are usually registered implicitly by putting them in specific places on the local file system. They are unregistered by removing the files from that location. It is therefore a registration process of the static type.

As flooding is not a scalable query mechanism, a second type of P2P systems uses a controlled search approach. Systems of this type designate responsibility

for a subset of the whole search space to certain nodes in the system. A responsible node can be found at a predictable location by applying a well-known algorithm to the service's metadata. An example of such a system is described in [2]. In such systems, all nodes act as registries, but each service is registered with just a (small) subset of them. Registrations are constrained in the temporal domain and therefore need to be renewed. This is necessary because not only the set of service providers (and with it, the set of services) changes, but so does the set of registries and therefore the set of registries responsible for a specific service. Registrations therefore follow a dynamic process type.

5 Architecture Selection

Besides offering a method for classification, a process-based viewpoint can also guide the selection of a particular discovery architecture for a given problem. For selecting a discovery system, a large number of different criteria exist, including description capabilities, performance, scalability, supported platforms, protocols used, etc. Most of these are of a low-level, technical nature. The process-based viewpoint proposed here is on a higher level of abstraction, working more on the architectural level than the implementation level.

The guidelines given here are not necessarily the most important ones. Depending on the circumstances in which a discovery system is to be used, different criteria have different priorities. Nevertheless, we think that looking at discovery process types gives a hint on what kind of discovery system is suitable to a given problem. As it works on a comparatively high level of abstraction, it might actually be the first criterion to consider.

When thinking of discovery, the processes for registration and look-up are not a natural starting point. And indeed, they are not the first thing to look at, as they only exist in conjunction with the entities that enact them, i.e., the services and clients.

Therefore, to select a discovery architecture, one should first analyze the services that are to be found by discovery. If these services are rather volatile, their registrations are as well. Dynamic process types for registration and look-up are therefore appropriate. A matching service discovery system should be chosen.

If the set of services is small and the services' metadata is static, a system using static registrations is sufficient. Whether to use a static or dynamic look-up process type in this case is a delicate trade-off. Static look-ups require some manual configuration effort when services migrate. Dynamic look-ups need some additional infrastructure that usually causes some initial set-up costs. The decision therefore depends on the estimate of how frequently service registrations change and thereby cause additional configuration costs.

It is important to note that even if individual services are rather static, but there is a large set of services, the set as a whole becomes volatile, because some changes in the set of services and their metadata always occur. Such a system would benefit from at least a dynamic look-up style. Making registrations

dynamic can be a good idea as well, because static registrations become stale eventually and their removal or update is often forgotten [5].

6 Related Work

There is a host of literature that compares various discovery systems. Most of this literature, e.g. [1, 4], compares systems primarily aimed at ubiquitous computing, especially SLP, UPnP, Bluetooth, and Jini. Among the criteria used for distinction are the network protocol used for communication, the type of service description, and the functionality of the systems. None of these comparisons deal with higher level abstractions or give guidance on where to use which architecture.

To the knowledge of the author, only one paper [10] compares service discovery systems on a more abstract level. Vanthournout et al. introduce a taxonomy of discovery systems distinguishing them by design aspects. They analyze discovery systems with respect to their structure, their foreknowledge, their registration behavior, their query routing, the supported resources, and resource naming. Although these represent a large set of criteria, the registration and look-up processes are neither mentioned nor evaluated. As described here, those processes influence the architecture of service discovery systems as well as the design of applications. In fact, some of the above mentioned design aspects are influenced by the processes. Namely, these are the foreknowledge, the registration behavior, and the supported resources.

7 Conclusion

In this paper, we have identified the registration and look-up processes as distinguishing aspects of service discovery systems. Registration as well as look-up can be either static or dynamic. A particular choice for either the registration or the look-up process type does not influence the choice of the other process type. The only caveat is that a static look-up process type does not match a dynamic registration scheme.

We showed that individual registration and look-up process steps are distributed differently across component lifecycle phases depending on the type of process used. We classified a number of well-known discovery architectures according to the process-based view and showed that for each viable combination of processes an examplar exists. We also described how the choice of a particular process type influences the functionality of the registry, the implementation of the entities, the performance of the system as a whole, and how foreknowledge is spread across the lifecycle phases.

We hope that the presented work makes developers adopting service-oriented architectures aware of the discovery processes and their interaction with component lifecycles. We think that a classification according to process types facilitates the selection of a particular style of discovery and therefore the selection of a particular discovery architecture.

Acknowledgements

This work has been supported by the NextGRID project and has been partly funded by the European Commission's IST activity of the 6th Framework Programme under contract number 511563. This paper expresses the opinions of the author and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

1. Christian Bettstetter and Cristoph Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In: Proceedings EUNICE Open European Summer School, Twente, Netherlands, September 2000.
2. Jun Gao and Peter Steenkiste. Design and Evaluation of a Distributed Scalable Content Discovery System. Journal on Selected Areas in Communications, 22(1):54–66, January 2004.
3. Cary G. Gray and David R. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In: Proceedings of the 12th ACM Symposium on Operating System Principles, pages 202–210, December 1989.
4. Sumi Helal. Standards for Service Discovery and Delivery. IEEE Pervasive Computing, 1(3):95–100, July 2002.
5. Mike Clark. UDDI – The Weather Report, November 2001. `http://www.webservicesarchitect.com/content/articles/clark04.asp`
6. Paul V. Mockapetris. Domain Names - Concepts and Facilities, November 1987. Internet RFC 1034.
7. OASIS Open. UDDI Version 3.0.2, October 2004. `http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm`.
8. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-Addressable Network. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001), pages 161–172, August 2001.
9. Sun Microsystems Inc. Jini Architecture Specification – Version 2.0, June 2003. `http://www.sun.com/software/jini/specs/jini2_0.pdf`.
10. Koen Vanthournout, Geert Deconinck, and Ronnie Belmans. A Taxonomy for Resource Discovery. Personal and Ubiquitous Computing Journal, 9(2):81–89, February 2005.