IT Report
Service Discovery

IT Members
A. Birukou, M. Boniface, K. Kearney, C. Kotsokalis, N. Matskanis, A. Micsik, V. Sanchez, T. Sellis, D. Skoutas, F. Torelli, Jesús Gorroñogoitia

Due date of deliverable: 16/01/2009
Actual submission date: 13/03/2009

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

This work is partially funded by EU under the grant of IST-FP6-034763.
Change History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Author (Partner)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>24/02/2009</td>
<td>draft</td>
<td>Atos</td>
<td>Template draft</td>
</tr>
<tr>
<td>0.2-0.9</td>
<td>24/02/2009-13/03/2009</td>
<td>Draft</td>
<td>IT contributors</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>16/03/2009</td>
<td>Final</td>
<td>Atos</td>
<td>Final version for release</td>
</tr>
</tbody>
</table>

Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP7 – 216446</th>
<th>Acronym</th>
<th>NEXOF-RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full title</td>
<td>NESSI Open Framework – Reference Architecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://www.nexof-ra.eu">http://www.nexof-ra.eu</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EU Project officer</td>
<td>Arian Zwegers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>Title</th>
<th>IT Report</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work package Number</th>
<th>Title</th>
<th>Service Centric System Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>final □</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature</td>
<td>Report □ Demonstrator □ Other □</td>
</tr>
</tbody>
</table>

Abstract (for dissemination)

Keywords

Authors (Partner)

| A. Birukou, M. Boniface, K. Kearney, C. Kotsokalis, N. Matskanis, A. Micsik, V. Sanchez, T. Sellis, D. Skoutas, F. Torelli, Jesus Gorroño-goitia |

Responsible Author

<table>
<thead>
<tr>
<th>Jesús Gorroño-goitia</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partner

<table>
<thead>
<tr>
<th>Atos</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

This document reports the results obtained by the IT on the topic Service Discovery, constituted in the first call of the NEXOF-RA open construction process. The document collects the contributions received and discussed within the IT, summarises the main design patterns extracted from the contributions, their similarities, relationships and dependencies, the most relevant standards considered in the contributions, and it concludes with an short review of IT objectives fulfilled by the IT.
# Table of Contents

- Executive Summary ........................................................................................................... 4
- Table of Contents ............................................................................................................... 5
- 1 Topic Introduction ........................................................................................................... 6
- 2 Topic Overall Results ....................................................................................................... 8
- 3 Topic Design Patterns ...................................................................................................... 16
  3.1 Shared Usage History pattern ......................................................................................... 16
  3.2 Contextualized B2B Service Discovery pattern .............................................................. 21
  3.3 SLA-Template-based service description pattern ........................................................... 29
  3.4 Group negotiation during discovery pattern .................................................................. 33
  3.5 Multi-phase discovery pattern ....................................................................................... 36
  3.6 Template based discovery pattern .................................................................................. 40
  3.7 Distributed (Peer to Peer) Registry Repository pattern ............................................... 44
  3.8 Service Matchmaking and Ranking pattern .................................................................... 48
- 4 IT Conclusions .................................................................................................................. 52
- 5 Appendix I: Design Pattern Template .............................................................................. 53
1 Topic Introduction

Overview

That services are “discoverable” is one of the key factors for the success of SOA. Services are fully described and these descriptions are published in publicly accessible SOA registries so the services may be later discovered. However, discovering services could be a cumbersome process if it had to be done manually by browsing in distributed registries with a large number of service descriptions, without the aid of semi-automatic integrated searching facilities, especially in the composite service development lifecycle.

Problem Statement

SOA strength is that services can be consumed as many times a particular user requires them. This is possible since services are discoverable. This capacity is based on the ability of services to describe themselves through well-defined formal descriptions that are published on SOA registries, and, once there, are publicly available for retrieval and inspection through browsing or other discovering mechanisms.

Discovering of services by browsing is not adequate for SOA registries with large content. Therefore, fast and accurate semi-automatic search and selection facilities are required which perform queries within local or remote, centralized or distribute, single or federate SOA registries.

Service discovery may rely on: a) the user’s ability of precisely describe its request, b) in the algorithms applied to match that user request with the capabilities of candidate services, and c) in the algorithms to rank and select the best candidate among those discovered.

Service discovery is intensively used by SOA practitioners when they compose other services or when they require invoking an external service from some application or process. Hence, the service discovery process is quite relevant in SOA engineering cycles.

Even if service discovery is, to some extent, well covered by the SOA techniques and tools, there are still some challenges, especially with techniques to specify user requests, semi-automatic service discovery, and the ranking and selection facilities. Also determining the role of service discovery in the SOA engineering phases like service composition specification, runtime, etc., requires still further attention.

Scope

This IT focuses on service discovery techniques and strategies and their applicability during certain service development phases (composition, invocation, etc.). It is not concerned with service description specifications, except in those cases where describing user request and services capabilities are relevant for the discovery process.
Contributions

This IT expects contributions for a general conceptual and technological framework for service discovery that should be aimed as much complete and self-consistent as possible. Therefore, it is expected:

- Techniques and language specifications for representing the user’s request. The technology should also support the creation of consumer’s goals, isolating as much as possible the underlying technological complexity.

- Strategies, techniques, best practices, etc. for accurate and precise matching between consumers’ goals and services’ capabilities. Most appropriate matching algorithms and guidelines to accommodate then to particular searching scenarios are also expected.

- Techniques, algorithms, best practices, etc. for semi-automatic service ranking and selection among those service specifications retrieved by the discovery process. As above, guidelines are expected to accommodate selection algorithms to particular searching scenarios.

- Strategies, techniques for service discovery applied to some frequent service discovery scenarios in composite services: requirement based service discovery (that is, based on stakeholders or analysts requirements elicitation process), architecture based service discovery (that is, based on design patterns and choreography constraints applied to the composite service) and runtime service discovery (that is, postponing concrete binding to services to execution time), among others.

- It is expected to explore other aspects concerning service discovery and selection such as consumers’ and providers’ context, SLA, negotiation, etc.

Baseline

Baseline for this IT is the WS technological stack; contributions are assumed to be full compatibility with this technology. There are no other assumptions or constraints.

Document purpose and structure

This report collects the contributions received and discussed within this IT. The report is structured as follows: 2 provides an overall integrated description of the received contributions, their location in the context of the Service Discovery concern, their relationships, dependencies and possible overlapping. Section 3 collects all the contributions analyzed within this IT. They are described using the design pattern template proposed by NEXOF-RA for submitting the contributions. Section 4 highlight main IT conclusions. Section 5 describes the design pattern template.
2 Topic Overall Results

This section introduces the main contributions submitted to and discussed within this IT, providing an overall and integrated view of them. This section describes the similarities, common patterns, dependencies and relationships among contributions. Finally, this section identifies a set of common technologies and standards used by the receiving contributions, but it doesn’t emphasize the benefits and drawbacks of them.

This topic IT has received the following contributions:

- Shared usage history
- Contextualized B2B service discovery
- SLA-template based service discovery
- Group negotiation during discovery
- Multi-phase discovery
- Template-based discovery
- Distributed (P2P) registry repository
- Service matchmaking and ranking

Errore. L’origine riferimento non è stata trovata. picture depicts the different contributions in the context of this IT topic. As shown, several contributions (B2B SD, SLA-template based SD, shared usage history) proposed general patterns that generally covers the complete Service Discovery problem, one contribution (matchmaking and ranking) describes a general pattern for the SD Engine core, and other contributions are specialized on other SD concerns, such as: a distributed (federated, P2P) registry, template-based SD (for consumer’s requirement specification), multiphase SD (for hybrid approaches, that is, combinations of IR and Semantic approaches), SD group negotiation (combined with service ranking).

Figure 1 Overview of Service Discovery IT contributions
From received contributions we can extract a first common pattern for SD matchmaking and ranking. This pattern is described in the Service Matchmaking and Ranking contribution, but it is presented in SLA Template based SD and Contextualised B2B SD contributions. Next UML2 composite structure with collaboration diagram depicts this pattern.

**Figure 2 SD Matchmaking and Ranking**

Given a set of consumer’s requirements which constitute the search criteria, this pattern returns a list of ranked candidate services and optionally selects the most appropriate which satisfy the criteria. The candidate services are selected from a repository of available services. Essentially the candidate selection is done by comparing the search criteria against each service capability which is described in the service description stored within the catalog.

This pattern involves a set of collaborations: matchmaking, ranking and selection, which involve some participants: a matchmaker, a ranker and a
catalogue. This pattern is quite well explained in Service Matchmaking and Ranking contributions (see section 3.8).

SLA Template based SD (see section 3.3) contribution shows a particular case of SD Matchmaking and ranking pattern, since the matchmaking is done between SLA templates which essentially describe both the consumer’s requirements and the service capabilities: both functional and non-functional (SLA) functionalities.

Contextualized B2B SD (see section 3.2) contribution proposes a complete solution for service procurement and advertising of WS in the B2B domain, not only for discovering appropriate services but also the accompanying artefacts and resources to consume them. Concretely it describes a technical solution for the SD process, sharing similar approaches to those described by the SD matchmaking and ranking, and the template-based SD. This contribution describes a particular template-based text-based SD solution. Users can use pre-existing templates to set their queries or make them by their own (see Ref [2] in section 3.2). Queries are performed against SLA, SLA templates, registered services, security resources and any other resource available within the registry.

SD matchmaking and ranking is complemented with the following SD template pattern, shown in the next picture, which is described in Template-based SD and SLA Template based SD (see section 3.3) contributions. This pattern provides a solution to specify the consumer’s requirements and derive from them the search criteria using specialised templates. This approach enables the consumer to specify using human-oriented UI the full-operative search criteria without tackling with the underlying discovery technology burden.

The consumer selects an appropriate template form a categorised template repository and fulfils its fields (using a UI form) with her search criteria. Then the template is automatically translated into a proper internal search query, suitable for the used SD engine. In Template-based SD contribution they use of WSMO goal templates, while in SLA Template based SD they use SLA templates.
The *Multi-phase discovery* (see section 3.5) contribution suggests a strategy to improve the SD performance, what is quite necessary in real-time environments, for instance during the execution of composite services, but also at design-time to speed up the development process. This pattern (see below picture) is especially suitable when the SD process uses techniques which are time-consuming, such as semantic reasoning (for matchmaking and ranking).

This pattern splits the SD process in sequential phases, performed by specialized SD engines, which constraints the target of potential candidate services in each phase. First phases use less accurate and time-consuming techniques and the latter phases more accurate and time-consuming techniques but upon a much more constrained target of candidate services. This pattern relies on the precision of the first phases to discard bad candidates, so the different phases should be tuned carefully. The *Multi-phase discovery* contribution describes a two-phase SD: an IR text-search SD phase followed by an intelligent reasoning SD, using WSMO and Prolog.

---

**Figure 3 SD Template**

![SD Template Diagram](image-url)
Figure 4 SD Multiphase

Shared usage history (see section 3.1) contribution complements the standard SD pattern (as described in the SD matchmaking and ranking) with an innovative approach based on previous experiences of searching (see below picture). In this case, a broker registers each search query (request) submitted by the consumer to the SD system and matches it to the subsequent selected candidate service. Besides, the consumer can report to the broker her feedback after the service invocation. That information is later used by the broker to suggest candidate services based on previous searches and invocation experiences. When the consumer submits a request to the broker, it search for similar stored requests, which are tied to particular candidate services (this link is determined by using previous invocations and feedback reported).
The Distributed (P2P) registry repository (see section 3.7) contribution proposes a federated registry pattern (see below picture). This pattern describes a federation of collaborative P2P registries which work altogether to provide features for both advertising and provisioning of WS. Regarding SD, each peer registry offers SD features to consumers, who may get access not only to the content of that particular local registry, but also to the content stored within other remote registries tied to the same federation. In that way, the search results are aggregation of results obtained from the local registry and external ones. Regarding the publication feature, the service can be published within the local registry or submitted to another registry through the local one.

This federate approach can be used to distribute the content (descriptions of services) amongst domain specialized registries, as well. With this approach the SD can be specialized by domain, offering more precise and fast results.
Last but not least, the Group negotiation during discovery (see section 0) contribution extends the SD pattern to embrace the negotiation phase. This pattern (shown in next figure) uses agents between consumers, providers and the SD feature. The consumer’s agent broadcasts a query to all services and obtains a list of offers from candidate services computed on-the-fly, then the customer’s agent ranks the list and starts to negotiate with selected best services (via the agents representing them). If the negotiation finished successfully, the consumer’s agent invokes the service, otherwise the process proceeds with another well-ranked candidate service. This pattern is relevant for runtime SD.

From the received contributions we can extract a set of standard technologies which can be considered relevant in this topic domain, for some specific purposes:

- **UDDI**: WS registry standard with a reference API for service publication (advertising) and full-text based discovery (provisioning)
- **WSDL**: a XML-based WS standard for WS description
• WSMO, WSML, WSMX: a semantic framework for WS: languages and technologies for describing, discover, compose and execute WS based on ontologies, goals, mediators and services.

• OWL-S: another semantic framework for WS.

• SAWSDL: an extension of WSDL with support for annotating WSDL descriptions using domain-specific ontology concepts.

• SPARQL: a W3C standard to create semantic queries against semantic repositories of RDF/OWL metadata.

• XQuery: a W3C standard to create XML-based queries against XML-based repositories.

• WS-Agreement: a WS standard to describe SLA templates.

• WS-Policy: a WS standard to describe SLA constraints.
3 TOPIC DESIGN PATTERNS

This section collects the most relevant contributions received and discussed within the IT. They are alphabetical ordered by the main author surname.

3.1 Shared Usage History pattern

Aliaksandr Birukou

Name

Shared Usage History

Requirements

How can data about using services by different people be shared in a systematic and automated manner to facilitate service discovery?

Icon

Problem

Alice would like to add a weather forecast service to her website. She finds serviceA that claims to provide worldwide weather forecast at a decent price. However, Bob has used serviceA already and he knows that it is often unavailable and predictions are poor. On the other hand, Bob knows that serviceB does the job and it is even cheaper than serviceA. Bob discovered serviceB after speaking with Carmen, an ex-colleague of his.

Alice knows Bob and Bob knows Carmen and they do not mind sharing their experience in using web services to avoid such situations in the future.

How can such information be shared with least user effort and most value gained from such experience exchange?

Solution

An information system for supporting collaborative service discovery can collect information about user requests for services and save results of the invocations of the services returned in response to the request. In addition, the system can allow users to express their satisfaction with the service explicitly, e.g. by means of ratings.

---

1 This pattern is derived from my previous internal research projects conducted with Enrico Blanzieri, Vincenzo D'Andrea, Paolo Giorgini, and Natallia Kokash (all – University of Trento)
If requests are connected with invocations and explicit feedback, for instance, by using unique request IDs, then for each request it is possible to find similar requests in the history and to find corresponding invocations and feedback. Such information about using services by their colleagues, friends, etc. can be much more useful to users than descriptions of services written by providers.

For instance, let us consider the example from the problem section. Figure 8 shows the logical view of the solution – three users: Alice, Bob, Carmen interacting with the information system by submitting requests, getting names of services relevant to the request, and reporting results of service invocations. The system stores information about user requests and service invocations in a database. The interaction diagram (process view) for this scenario is shown in Figure 9.

![Figure 8. Shared Usage History pattern – Logical view](image-url)
Application

The key issue in implementing the proposed solution is the possibility to observe user actions, preferably in non-intrusive manner, i.e. with minimal changes of existing practices. More specifically, it is necessary to collect information about user requests, service invocations, and feedback.

To collect user requests, the system can be integrated into the interface normally used to access the registry or the repository of services. When user selects a service, the code for the invocation of such service can be automatically generated. To connect invocations with requests, this code could contain request ID. Such behaviour could be implemented by modifying Apache Axis framework [1]. Finally, to collect user feedback, the system should provide an interface for accessing past user requests and expressing opinions or ratings on the invoked services.

We have implemented and evaluated a prototype of the system implementing this pattern [2]. The system is based on the IC-Service\(^2\), a domain-independent general purpose recommendation service [3]. User requests in the system are

\(^2\) The IC-Service will be released as an Open Source java project in 2009. If you would like to use it right now, please send a request to Aliaksandr Birukou at birukou@disi.unitn.it
expressed as plain text and we use textual description of service operations in WSDL. For the evaluation we have used a collection of 20 service operations from the XMethods.com\(^3\) registry. The matching between user requests and descriptions of service operations was performed by the internal recommendation mechanism of the IC-Service. The mechanism consists in calculating the similarity between the current and past user requests and recommending service operation that was invoked for similar request in the past. More information about similarity calculation is given in [2].

**Impacts**

It might be impossible to connect user requests with service invocations automatically. In this case, algorithms for mining connection between requests and invocations should be applied. Additional sources of information, such as explicit feedback, e.g. ratings, about users’ experiences with services, information about user preferences or the context where the service will be used, might be also taken into account.

**Relationships**

The described solution works on top of some existing service registry or repository, or any other source of information about existing web services. This is illustrated in Figure 10 by connecting the Shared Usage History pattern with a Registry or other similar pattern with the “uses” link. The proposed pattern can also use patterns proposing algorithms for matching requests with services and for ranking the list of results.

When applied on top of an existing service registry, the pattern can be used to select the service that is the best fit to user goals from the set of candidate services provided by the registry.

![Figure 10. Shared Usage History pattern - Relationships](image-url)

---

\(^3\) [http://xmethods.com/ve2/index.po](http://xmethods.com/ve2/index.po)
References


Summary Card

Shared Usage History

*How can data about using services by different people be shared in a systematic and automated manner to facilitate service discovery?*

**Problem**

Users are unaware of others’ experience with services and may often be in the situation of “reinventing the wheel”

**Solution**

An information system for supporting collaborative service discovery collects information about user requests for services and save results of the invocations of the services returned in response to the request. In addition, the system allows users to express their (dis)satisfaction with services explicitly

**Application**

The key issue in implementing the proposed solution is the possibility to observe user actions, preferably in non-intrusive manner.

**Impacts**

Users may value different services for similar requests, and may not desire to share data. There should be a trade-off between implicit and explicit feedback collected from users.

**Principles**

Experience reuse

**Architecture**

Discovery

**Status**

Release candidate

**Contributor Name**

Aliaksandr Birukou

**Contributor Notes**
3.2 **Contextualized B2B Service Discovery** pattern.
Mike Boniface, Nikolaos Matskanis

**Name**
Contextualized B2B Service Discovery

**Requirements**
How to discover services in the context of business relationships, procurement and project management functions? How can existing Web Service registries gain capabilities required by business today?

**Icon**

**Problem**
Service discovery is a core concept of service oriented architectures and an essential capability required to support dynamic service-oriented business relationships. Existing Web Service registries and associated discovery workflows lack the possibility to register and discover resources in the context of dynamic business relationships. Business to Business (B2B) collaboration demands not only discovery of services based on available metadata, but also discovery of business context (e.g. security tokens and SLAs) necessary to access a service at the point of use. In many situations, business context is not managed by the service consumer but by other actors with financial responsibility who place constraints on consumers.

**Solution**
In GRIA[^4^], we have developed a service-oriented infrastructure design to support dynamic service-oriented collaboration. We adopt a discovery through procurement approach that links service discovery to procurement processes by combining workflow, contextualised service registries [^1^] and security. Consumers automatically discover services they want to use and also discover and obtain the necessary federation contexts (SLAs) and security attributes to access them. Figure 11 shows the architectural pattern, which consists of three core packages and seven components[^5^].

The **Client Management** package provides support for organisation-level management of service consumers with centralised control and monitoring of service procurement and usage. The **RegistryService** component is a Web

[^4^]: GRIA, www.gria.org
[^5^]: Note that the components in Figure 1 are a subset of the components in the packages
Service supporting the contextualised registration and discovery of resources (WS-Addressing EPRs and metadata). The RegistryService has an extensible domain model that allows different resource relationship models to be supported. The RegistryService supports the lifecycle of multiple registries each maintained by a RegistryOwner. The RegistryOwner can register different kinds of services, resources and business context required by consumers and can optionally include security (WS-SecurityPolicy) and federation policies (WS-Policy) that describe the necessary context required to access a resource. The security policy describes where security attributes are to be obtained and refers to WS-Trust issuance services, the Token Service component in Figure 11. The federation policy describes which federation context can be used to access a resource and refer to management services, where context resources (SLAs) are issued. The discovery process is often recursive, for example, a consumer discovers a resource then the context to access the resource, which could also require a further context. The RegistryService uses an XML database backend that stores all the registered information as XML document.


The **Client package** provides a client API to GRIA management and application services that is used for integration with client applications. At the core of the client API is the **InvocationEngine** component that is responsible for building SOAP invocation messages through interactions with supporting services that provide policy, federation and security context.

The **AttributeSelector** component is responsible for automatically retrieving security tokens necessary to access resources in an invocation. Each resource in GRIA has access control rules defined on it to determine who can do what. For instance, you can define who can "use" an SLA that you own by adding...
access control rules to the SLA. There are several different types of rules, each defining the "who" in a different way. One rule type specifies who can access a resource by saying that they must present a SAML assertion of a certain type. Unlike the WS-Policy statements published by a service provider in a service's WSDL, the access control rules on a resource are only known to the resource's owner. This is because publishing the rules would be a security risk: you would be telling the whole world what was required to access your resource. The information to tell the client what token to use is obtained either from a local registry (controlled by the client user) or from a remote registry (controlled by a RegistryOwner). This is equivalent to going to your manager to find out how to obtain a password. When an invocation on a remote resource is made, the DefaultAttributeSelector will search the DefaultInvocationEngine's selected registry list to see if any of them contain any information about what token (if any) is required to access the resource. The sequence of steps for the DefaultAttributeSelector is provided below:

1. Work out where to get the security token from:
   a. See if the EPR passed to the selector has a WS-SecurityPolicy defined, and if so return it
   b. Otherwise, incrementally search the selected registries list of the InvocationEngine to try and find the WS-SecurityPolicy
   c. If it has still not found any WS-SecurityPolicy then check to see if the "fall-back group" has been defined.

2. Check if the security token is in the cache and if so return it

3. Invoke the WS-Trust request security token method on the token source endpoint we have discovered.

4. Add the token to the token cache and return it

The FederationSelector component is responsible for selecting federation context necessary for service operations. For instance, when proposing an SLA the client must already have a valid trade account and when creating a job the client must specify which SLA to use. The trade account or SLA the client chooses is known as the "federation context". The sequence of steps for the DefaultFederationSelector is provided below:

1. Request the WSDL of the target service
2. Find the WS-Policy statement for the operation being invoked (if any)
3. Search in the selected registries of the DefaultInvocationEngine to see if the client knows of a federation context (e.g. an SLA) that matches the policy
4. Use the AttributeSelector to attach any tokens required to use the federation context

The Service package represents any service that is secured and managed by GRIA. For example, this could be a general job submission service for providing access to applications installed on a cluster or a business operation providing
access to data management system. The *FederationPolicyManager* component allows a *ServiceManager* to administer the federation policy for each service operation. The *ServiceManager* defines a federation policy by providing a set of "trusted management services" responsible for managing the service. The federation policy is expressed in WS-Policy and allows a client to discover the federation contexts necessary to access the service operation. Zero to many federation contexts can appear in the policy. The *FederationInterceptor* component is responsible for enforcing the federation policy by intercepting incoming SOAP invocations, extracting the federation context and checking with services in the *Service Provider Management* package that the client can perform the operation.

Figure 12 shows an example process view for the pattern. In this scenario, a client is using the invocation engine to access a resource at a service that is managed by an SLA. The client requires an SLA to access the resource, a security token to access the SLA (from STS1) and a different security token to access the resource (from STS 2).

---

**Application- Collaborative Engineering**

This example focuses on applying the pattern to different collaboration scenarios in the aerospace industry. These scenarios have been explored with BAE, EADS, Airbus and MBDA in the context of the SIMDAT and CFMS projects.

The first scenario looks at a collaboration model based on creating a virtual organization from federated security token services (See Figure 13). Each organisation created an internal group to manage engineers working in their
team. An additional group was created at the prime contractor that logically represented the virtual organisation (VO). Each group was added to the VO group so that members of each organisation group could get access to a VO security token. In this example the pattern allows the client to automatically discover the security tokens required to access the VO group and resources within that group.

**Figure 13: Automatic discovery of tokens in federated security scenario**

The second scenario is where the prime contractor wants to share data with a subcontractor. In this scenario the subcontractor maintains a team of engineers working on a project but needs access to specific design data for their work. A manager at the prime contractor adds the membership group rule to the data service policy and a manager at the subcontractor adds the design data EPR to the registry along with the WS-SecurityPolicy stating that a “Project” security token is required to access the design data. In this example the pattern allows an engineer at the subcontractor to automatically discover the security tokens required to access the engineering data resource at the prime contractor.

**Figure 14: Sharing data with a subcontractor**

The third scenario is where the prime contractor wants to share computation, storage and application resources with a subcontractor. The prime contractor ensures that appropriate federation policies are defined on application services. The prime contractor then defines the SLA templates and creates an SLA for their collaboration with the subcontractor. The subcontractor manager then
registers the SLA EPR resource in the registry with a WS-SecurityPolicy stating that a “Project” security token is required to access the SLA. The sub-contractor manager can also register other resources created in the context of the SLA and share these with his team. The pattern allows engineers at the subcontractor to automatically obtain access to data, computation and SLA resources at the prime contractors site and necessary security tokens.

Figure 15: Providing resources to a subcontractor

The final scenario shows how the pattern supports 3rd party licensing scenarios where for example, a customer procures a license from an Independent Software Vendor (ISV) and uses that license at a 3rd party service provider.

Figure 16: 3rd party licensing architecture

Figure 16 shows the stakeholders and the services that need to be deployed. The sequence of events is described below:

1. The service provider agrees an SLA with the ISV to run their software
2. The manager at the consumer agrees a license with the ISV (this is effectively an SLA)
3. The manager at the client organisation agrees an SLA with the service provider to use their resources (computation, storage and applications)
4. The resources (SLAs) are placed in the client organisation’s client management system, groups and policies are set up
5. A user discovers SLA, license and any required security tokens from the client management system
6. A user runs a job, passing over license and SLA to the service provider
7. The service provider checks the SLA to ensure that the activity is permitted and sufficient resource is still available to the client
8. The service provider checks with the ISV that they are permitted to run the application and that the customer has a valid license.
9. If all checks are successful the service provider can run the application

The scenarios above focused on collaborative engineering, however, the pattern has been verified and validated in industrial proof of concepts other important application sectors, for example:

- 3D Internet (Edutain@Grid)
- Business Intelligence (BEinGRID, NextGRID)
- Collaborative Engineering (SIMDAT, BRIDGE, BEinGRID, CFMS, CRISP)
- Finance (NextGRID, BEinGRID)
- Meteorology (SIMDAT, BRIDGE)
- Life sciences (SIMDAT, BRIDGE)
- Post production (MUPPITS)

**Impacts**

The impact of the pattern is greater system dynamics when orchestrating services at runtime, business stakeholder policy injection and policy administration efficiencies in service-oriented relationships.

**Relationships**

This pattern depends upon the WS-Trust/WS-Federation pattern

**References**

Summary Card

(Contextualized B2B) Service Discovery

How to discover services in the context of business relationships, procurement and project management functions? How can existing Web Service registries gain capabilities required by business today?

Problem

Business to Business (B2B) collaboration demands not only discovery of application resources based on available metadata, but also discovery of these resources in the context of security policy and agreed contracts, like Service Level Agreements.

Solution

GRIA’s services and client API provides discovery through procurement approach that links service discovery to procurement processes by combining workflow, contextualised service registries and security. Consumers automatically discover services they want to use and also discover and obtain the necessary federation contexts (SLAs) and security attributes to access them.

Application

Verified and validated across a range of sectors including collaborative engineering, life sciences, post production, finance, etc.

Impacts

Greater system dynamics when orchestrating services at runtime, business stakeholder policy injection and policy administration efficiencies.

Principles

Service reusability, autonomy, quality, agreement, loose coupling, and discoverability.

Interoperability, security, extensibility, usability.

Compliance to standards, governance.

Architecture

Service (Service Package)
Message (Client Package)
Security (Client Management Package)
Discovery (Client Management Package)

Status

Contributor Name

Mike Boniface, Nikolaos Matskanis

Contributor Notes
3.3 **SLA-Template-based service description** pattern.

Keven Kearney, Costas Kotsokalis, Francesco Torelli

**Name**

SLA-Template-based service description

**Requirements**

How can a service be described and discovered, with regard to its functional and non-functional specification, including indications about agreements it is willing to establish.

**Icon**

**Problem**

While in the traditional view a customer only searches for a service endpoint providing specific functional and QoS properties, in a SLA context the consumer searches for a provider able to supply a specific kind of service and also willing to accept specific guarantees and agreement terms. In other terms, we assume that in real business the customers search for a specific kind of SLA more than just a specific kind of service. Suitable representations must be used to express the consumer requirements and the provider offers.

**Solution**

The solution proposed, is to use the SLA templates which may be advertised by a service, as its service description.

A SLA Template (SLAT) responds, partially or in complete, to the following questions:

- What exactly does the service do? What is its exact functionality?
- How well does the service function? How efficient is it in completing its tasks?
- Which parts of this functionality and the associated efficiency/performance may be the topic of a negotiation with potential customers?
- What are the acceptable limits that must be respected by the potential customers, when customising the template in anticipation of negotiating certain service properties?

A proper registry will hold these templates. Following a request (with a template as input) it will provide all matching templates as output. The registry will have to implement smart matchmaking algorithms, so that the input template (containing functional/non-functional properties) can be matched to templates that may be structurally different, but hold the same information, semantically. For instance, the query template carrying the property “service must provide response time less than 5 seconds” should be recognised as semantically
compatible to the advertisement “service response time may be negotiated and cannot be less than 2 seconds”. Therefore, the SLAT corresponding to the latter statement should be returned as response to a SLAT-based query based on the former statement.

Service customers will search the SLAT registry to discover available SLATs (i.e. available service offers). After selecting a SLAT the customer modifies the free variables (terms) available in it in order to define a SLA for a specific service provisioning and propose such a SLA to the provider. Based on a predefined negotiation scheme, there may be various counter-offers, until the two parties reach an agreement of give up.

Using the service description part of the SLA template, it is possible to encode in there the information on functional properties. Therefore, the SLA template registry may be used as a service registry, allowing to take into account both functional and non-functional service properties alike.

**Application**

A concrete syntax of SLATs can be based on the WS-Agreement standard. WS-Agreement is, however, an empty framework. As such, it must be complemented with other languages for the description of both functional and non-functional service properties, as well as policies (with reference to acceptable limits of negotiable properties). Various such languages for the properties description exist, e.g. SAWSDL, WSMO, OWL-S. Additionally, WS-Policy may be used for defining constraints in the SLAT.

**Impacts**

Describing a service through a commonly understood template provides the potential customer not only with information about the service itself but also, to some extent, about the policies of the provider with regard to offering and providing this service. For instance, the exact same service (with regard to its functional and non-functional properties) may be offered by two different providers; however, the corresponding SLATs may advertise different negotiable properties and limits for these properties, effectively resulting to two different service offerings.

The SLA context of service description also has impact on the boundaries of the discovery & selection process. Usually it is assumed that the discovery process ends with the identification of one or more service descriptions, matching with the customer requirements, and optionally with a ranking of the matching results. The provider having the highest ranking is usually selected. Ideally, it is assumed that if the sources of such descriptions are up to date and trustable, then the described services are in general ready for use (i.e. the unavailability of the service is treated as an exception).

In a SLA context the use of a service is preconditioned by the existence of a SLA and this is in general the result of a negotiation between the consumer and the customer (possibly a trivial negotiation consisting just of the implicit
acceptance of the SLA by the customer). The information that the providers publish (by registries or other advice means) is just an assertion of general intentions. Therefore it is, in a sense, incomplete and uncertain. Incomplete because the exact service provision will be in fact defined by the result of the negotiation and uncertain because the negotiation could fail, i.e. the provider is free to refuse an agreement also if this is compliant with its advice.

As a consequence, we cannot consider the discovery & selection problem resolved by just selecting the best matching description. The problem really ends when an agreement is established. After that, the agreed upon service may be considered ready for use at the agreed time and conditions (note also that in spite of the agreement, the unavailability is still possible and must be treated as an exception).

Relationships
Service description in a SLA context is directly related to many other patterns in a SOA environment. The more evident ones are:

- SLA Template definition
- Customer-side creation of service queries
- Service matchmaking and discovery
- SLA negotiation

References

Summary Card

<table>
<thead>
<tr>
<th>SLA-Template-based service description</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can a service be described and discovered, with regard to its functional and non-functional specification, including indications about agreements it is willing to establish.</td>
</tr>
</tbody>
</table>

Problem
Coming up with a service description methodology so that customers can search not just for services, but rather for SLAs, using SLA templates as input and output of the service discovery process.

Solution
Describing services through templates which in turn describe all possible SLAs for a service. Designing proper algorithms for matching structurally different templates, so that a customer’s query can be matched to semantically compatible (but structurally different) templates.

Application
WS-Agreement, WS-Policy, and QoS-

Impacts
Flexibility (same services offered by
<table>
<thead>
<tr>
<th>enabled service description languages such as SAWSDL, OWL-S, WSMO etc.</th>
<th>different providers and corresponding to different possible SLAs); boundaries of discovery and selection process, involvement of negotiation process.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Principles</strong></td>
<td><strong>Architecture</strong></td>
</tr>
<tr>
<td></td>
<td>Discovery</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td><strong>Contributor Name</strong></td>
</tr>
<tr>
<td></td>
<td>Keven Kearney, Engineering (Ingegneria Informatica), Italy</td>
</tr>
<tr>
<td></td>
<td>Costas Kotsokalis, Dortmund University of Technology, Germany</td>
</tr>
<tr>
<td></td>
<td>Francesco Torelli, Engineering (Ingegneria Informatica), Italy</td>
</tr>
</tbody>
</table>
3.4 **Group negotiation during discovery** pattern.

Andras Micsik

**Name**
Group negotiation during discovery

**Requirements**
How can we apply dynamic negotiation during service discovery?

**Icon**

**Problem**
The decision on accepting a service request (SP side) or selecting a service (customer side) may depend on the dynamically changing current situation. Furthermore, making this decision may require the use of artificial intelligence on the fly. Searching in static repositories may not be the most adequate approach in rapidly changing environments. For example, if many requests arrive in a short period to a service, then the capacity of the service is blocked for a period of time, when it should not accept more jobs, and allow other services to “take their part”.

**Solution**
The solution is to represent each service request and service with a piece of “intelligent” software, for example with an agent. Then traditional agent coordination mechanisms can be used, such as auctions, Contract Net protocol, etc. A service or SP can intelligently decide whether to accept a service request and with what SLA. The essential change compared to the traditional “search – read results – select” method is the dynamic negotiation. For example, first a request is broadcasted to all possible services. Services which want to take the request respond with an offer containing pricing and SLA suggestions. The offers are compared and a next round of negotiation can be accomplished with a smaller set of services. Or, the agreement can be made with a single selected service.

**Application**
In a typical case, agents can take the role of customer and service provider. With a limited built-in intelligence they can decide on what offers or requests to accept. The usual protocols used in this case are auctions and contract net. If several cycles are planned, then care must be taken to finish the negotiation within a given timeout.

**Components:**
- Agents or other software components representing customers (and/or requests)
- Agents or other software components representing service providers (and/or offers)
- Agent communication mechanism (e.g. whiteboard, broadcast facility)

**Impacts**

**Benefits:**
- More flexible matchmaking
- Dynamic optimization is possible for various purposes (income, speed, capacity, etc.)

**Drawbacks:**
- Slower response time, possible indeterminism in discovery result
- More communication over the wire is needed

**Alternatives:**
- Using a centralized algorithm

**Relationships**

This pattern assumes certain communication channels exist, and also it interferes with patterns related to service selection and negotiation.

**References**


**Summary Card**

<table>
<thead>
<tr>
<th>Group negotiation during discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>How can we make discovery more dynamic for service providers and consumers?</em></td>
</tr>
</tbody>
</table>

**Problem**

The decision on accepting a service request (SP side) or selecting a service (customer side) may depend on the dynamically changing current situation. Discovery results based on static service descriptions may not give the best results in rapidly changing environments.

**Solution**

The solution is to represent each service query and service offer with a piece of “intelligent” software, for example with an agent. Then offers and queries can be formulated dynamically depending on the client, current workload, etc. Furthermore, traditional agent coordination mechanisms can be used, such as auctions or Contract Net protocol.

**Application** | **Impacts**
--- | ---
<p>| | |
| | |</p>
<table>
<thead>
<tr>
<th>Principles</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-Agreement or FIPA protocols can be used.</td>
<td>Discovery</td>
</tr>
<tr>
<td></td>
<td>Selection</td>
</tr>
<tr>
<td>More flexible matchmaking</td>
<td></td>
</tr>
<tr>
<td>Dynamic optimization is possible</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Contributor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributor Notes</td>
<td>András Micsik, SZTAKI</td>
</tr>
</tbody>
</table>
3.5 **Multi-phase discovery** pattern.

**Name**
Multi-phase discovery

**Requirements**
How can good quality discovery results be produced in short time?

**Icon**

**Problem**
For complex service queries, when several criteria are given, or logical querying is used, it is hard to produce the results in short time. Logical querying has typically slow response time.

**Solution**
A possibility is to narrow down the set of possible matches step-by-step, applying different query mechanisms in each step.

This pattern is neutral to service modelling technique (user goal/requirements and service capabilities specification). It has been tested with WSMO, but it should work with OWL-S and any other service modelling technique as well.

The choice of service description registry does not affect the pattern significantly. It is an advantage if the registry supports service description using flexible data structures, or using different service description schemas.

The query language to be supported depends on the filtering technique and the data storage mechanism used in the different phases of service discovery. If service descriptions are indexed using XML database, then XQuery is the right choice, if service descriptions are indexed using keywords, then SQL or Google-like queries might be proper. In case of semantic service descriptions, some kind of semantic query facility should be established. SPARQL is the most obvious choice here, as it is standardized. However, querying based on Datalog or Prolog is similarly feasible, and has been used in our test cases.

**Application**
For example, the first phase may use textual queries run against full-text indexes, or an SQL query filtering on simple criteria. The second phase then runs the slower query algorithm on a much smaller set of data, which can significantly reduce response time. In a concrete example we implemented this approach using text-based query in first phase and Prolog in the second phase.

**Components:**
Service Discovery Engine embedding:

- Several types of service databases or indexes
- The implementations of the algorithms used in each query phase

Multi-phase discovery typically involves:

- Service description registries
- Indexes or querying facilities
- Optionally converters between different description types
- Discovery engine

A full implementation of such a scenario exists for WSML, which can be downloaded from the webpage of the INFRAWEBS project\(^6\). The DSWSR component provides the service description registry, while the Service Access Middleware (SAM) provides the multi-phase discovery solution. This solution requires user goals and service capabilities described using WSML (some examples included).

Additionally, for each part we list possible software to be used:

- **Service description registries:**
  - Any kind of registry with an open query interface and flexible description schema support.

- **Indexes or querying facilities:**
  - For keyword-based techniques: Apache Lucene, any SQL server or full-text search engine.
  - For semantic techniques:
    - OWL/RDF tools: Jena, Sesame, SPARQL query engines, etc.
    - LP tools: various Prolog implementations, e.g. XSB, SWI-Prolog

- **Optionally converters between different description types:**
  - Keyword extractor, WSML to Prolog converter can be found in the INFRAWEBS SAM component, as example. Furthermore, custom scripts, XSL, or any other usual text processing tool can be used.

- **Discovery engine:**
  - The INFRAWEBS SAM component contains an example. Similar paradigm exists in the OWLS-MX\(^7\) software.

---


\(^7\) [http://www.ags.dfki.uni-sb.de/~klusch/owls-mx/](http://www.ags.dfki.uni-sb.de/~klusch/owls-mx/)
**Impacts**

**Benefits:**
- Faster response

**Drawbacks:**
- The phases must be planned carefully not too loose some matches during the phases
- More complex architecture

**Alternatives:**
- Using a single query algorithm which is fast enough

**Relationships**

This pattern is embedded by the basic “Service discovery” pattern, which accepts a query and produces a list of matching services.

**References**


**Summary Card**

<table>
<thead>
<tr>
<th>Multi-phase discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>How to combine several techniques in order to reduce the calculation time or improve the quality of discovery results?</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Problem</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Each query technique has its drawbacks and advantages. Logical querying, for example, has typically slow response time, but good quality.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Solution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Several query techniques can be combined in a step-by-step manner so that the set of possible matches is narrowed in each step.</td>
</tr>
<tr>
<td><strong>Application</strong></td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>In any discovery engine</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Principles</strong></th>
<th><strong>Architecture</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Discovery</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Status</strong></th>
<th><strong>Contributor Name</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>András Micsik, SZTAKI</td>
</tr>
</tbody>
</table>

**Contributor Notes**
3.6 **Template based discovery** pattern.
Andras Micsik

**Name**
Template based discovery

**Requirements**
How can a user create powerful service queries?

**Problem**
When a user wants to find a service with required capabilities, it is not necessarily easy for her to construct that query. If the query contains only keywords as input, then the results may not be satisfactory. If the system can accept several search criteria, then it usually has an internal language for accepting these criteria. This internal language is often too complex for average users to be used (e.g. SPARQL, OWL or some other logical language). So the user needs help to be able to create powerful queries.

**Solution**
A practical solution is to allow users to use predefined templates for assembling queries. These templates may take the form of an on-line form or any other visual (e.g. Flash) interface. The interface records various criteria input by the user. The filled template is then converted to the internal language of the service discovery engine. The engine responds with relevant results, which may need some translation before display to a format corresponding to the template.

**Application**
A real use case example may be a semantic discovery engine which requires queries in the form of WSML goals. The user first selects basic goal type, which has a corresponding WSML template. In the next step the user fills a dynamically generated form, which is generated from the previously selected WSML template. The form contains input fields for specific criteria, which may be filled or left empty. The field values are then inserted into the WSML template, and thus a complete WSML goal is formed. This WSML goal can be fed into the discovery engine. The result is a set of matching WSML service descriptions. Concrete values from these descriptions may be extracted and displayed to the user together with the list of matching services.
Components:

- Service Discovery Engine
- Template Repository
- Template Translator: translates between user-understandable and machine-understandable formats (e.g. from WSML to HTML and backwards)
- Service Discovery User Interface

This pattern is most suitable for semantic service modeling techniques (user goal/requirements and service capabilities specification). It has been tested with WSMO, but it should work with OWL-S and any other semantic service modeling technique as well.

The choice of service description registry does not affect the pattern significantly. It is an advantage if the registry supports storing service discovery templates as well.

The query facility that can be used depends on the semantic service description used. In case of OWL-S or other RDF or OWL-based technique SPARQL can be a natural choice. In case of WSMO the WSMO goals may be translated into various query languages depending on the WSML variant applied. However, it is possible to implement the pattern using Datalog or Prolog style queries as well.

Template-based discovery typically involves the following components (with list of applicable software attached):

- Discovery User Interface
  This is usually a custom solution, it may be solved using HTML forms, for example.
- Template Repository
  The INFRAWEBS DSWSR component is able to store templates, but any usual file-based storage may be suitable.
- Template Translator
  The INFRAWEBS Service Access Middleware (SAM) provides an example for template translator functionality for the case of templates based on WSML goals. The functionality of the translator is to convert the template filled by the user into a machine-understandable service query. An alternative solution might be to convert HTML forms into SPARQL queries.
- Discovery engine

---

The INFRAWEBS Service Access Middleware (SAM) provides a discovery engine which works together with a template translator.

**Impacts**

**Benefits:**
- Easy to learn and use for average users
- The user interface becomes more independent of internal query language

**Drawbacks:**
- The expressive power of the internal query language cannot be fully exploited using templates (i.e. not all possible queries can be formulated using templates)
- The translation process may add errors to the results

**Relationships**

This pattern uses the basic "Service discovery" pattern, which accepts a query and produces a list of matching services.

**References**


### Summary Card

<table>
<thead>
<tr>
<th>Template based SD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>How can a user create powerful service queries?</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>When a user wants to find a service with required capabilities, it is not necessarily easy for her to construct that query, especially if the query has to be constructed in a specific language such as SPARQL or WSML.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A generic solution is to allow users to use predefined templates for assembling queries with a certain generic goal. These templates may take the form of an on-line form or any other visual interface, where the template parameters can be provided.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any discovery engine. For example, creating WSML goals.</td>
<td>Easy to use for non-expert users</td>
</tr>
<tr>
<td></td>
<td>User interface becomes more independent of underlying query language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Principles</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Discovery</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Contributor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>András Micsik, SZTAKI</td>
</tr>
</tbody>
</table>

| Contributor Notes |
3.7 **Distributed (Peer to Peer) Registry Repository** pattern.
Valentín Sanchez et al.

**Name**
Distributed (Peer to Peer) Registry Repository

**Requirements**
On of the main Service-Oriented Design Principles is the so-called Service Discoverability. This design principle implies that services should be supplemented with communicative metadata by which they can be effectively discovered and interpreted. It also leads to the definition of a SOA module used to store services metadata and provide browsing and searching capabilities: the Service Registry/Repository.

**Icon**

**Problem**
Two of the main concerns of the service registry are “availability” and “scope”. In a SOA environment, the service registry becomes a single point of failure, which can compromise the availability of the whole system. The other feature, scope, is related to the number of registered services. In order to increase the scope a federated service registry is a common solution.
We provide another solution which improves the availability and scope of the service registry. The solution is complementary to the other two approaches.

**Solution**
What we propose is another kind of SOA Architecture where each node can act as a service provider, service consumer and a service registry/repository. It defines a peer to peer distributed service infrastructure (SOA nodes network).
Figure 17 Distributed (Peer to Peer) Registry Repository Class Diagram

Figure 18 Peer to peer service discovery sequence diagram
Application

In the context of Service Discovery this pattern is used to find new services the same way a “normal” service registry is used. It can be used in “requirement time”, in design time or in runtime to provide automatic composition of services.

Impacts

The peer-to-peer SOA approach is not a single design pattern, but a new architectural solution to implement the SOA strategy. This new approach can be useful not only for service visibility issues but also for security, policy and trust management, services governance and management, quality of service issues, etc.

What we call “service infrastructure” appears as a first-class SOA concept, like “service provider”, “service consumer” or “service registry”. Another important point is a new classification of services: infrastructure or “core” services and business services. The relationship between the service implementation and the service infrastructure it lives is a very important issue to take into account when defining a SOA system.

Relationships

References


Summary Card

<table>
<thead>
<tr>
<th>Distributed (Peer to Peer) Registry Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
</tr>
<tr>
<td>Two of the main concerns of the service registry are “availability” and “scope”. In a SOA environment, the service registry becomes a single point of failure, which can compromise the availability of the whole system. The other feature, scope, is related to the number of registered services.</td>
</tr>
<tr>
<td>Solution</td>
</tr>
<tr>
<td>A new SOA architecture where each node can act as a service provider, service consumer and a service registry/repository. It defines a peer to peer distributed service infrastructure (SOA nodes network).</td>
</tr>
<tr>
<td>Application</td>
</tr>
<tr>
<td>This pattern is used to find new services. It can be used in “requirement time”, in design time or in runtime to provide automatic composition of services.</td>
</tr>
<tr>
<td>Impacts</td>
</tr>
<tr>
<td>No single point of failure for discovering new services. SOA infrastructure in each node: services clients and providers.</td>
</tr>
</tbody>
</table>

NEXOF-RA • FP7-216446 • D7.2 • Version 1.0, dated 22/05/2008 • Page 46 of 57
<table>
<thead>
<tr>
<th>Principles</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distributed. Peer to peer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Contributor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valentín Sánchez</td>
</tr>
</tbody>
</table>

Contributor Notes
3.8 **Service Matchmaking and Ranking** pattern.
Dimitrios Skoutas et al.

**Name**
Service Matchmaking and Ranking

**Requirements**
How can appropriate services be identified and ranked with respect to the description of a requested service.

**Problem**
Due to the increasing popularity, adoption, and availability of services, the problem of effectively and efficiently discovering and selecting appropriate services to meet specific user or application requirements or to compose complex workflow processes becomes a critical issue. Typically, the user provides, through some comprehensive interface, a definition of the desired service and poses a query on a system maintaining a repository of advertised service descriptions. Alternatively, the user could identify a service, e.g., from some previous query or simply by browsing the registry, and request similar results. Then, the search engine needs to employ a matchmaking algorithm in order to identify appropriate services, and rank them based on their relevance to the request.

**Solution**
This problem is addressed by the service discovery engine. The discovery process takes place in two main steps, described in more detail in the following. First, a matchmaker component applies one or more matching criteria to assess the degree of match between service parameters. Then, a ranking component returns a ranked list of results, according to their overall degree of match with the request.

Regarding the first step, matching can take into account a variety of criteria, ranging from string matching between parameter names and/or associated comments to matching using auxiliary information from a domain thesaurus. In particular, there are two major paradigms for assessing the degree of match between service parameters. The first views the service discovery and selection process from an Information Retrieval (IR) perspective. It treats the parameter descriptions as documents, and it employs basic IR techniques to perform keyword-based search on them. In this case, the user issues a query comprising one or more keywords describing his/her information need, and the search engine returns available services which contain these keywords in their descriptions. The relevance of a result is typically determined based on term frequency measures. Also, typical schema matching techniques can be incorporated, taking into consideration the structure of service parameters,
comparing the corresponding data types, as well as performing linguistic analysis, e.g., applying string similarity measures on parameter names or associated comments, to compute the degree of match between the service request and the service advertisements. However, since service descriptions are essentially very short documents with few terms, keyword-based matchmaking often fails to properly identify and extract semantics.

The second paradigm follows the Semantic Web vision. Services are enriched by annotating their parameters with semantic concepts taken from domain ontologies. Then, estimating the degree of parameter match reduces to a problem of logic inference: a reasoner is employed to check for equivalence or subsumption relationships between the concepts of the parameters under investigation. Based on the result of this inference, a degree of match is assigned to each parameter pair. More specifically, typically the following discrete types are used to characterize the match: exact, if the request is equivalent to the advertisement; plug-in, if the request is subsumed by the advertisement; subsume, if the request subsumes the advertisement; intersection, if the intersection of the request and the advertisement is satisfiable; and disjoint, if the two concepts are disjoint. More elaborate match results can be achieved, depending on the expressiveness of the ontology language used, i.e., whether a lightweight ontology (i.e., one consisting only of IS-A relationships) or a more expressive ontology (e.g., an OWL ontology, comprising properties, restrictions, etc.) is used.

Although this approach takes into consideration the semantics of the service parameters, it still suffers from several limitations, mainly due to the lack of available ontologies, the difficulty in achieving consensus among a large number of involved parties, and the considerable overhead in developing and maintaining an ontology, and using it to semantically annotate the available services. Thus, hybrid techniques, employing criteria from both the aforementioned approaches, can also be considered, applying both IR-based search and logic-based matching. Then, the overall degree of match is determined as an average from the two types of measures, e.g., a string similarity on the parameter names and a semantic similarity determined from an accompanying ontology.

Regarding the second step, the service discovery engine computes the overall degree of match for the requested and the available services, taking into consideration the individual scores of corresponding parameters. There are various approaches to this step, as well. First, an overall degree of match can be derived from the individual scores by applying some aggregation function, such as min, max or mean, which, however, leads to loss of information that may significantly affect the quality of the retrieved results. To avoid this, appropriate weights should be assigned to the degrees of match of individual parameters, reflecting the relative importance of each parameter. However, choosing the appropriate weights for each contributing factor requires either a-priori knowledge of the user’s preferences or relying on user feedback and machine learning techniques. Another approach that overcomes these problems is to address service selection based on the notions of dominance and skyline queries. Skyline queries aim at identifying “interesting” objects from
a set of objects described in a multi-dimensional space. More specifically, a skyline query retrieves all those objects that are not dominated by any other object. An object is said to dominate another object if it is better or equal to it in all dimensions and strictly better in at least one dimension. In this context, a given service advertisement is preferred if it provides a closer (or at least equal) match in all the requested parameters. Thus, this approach allows retrieving the best match results in a way that does not require a priori knowledge of specific user preferences.

Finally, to allow for fast service selection at run-time, the service discovery engine should support an appropriate indexing of the service descriptions. The type of the index depends on the type of the descriptions and the type of the matchmaking used. In particular, an inverted index can be used to support IR-style retrieval, while labelling schemes can be used to speed-up the matchmaking process in the case of ontology-based match.

Application

The application of this design pattern may vary depending on the description language and the matchmaking criteria used. Current industry standards for describing and locating Web services, namely WSDL and UDDI, focus on describing the syntactic aspects of a service and on performing keyword-based matching. On the other hand, three main approaches exist to semantically enhance Web service descriptions: SAWSDL, WSMO and OWL-S. The main idea underlying these approaches is to use appropriate ontologies to semantically annotate several aspects of a Web service, such as inputs, outputs, preconditions, and effects, as well as non-functional parameters (e.g., QoS aspects).

Given a service request, the discovery engine first selects one or more matching criteria to apply. For each requested parameter, it queries the underlying registry(-ies), applying the selected criteria, to identify services providing a match for this parameter. Then, those services having a match for all (or perhaps most) of the requested parameters are selected, and they are ranked accordingly. For this purpose, the discovery engine can compare and rank the match results using the notion of dominance. A match result $S_1$ dominates another match result $S_2$, if $S_1$ is better or equal to $S_2$ in all the requested parameters and strictly better in at least one parameter. Then, a match result is ranked high, if it dominates as many other matches as possible, and it is dominated by as few other matches as possible.

Impacts

It may occur that no available service provides a sufficient match for a given request. However, it is often possible to achieve the requested goal by composing together more than one services to provide the overall functionality. Thus, in that case, the discovery engine should support functionality for composing existing services, for example by matching the outputs of one service with the inputs of another. This, however, increases the complexity of the discovery process and may incur a significant additional cost.

Relationships
Since the type of matchmaking that can be applied between service requests and service advertisements depends on the information provided in service descriptions, this design pattern depends on the pattern followed for service discovery. Also, there is relationship between service discovery and service composition, since (a) searching for a service may lead to the need for composing existing services, and (b) during service composition appropriate services may have to be discovered.

References

Summary Card

<table>
<thead>
<tr>
<th>Service Matchmaking and Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can appropriate services be identified and ranked with respect to the description of a requested service?</td>
</tr>
</tbody>
</table>

Problem

When a user requests a service, the service discovery engine should identify potential matches and select the best ones.

Solution

The service discovery engine can employ various matching criteria to find services matching the requested parameters, and then exploit these partial match results to determine the overall degree of match between the requested and the available services.

Application

The user issues a service request, by specifying the desired parameters in a search interface or browsing for an existing service and requesting similar results. Then, the engine retrieves potential matches performing keyword-based or logic-based match.

Impacts

The discovery engine should be able to compose existing services in case there is no single service that provides a sufficient match to the request.

Principles

- Service discoverability

Architecture

- Discovery

Status

Contributor Name

- Dimitrios Skoutas

Contributor Notes
4 IT CONCLUSIONS.

This IT has received and discussed on several important and relevant contributions which cover most of the topic aspects that were considered when constituting the IT. An important number of contributions have addressed the service discovery problem for provisioning usage but only one has addressed the advertisement usage, whereby only one design pattern for service catalogues has been analysed.

Regarding the provisioning usage of service discovery, some contributions have proposed design patterns for representing the user's request using similar specialized template-based approaches. Most of contributions have described patterns for service matchmaking, ranking and selection, ranging over the most relevant current approaches based on IR and semantic modelling and reasoning techniques. Besides, special usages and techniques for service discovery such as SLA-based service discovery, negotiation-phase service discovery or multi-phase service discovery were considered by some contributions.

We have missed contributions in some scenarios identified in the IT invitation, such as requirement-based service discovery and architecture-based service discovery. However, we believe that the design patterns analysed by the IT cover the most relevant aspects of the service discovery concern and provides a representative survey of the current research trends and achievements.
5 APPENDIX I: DESIGN PATTERN TEMPLATE.

This DP template was extracted from NEXOF-RA D7.2 authored by Piero Corte and Debora Desideri.

Each of the patterns that are part of the NEXOF-RA are described using the same profile format and structure based on the following parts:

- Name
- Requirement
- Icon
- Problem
- Solution
- Application
- Impacts
- Relationships

A further part is provided to summarize the overall description and it is named

- Summary Card

The following sections describe each part of a pattern profile individually.

Name
It is the name of the pattern.
An example of pattern name:

Canonical Data Format

Requirements
A concise, single sentence statement that presents the fundamental requirement addressed by the pattern in the form of a question. Every pattern description begins with this statement.
An example of a requirement statement:

How can a service be designed to avoid data format transformation?
UML use cases and/or activity diagrams can be added to better describe the requirements in terms of usage scenarios.

**Icon**

Each pattern description is accompanied by an icon image that acts as a visual identifier.

An example of a pattern icon:

![Pattern Icon](image)

**Problem**

The issue causing a problem and the effects of the problem are described in this section, which is often accompanied by a figure that further illustrates the “problem state.” It is this problem for which the pattern is expected to provide a solution.

**Solution**

The design solution proposed by the pattern to solve the problem and fulfil the requirement. Often this is a short statement accompanied by a diagram to concisely communicate the final solution state. “How-to” details are not provided in this section, but are instead located in the Application section.

UML diagrams (Logical and Process view) must be provided to describes the internal structure and behaviour of the pattern as well as its Provided and Required interfaces.

**Application**

This part is dedicated to describing how the pattern can be applied. In can include guidelines, implementation details, and sometimes even a suggested process.

UML diagrams (Logical and Process view) can be provided to describes the internal structure and behaviour of the pattern as well as its Provided and Required interfaces.

**Impacts**

Most patterns come with trade-offs. This section highlights common impacts and requirements associated with the application of a pattern and may also provide alternatives that can be considered.
Note that these consequences are common but not necessarily predictable. For example, issues related to performance requirements are often raised; however, these issues may not impact an environment with an already highly scalable infrastructure.

**Relationships**

The use of design patterns can tie into all aspects of design and architecture. It is important to understand the requirements and dependencies a pattern may have and the effects of its application upon other patterns. This section is therefore dedicated to documenting key pattern relationships.

The content in this section is not exhaustive in that not all possible relationships a given design pattern can have are covered. This section highlights key relationships only, with an emphasis on how patterns support or depend on each other.

Each Relationships section is accompanied by a diagram illustrating the pattern relationships.

Details regarding the format of pattern relationship diagrams are covered in *Errore. L'origine riferimento non è stata trovata.* section.

UML diagrams can be used to describes such relationships. The following table shows how to model the different type of pattern relationships.

<table>
<thead>
<tr>
<th>Relationship type</th>
<th>Style</th>
<th>UML diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses relation, where a pattern &quot;uses&quot; another pattern</td>
<td><em>Use Style</em></td>
<td>Class Diagram</td>
</tr>
<tr>
<td>PartOf relation, where a pattern is part of another pattern</td>
<td><em>Decomposition Style</em></td>
<td>Class Diagram</td>
</tr>
<tr>
<td>Refines relation, where a pattern &quot;refines&quot; another pattern</td>
<td><em>Generalization Style</em></td>
<td>Class Diagram</td>
</tr>
<tr>
<td>Conflicts relation, where a pattern &quot;conflicts&quot; with another pattern</td>
<td>-</td>
<td>Class Diagram using an association with the following stereotype: &lt;&lt;conflicts&gt;&gt;</td>
</tr>
</tbody>
</table>

**References**

List of complementary references
Summary Card
The summary card is used to summarize the overall description of the pattern and provides a concise synopsis of it.

It contains:

- information extracted from the following sections of the pattern profile:
  - Name
  - Icon
  - Requirement
  - Problem
  - Solution
  - Application
  - Impacts

- references to related service-orientation design principles and service-oriented architectural types via the following sections:
  - Principles (see Errore. L'origine riferimento non è stata trovata.)
  - Architecture

- other information concerning
  - Status (draft/final)
  - Contributor Name
  - Contributor Notes

Hereafter an example of summary card.

### Canonical Data Format

<table>
<thead>
<tr>
<th>How can a service be designed to avoid data format transformation?</th>
</tr>
</thead>
</table>

**Problem**
Services that use different formats to represent data cause interoperability concerns, raise difficult runtime exceptions, and require the need for undesirable data format transformation measures

**Solution**
The format used to deliver data between services is standardized across all services within an inventory, thereby guaranteeing a base level of
interoperability.

<table>
<thead>
<tr>
<th>Application</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depending on the communication protocols being used, this pattern may dictate certain file formats or, in the case of Web services, the use of the WS-I Basic Profile.</td>
<td>Data format standardization can lead to communication limitations associated with the chosen format type.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Principles</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardized Service Contract</td>
<td>Registry</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Contributor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draft</td>
<td>Thomas Erl</td>
</tr>
</tbody>
</table>

**Contributor Notes**

While there’s not harm in standardizing on common data formats, this form of standardization may not warrant a separate design pattern. Data Format Transformation is often carried out within service boundaries to overcome legacy compliance issues. In many of these situations, the data formats cannot be standardized until the legacy system is replaced. Also, the application of the Canonical Protocol pattern often will result in the standardization of data formats that are established with the communication protocol technology.