Towards an Architecture for Distributed Reputation-Enhanced Web Service Discovery

(version 1.0, 22/12/2008)

Abstract –
This deliverable introduces the idea of an architecture for distributed Web Service discovery, describing the scenario and related technologies, defining a model to represent the information involved in the process and sketching a way to represent, store and retrieve it in a distributed architecture. It also introduces the idea of using reputation as part of this architecture, sketching a preliminary design of the reputation system.
Document information

Document ID code | D4.1.2v1
---|---
Keywords | Web Services, service discovery, P2P, reputation
Classification | FINAL
Distribution level | NeP4B Consortium

<table>
<thead>
<tr>
<th>Editor</th>
<th>Davide Cerri</th>
<th>CEFRIEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Davide Cerri</td>
<td>CEFRIEL</td>
</tr>
<tr>
<td></td>
<td>Alessio Carenini</td>
<td>CEFRIEL</td>
</tr>
<tr>
<td></td>
<td>Emanuele Della Valle</td>
<td>CEFRIEL</td>
</tr>
<tr>
<td></td>
<td>Francesco Corcoglioniti</td>
<td>CEFRIEL</td>
</tr>
</tbody>
</table>

Version history

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25/7/2008</td>
<td>DRAFT 0.1</td>
<td>Initial structure</td>
</tr>
<tr>
<td>3/10/2008</td>
<td>DRAFT 0.3</td>
<td>First draft version</td>
</tr>
<tr>
<td>22/11/2008</td>
<td>DRAFT 0.6</td>
<td>Draft version</td>
</tr>
<tr>
<td>22/12/2008</td>
<td>FINAL 1.0</td>
<td>Final version</td>
</tr>
</tbody>
</table>
1 Introduction

Deliverable D4.1.1 [6] provides requirements and state-of-the-art analysis for NeP4B reputation system, discussing some initial ideas about how a reputation distribution system for NeP4B can be designed. In order to proceed with the design of such reputation system, it is however necessary to think about its objective, i.e. how this is going to be used in the NeP4B platform.

The primary “objects” to which a reputation system can be applied in NeP4B are the “resources” that the semantic peers make available on the network, i.e. services and data sources. A fundamental aspect of WP4, however, deals with the problem of finding these resources in the network of semantic peers. Dealing with services, in particular, involves a discovery process accomplished by a discovery engine. In NeP4B network, however, many discovery engines exist (each semantic peer can have its own discovery engine), which leads to the need for a kind of “distributed discovery” process. It is therefore worthwhile to design the reputation system taking into account also this process rather than just the “resources”, and build a “reputation-enhanced” discovery mechanism.

In order to do this, it was necessary to start designing the distributed Web Service discovery process, identifying the involved actors, the requirements, the needed information and its representation, and the basic discovery mechanism. This is the reason why we chose to anticipate some work on the distributed Web Service discovery mechanism, and at the same time to delay some work on the reputation mechanism, so that the mutual relationships between the two tasks can be better exploited.

This deliverable reflects this choice, and it is divided into two parts. The first part deals with distributed Web Service discovery, describing the scenario and related technologies, then defining a model to represent the information involved in the process and sketching a way to represent, store and retrieve it in a distributed architecture. The second part deals with reputation, sketching a preliminary design of the reputation system. The design of this system will be finalised in an upcoming second version of this deliverable (D4.1.2v2), which will be delivered along with the design of the distributed Web Service discovery mechanism (D4.4.1).
2 Distributed Web Service Discovery

2.1 Scenario

The NeP4B vision depicts an Internet-based structured marketplace scenario where companies can access the huge amount of information already present in vertical portals, service and corporate databases and use it for dynamic, value-adding collaboration purposes. In this scenario, different Discovery engines allow a user to obtain ranked lists of services that are able to fulfill a user’s desire (Goals). In this decentralised scenario there can be multiple discovery engines, each one understanding just a subset of the possible user’s Goals, a particular category of services also exist, namely the mediation services. Those services are able, given a Goal, to produce another Goal that is more or less equivalent in its content, but expressed with a different structure. The aim of these services is to enable Goals interoperability just like a conversion filter transforms a document from its native format into a different format, while keeping the meaning of the content of the document itself.

In this scenario the concept of a Goal class represents an interface used to express a desire to a discovery engine, and as there exist different actors providing services, different actors can provide different Goal classes. Purchase groups or heavy buyers (large enterprises) are a first clear example of an actors category which can define a “standard interface” thus forcing service providers to write compatible service descriptions and discovery engine to accept the related Goal instances. At the opposite side of the producer-consumer relationship, Service providers, alone or grouped in a consortium, can also define a Goal class as a mean to be discovered, the description of the Goal class fully representing all the features offered by the service provider. A third category of Goal class provider is represented by service brokers, which can transparently aggregate homogeneous services (and thus providing a description of the common features inside the Goal class) and provide value-added services. Once a certain number of Goal classes linked to the same domain is deployed, new service providers can provide mediator services to enable goal instances to be processed and consumed by existing discovery engines.

Given the described distributed scenario, the problem of allowing service discovery turns into knowing which discovery engines are “reachable” (that is, which discovery engine can understand the goal instance expressed by the user) either directly or via a series of mediation services that transform a goal instance into an equivalent one. Proceeding step-by-step doesn’t beforehand guarantee any success of the overall discovery process, and as each transformation can be costly both in time and in money, it is important to reconstruct the connections graph to navigate it in order to choose the right path of mediators and avoid dead ends. Because of the various parameters affecting the scenario, we cannot calculate an optimal route from the user’s goal to the discovery engine by just knowing the elements of the connection graph, as:

- each mediator can introduce an information loss when processing the input goal instance;
- each mediator has a different cost both in terms of service fee and processing time;
- when more than one discovery engine can be reached, there is no hint about the quality of the services pool used by the single discovery engine before actually performing the actual discovery.
In this deliverable we deal with providing the user with a list of services which, invoked in the provided order, allow the user to fulfill his goals, while with distributed discovery we mean allowing service discovery in a distributed scenario with different actors responsible of different services rather than distributing the discovery process.

2.2 Service discovery in SOA

As shown in figure 1, the SOA paradigm for finding services is a two-step process that includes three actors, namely a Service Provider, a Service Consumer and a Service Broker. In order to be found by the consumer, the service provider must register his service(s) to a broker, which represents a third-party entity w.r.t. both the provider and the consumer (even if nothing forbids the provider to also act as a broker). Different service providers can register to the same broker, which thus acts as a form of central repository, or to different brokers, thus multiplying the information repositories at the expenses of the number of services that can be obtained with a single query. We will now introduce the basic concepts of two main standards that can be used: UDDI and ebXML standards define specifications for the Service Broker in different ways offer solutions for federating different repositories, but both require a certain trade-off between

![Diagram of Service Oriented Architecture](image)

Figure 1: Discovery in Service Oriented Architecture

2.2.1 UDDI

**General concepts**  The Universal Description, Discovery and Integration (UDDI) project is a worldwide project and initiative comprising companies such relevant as Computer Associates, IBM, Microsoft, Oracle and SAP that defines open and standard specifications for key Web Services Registry functionalities:

- Description of services and their providers
- Publication of information about services and their providers
• Discovery of services and/or providers of those services

• Creation and deployment of XML-based distributed registries

Nowadays the UDDI family of specifications are managed and published under the Organization for the Advancement of Structured Information Standards (OASIS) at http://uddi.xml.org.

The current version of the UDDI family of specifications is 3.0.2. published on the 3rd of February of 2005 ([9]). This is the version covered by this document.

Information model and categorization

The UDDI family of specifications is not a set of specifications developed exclusively to solve the registry problem in the Web Services area. In fact, the UDDI specifications are platform agnostic and can interact with businesses and services deployed on a variety of platforms. This much broader approach is clearly stated in the launching documents of the consortia set up to define the UDDI specifications: "The UDDI Project is an open industry initiative in which any organization can participate and implement the specifications. The specifications build on core Internet standards including TCP/IP, HTML, and XML and are independent of any underlying platform, language, object model, business application, or marketplace."

To manage this general purpose registry solution, UDDI introduces the concept of information model in the sense that every entity susceptible of being published in a UDDI registry has to be previously modeled using the tools and mechanisms defined by the UDDI specifications and presented further on. These tools and mechanisms allows the modeling of businesses and services respecting the platform agnostic principle characteristic of the UDDI specifications.

The UDDI information model is based in the concept of categorization. Every entity published in a UDDI registry can be categorized in depth using a fairly basic mechanism based on key-value pairs. To facilitate the categorization of the entities, the UDDI specifications include a set of standard categories that can be directly referenced and used like, for example, UNSPSC for product and service categorizations and ISO 3166 for the characterization of geographical regions. Anyway, nothing impedes the publisher of an entity to define new category sets or even to enrich existent ones.

UDDI information model

The UDDI specifications identify a set of entities that are susceptible of being registered in a UDDI registry. These entities are:

Provider: Entity that offers a service or set of services, including relationships with other providers of services.

Service: A set of operations or functionality grouped as a unit.

Binding: Technical information about how to interact with a service.

Technology Model (tModel): Service interaction specifications an categorization of the entities.

The service clients or consumers are not necessarily described. At least the UDDI specifications do not mandate that a service consumer should be registered to consume a service.
The UDDI specifications define a set of XML Schemas to describe each one of the preceding entities. These XML Schemas make it possible to characterize these entities in depth. We summarize the entities defined in the following table:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
<th>Attributes</th>
<th>Subentities</th>
</tr>
</thead>
<tbody>
<tr>
<td>businessEntity</td>
<td>A business or other organization that typically provides Web Services</td>
<td>businessKey</td>
<td>discoveryURL name description contacts businessServices identifierBag categoryBag signature</td>
</tr>
<tr>
<td>businessService</td>
<td>A collection of related Web Services offered by an organization described by a businessEntity</td>
<td>serviceKey businessKey</td>
<td>name description bindingTemplates categoryBag signature</td>
</tr>
<tr>
<td>bindingTemplate</td>
<td>Technical information necessary to use a particular Web Service</td>
<td>bindingKey serviceKey</td>
<td>description accessPoint or hostingRedirector tModelInstanceDetails categoryBag signature</td>
</tr>
<tr>
<td>tModel</td>
<td>A &quot;technical model&quot; representing a reusable concept, such as a Web Service type, a protocol used by Web Services, or a category system</td>
<td>tModelKey deleted</td>
<td>name description overviewDoc identifierBag categoryBag signature</td>
</tr>
</tbody>
</table>

Table 1: First level UDDI entities

The relationships amongst this entities are depicted in Figure 2 (extracted from [3]).

![Figure 2: Relationship amongst the UDDI first level entities](image)
Registry distribution  The UDDI specifications state that a set of Web services that implements at least one UDDI API set (for instance, the “publisher” API) is termed a UDDI node. One or more UDDI nodes may form a UDDI registry, with the restriction that a node may belong to one, and only one, registry. Registries can moreover form hierarchical relationships. While a UDDI node corresponds to a company branch office that maintains phone numbers for its employees, a registry is similar to a company-wide phone directory, of which the branch office directories are a part.

As the purpose of this standard is to provide a lightweight registry for SOA, not all the issues of distribution have been tackled yet. In particular, all the resource identifiers used by the registry are defined as local with respect to the registry itself, and this affects how distribution is achieved. Rather than distribution, the approach chosen in the UDDI specifications is indeed replication of data among the nodes forming a registry. Once a new node joins the registry, it asks all the available information from all the other nodes and then synchronizes its data with those of the other nodes.

The issue of overcoming the limitations of UDDI distribution, by either distributing queries or content, is discussed in several works. In [13] a P2P system is proposed to enable query distribution to the nodes of the registry, while the heterogeneity of service descriptions is tackled by using DAML-S ontological description. [5] also proposes a P2P architecture, but instead of using it to be aware of the nodes of the registry, it is used to better route the queries to the right UDDI node.

To embrace the UDDI vision, the NeP4B scenario should be formed by a series of UDDI nodes federated into registries, but, being decentralised by definition, each service provider should also maintain its own UDDI node. Moreover, in case of aggregating nodes and registries by domain, as each node can belong to one and only one registry, a service provider offering two services belonging to two different domains should also maintain two different registries. A third argument against the adoption of UDDI as the mean to implement distributed discovery is that, given the two previous arguments, a new node joining a registry should obtain a permission to interact with the other nodes, but no central authority can take the decision in a decentralised network.

2.2.2 ebXML Registry/Repository

ebXML is a set of standards intended to provide an open, interoperable and secure electronic business infrastructure for trading partners layered on top of Web Service standards. Thereby the ebXML specifications are focused on defining concepts and common methodologies that can be applied to enable efficient and interoperable implementation of eBusiness solutions by covering functionality from various aspects of B2B interactions [11]. In Figure 3, an overview of the ebXML specification technical architecture is presented.

Business interactions can be seen as series of exchanges of business documents between a set of actors (i.e. business services) and are described through business process descriptions. To establish a common understanding of business terms between partners, business documents are related to so-called ebXML core components that define a standard vocabulary through a common set of metadata. The Collaboration Protocol Profile (CPP) defines interaction partners with respect to e.g. their contact information, industry classification and their capabilities such supported business processes or provided service interfaces as well as their requirements on communication partners, e.g. regarding their interface or the underlying messaging service. A Collaboration Protocol Agreement (CPA) describes the in-
interaction properties mutually agreed by business partners. The rules governing the actual communication between business services (such as ensuring security, reliability and message sequence) is described in the ebXML messaging specification.

All data objects that need to be shared between interaction partners are stored/exchanged via an ebXML registry, defined in the Registry and Repository specification [2].

An ebXML registry provides a registry and repository service for service-oriented architectures, offering ways to efficiently and securely classify, store, manage, query and browse “any information that needs to be widely, yet securely shared among trading partners or potential trading partners” [14]. By providing a platform for exchanging classified business documents, the ebXML registry enables business partners to establish the shared context necessary for business interactions.

Thereby an ebXML registry supports advanced features like logging of lifecycle management operations in an audit trail, notifications, automatic versioning of stored information and means for distributing registries through federation. The ebXML Registry and Repository specification consists of the ebXML Registry Information Model (RIM) [1] specification and the ebXML Registry Services and Protocols (RS) [2] specification.

Use Cases and Example Since an ebXML registry essentially provides generic services for content and metadata management [2], it can support a number of potential use cases. In the context of a service-oriented architecture (SOA), an ebXML registry can be used for instance as (i) a Web Service Registry to support publication and discovery of Web Service descriptions (e.g. in form of WSDL [7] descriptions) and (ii) as a registry and repository for business process descriptions (e.g. in form of BPEL [4] processes). Furthermore, the ebXML registry can be used for serving a controlled vocabularies such as taxonomies or XML schema definitions.

In Figure 4, a simplified use case scenario for an ebXML registry is presented, describing its role as a service registry in the context of a service-oriented architecture [15]. Company
A wants to provide business partners with an externally accessible, aggregated Web Service. Prior to service development (step 1), Company A queries the ebXML registry for existing services that can be aggregated to form a new composite service. On the basis of the discovered services, the composite service is developed (step 2). After development, the aggregated service is registered at the ebXML registry along with necessary metadata, e.g. its service description (step 3), where the administrator of the registry defines policies regulating e.g. its visibility or rules for access (step 4). A potential service consumer (represented by Company B) can query the registry for suitable services and processes and download their metadata (e.g. their descriptions, associated policies) from the registry (step 5). After an agreement between interaction partners has been reached, the actual interaction is performed.

**Federation of registries/repositories** ebXML registry/repository specifications deals with the issues of distributing a registry on multiple nodes, or rather, with the issues of allowing a group of registries to voluntarily form a loosely coupled union. The registries federations can be based on common business/domain interests and specialties that the registries might share, and are based on a peer-to-peer (P2P) model where all participating registries are equal.

This enables operations such as:

- Cross-registry associations
- Federated queries
- Local caching of data from another registry
- Object relocation

While in UDDI replication was the only form of distribution, in ebXML it is just an option. Replication of RegistryObjects in other registries within a federation can improve access time and fault tolerance through local caching of remote objects, as it involves creation of a “local replica” that can be kept current using the event notification feature, or through periodic polling.

Even if ebXML specifications allow more complex ways of distributing content among the participants of a federation, there still exists the problem of deciding whether a new node is allowed to join, as there must be an explicit agreement among the participants of the federation for accepting a new node. Moreover, as in UDDI, it is not clear which actor in our scenario should be responsible of maintaining the infrastructure (either each provider
maintain its node in a pure P2P fashion or there is some sort of central or local authority). These issues prevents from adopting this standard for distributed discovery in the project scenario.

2.3 Towards a model for distributed discovery

Discovery engines, goal classes and ggMediators can be represented with a directed graph, as shown in figure 5 and as introduced in 2.1 with the concept of the “connection graph”. This graph shows the “connections” between these elements, i.e.:

- an arc from a goal to a discovery engine means that the discovery engine understand that goal class, i.e. it is able to discover services that satisfy a goal belonging to that class;

- arcs between goal classes and ggMediators mean that the mediator is able to translate a goal belonging to one class to a goal belonging to the other class.

Using the graph, a client having a goal is able to find out which discovery engines it can use, either directly (because the discovery engine understands that goal class) or indirectly (through ggMediators). For example, a client that has a goal belonging to class $C$ in figure 5 can submit its goal to discovery engine $\alpha$, or it can use a mediation service to translate the goal to class $D$ and then submit it to discovery engine $\beta$.

The client may actually not be interested in the intermediate goal classes, but only in the discovery engines that can be “reached” starting from a goal, and the mediation services (if...
any) that need to be invoked in order to reach a discovery engine. For this reason, partial “views” of the graph can be used, representing only this subset of information and related to each one of the goals. For example, figure 6 shows the views related to goals \( A \) and \( C \) in the graph shown in figure 5. In this representation, a part from the starting point\(^1\), all nodes correspond to services (discovery services and mediation services) and are thus related to peers in NeP4B network that offer them\(^2\).

![Figure 6: Different views of the graph in figure 5, w.r.t. to goal \( A \) (left) and goal \( C \) (right).](image)

The complete graph can contain cycles, because a chain of ggMediators can lead to the starting goal class. This is possible, at least in principle, also for the views. The client will anyway want to avoid going through such cycles, because each mediation step will probably imply some loss of quality. Therefore, when building a view from the graph, possible cycles can be cut in order to obtain an acyclic graph.

### 2.3.1 Design of the distributed discovery metamodel graph

In the previous paragraph we introduced the basic concepts and the requirements of discovery in a distributed environment. We can now provide a complete representation of all the entities involved in the process, starting from the Glue metamodel (figure 7). This metamodel can be represented as an RDF graph, as shown in figure 8, using the following triples:

- \(<\text{GC}\_\text{URI} \text{ rdf\_type glue\_GoalClass}>\>
- \(<\text{WG}\_\text{URI} \text{ rdf\_type wsml\_WGMediator}>\>
- \(<\text{WG}\_\text{URI} \text{ glue\_wg\_source \text{GC}\_\text{URI}}>\>
- \(<\text{WG}\_\text{URI} \text{ glue\_wg\_target \text{WSC}\_\text{URI}}>\>

\(^1\)The starting point is a goal class, and thus does not correspond to a service or a peer in NeP4B network.

\(^2\)This does not mean, however, that nodes in this graph represents peers (a peer can offer many services), and this graph does not represent NeP4B P2P network.
Given our goal of navigating the graph to build a series of paths from the starting goal to the reachable discovery engines, in figure 8 we can notice a first limitation of the basic metamodel, as it is not possible to navigate the graph following the arcs from the Goal to the Web Service because the Mediators have only outgoing arcs. To overcome this limitation and thus allow navigability, we must add two triples:
A second issue that can be noticed is that the metamodel doesn’t take into account the existence of the discovery engine(s). Glue metamodel is derived from the WSMO specifications, but while the services described using the WSMO metamodel are meant to be distributed on the network, the descriptions themselves are intended to be all known by a single WSMO execution engine. The main difference w.r.t. our scenario is that our major requirement is not to have a central entity that is responsible of collecting the service descriptions, so we must enrich the metamodel with entities that comes from an “infrastructural” layer, such as the Discovery Engines. A Discovery Engine is able to accept instances of one or more Goal Classes, and the triples that represent both the existence of such an entity and its relationship with the Goal Classes are:

- `<DE_URI rdf:type glue:DiscoveryEngine>`
- `<DE_URI glue:hasGoalClass GC_URI>`
- `<GC_URI glue:acceptedBy DE_URI>`

As all the identifiers of the entities are URIs, we have to add a triple to bind the URI of a given entity with the URL where its content can be found:

- `<Entity_URI glue:hasDescription Entity_URL>`

The complete metamodel resulting from the analysis of service discovery in a distributed scenario is shown in figure 9. We can note that the presence of both the Discovery Engine and the entities that can be reached from the GoalClass is partly redundant, because once the DE is introduced it masks these other entities. Even if redundant, this metamodel allows the existence both of Discovery Engines that performs crawling in order to find the services on the network (as the whole required information is represented) and of Discovery Engines where the inclusion of a service inside the service pool is ruled by an explicit agreement between the service owner and the DE owner.

### 2.3.2 Graph storage and retrieval using a DHT

A client, in order to discover which discovery engines can be reached and how (i.e. through which ggMediators), needs to obtain the graph introduced in the previous paragraph (or, more precisely, the view of the graph w.r.t. to its goal class). However, as discussed in section 2.1, in our decentralised scenario we cannot rely on any centralised system to store this information. Moreover, the graph cannot be associated with a particular peer of the network, because it represents information related to many peers. A distributed, peer-to-peer, infrastructure is thus a natural choice for the storage and retrieval of the information represented in the graph.

The graph information defined by the complete metamodel described in the previous paragraph can be stored in a DHT (Distributed HashTable) system using the following keys:

- `Entity_URI rdf:type` → `{ glue:GoalClass | wsml:WGMediator | wsml:GGMediator | glue:WebServiceClass | glue:DiscoveryEngine | WSC_URI }`: these entries are published by the entity owners;
Figure 9: Graph representation of the complete NeP4B discovery scenario in RDF format

- **Entity_URI glue:hasDescription** → **Entity_URI**: each entity owner (mediation service provider, discovery engine provider, Goal class provider) is responsible of defining where the description of the entity itself can be retrieved;

- **glue:wg_source GC_URI** → **WG_URI**: the WG Mediator provider is responsible of publishing information regarding which Goal classes are connected with the mediator;

- **glue:wg_target WSC_URI** → **WG_URI**: same as above;

- **WG_URI glue:hasWSCOutput** → **WSC_URI**: same as above;

- **glue:gg_source GC_URI** → **Med_URI**: the GG Mediator provider is responsible of publishing information regarding which Goal classes are connected with the mediator;

- **glue:gg_target GC_URI** → **Med_URI**: same as above;

- **Med_URI glue:hasGoalOutput** → **GC_URI**: same as above;

- **GC_URI glue:acceptedBy_GGMediator** → **Med_URI**

- **WSC_URI glue:acceptedBy_WGMediator** → **WG_URI**

- **glue:hasGoalClass GC_URI** → **DE_URI**: the discovery engine provider is responsible of publishing information regarding which Goal classes are handled by the engine;

- **glue:acceptedBy DE_URI** → **GC_URI**: same as above.
As introduced in paragraph 2.3.1, the entities represented in the complete metamodel are partly redundant. In order to choose the representation of the information required by the discovery process that guarantees a lower number of queries to obtain a path between a Goal Class and the target entity (Web Service of Discovery Engine), we can evaluate two cases for each of the two alternatives: and optimal case, with a direct path from a Goal Class to the target entity, and a fairly normal case, with a chain of GG Mediators connecting the source Goal Class with the target Web Service Class.

**Complete metamodel** The complete metamodel comprises all the entities defined in 2.3.1 that are also represented in the DHT. The test queries reported below try to obtain a path between a Goal class and the Web Services that are reachable according to the published information (as shown in figure 10):

- **optimal case:** direct GC → WS path
  - `<glue:wg_source GC_URI> → WG_URI`
  - `<WG_URI glue:hasWSCOutput> → WSC_URI`
  - `<rdf:type WSC_URI> → WS_URI`

- **normal case:** at least one GG Mediator in the GC → WS path
  - `<glue:wg_source GC1_URI> → null`
  - `<glue:gg_source GC1_URI> → Med_URI`
  - `<Med_URI glue:hasGoalOutput> → GC2_URI`
  - `<glue:wg_source GC2_URI> → WG_URI`
  - `<WG_URI glue:hasWSCOutput> → WSC_URI`
  - `<rdf:type WSC_URI> → WS_URI`

**Reduced metamodel** The reduced metamodel is obtained by deleting everything but Discovery Engines, Mediators, Goal Class and their relationships. The obtained metamodel contains far less entities w.r.t. the previous case, and this causes a lesser number of queries required to obtain a path from the source Goal class to the target Discovery Engine (as also shown in figure 11):

- **optimal case:** direct GC → DE path
  - `<glue:hasGoalClass GC_URI> → DE_URI`

- **normal case:** at least one GG Mediator in the GC → DE path
  - `<glue:hasGoalClass GC_URI> → null`
  - `<glue:gg_source GC_URI> → Med_URI`
  - `<Med_URI glue:hasGoalOutput> → GC2_URI`
  - `<glue:hasGoalClass GC2_URI> → DE_URI>`
As can be seen from comparing the number of queries required by the two versions of the metamodel to obtain the needed information, and considering the basic requirement of minimizing the number of queries to the DHT, we can conclude that the reduced metamodel can be used as a basis for implementing the distributed discovery in the NeP4B scenario. We can further optimize the number of interactions that allow the user to obtain a series of paths by allowing a GG Mediator to publish information regarding an entire path rather than publishing the existence of a single node in the graph by using
\[ Med\_URI\ glue:leadsTo \rightarrow Mediator1\_URI, Mediator2\_URI, Mediator3\_URI, DE\_URI. \]

When published, this information allows a user to directly obtain a path from the mediator to the discovery engine. As the mediator provider is responsible for the information, the
user can then decide whether to trust to information and then stop continuing the search or to continue exploring the graph to obtain other mediators to reach one or more discovery engines.
3 Preliminary design of the reputation architecture

In this chapter we sketch a preliminary design of the reputation system, which will be finalised in the second version of this deliverable (D4.1.2v2).

3.1 Entities, subjects and objects

In NeP4B scenario we can identify the following entities that are relevant to the reputation system:

- **economic actors** are the entities that federate themselves in semantic peers, cooperating among them in order to provide and use services and data, exploiting mediation and search functionalities provided by NeP4B infrastructure;

- **semantic peers** are the entities that aggregate the economic actors and the data sources and services they provide, making those available on the NeP4B network;

- **resources** are the “objects” (in a broad sense) exposed by the semantic peers; in NeP4B resources can be services or data objects.

Most P2P reputation system take into account peers only, thus having in principle the three entities listed above requires some discussion.

We decide not to take into consideration the economic actors, which therefore will be neither subjects nor objects of the reputation system. There are two main reasons for this:

- Resources are always exposed by semantic peers, not by economic actors (which are not “visible” on the P2P network). Moreover, a single resource can be built by a semantic peer as an aggregation of resources provided by different economic actors. Therefore, semantic peers, and not economic actors, appear as resource providers in NeP4B network and can thus be objects of the reputation system.

- A certain degree of anonymity is desired by the economic actors, which may want not to disclose even the existence of some relationships among them. Thus, economic actors will not appear as subjects of the reputation system (i.e. entities that express opinions/feedbacks about objects), but they will be “masked” by the respective semantic peers.

For the above reasons, only semantic peers will be the subjects of the reputation system, while both semantic peers and resources can be the objects (we will discuss this in a later section). It is worth noting that if an economic actor does not want aggregation with other actors and does not want to be “masked” behind a peer together with other actors, it can participate to NeP4B network acting as a peer of its own.

3.2 Feedbacks vs. opinions

Feedbacks and opinions are two different ways to deal with subjects’ judgements about objects. A feedback is a judgement expressed as a result of a single interaction (often called transaction in this context), and concerning the outcome of that single interaction. A subject can thus give a feedback about an object only after an interaction with that object; moreover,
the same subject can give many different feedbacks about the same object if it interacts with it multiple times. EBay provides a typical example of a feedback-based system. An opinion, on the other hand, expresses a generic judgement of a subject about an object, not related to a particular interaction and usually with no particular restriction (i.e., any subject may be able to express an opinion about any object, not depending on actual interactions). Opinion-based systems often restrict each subject to have at most one opinion in the system about each object: the opinion may change over time, but when it changes it supersedes the previous one (which does not happen with feedbacks).

In NeP4B, we decide to use a feedback-based system. This is simpler, and more adequate to the actors and the kind of interactions occurring in NeP4B scenario (which can be economic transactions). Moreover, feedbacks are compatible with the idea of using semantic peers as subjects: if two economic actors belonging to the same peer interact with the same resource, the peer will be able to give two feedbacks about that resource, each one expressing the judgement by one of the actors. This would not be possible in an opinion-based system with the usual restriction of one opinion per subject: in this case the semantic peer (i.e. the subject) would have to express just one opinion, somehow “mediating” between the (possibly) different opinions of different economic actors.

Another advantage of using feedbacks is given by a possible “internal” use inside the semantic peer. As we said in the previous section, resources and peers, but not economic actors, can be the objects of the reputation system, so feedbacks will be related to peers and/or resources rather than to economic actors. Inside the semantic peer it may however be possible to map each interaction with a single economic actor that provided the resource for that interaction, and thus directly and individually related to that feedback. This information can be used internally in the semantic peer, e.g. in order to know if a particular actor providing resources through the peer is misbehaving or giving a bad service.

In a feedback-based system a subject is allowed to give a feedback about an object only if it actually interacted with that object. This is easy to do if, like in EBay, all interactions can be monitored through a central system, which also manages the reputation system. But there is no such central authority in NeP4B network, so we must relax this requirement. Therefore, we will assume that an interaction has happened if both involved parties (plus, possibly, a witness) confirm that it has happened. This is enough to prevent a malicious peer from sending a lot of negative feedbacks about a resource/peer in order to damage its reputation\(^3\), but does not resolve the problem of colluding entities that have a mutual interest\(^4\).

### 3.3 Resources vs. peers as feedback objects

Both semantic peers and resources can be reputation objects. Since we choose a feedback-based approach, however, it is more appropriate to associate the feedback to a resource rather than to the semantic peer that offers that resource. Indeed, the client interacts primarily with a resource, and its judgement about that single interaction (i.e. the feedback) is more appropriately applied to that single resource rather than to the semantic peer which provides the resource. Therefore, when a semantic peer (on behalf of an economic actor) interacts with a resource, it is able to give a feedback that has the resource as object, and not the

---

\(^3\)The peer that provides the resource (i.e. the victim) will be able to detect such behaviour after a couple of interactions, and thus will probably refuse subsequent interactions with the malicious peer.

\(^4\)This problems exists to a certain extent also in centralised systems, anyway.
3.4 Reputation values

Using resource-related feedbacks, it is possible to build resource-related reputation values, which can then be used by clients when ranking resources. We choose a one-to-one relationship between resources and resource reputation values, i.e. each resource can have just one reputation value. It would in principle be possible to have different reputation values for different “facets” of a single resource; this would however require defining such facets, which is not simple in a general-purpose system like NeP4B network and requires some common understanding among the different involved parties. Moreover, it would make comparison between different resources more difficult, because different resources could have reputation values defined according to different facets. If a resource can however be clearly divided into “sub-resources” and it is appropriate to have a reputation value for each of them, these can simply be exposed by the semantic peer as individual resources.

Besides resource reputation values, it is useful to have also peer reputation values, representing general trustworthiness of that peer as a resource provider. For example, a new resource (thus without an established reputation) offered by a reputable peer can be considered positively on the basis of the peer reputation, thus allowing a quicker acceptance for “good” resources offered by already known “good” peers. Since feedbacks are related to resources only, peer reputation must be built by aggregating feedbacks/reputation values for all the resources offered by a peer.

Whereas the resource is always “passive”, the peer can be either “passive” (when it provides a resource) or “active” (when it interacts with a resource offered by another peer, acting as a client on behalf of an economic actor). It is useful to distinguish between the two possible roles a peer can play in dealing with reputation values; we therefore choose to have two different reputation values for the peer: one when acting as a resource provider (i.e. the “passive” case) and one when acting as a client (i.e. the “active” case). The former will be derived from the feedbacks/reputation values of the resources offered by the peer, because there is no feedback related to the peer as a resource provider, as we discussed above. The latter will be derived from feedbacks related to the peer when acting as a client, which can be given by the resource providers after any interaction.

Besides the client and resource provider roles, semantic peers act in the system also playing a third role: the “recommender”, i.e. feedback author (subject). In a further improvement, this role could be taken into account too, and opinions about peers with respect to this role could be given, building an aggregate value that would assume the meaning of peer credibility when giving feedbacks.

3.5 Information storage and retrieval

Which data are stored, where they are stored, and how they can be retrieved are crucial issues for a P2P reputation system.

Dealing with the first question (which data are stored), in a feedback-based system we can identify the following alternatives:

---

5This mechanism may have the drawback that it allows a malicious peer to build a good reputation on some resources and then to “spend” this reputation starting to offer a bad resource. This is anyway a general problem, because the same kind of behaviour can be applied to a single resource (e.g. a service).
• The systems only stores the feedbacks: each peer interested in the reputation of a peer/resource must retrieve the relevant feedbacks and compute the reputation value by itself. This has the advantages that each peer can use a custom algorithm to compute the reputation (thus taking into account subjective preferences). The drawback is that the process of collecting the feedbacks and computing reputation can be quite expensive, and that the amount of data to be stored in the system can become very large (all the feedbacks).  

• The system only provides a reputation value: feedbacks are used by the system to compute a reputation value, but are not stored. This has the advantages of reducing the amount of stored information and giving a reputation value with no additional processing required on the requester's side. Computing the reputation value becomes however a crucial task, and it is very difficult to guarantee that this is done properly (i.e. rightly and not maliciously) without a trusted authority. Moreover, since the “raw” data (the feedbacks) is not available, peers cannot apply custom algorithms to compute reputation.

• The system provides both the feedbacks and a reputation value: this is a mixed approach that somehow combines the advantages and disadvantages of the other two. Following this approach, the client can first quickly get a reputation value and use it for a coarse-grained filtering, and then, if desired, use the raw feedbacks for more fine-grained evaluation. The trust issue about who computes the system-provided reputation value still exists, but the availability of the raw data starting from which it is computed clearly mitigates the problem, because anyone can check the correctness of the value.

Dealing with the second question (where the data are stored), we can identify the following alternatives:

• feedbacks about an object are stored by the respective authors (i.e. subject peers): this has the advantage that feedback integrity is not an issue (feedbacks are stored by respective authors), but has the drawback that collecting feedbacks about an object requires contacting many different peers, and is hardly compatible with the possibility of having a reputation value computed by the system;

• feedbacks about an object are stored by the object peer: this has the advantage that feedbacks and reputation information are easy to find, but making each peer responsible for its own reputation makes it more difficult to guarantee the integrity of such information (a peer is probably interested in dropping negative feedbacks and, in general, in positively altering its reputation);

• feedbacks about an object are stored by a third part (i.e. neither the subject nor the object peer): this allows storing all information about an object in the same place, without being that place controlled by someone obviously interested in tampering with that information.

---

6 For this reason, old feedbacks are usually discarded after a certain amount of time.
7 Of course, integrity of feedbacks must be guaranteed.
We discard the first option because it is not compatible with our requirements and other design decisions. The third option is quite common in P2P reputation system, but it has a significant drawback. The third-party peer that is responsible for the reputation of an object has nothing to do with that object, thus it is not directly interested in maintaining that value. Moreover, it can leave the network at any time, or behave selfishly or maliciously. The usual way to cope with this sort of problems is resorting to a significant amount of replication, which however introduces significant additional complexity. For these reasons, we pursue the second option, and store the feedback information in the object peer. The object peer is of course directly interested in keeping information about its reputation if it has a good one, so this resolves the problem of selfish peers not willing to store much data they are not interested in. If the reputation is not very good, we just need to prevent the object peer to tamper with this information, whereas the possibility of totally dropping own reputation information is less problematic. Moreover, reputation information follows the same history as the resource/peer it refers to: if the peer goes offline its reputation information become unavailable, but this is not a big issue because it would anyway be impossible to interact with that peer.

R-Chain [12] is a reputation management system for structured P2P overlays in which reputation information is stored in the object peer using a chain of “transaction records”. We plan to build NeP4B P2P reputation system starting from R-Chain, and exploring the possibility to enhance it with a system-computed reputation value. A possibility could be the inclusion of an updated reputation value in every record, computed by both involved parties (and possibly by the involved witness peer).

### 3.6 Summary of design principles

In this section we summarise the design principles of the reputation system we discussed so far:

- the system is based on feedbacks;
- semantic peers appear as authors of the feedbacks (subjects);
- resources offered by semantic peers are the object of the feedbacks;
- a resource reputation value can be computed starting from feedbacks given for that resource;
- a peer reputation value as a resource provider can be derived from the reputation of the resources offered by that peer;
- a peer reputation value as a client can be computed starting from feedbacks given to the peer when acting as a client;
- each resource can have only one reputation value;
- feedbacks and reputation information is stored by the object peer.

---

8 This would be the same as acquiring a new identity (white-washing).
9 R-Chain only provides the list of feedbacks.
References


