ENHANCING CORBA'S NAMING SERVICE WITHOUT MODIFYING ITS IDL-INTERFACE

MARKUS ALEKSY, AXEL KORTHAUS, MARTIN SCHADER

Department of Information Systems III
University of Mannheim, Germany
Schloß (L5,6)
D-68131 Mannheim, Germany
{aleksy|korthaus|mscha}@wifo3.uni-mannheim.de

ABSTRACT

In this paper, we would like to present a smooth way of enhancing the functionality of the CORBA Naming Service. The key element of our design is to leave the IDL interface of the Naming Service unchanged. The enhancements introduced in this paper are based on the exploitation of several underspecified parts of the CORBA standard, in a way that conformity with the standard and the corresponding benefits of reusability (of existing clients and servers) are guaranteed.

KEY WORDS: CORBA, Naming Service, Service Discovery

1 INTRODUCTION

Since version 2.0 of the CORBA specification, CORBA-based applications have gained increasing popularity. One reason for this success is that CORBA enables different ORB products to be used together and integrated. Furthermore, it allows the collaboration of CORBA-based systems with other systems based on different kinds of architectures. Thus, the main foundation of CORBA's popularity is the specification of an interoperable system architecture which regulates the exchange of data between software implementations that can be built on top of middleware products from different vendors.

Besides its core architecture [1], the CORBA standard specifies a number of different services which augment the basic functionality provided by the CORBA core. Among the services specified by the standard are typical functionalities often needed by software developers, concerning persistency, transactions, security, and other aspects. Due to the standardization of a set of interfaces to these services, application portability was improved significantly.

Being capable of localizing distributed objects, the Naming Service [2] may well be the most fundamental CORBA service. It was one of the first services to be specified by the Object Management Group (OMG), and has been implemented by most of the current ORB product vendors in the meantime. The purpose of the Naming Service can be compared to the purpose of a telephone directory. With its help, a freely selectable name can be attached to an object (i.e., an object reference). This mapping of names to object references (or contexts, respectively,) is called "binding". A context object contains a set of bindings. Any name is unique within its context and can be resolved into exactly one subcontext or object reference.

The Naming Service consists of two basic interfaces, called `NameContext` and `BindingIterator`. `NameContext` objects manage "named" object references. The interface provides a number of operations for registration (`bind`, `rebind`) and unregistration (`unbind`) of CORBA objects, and for determination of the reference to a certain CORBA object (`resolve`). Furthermore, the `NameContext` interface provides the possibility to create (`bind_context`, `rebind_context`, `bind_new_context`, `new_context`), unregister (`unbind`), determine (`resolve`), and delete (`destroy`) additional `NameContext` objects within the parental `NameContext`. `BindingIterator` is used to iterate over the different entries of a certain name context object. To this end, two operations `next_one` and `next_n` are provided. In order to get a reference to the Naming Service, an application can call the ORB operation `resolve_initial_references("NameService")`. The result of this call is of type `CosNaming::NamingContext` and represents the initial context (root context) of the Naming Service.

By specifying an Interoperable Naming Service, which has replaced the original Naming Service in the meantime, the OMG reacted to the frequently criticized mechanisms for resolving the initial reference (cf. [3], [4]). The architecture of the new Naming Service is largely equivalent to the original Naming Service, but it includes several enhancements. Among these are:

- the introduction of new addressing schemes for CORBA object references (e.g., `corbaloc` and `corbaname`),
- the extension of the service with a new interface (`NamingContextExt`), which contains opera-

tions for converting the new addressing schemes into object references, and

- the standardization of the command line options (`-ORBinitRef` and `-ORBDefaultInitRef`), by which the new addressing schemes can be applied in a vendor-independent way.

In the following sections we are going to describe our own enhancements to the Naming Service and deal with other alternatives as well as with the advantages and disadvantages of the different approaches.

2 MOTIVATION FOR ENHANCING THE CORBA NAMING SERVICE

In open, heterogeneous, distributed systems, locating a service provider by name is not always a sufficient solution, because the service client has to know the exact name of its communication partner. This criticism is also true of the CORBA Naming Service, since its querying features are very limited. When looking up a server's reference by name, the developer can neither use any "wildcards" such as "*" or "?", nor are there any operations implementing a name-based search restricted by specific search criteria such as independence of upper case or lower case letters. Although clients often know the type of the servers they need to connect to, they might not know the exact names the servers used for registration with the Naming Service.

The only way to solve this problem is to transfer and subsequently traverse all the existing `NameContext` objects and the names registered with them by using the `BindingIterator` interface. Since this solution usually entails a high network load, a relatively high expenditure of time, and a complex programming of the client, different alternatives should be considered.

3 ALTERNATIVES BASED ON CORBA'S TRADING SERVICE

Under certain circumstances, a query based on service attributes as specified by the CORBA Trading Service [5], followed by a selection of the preferred result, can be a better solution. The purpose of trading is to match a service demand with an appropriate service offer. The clients of the Trading Service can be subdivided into two categories: service exporters and service importers. These categorization is not strict, i.e., an exporter can as well take over the role of an importer and vice versa.

A service provider must export its functionality by publishing its access information, its service type, and several attributes. An attribute represents a property of a service and consists of an attribute name and an attribute type. Attributes can be static or dynamic. While static attributes serve to represent the features of a service, dynamic attributes reflect the current state of the service. After registration with the Trading Service, an exporter is ready to process incoming requests by potential service clients.

By stating the required service type and the relevant service attribute values, a service client has to provide the access information needed by the Trading Service to find an appropriate offer. The interceding trader may return no, one, or more service offers. The trader checks each registered service provider for conformity with the required service type and supplies the importer with all bindings that match and therefore come into question. The trader can apply different policies for the query which can be set by the importer. Thus, e.g., the total number of offers to be returned or their order can be influenced.

The number of matching service offers may possibly be very large. By providing means to specify policies, constraints, and preferences, the Trading Service enables the user to limit the query result set appropriately.

However, using the Trading Service has several disadvantages:

- client programs soon become very complex, because the developer has to consider many details and
- most of the existing applications are based on the Naming Service. In case these legacy applications are to be modified to be able to collaborate with the Trading Service, an increased porting effort has to be put up with.

These criticisms show that transforming a Naming Service-based application into a Trading Service-based application is always bound up with significant changes to the application. For this reason, we preferred enhancing the Naming Service to using the differently targeted and more complex Trading Service. This was done such that

- the new Naming Service provides higher flexibility,
- existing legacy applications can be left unchanged, and
- migration of legacy applications to the new environment can be achieved with minimum program modifications.

4 OPEN ISSUES IN THE NAMING SERVICE SPECIFICATION

The CORBA services specifications leave several aspects underspecified and open to interpretation. An example of how such open points in the CORBA standard can be exploited is given by the OmniORB Naming Service. The standard gives no hint whether the Naming Service should store object references persistently or transiently. While the Naming Services of JavaIDL and OmniBroker store object references transiently, the OmniORB Naming Service stores them persistently, so that they are still available after a crash or shutdown and subsequent restart of the service. This is not true of the other two Naming Services, so that the bindings would

have to be renewed after a crash or shutdown in their case. However, this difference affects the way applications that use these Naming Services have to be designed, because the renewal of a binding to the Naming Service provides different results: OmniORB reports an „`AlreadyBound`" exception, whereas the other products perform the binding operation without any complaints.

In our approach, we make use of the "freedom" granted by the specification, too. All the enhancements introduced by our design were realized on the foundation of such open points in the standard.

5 DESIGN GOALS

As already mentioned, we did not aim at a complete redesign or comprehensive extension of the Naming Service, but tried to provide the existing Naming Service specification with additional flexibility. Therefore, we put emphasis on the following aspects:

- standard compliance, i.e., the increased flexibility had to be realized with no need to modify or extend the IDL interface of the Naming Service. The main reason for this design goal was to be able to keep standard compliance and portability of existing applications, so that they can be continued to operate;
- realization of the functional enhancements on the basis of "open issues" in the specification of the standard in a way that they can be used comfortably by newly developed clients;
- no changes should be necessary to existing servers, independently of the question whether they are based on the Naming Service or on the Trading Service.

6 THE ENHANCED NAMING SERVICE

The existing functionality of the Naming Service can be enhanced in different ways without modifying the corresponding IDL interface.

On registration, a server can pass its name (element `id`) and its kind (element `kind`). Since the latter information has rarely been used by existing applications, it seems to be very suitable for a semantic enhancement of the Naming Service. For example, the following values of the `kind` element could lead to a special treatment by the Naming Service:

- `type:` the name element (`id`) contains the required type of the target server,
- `wildcards:` the `id` element may contain wildcards such as "`*`" and "`?`",
- `like:` the value of the `id` element has to be "similar" to the registered name,
- `ignoreCase:` if this keyword is specified in the `kind` attribute, the name in the `id` element has to be matched with a registered name without distinguishing between upper and lower case letters,
- `startsWith:` the registered name has to start with the string provided in the `id` field,
- `endsWith:` the registered name has to end with the string provided in the `id` field.

Additionally, application-specific declarations are possible. Choosing the appropriate kind of enhancement, the user should always keep in mind the resulting effects to the Naming Service's performance. While policies such as `wildcards` or `ignoreCase` only bring about a minor overhead, a type-based search is relatively expensive, because it requires the invocation of the remote operation `is_a` for each reference.

Furthermore, it should be possible to restrict the search space. Two basic policies could be distinguished:

- `complete:` the complete name graph is searched which can be very time consuming and
- `partial:` only the current branch of the name graph is searched. This both limits the number of entries potentially found and increases the search speed.

The enhanced semantics can be based on different approaches to increase the flexibility of the Naming Service. The enhancements we propose can, for example, be implemented without using any further components or services. Another possibility is to have the search requests that are directed to the Naming Service both processed locally and forwarded to one or more Trading Services. Therefore, a suitable mapping from the values provided in the `kind` field to a corresponding request to the Trading Service is needed.

7 RUNTIME BEHAVIOR IF NO ADDITIONAL SERVICES ARE USED

In this section we explain our basic design which is not based on using the Trading Service. The following code snippet is to illustrate the approach:

```
// nc is a NamingContext object
// ...

NameComponent[] aName =
  new NameComponent[1];
aName[0] = new NameComponent();
aName[0].id = "RoomBooking";
aName[0].kind =
  "policy:complete, " +
  "ignoreCase";
org.omg.CORBA.Object obj =
  nc.resolve(aName);
NamingContext contxt =
  NamingContextHelper.narrow(obj);
```

The invocation of operation `resolve()` causes the enhanced Naming Service to check whether the name sought after already exists. If this is the case, the corresponding reference can be returned immediately. If the name does not exist, the service checks, whether the `kind` component indicates some special semantics. If not, a `NotFound` exception is thrown, as demanded by the standard. In the example above, the `kind` component specifies special semantics, so that the complete (`policy:complete`) name graph has to be searched for the name "`RoomBooking`", ignoring case differences (`ignoreCase`). As soon as an entry is found, a new `NamingContext` object is created with the specific name (here: "`RoomBooking`"), but it is not yet inserted into the name graph. During the next step, all entries found are registered with the new `NamingContext` object "`RoomBooking`", and, subsequently, the latter is inserted into the name graph. During the last step, its reference is returned to the client. Should the search fail, a `NotFound` exception is thrown at that point.

To avoid recurrent result determination processes in case of frequent, similar queries, search results should at least be stored transiently. Caching can be done according to different policies in order to increase the query efficiency or to decrease the load of the computer on which the Naming Service resides. Among others, the following techniques can be used to prevent an exceeding number of results to be temporarily stored:

- Life-Time-Policy
  query results are stored not exceeding a certain (maximum) amount of time,
- Maximum-Size-Policy
  only a certain (maximum) number of query results is buffered. If this number had to be exceeded, the oldest query result would be replaced by the current one.

Using one of these two techniques or a combination of both increases the efficiency of the modified Naming Service.

8 RELATED WORK

In [6], another approach is presented which is very interesting from the client's point of view. The authors extend the Naming Service with load balancing capabilities. After the invocation of operation `resolve()`, the IOR of the currently least loaded server is selected from the set of servers registered with the specified name and returned to the client. Prerequisite of this approach is the possibility for different servers to register with the Naming Service using the same name. However, according to the Naming Service specification this is not allowed and has to cause an `AlreadyBound` exception.

9 RUNTIME BEHAVIOR IF THE TRADING SERVICE IS USED

The enhancements to the Naming Service described so far significantly extend the functionality of the service without the need to fall back on additional services. For some applications, this extended functionality might still be insufficient or the load of the service component might be too high, because it has to perform all the query operations itself. As mentioned before, a more comprehensive solution could be to delegate an "enhanced" client request to one or more Trading Services. Although this means that another CORBA service becomes involved, flexibility and speed of search can be further improved, especially if trader federations are used.

9.1 MAPPING OF NAMING SERVICE'S `KIND`–ELEMENT TO THE TRADING SERVICE

In order to make an enhanced query involving the Trading Service possible, a mapping of the `kind` element's content to the parameters of a query understood by the Trading Service has to be provided. Since the Trading Service supports much more complex queries than the Naming Service (see below a partial view of the Trading Service's IDL interface), we now have to consider possible ways to achieve that goal.

```
void query (
  in ServiceTypeName type,
  in Constraint constr,
  in Preference pref,
  in PolicySeq policies,
  in SpecifiedProps desired_props,
  in unsigned long how_many,
  out OfferSeq offers,
  out OfferIterator offer_itr,
  out PolicyNameSeq limits_applied
) raises (
  IllegalServiceType,
  UnknownServiceType,
  IllegalConstraint,
  IllegalPreference,
  IllegalPolicyName,
  PolicyTypeMismatch,
  InvalidPolicyValue,
  IllegalPropertyName,
  DuplicatePropertyName,
  DuplicatePolicyName
);
```

In a query request directed to the Trading Service, several parameters can be specified, e.g.:

- `ServiceTypeName`: denotes the type of the target server (similar to the Interface Repository structure `IR::Identifier`),

- `Constraint`: denotes a query constraint expression. The constraint is used to filter offers during a query, and must evaluate to a boolean expression. The set of query results can be restricted using operators such as `==`, `!=` , `>`, `>=`, `<`, `<=`, `~`, `in`, `and`, `or`, `not` etc.,
- `Preference`: denotes a query preference expression. The preference is used to order the offers found by a query. Possible values of `Preference` are: `min`, `max`, `with`, `first`, and `random`.
- `PolicySeq`: denotes a sequence of names of policies together with the corresponding policy values used to control the trader's behavior.

In order to provide the Trading Service with the parameter data needed for query processing, the `kind` element of the Naming Service request has to carry the relevant information. A mapping could be based on the use of name-value pairs inserted into the `kind` element. Each name should indicate the corresponding Trading Service `query` parameter, and the value has to be of a specific type in order to match the `query` operation's requirements. The following table depicts a possible choice of names and value types suitable for realizing the required mapping:

Keyword in `kind` element	**Type of allowed values**
`ServiceTypeName`	`string`
`Constraint`	`string`
`Preference`	`string`
`PolicyName`	`string`
`PolicyValue`	`any`
`SpecifiedProps`	{"none"|"some"|"all"}

Table 1: Keywords and value types for the `kind` element

Figure 1 illustrates the dynamic, sequential flow of communication messages.

A query to the Naming Service, which is to be delegated to the Trading Service, might look like this (cf. Figure 1):

```
NameComponent[] aName =
  new NameComponent[1];
aName[0] = new NameComponent();
aName[0].id = "UseTrader";
aName[0].kind =
  "ServiceTypeName=Counter," +
  "Preference=first," +
  "PolicyName=exact_type_match," +
  "PolicyValue=true," +
  "SpecifiedProps=all";
org.omg.CORBA.Object aObj =
  nc.resolve(aName);
```

In the example above, setting the `id` value to "`UseTrader`" has the effect, that the Trading Service is involved in processing the query. Thus, the value of the `kind` component gets a special meaning and has to be evaluated. Should one of the values provided be invalid, the Trading Service can report the precise cause of the error, e.g. by throwing exceptions of types `IllegalServiceType`, `IllegalConstraint`, `IllegalPreference`, or `IllegalPolicyName`. Since a Naming Service-based client does not expect any of these exceptions and therefore cannot handle them, they have to be mapped to valid Naming Service exceptions. To achieve this, several Naming Service exceptions come into question:

- `NotFound`,
- `InvalidName` or
- `CannotProceed`.

With respect to the semantics, the `InvalidName` exception appears to be the most suitable one. It does not allow the specification of additional details, though, so it is not really appropriate to support the identification and resolution of errors. The `NotFound` exception and the `CannotProceed` exception, on the other hand, give the possibility to specify more details with the help of an element of type `string`. Because of its semantic connotation, the `NotFound` exception seems to be unsuitable for representing one of the Trading Service exceptions mentioned above. The last alternative is the `CannotProceed` exception that might be used to carry

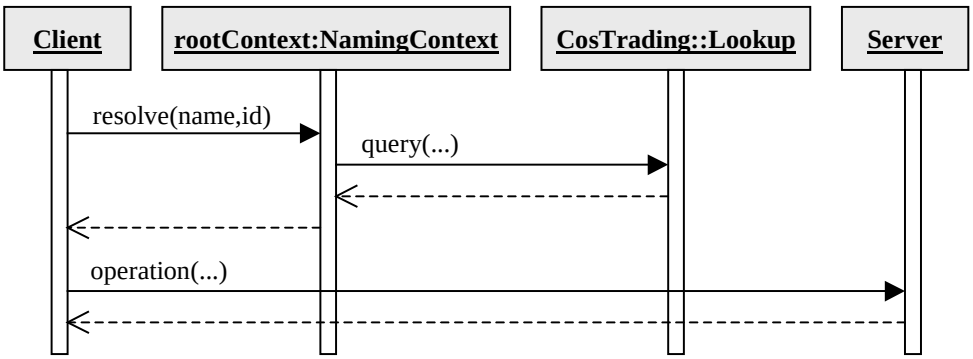

Figure 1: Example Scenario using Trading Service

detailed information about the kind of error in its `rest_of_name` element (of type `string`), e.g., `IllegalServiceType`, `IllegalConstraint`, `IllegalPreference`, or `IllegalPolicyName`.

9.2 RESTRICTIONS

A query to the Trading Service might produce more than one object reference as its result. A Naming Service-based query, on the other hand, always returns exactly one reference. For this reason, the enhanced Naming Service should be able to handle a set of references returned by a Trading Service and to select one reference out of this set to return to its client. For example, the Naming Service might perform a specific selection based on some semantic criteria or it might simply return the first reference. A simple solution to this problem could be to create a temporary naming context object, register the results of the trading service query with it and only return the reference to this naming context object to the client. In this case, the client has to iterate over the entries and to make a selection by itself. The client is also responsible for deleting the naming context object when it is no longer needed.

Another restriction is that the `kind` element is of type `string`, so that it cannot store arbitrary numbers of characters, i.e., the string length is restricted in actual language mappings. However, the complex type structures used in the Trading Service's `query` operation have to be mapped to one single flat string (the `kind` element) in our approach, so that it is not possible to make use of the full power of the Trading Service's query facilities. Queries transferred to the Trading Service are restricted by this size limitation, but from a practical point of view this restriction should be irrelevant in most cases.

10 CRITICAL REVIEW

By utilizing underspecified parts of the Naming Service specification and omitting any changes to its IDL interface, an enhanced but still standard-compliant solution was achieved. The different approaches discussed in the paper show several benefits and drawbacks. In the case of the Naming Service, not used together with other services, the advantages are:

- clients benefit from improved flexibility and comfort, and
- the service is not dependent on any other component (such as the Trading Service), so that in case of any errors no special fault-tolerance strategies have to be applied.

The main disadvantage concerns the Naming Service architecture itself, because it is designed as a centralized component. If there is a large number of client requests invoking the "enhanced" search functionality within a small time frame, the load becomes very high, because the service component has to process the queries and return the search results on its own.

If a collaboration between the Naming Service and the Trading Service is preferred, the following aspects can be considered advantageous:

- very high flexibility and comfort on the client side, concerning the search of suitable servers,
- accelerated search based on the use of trader federations.

Since this solution, as opposed to the former, requires the availability of the Trading Service, possible fault-tolerance strategies have to be considered in advance. Another source of potential errors is the mapping of the Naming Service's `kind` value to a Trading Service request. Since the maximum number of characters in the `kind` element is restricted, it is not possible to make use of the Trading Service's full functionality.

As shown by the result of our work, the CORBA Naming Service can easily be functionally enhanced without the need to modify the standardized IDL interface. Thus, it becomes possible to:

- reuse existing legacy applications (clients and servers),
- provide existing legacy applications with new functionality effortlessly, and
- improve the flexibility of the whole system.

Therefore, investments in existing clients and servers can be leveraged and additional functionality can be integrated in a simple way.

11 REFERENCES

[1] OMG, CORBA/IIOP 2.4 Specification, OMG Technical Document Number 00-10-01, 2000, ftp://ftp.omg.org/pub/docs/formal/00-10-01.pdf

[2] OMG, Naming Service Specification, OMG Technical Document Number 00-06-19, 2000, ftp://ftp.omg.org/pub/docs/formal/00-06-19.pdf

[3] M. Aleksy, M. Schader, C. Tapper, Interoperability and Interchangeability of Middleware Components in a Three-Tier CORBA-Environment – State of the Art, *Proc. Third Int. Enterprise Distributed Computing Conf. EDOC'99*, Mannheim, Germany, 1999, IEEE, 204-213

[4] M. Schader, M. Aleksy, C. Tapper, Interoperabilität verschiedener Object Request Broker nach CORBA2.0-Standard; *OBJEKTspektrum*, 3/98, 72-77, http://www.wifo.uni-mannheim.de/IIOP

[5] OMG, Trading Object Service Specification, OMG Technical Document Number 00-06-27, 2000, ftp://ftp.omg.org/pub/docs/formal/00-06-27.pdf

[6] T. Barth, G. Flender, B. Freisleben, F. Thilo, Load Distribution in a CORBA Environment, *Proc. of the Int. Symposium on Distributed Objects and Applications*, Edinburgh, Scotland, 1999, IEEE, 158-166