JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 1, No. 1, May-June 2002

What is Common Between Generic Services and Interface Navigation?

Rushikesh K. Joshi, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay

Abstract

Generic services and interface navigation in object oriented systems can be formulated in terms of specific typecasting sequences over narrowing and widening operations. With examples, the commonality and difference between the two are brought out with reference to typecasting. Consequences of specific typecasting sequences to their implementation in a distributed environment are highlighted.

1 A GENERIC SERVICE

A generic service is applicable to various contexts, since it is designed to cover a wide range of types. A naming service such as CORBA naming service [5] is an example of a generically applicable service in a distributed environment. The service essentially maintains mappings from object names to object handles, with object name as key. To make naming service a generically applicable component, it must not be made to handle *actual* object types. For example, with reference to the class hierarchy shown in Figure 1, a naming service that maps object names to handles of type LocatableObject is a generic component. Whereas, a naming service designed to map object names to objects of type ObjectType2 is not a generic component. The former is applicable to hold object name, object handle tuples for any type of object in the hierarchy, whereas, the latter can only handle objects of type ObjectType2.

Cite this article as follows: Rushikesh K. Joshi: *What is Common between Generic Services and Interface Navigation?*, in Journal of Object Technology, vol. 1, no. 1, May-June 2002, pages 125-132, http://www.jot.fm/issues/issue_2002_05/article4

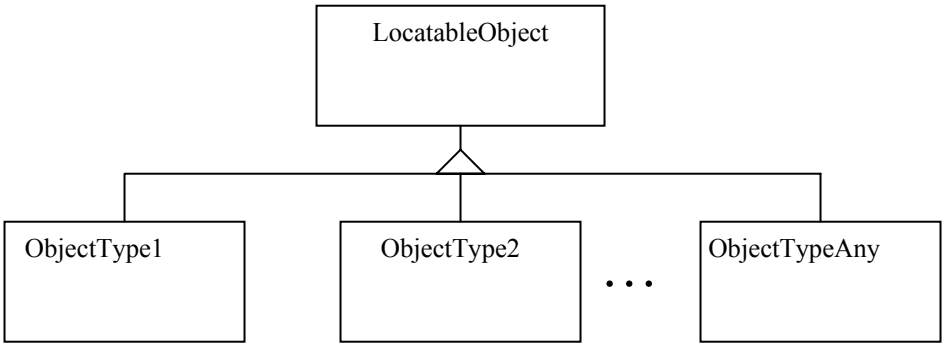

Figure 1. Hierarchy in a Generic Service

2 INTERFACE NAVIGATION

Interface navigation allows a client having a handle to an object through one of its multiple interfaces to navigate through its other interfaces. Consider the class hierarchy shown in Figure 2. By interface navigation, a client having a handle to one of all the interfaces implemented by a shared implementation may be allowed to obtain a handle to any of the N interfaces implemented by the shared implementation.

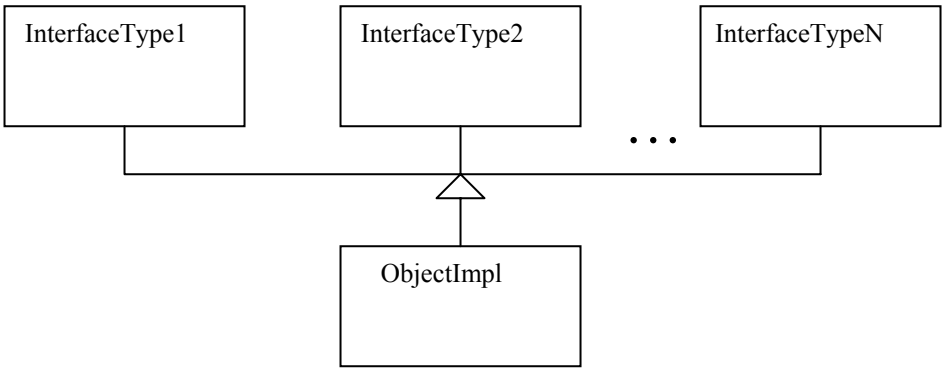

Figure 2. Hierarchy in Interface Navigation

3 TYPECASTING SEQUENCES

The above two applications are based on two different object gluing patterns. A generic service uses multiple sub-classing, whereas, interface navigation uses multiple super-classing, or multiple inheritance. Both use sequences of typecasting specific to them. In an inheritance hierarchy, typecasting is either a widening operation, or a narrowing operation. The widening and narrowing operation sequences are subjected to type safety rules followed by the programming environment used for implementation. For example, JAVA [1] throws an exception upon a type-mismatch, whereas, Smalltalk [2] deals with this problem during method binding time, while Meyer prescribes an *assignment attempt statement* [3], which results in null assignment upon violation of type conformance.

A generic service-based application uses two typecasting operations, widening followed by narrowing as shown in Figure 3. Operation 2 narrows the result of operation 1. It can be noted that operation 2* shown in the figure is an incorrect operation. An incorrect operation needs to be prohibited at programming level through an implementation of a type-safety mechanism.

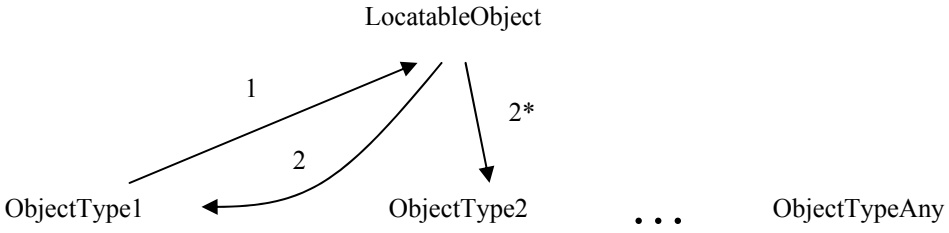

Figure 3. Typecasting Sequence in a Generic Service-based Application

An example of interface navigation is a sequence of three typecasting operations, widening, narrowing and widening, as shown in Figure 4. The third operation widens a result of narrowing operation performed on a result of widening operation. In the figure, 3, 3* and 3** are examples of permissible widening operations performed on result of operation 2.

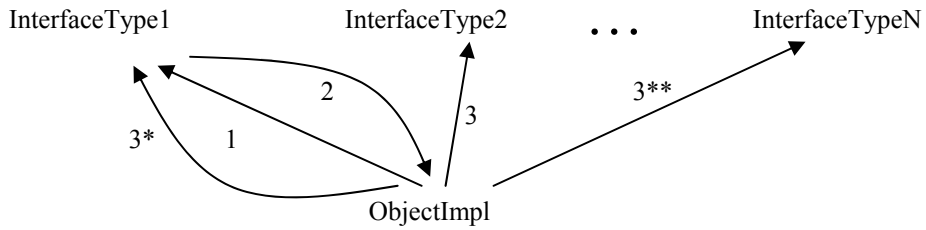

Figure 4. Typecasting Sequence in an Interface Navigation-based Application

4 REALIZING A GENERIC SERVICE

Table 1 identifies the responsibilities of narrowing and widening operations in a generic service-based application. The arrow ids are as shown in Figure 3.

Table 1: Typecasting Responsibilities in a Generic Service-based Application

Arrow Id	Operation	Who performs the operation
1	Widening	Client of Generic Service (source client, e.g. server hosting object)
2	Narrowing	Client of Generic Service (destination client, e.g. process which wants to use the object)

The first operation is a widening operation, which needs to be performed by the client of the generic service. The interface of the generic service expects from the client a handle of type LocatableObject. The client in this case is typically the server hosting the object, or the object itself.

In response to a quarry, the generic service returns to the caller, a handle of type LocatableObject that needs to be typecasted to a handle of the desired object type. This step also is performed at the caller, which is another client of the generic service. The second type of client is the user of the object that is registered with the generic service. Since clients of a generic service perform both operations, the latter does not need to know the actual types of objects registering with it.

In a distributed environment, implementation of arrow 2 at client side conflicts with hiding of implementation types. Addition of one more layer of interface type as shown in Figure 5 solves this problem. The additional interface layer allows the implementation types realizing the interface to be hidden.

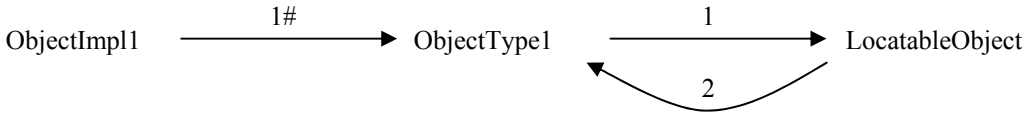

Figure 5. Generic Services in Distributed Environment

In Figure 5, a generic service client acting as a user of the registered object accesses the object through the intermediate interface type rather than the implementation type. The new responsibilities of typecasting may be modeled as given in Table 2. This solution is typical to a CORBA-based application.

Table 2: Typecasting Responsibilities in a Generic Service-based Application in a Distributed Environment

Arrow Id	Operation	Who performs the operation
1#	Widening	Client of Generic Service (source client, e.g. Server hosting the object)
1	Widening	Client of Generic Service (source client, e.g. server hosting the object)
2	Narrowing	Client of Generic Service (destination client, e.g. process which wants to use the object)

5 REALIZING INTERFACE NAVIGATION

The typecasting responsibilities in an interface navigation based-application are captured in Table 3. The arrow ids are as shown in Figure 4.

Table 3: Typecasting Responsibilities in an Interface Navigation-based Application

Arrow Id	Operation	Who performs the operation
1	Widening	Server hosting object's implementation
2	Narrowing	Client accessing the object through an interface
3	Widening	Client accessing the object through an interface

The widening operation at arrow 1 provides the client of the object a handle to the object through the desired interface type. The server, which hosts object's implementation, is

responsible for the first widening operation since the server hosting the object implementation knows the implementation type.

Navigation is an operation performed by the client of an object. The client typically holds an interface handle and obtains another. Arrows 2 and 3 together perform the navigation. Since the client is the originator of a navigation request, the client itself must perform both the typecasting operations.

A consequence of this constraint is that the client of the object needs to know what the implementation type is. While in a single address space environments such as a C++ [6] process, this solution does not pose considerable difficulties; the constraint becomes a bottleneck for distributed programming environments desiring to protect implementation types from being exposed to clients of the implementations. This problem may be solved in two ways.

In the first solution, an additional interface layer may be supported as shown in Figure 6. This solution requires that the implementation must support a unifying intermediate interface, which explicitly inherits from all its multiple interfaces. This solution is similar to the solution adopted for generic services (Figure 5).

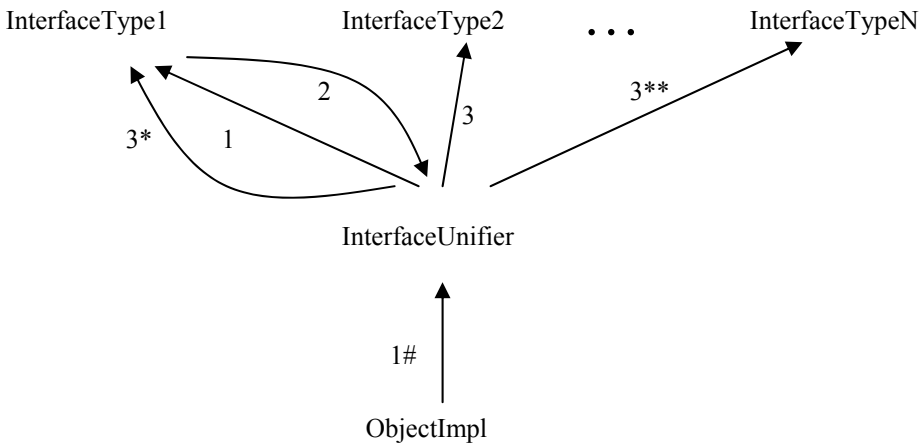

Figure 6. Navigation in a Distributed Environment

The typecasting responsibilities are shown in Table 4. Since an intermediate layer of interface separates the implementation, the clients located on a remote machine are not required to narrow to an implementation type hidden on a remote machine. Typecasting performed at client is limited to interface types.

Table 4: Typecasting Responsibilities in an Interface Navigation-based Application in a Distributed Environment

Arrow Id	Operation	Who performs the operation
1#	Widening	Server hosting object's implementation
1	Widening	Server hosting object's implementation
2	Narrowing	Client accessing the object through an interface
3	Widening	Client accessing the object through an interface

Alternatively, in absence of enforcement of a unifying interface, typecasting maybe performed by sending the typecasting request back through the interface handle available with the client. The implementation then performs a local type-cast to the desired interface and returns the typecasted handle in response to this query. This solution can be seen in COM-based systems [4]. A constraint on this solution is that the query for typecasting must indicate the desired interface type as an argument. If metatypes are not supported in the programming environment, the typecasting queries need to encode the types in terms of other recognized types such as character strings.

6 CONCLUSIONS

Generic services and interface navigation are modeled as typecasting sequences. Implementation consequences of typecasting sequences in distributed environments are discussed. Since a desire for hiding of implementation types conflicts with a typecasting request, if the latter has to be performed by the client, a suitable solution needs to be adopted by the distributed environment. Some solutions were discussed and related to existing paradigms.

REFERENCES

[1] K. Arnold and J. Gosling, *The JAVA Programming Language*, Addison Wesley, 1998.

[2] A. Goldberg, D. Robson, *Smaltalk-80: The Language and its Implementation*, Addison-Wesley, 1983.

[3] B. Meyer, Object Oriented Software Construction, Second Edition, Prentice-Hall, 1997.

[4] Microsoft Corporation, *The COM Specification*, 1995.

[5] Object Management Group, *COS Naming Service Specification*, February 2001.

[6] Stroustrup, *C++ Programming Language, Third Edition*, Addison Wesley, 1997.

About the author

Rushikesh K Joshi is with the Department of Computer Science and Engineering at Indian Institute of Technology, Bombay. He works on object oriented systems and architectures. His recent efforts have been on the design and development of dynamically pluggable first class filter objects for various object oriented programming environments. He is also interested in the development of teaching methods for imparting classroom education in the field of object oriented software development, especially analysis and design. He can be reached at rkj@cse.iitb.ac.in.