An Efficient Naming Service for CORBA-based Network Management

Joon-Heup Kwon, Moon-Sang Jeong and Jong-Tae Park
School of Electronic and Electrical Engineering,
Kyungpook National University
1370 SanKyug-Dong, Buk-Gu, Taegu, 702-701, Korea
Tel: +82-53-950-5543, Fax: +82-53-950-5505
{jhkwon, msjeong}@ain.knu.ac.kr, park@ee.knu.ac.kr

Abstract

There have been growing research interests for the realization of CORBA-based network management system for telecommunication network, and several methods have been devised to apply CORBA technology to telecommunication management network. We concern about the pure CORBA-based network management system. In this system, there is no conventional CMIP agent in a managed system. Otherwise, a managed system contains CORBA-based managed objects with which CORBA-based managing objects interact and perform the management task. To represent the containment relationship between CORBA-based managed objects in a managed system, naming tree is used. To form the tree, CORBA naming service is utilized. However, OMG's CORBA naming service requires the entire CORBA-based management objects to be instantiated at all the time. Since network elements usually do not have sufficient system resources, the applications running on them may experience severe degradation in performance. In this paper, we design a new CORBA naming service. By modifying the OMG's naming service and adding an IDL definition for the new naming service, it can efficiently construct a naming tree without instantiating all the managed objects all at once. CORBA IDL is defined for the new naming service, and performance of the implementation is evaluated to demonstrate its efficiency.

Keywords

network management framework, CORBA-based network management system, naming service, naming tree, TMN, CORBA

1. Introduction

Telecommunication Management Network (TMN) [1] architecture has been applied for the management of telecommunication network. It is known that the

realization of the higher layers of TMN architecture is very difficult task. Recently, Common Object Request Broker Architecture (CORBA) [2] has been considered as the base technology for the realization of higher layer TMN functions and services [3,4,5]. A gateway approach was the first step for adoption of CORBA into TMN system [6,7,8]. Since the approach mainly focused on the interoperability between pure CORBA-based manager systems and legacy CMIP-based agent systems, the capability of CORBA was not fully utilized.

There have been active research works to build pure CORBA-based telecommunication network management. Working group T1M1.5 of committee T1 and Study Group 4 (SG4) of ITU-T are leading these research works. T1M1.5 is currently working on two draft standards: a framework similar in nature to X.700 series for CMIP [9,10], and a generic network information model based on M.3100 [5,11].

In the framework, there is no classical CMIP agent system at network system. Instead, there is a set of CORBA-based Managed Objects (MOs) with which a CORBA-based managing object can interact. It requires network elements to be equipped with Object Request Broker (ORB), CORBA Common Object Services (COS) such as naming service and notification service.

To represent the containment relationship between these CORBA-based MOs in a network element, a naming tree, which is similar to the containment tree for CMIP, should be formed using CORBA naming service [12].

CORBA naming service enables managing object to access an object with the name assigned to the object instance. This name-to-object association is referred as name binding. A name binding is always defined relative to a Naming Context [12]. With the classic OMG naming service, all CORBA-based MOs in a network element should be instantiated to form a naming tree because the naming service uses the Interoperable Object Reference (IOR) of an object “instance” [2,12] to build a name binding.

However, managed systems, e.g. ATM switch, usually do not have enough system resource. If all the CORBA-based MOs in a network element are instantiated at the same time, most of system resources may be assigned to network management operations.

As a solution to this problem, we propose an extended version of conventional OMG naming service, Smart Naming Service (SNS). SNS does not instantiate the entire MOs to form a naming tree. Instead, SNS uses the instantiation information for each object to build a name binding, to allow the object to be instantiated when there is a request for it.

To implement these functionalities, we modify the structure of name binding and the structure of naming context, and introduce a number of new building blocks. As mentioned before, since SNS is an extended version of conventional OMG naming service, managing objects can access MO instances without the knowledge of SNS.

In Section 2, a brief introduction to CORBA-based telecommunication network management framework is presented. The naming scheme suggested by T1M1.5 and the problem with conventional OMG naming service is described in Section 3.

As a solution to this problem, we present SNS in Section 4. We also discuss the requirements, and design SNS, and define the interface of it using the Interface Definition Language (IDL) [2]. Section 5 describes the implementation architecture, and in Section 6, we present the performance evaluation of our implementation. Finally, we drew a conclusion in Section 7.

2. CORBA-based TMN Framework

The TMN architecture defined in Recommendation M.3010-2000 introduces concepts from distributed processing and includes the use of multiple management protocols such as CORBA [1]. It has been known that, with CMIP, it is hard to implement the higher layer of TMN architecture for more sophisticated, extensible, and flexible network management system. Working Group T1M1.5 of committee T1, and ultimately ITU-T, are working on framework for CORBA-based TMN interfaces now.

To date, manager/agent paradigm has been adopted to make TMN network management system. Every managed system has had a CMIP agent. All management operations are performed through the agent. Figure 1 shows the CMIP-based TMN network management system.

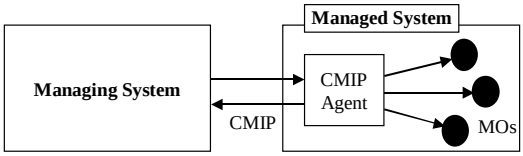

Figure 1: CMIP-based management framework

In T1M1.5 Framework, there's no such classical agent system. Instead, in an managed system, there are CORBA-based MOs connected directly to ORB, i.e. managing system can directly access to an MO and perform network management operations. This is called access granularity [8]. The network management system based on the T1M1.5 framework is shown in Figure 2.

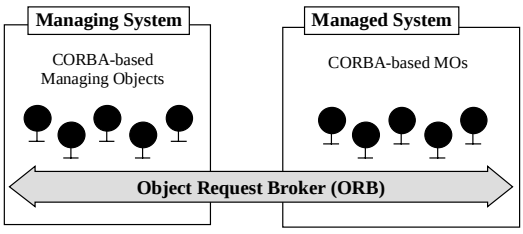

Figure 2: CORBA-based management system framework

3. Naming Scheme of T1M1.5

T1M1.5 is defining the CORBA-based network management framework through two draft standards: T1M1.5/2000-029 [4] and T1M1.5/2000-030 [5]. The first document covers framework requirements, CORBA COS usage requirements, and information modeling guidelines. The second specifies information models, defining a set of generic interfaces and constants based on M.3100.

This section presents the naming scheme suggested by T1M1.5. At the end of this section we point out the problems occurring when the conventional naming service is adopted to implement the naming scheme.

3.1 Naming Scheme Suggested by T1M1.5

As stated before, CORBA naming service is used to achieve access granularity, and to represent the containment relationships between CORBA-based MOs. Figure 3 gives an example of naming scheme according to the framework proposed by T1M1.5 using conventional OMG naming service [12]. Example deals with an ATM switch, ATM_switch_1, which belongs to a network, ATM_network_1.

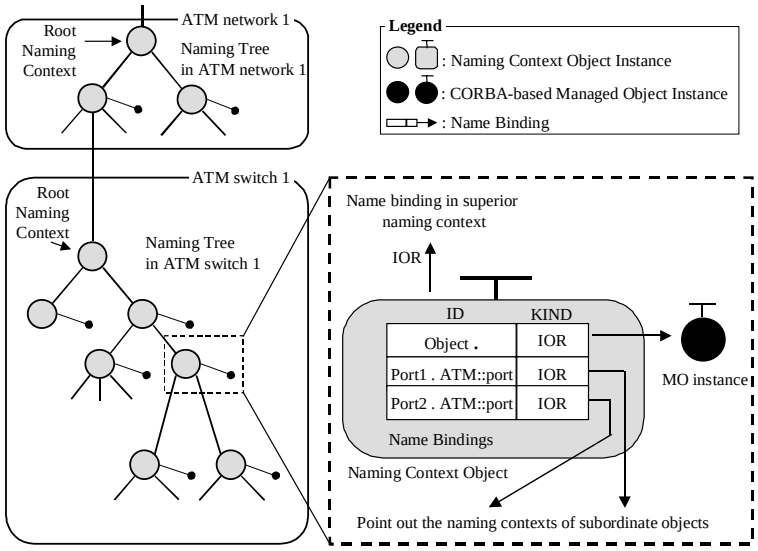

Figure 3: Naming scheme of T1M1.5

A Containment Tree defines the relationship between the MOs. As depicted in Figure 3, one CORBA-based MO instance has a corresponding naming context object since a simple name binding cannot represent the containment relationship. While containing other MO, or being contained to other MOs, these naming contexts forms a naming tree, with single-rooted hierarchy.

Each managed system should provide at least one local root naming context.

This naming context will be bound to superior naming tree as an MO. In Figure 2, the local root naming context in ATM_switch_1 is contained to an MO in naming tree of ATM_network_1. This results in a huge naming tree. Through the naming tree, CORBA-based managing object easily access to a MO directly.

The name of a naming context contains the unique ID in ID field and the scoped class name of a MO in KIND field. The actual CORBA-based MO has special name binding whose ID field value is "Object" and KIND field value is NULL [4].

3.2 Conventional OMG Naming Service and Its Problem

Figure 4 shows the procedure of getting an IOR to a CORBA-based MO instance assuming that the conventional naming service is adopted. Managing object in the figure has already accessed to the naming context of an MO.

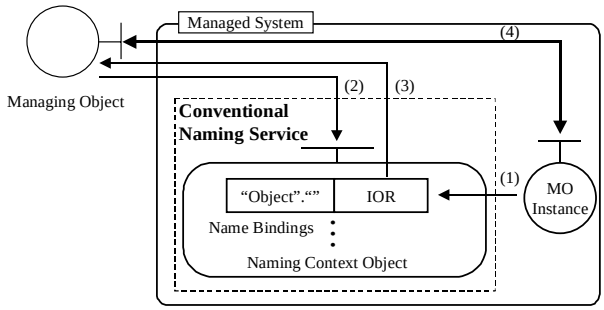

Figure 4: Resolving an MO with the conventional OMG naming service

The steps for accessing an MO with conventional OMG naming service is as follows [8]:

(1) Since OMG naming service requires an object's IOR to build a name binding and an IOR of an object is created when the object is instantiated, the CORBA-based MO must be instantiated to build the name binding of it. (2) When a managing object asks the naming context to return the IOR of the MO instance, (3) the naming context first locates the name binding corresponding to the name, and then, returns IOR in the name binding to the requester, the Managing Object. (4) Once the managing object has the IOR of an MO instance, it can access to the MO instance and perform network management operation.

However, since network equipment has limited system resources, instantiating all managed objects may cause severe degradation in performance. Each network element is in charge of performing its own tasks such as data transmission. If all the CORBA-based MOs in a network element are instantiated at the same time, the network element may not be able to do its own task.

Usually, the MOs do not participate in network management operations at the same time. Consequently, with conventional OMG naming service, part of precious

system resource is assigned to MOs that are not working. So, there must be another way for the efficient use of the system resources.

4. Smart Naming Service (SNS)

As an effective solution to this problem, we introduce a new naming service, Smart Naming Service (SNS). SNS provides an efficient way of using system resources in a network element while forming a naming tree without instantiating all CORBA-based MOs at the same time. It instantiates an object when it is needed for network management operation.

SNS is an extended version of conventional OMG naming service. In other words, It does not change any functionality of conventional OMG naming service.

4.1 Requirements for efficient Naming Service

In this section, we present system requirements, which SNS should satisfy.

R 1: The naming context object of SNS should be able to form a name binding with the instantiation information or the IOR of an MO.

R 2: MOs, which are needed to be persistently instantiated, should be able to be instantiated all the time.

R 3: The structure of name binding should be modified to contain the name, the instantiation information, and the IOR of an MO.

R 4: There should be a way for naming context object to instantiate an object with the instantiation information in the object's name binding.

R 5: Even if the SNS is applied, managing object should be able to get an IOR of an object instance without the knowledge of SNS.

R 6: Each CORBA-based MO should be synchronized with the real device or re-initialized by the previous state information when it is instantiated.

R 7: SNS should have internal DB or memory space for MOs to save its current state when it is terminated.

R 8: After the synchronization or re-initialization, the CORBA-based MO instance becomes real MO; any changes in MO instance should be reflected to the real device, and vice versa.

R 9: If the MO instance already exists, naming context object should just return the IOR of existing MO instance without instantiation of it.

R 10: Any instance, which is not used for a long time, should be deleted returning system resources for other MO instances.

R 11: Among MO instances, there're several MOs that are persistently launched. Such MOs should not be deleted by SNS. So there should be a way to distinguish whether an MO is persistently instantiated or not.

4.2 Design of SNS

In this section, we design SNS satisfying to these requirements above. We use the original OMG naming service IDL definition, `CosNaming.idl`, and, additionally, add one IDL definition, `SNS.idl`, that contains some structures and interfaces for SNS. By doing so, managing objects can access to an MO without the knowledge of SNS. We implement both interfaces.

4.2.1 IDL Definition for Instantiation Information

Firstly, we define IDL level structure for the instantiation information of an MO. Figure 5 shows the IDL definition of the structure containing the instantiation information. `MO_Class_Name` in the figure is to contain the scoped class name of the MO, and `Real_Resource_ID` is to indicate the real MO in the network element. With this information, the object can be instantiated by the naming context object. This enables SNS to build a name binding of an MO without instantiating.

```
// SNS.idl

#include "CosNaming.idl"

module SNS
{
  // . . .

  struct Inst_Info{
    string          MO_Class_Name;
    unsigned long   Real_Resource_ID;
  };

  // . . .
};
```

Figure 5: IDL definition for `Inst_Info`

4.2.2 Name Binding of SNS

Figure 6 shows the definition of the structure for the name binding of SNS. Name binding is internal data structure of naming service. So there is no IDL level definition for name binding. The structure of the name binding of SNS contains three fields for name, instantiation information, and IOR of an MO.

```
// SNS_Impl.h

#include "SNS.hh"

// . . .

struct NameBinding{
  SNS::Name_var       name;
  SNS::Inst_Info_var  inst_info;
  CORBA::Object_var   obj_Ref;
};

// . . .
```

Figure 6: The structure of SNS Name Binding

In the definition, the field named `obj_ref` is to indicate that the object already exists or not. If the object is not instantiated, the value in the field is NULL. Otherwise, the field contains the IOR of the object.

4.2.3 Naming Context of SNS

Naming context of SNS is modified to use the name binding defined section 4.2.2. The naming context of SNS inherits from the naming context of conventional OMG naming service. We just add one method, `bind_sns.` The IDL definition for the naming context of SNS is in Figure 7.

```
// SNS.idl

#include "CosNaming.idl"

Module SNS
{
  // . . .

  interface NamingContext : CosNaming::NamingContext{
    void bind_sns(in Name n, in Inst_Info i)
      raises( NotFound, CannotProceed,
              InvalidName, AlreadyBound );
  };

  // . . .
}
```

Figure 7: Definition of SNS naming context

Using `bind_sns` of SNS, developer of a managed system can make the name binding of an MO, which does not need to be instantiated all the time. For MOs that should be persistently instantiated, the method, `bind` of conventional OMG naming context object should be utilized to build name binding of persistent MO.

4.2.4 Implementation Storage of SNS

```
// SNS.idl

#include "CosNaming.idl"

Module SNS
{
  // . . .

  interface IS{
    Object inst(in Inst_Info order)
      raises( NotFound, CannotProceed,
              InvalidName, AlreadyBound);
  };

  // . . .
}
```

Figure 8: Definition of SNS Implementation Storage

Implementation Storage (IS) is an object that actually instantiates an object according to the information given by naming context objects. IS contains the implementation for each MO class. Figure 8 shows the IDL definition of it.

If an MO needs to be instantiated, SNS naming context object invoke inst operation of IS giving the instantiation information of the MO. IS finds the MO implementation according to the information and instantiate it. And synchronize the instance with the real device or initialize the instance.

4.3 The Architecture of SNS

Figure 9 shows the overall architecture of SNS. The situation here is also the same as that shown in Figure 4.

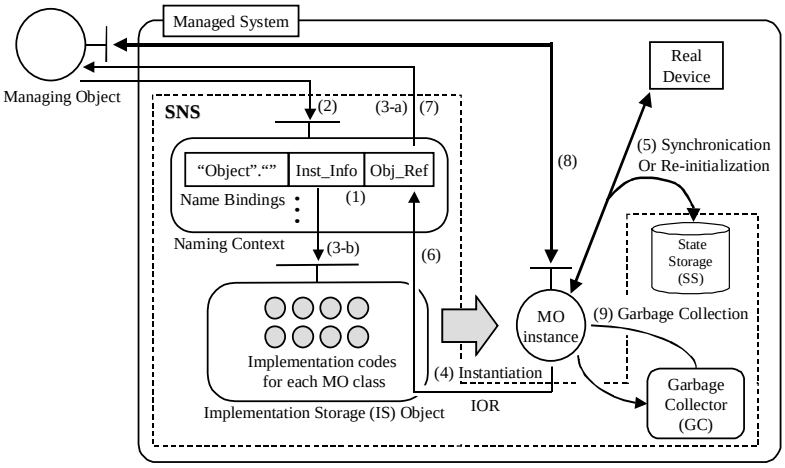

Figure 9: The architecture of SNS

In the figure, there is an object, Garbage Collector (GC). GC is an internal function block of SNS. It releases the system resources assigned to object instances that are not used for a long period of time. The term, garbage, means the MO that is not used for a certain time. GC saves the system resources while releasing the resources that has been assigned to garbage. GC has the list of the object instances that were created by IS. The list contains the names of object instances, and the time-stamps for each instance. The time-stamp is refreshed every time the instance participates a management operation. Before GC terminates a garbage object, GC saves the object's current internal state in State Storage (SS) for next time the object participates any network management operation. The detailed description about the procedure of the `resolve` operation of SNS is as follows:

(1) If a developer of a network element system decides that an MO doesn't need to be instantiated persistently, the name binding of the MO is built with the instantiation information, and the information is to be stored in `Inst_Info` field.

(2) When a manager object asks the naming context to return the IOR of the MO, the naming context first locates the name binding and checks the existence of the MO instance while checking `obj_ref` field. If the value in `obj_ref` field is not NULL, in other words, the object already exists, (3-a) the naming context object should return the IOR of the requestor, the managing object. This prevents the MO instance from being duplicated.

Otherwise, if the value in `obj_ref` field is NULL, (3-b) the naming context should transfer the instantiation information in `Inst_Info` field to the IS to instantiate the MO. (4) With the `MO_Class_Name`, IS decides which MO class is to be instantiated, and then instantiate the MO class. After that, (5) IS synchronize the MO instance with real device, or re-initiates the MO with the previous state information in SS of SNS. And then, (6) the IOR of new instance is returned to naming context. The naming context inserts the IOR in the `obj_ref` of the MO's name binding. This enables the naming context object not to create multiple instances for one MO. After all, (7) the naming context object returns the IOR, (8) letting the managing object to perform management operation with the object instance. GC periodically checks the MO instances listed. (9) If GC finds garbage, i.e. the instance that has not participated in any network management operation, GC releases the system resources assigned to the instance, and sets the value in `obj_ref` field of name binding of garbage instance to NULL.

4.4 IDL definition for SNS

```
// SNS.idl
#include "CosNaming.idl"


module SNS
{

   typedef CosNaming::Istring Istring;
   typedef CosNaming::Name Name;
   typedef CosNaming::BindingList BindingList;
   struct Inst_Info{
     string            MO_Class_Name;
     unsigned long     Real_Resource_ID;
   };

   interface BindingIterator : CosNaming::BindingIterator;

   interface NamingContext : CosNaming::NamingContext{
     void bind_sns(in Name n, in Inst_Info i)
       raises( NotFound, CannotProceed,
               InvalidName, AlreadyBound );
   };                                                          A

   interface IS{
     Object inst(in Inst_Info order)
       raises( NotFound, CannotProceed,
               InvalidName, AlreadyBound );
   };                                                          B
};
```

Figure 10: IDL definition of SNS

Figure 10 shows the IDL definition of SNS. As stated before, we used the original OMG naming service IDL definition, and IDL definition in Figure 10 is the additional definition for SNS.

In the definition, we just add one data structure, `Inst_Info`, one interface, `IS`, and the naming context of SNS inherits from that of OMG's, and have one additional method, `bind_sns`.

A box named 'A' in the figure. Since the naming context of SNS inherits from that of conventional OMG naming service, the managing object can resolve an MO instance as in the same way of conventional OMG naming service [3,8]. The method, `bind_sns`, is to build a naming context with the information of an MO. With this method, the name binding of an MO is built without instantiating the object.

The definition of IS of SNS is in the box named 'B' in the figure. The implementation of IS should have all implementation codes of all management object classes in a network element. The method, `inst`, is for instantiating the object class specified in the parameter, `order`, and IS then synchronizes the new instance and the real device, or re-instantiate the instance using information in SS according to the parameter, `order`. After the synchronization or re-initialization, it returns the IOR of the instance to the naming context.

5. Implementation

We have two IDL definitions for SNS: `CosNaming.idl` and `SNS.idl`. We have implemented both interfaces. With `CosNaming.idl`, managing object can access and perform network management operations without any knowledge of SNS.

We implemented `CosNaming.idl`. However, we used the name binding structure defined in `SNS.idl`. Since the name binding structure is extended from that of original one, there's no problem in using the structure. The `bind` operation of the naming context object is implemented to support the functionality of SNS. Though the naming context uses the name binding structure defined in `SNS.idl`, it uses an object's IOR instead of instantiation information.

SNS.idl is then implemented. `bind_sns` of SNS naming context is implemented to create a name binding without instantiating an object. It uses the instantiation information instead of IOR of an object. IS of SNS contains several object's implementation. It contains functionality for synchronization and re-initialization of an object instance.

As described in Section 4.3, there's a internal function block, the Garbage Collector. In our implementation, GC has its own table which contains the list of the instances that was created by IS. When IS instantiates an MO, time stamp is marked in the table for the MO instance representing the instantiation time. The time stamp is updated every time the MO instance is used. GC periodically, 4 seconds in our implementation, checks the time stamps in the MOs listed in the table, and checks the difference between the time-stamp and the current time

exceeds the predefined time-out value. The time-out value is set to 10 seconds in our implementation. Once GC finds an object instance timed-out, GC releases the resource that has been assigned to the object instance, and set the value of obj_ref field in the name binding of the MO to NULL. While releasing the garbage object, GC save the object's current state information into SS.

6. Performance Evaluation

Figure 11 and Figure 12 show the result of performance evaluation. The performance of SNS and conventional naming service are comparatively evaluated. We use Orbix 2.3 and implemented in SUN SPARC 1000 system. For the performance evaluation, we assumed that there are 1000 MOs in a managed system. With the conventional naming context, 1000 MOs and 1001 naming context objects are instantiated, and, otherwise, with SNS, only 1001 naming context objects are instantiated initially. In other words, all the name binding was built with the instantiation information for each MO at bootstrapping time.

The managing object accesses 100 objects at a time, and the target objects are selected by uniform random sampling. With 2 seconds time interval, the managing object perform the operation 50 times. The time out value for GC is set to 10 seconds.

Figure 11 shows the average response time required to get the IORs of 100 MO instances comparing that of the conventional naming service and that of SNS. As time goes on, the average response time merge.

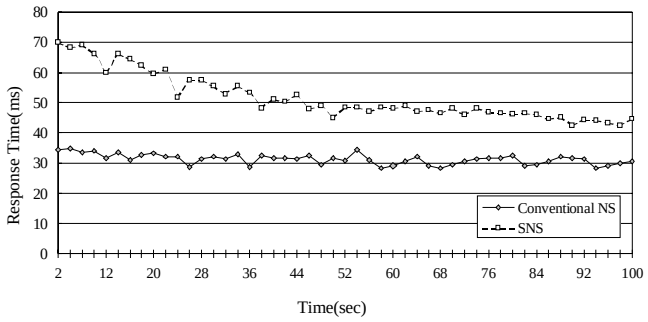

Figure 11: Response time of conventional naming service and SNS

Figure 12 shows that run-time memory usage by network management operations. The amount of memory required by SNS is smaller than that of conventional OMG naming service.

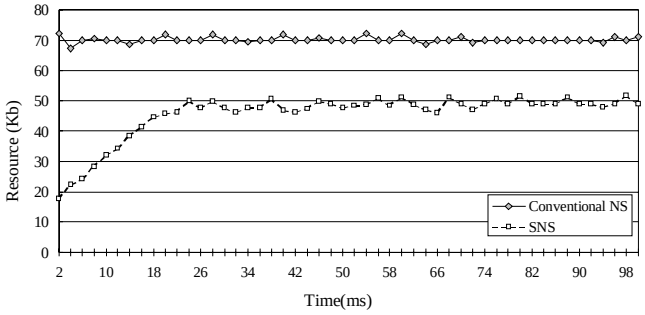

Figure 12: The volume of memory resource of conventional naming service and SNS

Though the response time of SNS is longer than that of the conventional naming service at first, as time goes on, the response time get reduced, and finally it becomes slightly longer than conventional OMG naming service. The reason is that once an object is instantiated, it lasts before the time stamp in the instance expires. However, the volume of the system resources assigned to network management operations with SNS was quite smaller than that of the conventional OMG naming service.

7. Conclusion

The OMG naming service is not suitable for network element that has limited system resources. We introduce new effective naming service, SNS. SNS provides an efficient way to build a name binding for the MO that doesn't need to be always instantiated. MOs that are not participating in network management operation are not instantiated. Consequently, the system resources of a managed system can be efficiently assigned to MOs that participates in network management operations.

We implemented a prototype of SNS satisfying the requirements listed in Section 4.1. Though SNS reveals longer response time, the performance evaluation shows that the volume of system resources assigned to network management operations is reduced with SNS.

Because we still keep the functionalities of conventional OMG naming service, some MOs can be instantiated persistently for shorter response time. The name bindings of the objects are built in a conventional way.

Though SNS provides longer response time, through code optimization and proper choice of persistent objects, we may improve the performance.

SNS can be utilized for not only managed system in network management system, but also systems whose system resource is limited; the mobile phone in wireless communication, electric devices in home networking, and so on.

8. References

[1] ITU-T Recommendation M.3010, *Principles for a Telecommunications management network*, February 2000.

[2] The Object Management Group (OMG), "The Common Object Request Broker: Architecture and Specification," OMG Document formal/99-10-07, Revision 2.3.1, October 1999.

[3] G. Pavlou, "A Novel Approach for Mapping the OSI-SM/TMN Model onto ODP/OMG CORBA," International Symposium on Integrated Network Management, San Francisco, USA, Boston, USA, May 1999.

[4] Working Group T1M1.5, *Working Document for Draft Standard ANSI T1.2xx-2000, Framework for CORBA-Based Telecommunications Management Network Interfaces (T1M1.5/2000-029r1)*, ANSI, July 2000.

[5] Working Group T1M1.5, *Working Document for Draft Standard ANSI T1.2xx-1999, CORBA Generic Network and NE Level Information Model (T1M1.5/2000-030r2)*, ANSI, August 2000.

[6] The Object Management Group (OMG), "JIDM Interaction Translation," Edition 4.31, OMG Document telecom/98-10-10, October 1998.

[7] Moon S. Jeong, Jeong W. Kim, Joon H. Kwon, Jong T. Park, Seong B. Kim, and Chan K. Hwang, "Design and implementation of CORBA-based network management applications within TMN framework," APNOMS'99, Kyungju, Korea, Sept. 1999.

[8] Jong T. Park, Moon S. Jeong, and Seong B. Kim, "A Platform Architecture for the Integration of CORBA Technology within TMN Framework," IEICE Transactions on Communications, Vol.E82-B, No.11, pp.1770 ~ 1779, Nov. 1999.

[9] ITU-T Recommendation X.703, *Information Technology-Open Distributed Management Architecture*, October 1997.

[10] ITU-T Recommendation X.711, *Common Management Information Protocol Specification for ITU-T Applications*, October 1997.

[11] ITU-T Recommendation X.3100, *Generic network information model*, July 1995.

[12] The Object Management Group (OMG), "Naming Service Specification," OMG Document formal/2000-06-19, Version 1.0, April 2000

[13] IONA Technologies, *Orbix Programmer's Guide, Orbix 2, Distributed Object Technology*, October 1997.

[14] Mitch Henning and Steve Vinoski, "Advanced CORBA Programming with C++," Addison-Wesley, ISBN 0-201-37927-9, Jan. 1999.