Plug-and-Play Network Service Configuration Using CORBA

Syed Kamran Raza, Bernard Pagurek, Tony White

Dept. of Systems and Computer Engineering,
Carleton University
1125 Colonel By Drive
Ottawa, ON. Canada K1S 5B6

Email: {skraza, bernie, tony}@sce.carleton.ca

Please address all correspondence to B. Pagurek
Phone: (613) 520-5724
Fax: (613) 520-5727

Abstract

Configuration of network services is a difficult problem requiring considerable human involvement. Recently, the hardware configuration of a personal computer has been simplified by the Plug and Play specification and the addition of the universal serial bus. Also recent developments in the CORBA framework, the Java language, mobile agent facilities, and Jini, have provided new tools that facilitate the movement of code from device to device, and help to provide a framework for service subscription and utilization on demand. This paper brings together the principles of the Plug and Play specification, CORBA, and mobile code in a design for plug and play network services that should provide more interoperability and language independence than the Java-centric approach.

Keywords: Network service configuration, mobile code, mobile agents, plug and play

Plug-and-Play Network Service Configuration Using CORBA

Syed Kamran Raza, Bernard Pagurek, Tony White
Systems and Computer Engineering, Carleton University
{skraza, bernie, tony}@sce.carleton.ca

Abstract

Configuration of network services is a difficult problem requiring considerable human involvement. Recently, the hardware configuration of a personal computer has been simplified by the Plug and Play specification and the addition of the universal serial bus. Also recent developments in the CORBA framework, the Java language, mobile agent facilities, and Jini, have provided new tools that facilitate the movement of code from device to device, and help to provide a framework for service subscription and utilization on demand. This paper brings together the principles of the Plug and Play specification, CORBA, and mobile code in a design for plug and play network services that should provide more interoperability and language independence than the Java-centric approach.

1 Introduction

In May of 1994, Microsoft introduced the Plug and Play (PnP) specification [9], [6] for PCs. PnP had three immediate goals: to make PCs easier to setup and configure; to ease the task of installing new software and hardware, and to facilitate configuration of the PCs while operating.

Microsoft's Windows '95 was the first operating system that implemented the specification. The framework was immediately welcomed by PC manufacturers and peripheral vendors and it has gained wide acceptance since then. Very recently, Windows '98 was released and it has been described as a *real* PnP OS, supporting full plug-and-play functionality with the help of new technologies such as the Universal Serial Bus (USB). Networks present a number of challenges with regard to the integration of new devices and services when compared to the PC. The concept of PnP from the PC world requires an acknowledgement of added complexity seen in networks. The added complexity includes:

- Keeping a registry for all the network devices and their interaction schemes is almost impossible.
- Provisioning a network component is more complicated than installing a simple PC driver.
- There is no centralized OS available over the network that could handle automatic installation.
- Services are often distributed; e.g., email.

Technologies are only now becoming available that could be used to support a plug-and-play capability in networks. First, CORBA has introduced naming and trader features [10] that provide for service discovery that allows potential clients of a service to locate a service entity. Second, CORBA 3.0 [19] offers a form of code mobility through exploitation of object-by-value message passing that would facilitate the deployment of client-side code[1] on the client device thus enabling service utilization. The research reported in this paper is part of an overall effort to evaluate recent emergent technology and its use in the implementation of plug-and-play services. Our previous work on plug-and-play network services has included the evaluation of a Jini-based implementation [14], mobile agents [12] and code mobility [17]. A CORBA evaluation is important in that mobile agents and Jini [15], [16] represent (largely) Java-centric solutions; interoperability thus suffers. A CORBA approach therefore promises to provide language independence and interoperability with legacy components.

This paper consists of the following sections. The paper continues with a description of the motivations for plug and play network services and the objectives for the scheme presented. The requirements for the PnP framework are then presented, followed by a section containing a brief

[1] Client-side code can be thought of as an agent acting on behalf of the server side of a service-providing component.

description of relevant features of CORBA. A section providing detailed descriptions of PnP scenarios using CORBA then follows. The paper concludes with sections that provide an evaluation of the PnP framework presented and conclusions drawn from the research.

2 Motivations and Objectives

A LAN consists of several servers and many hosts that are connected over an Ethernet using TCP/IP for the communication. These workstations and servers may run different operating systems (Windows 95, Windows NT, Linux and Solaris) and may use different platforms (PC, Unix, Mac etc.). The services over a network are usually installed on a server and shared among clients. For example, we have print servers, file servers, application and domain servers etc. that allow clients access to services using appropriate protocols. These servers usually provide access to some resource on the network such as a printer, a storage disk, or router. However, there is often a need for installation of corresponding software on the client for that resource. We use the terms *service* and *service module* respectively for the resources and their associated software.

The focus of this paper is the Network Management problem of *configuration and related software distribution* that arises with the installation of a new service provider component. On a broader scale, our objective is:

- To make service provision easier to setup and configure in networks.
- Continuous supervision of the networks for service provisioning due to their dynamic nature.
- To minimize the interactions between a user and a network component during its installation.
- To minimize network administrator intervention in a component's configuration.

These objectives can be met with a use of current and proposed features of CORBA [10], [11] and mobile code. Other approaches to PnP we have explored include mobile agents [2], [12] and Jini [14]. Software agents [4] and, specifically, mobile agents[2] [3], [5] have also been shown to be widely applicable to network management tasks [1].

We propose a Plug-and-Play (PnP) framework for networks that is composed of smart, intelligent service provider components. After some manual steps that are required to install these components in the network, the components would make their installation and configuration arrangements themselves without user intervention. We associate the term *Plug-and-Play component (PnP)* with such components. It should be noted that plug-and-play is a server-side characteristic and thus this term refers exclusively to a service provider component.

A network printer is chosen to represent a service provider component for describing the network PnP framework in forthcoming sections. However, more general plug-and-play services can easily be accommodated within the framework presented here.

The system of our concern is the framework that provides plug-and-play capability and its environment includes the actors that use or affect this system via interfaces (physical network topology, network cards, console etc). On inspection, we discover five major actors that interact with our framework:

1. Vendor
2. Network Administrator
3. Service provider network component
4. Client network component
5. Broker

As stated earlier, a network component that offers some kind of service(s) to the other network components is called a Service Provider component while one that requires or uses an offered service is called a Client component. For example, a network printer is a service provider while all other workstations that wish to use the printer are its clients. The vendor is the party that supplies a service provider component and its associated software. The Network Administrator is the person who is in charge of managing the network. Although, one of our goals is to minimize the interactions of the administrator with the network, we still need him for monitoring the overall functionality of our PnP framework. Finally, the

[2] We will use the terms mobile agent and code interchangeably in this paper. We use Lange's definition of an agent here.

broker serves as glue between clients and server components; i.e., components find each other using the broker.

3 Requirements and Facilities

Requirements

The PnP framework is the system that is required to provide the following functionality and capabilities in the components:

- Auto-installation (client/server)
- Locate-and-Join the network
- Service provision (PnP component)
- Service discovery and its use (client components)
- Service upgrades
- Dynamic configuration provision

Installation and configuration cannot be automated completely; there are tasks that require user intervention. Examples of these tasks include:

- Establishing a physical network connection.
- Supplying initialization information related to the configuration, setup and security.
- Monitoring System messages, error messages and logs.

The major requirements of our system have been determined and now they have to be organized in some manner. Organizing the requirements by *roles* is a natural choice since it reflects the actual source where the requirements come from. We try to organize requirements by assigning appropriate responsibilities and functionality to each actor. Following is a list of responsibilities and requirements that is constructed for our system.

Vendor

1. Supply a service provider component with the required software pre-installed on it.
2. Program some hard-coded information into the component at the manufacture time.
3. Provide a repository for the service modules (client and server sides).
4. Accept registrations from service provider PnP components.
5. Notify about service module upgrades.
6. Control security and access over registering PnP components and providing service modules.
7. Third-party provision (in case the vendor goes out of business, it must notify the registered components).

Network Administrator

1. Set up initial configuration parameters, hardware arrangements.
2. Monitor overall system behavior, response, logging etc.

Service Provider PnP Component

1. Physical installation
2. Locate and Join the network
3. Initialize: the parameters for setup and security controls
4. Registration with the vendor and obtaining server side service module
5. Service advertisement (offer) over the network using some broker service
6. Service Provision and Up-gradation
7. Accept and register the client requests
8. Make arrangements for the distribution of client-side service module
9. Security and access control for client registration
10. Inform all the clients and the vendors before withdrawing its service

Client Component

1. Physical installation
2. Locate and Join the network
3. Initialize: the parameters for setup, service requirements and security controls
4. Service Requirement
5. Find and lookup the appropriate service provider and register with it using a broker service
6. Receive client-side service module
7. Use required service

8. Get updates on client side modules
9. Security and access control
10. If a service provider is no longer available, find/look up some other provider

Broker

1. Facilitate service advertisement of service provider PnP components.
2. Accept requests from client components for locating other components and services.
3. Bind a service and client component.

Facilities

We can now summarize the main facilities that are required by our framework. These facilities are:

- Naming / Trading
- Security
- Notification (events)
- Persistence (storage)
- Transaction
- Logging

Component Interfaces

The framework consists of many components that need to collaborate to achieve the PnP functionality. However, these components need to know each other's interfaces. The main participants are the PnP-component, the Client component, and the Vendor component, and their proposed interfaces are:

1. **Vendor-Interface**: A vendor implements the interface that defines operations for PnP-component's registration, de-registration and, most importantly, module interactions.
2. **PnP -Interface**: This interface has to be implemented by component that needs to provide a service. The operations supported by this interface allow client's registration/de-registration, notifications from the vendor and for the module reception.
3. **Client-Interface**: This interface has to be implemented by a client component that needs to use some service. The typical operations defined are much the same as the operations provided by a PnP Interface; the only exceptions being the registration and de-registration operations.

In addition to these essential interfaces, we may think of interfaces for supporting facilities like security, naming etc. Note that a component may implement more than one interface. For example, a service provider PnP component may also function as a client component for other services.

4 CORBA Overview

It is becoming difficult to get software and hardware to work together with the increasing complexity and heterogeneity of modern networks. Moreover, different systems use different languages, which also makes software inter-working very difficult. The solution to these problems requires a distributed framework that would satisfy the requirements of distribution, scalability, heterogeneity, and reliability. In recognition of the problems associated with distributed environments, the Object Management Group (OMG) was formed in 1989. Application integration as its main goal, the OMG has defined open-ended distributed environment specifications, known as the Object Management Architecture (OMA), which addresses the major issues of multi-vendor portability and inter operability. One of the essential components of OMA is the Common Object Request Broker Architecture (CORBA). CORBA provides a great deal of flexibility in distribution and implementation of today's networked applications through the transparencies related to programming languages, platforms, operating systems, and network protocols. The key feature of CORBA is its cross-platform communication architecture, which enables definition, transportation, implementation, and invocation of objects in many programming languages; e.g., C, C++, Smalltalk and Java. All these features make CORBA a powerful enabling technology for network management purposes.

CORBA is described adequately in many places [10], [11], [19], so we omit a general overview here. Several of the CORBA services are especially of interest in the research reported here; i.e., the Naming and Trading, Event, Security, and Property Services so these are discussed briefly later in section 5.3. Those aspects of CORBA that promote interoperability are also quite important. These include the specifications

for ORB interoperability protocols and portable object adapters. These features and the OMA are described in the next three sections.

Object Management Architecture

The Object Management Architecture provides a common language for applications and enables *interoperability at the application level* by defining standard services and interfaces. OMA gives us the big picture of the architecture whose core element is CORBA. Figure 1 illustrates the OMA and shows the object interfaces that can be categorized into four main categories:

1. **Object services:** These are standardization of basic services, that almost every object needs. Examples of these services include Naming, Trading, Security, Event, Life Cycle, Concurrency, Transaction, and several others.
2. **Common facilities:** These are the horizontal facilities that provide an intermediate level services to applications. Compound Document management facility is one of the examples of these facilities.
3. **Domain interfaces:** Many industry groups can benefit from standards of their own by defining domain interfaces (vertical facilities) in collaboration with OMG's Special Interest Group (SIG). These domain interfaces provide the standard interfaces that are oriented towards specific application domains, such as health care, financial services, and telecommunication.
4. **Application interfaces:** They are developed specifically for a given application, such as the interfaces of this PnP application. These interfaces are defined by the end-user and not by the OMG.

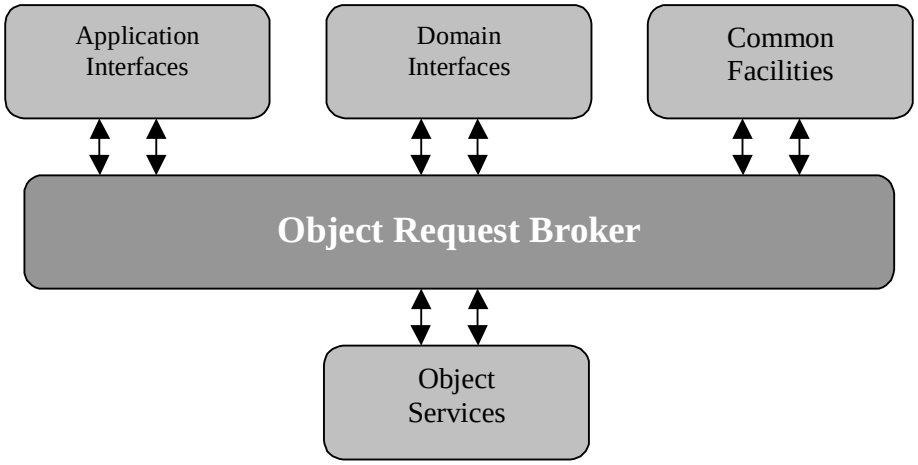

Figure 1: Object Management Architecture

Interoperability and the ORB

When communicating among objects that reside on different ORBs, there is a need for an inter-ORB communication protocol to fulfil the heterogeneity needs. There are two ways of putting this heterogeneity into perspective: either get all the ORBs to speak the same protocol or use bridges to translate protocols when ORBs use different protocols.

For ORB-to-ORB interactions, the OMG has defined the General Inter-ORB Protocol (GIOP) that is an abstract, simple, and easy-to-implement connection-oriented protocol. Due to the ubiquity of the Internet, the Internet Inter-ORB Protocol (IIOP) has been developed for TCP/IP as a concrete representation of the abstract GIOP to support the Internet world. Like IIOP, the mappings of GIOP are also possible for other protocols, such as IPX. Interoperability issues can also be addressed by defining an open-ended set of Environment-Specific IOP (ESIOP). One such example is the DCE Common IOP (DCE-CIOP).

Figure 2 shows various Inter-ORB protocols and their relationships to one another. This PnP application assumes the use of IIOP for addressing interoperability among various ORBs.

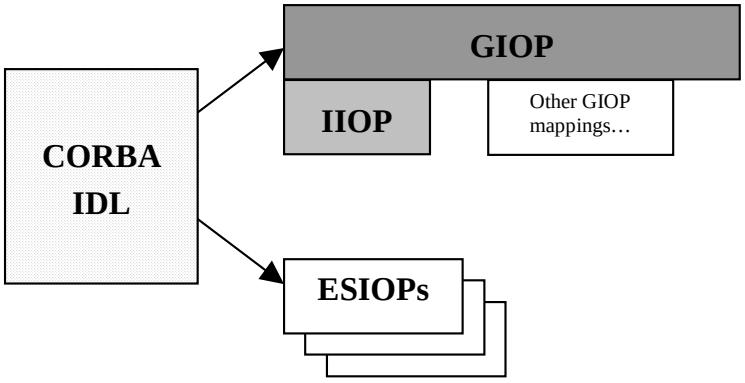

Figure 2: Inter-ORB Protocol Relationships

Portable Object Adapter

The current CORBA specification describes a Portable Object Adapter (POA), which allows applications and their programming language object implementations to be portable between ORBs supplied by different vendors. A POA is an enhancement of the Basic Object Adapter (BOA) and, in fact, is its replacement; both representing a reification of the Adapter design pattern. The newly defined POA specification supports a wide range of features, including: user- or system-supplied object identifiers, persistent and transient objects, explicit and on-demand activation, multiple servant CORBA object mappings, total application control over object behavior and existence, and static and DSI servants.

5 CORBA Based Plug-and-Play Approach

5.1 Introduction

A plug-and-play approach for network services can be implemented using CORBA. The approach is realized as a distributed application, where the participant components reside as CORBA objects on an ORB. Following is the description of some of the major concepts in the CORBA-based approach:

1. Every participant entity and network component in this PnP scheme is a CORBA object that has a unique object reference.
2. The components define standard interfaces for plug-and-play negotiations.
3. The components communicate with each other through interface invocation for achieving the PnP functionality.
4. The ORB acts as the underlying communication structure and middleware component, which takes care of all this communication by providing the appropriate transparencies.
5. Several CORBA services (Naming, Security, Events, etc.) are used to provide the basic object services to this distributed application.

Figure 3 on the next page illustrates the deployment of the CORBA-based PnP approach. The diagram does not show all the building blocks of an ORB -- such as stubs, skeletons, ORB interface, and POA -- for the sake of simplicity.

5.2 Main Building Blocks

Before describing the PnP scheme, we describe the major building blocks of the PnP scheme.

Domains and ORBs

The PnP application is divided into two domains -- the vendor domain that comprises the vendor site and the vendor component, and the enterprise-domain that involves the network of interest, running various clients and server (PnP) network components. Each of these domains has an ORB on which the respective components reside. These ORBs not only bind local components but also inter-operate for enabling remote bindings. Although the ORBs may be different from a technology perspective (network, protocol, and

platform), it is assumed that both the ORBs use the same protocol and communicate using the IIOP. Given the existence of a specification for bridges, the above assumption does not pose a significant limitation.

Participants as CORBA Objects

Of the identified actors, the PnP client, vendor, and admin components are represented as CORBA objects, whereas the underlying ORB and its directory service play the role of a broker actor. Every hardware component (PnP and other network components) needs to have appropriate ORB functionality pre-installed on it in order to function as a CORBA object. Furthermore, every CORBA object in this system has a unique Object Reference (OR), assigned by its host ORB, which remains valid until the object is explicitly deleted. The participating components may be implemented in a variety of ways using different programming languages but they define the standard PnP interfaces proposed earlier in section 3.

In CORBA's method invocation scheme, a caller is termed a *client* and the callee is termed a *server*. Note that CORBA allows an object to play the role of a client and server simultaneously. For objects to communicate and collaborate in this manner, the client object needs to know the server component's interface and its semantics, and the server component's reference.

The publication of well-defined interfaces by each server component solves the first requirement, whereas, the second problem is addressed using an interoperable object reference (IOR).

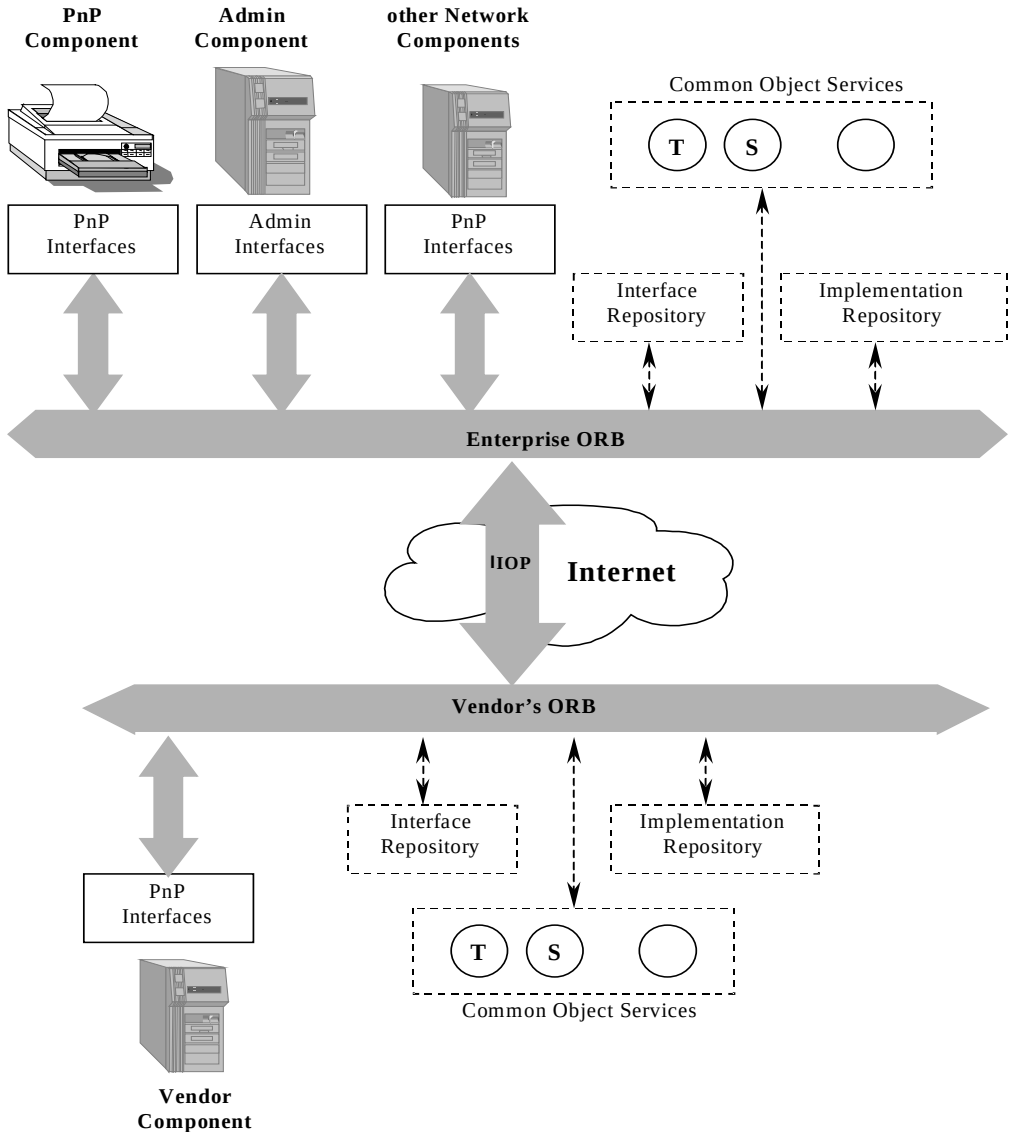

Figure 3: CORBA Based PnP Approach

Interfaces

For objects to 'plug-and-play' together in a useful way, clients must know exactly what they can expect from other objects. Therefore, the interface definitions of each object are used to define the required contract/agreement among the clients and servers. The OMG-IDL can be used to write the PnP interfaces in CORBA. An example PnP interface, written in IDL, is shown below. The shown interface is implemented by a vendor component and helps the PnP components in registering and de-registering with their vendors.

```
module PnP {

    struct ComponentInfo {
      string name;
      string owner;
      string address;
      ….
    };

    interface vendor {

      readonly attribute vendorName;
      ……

      boolean register(in ComponentInfo pnpc) raises SecurityException;
      void deregister(in long long registrationID);

    };
};
```

Figure 4: Example PnP Interface (Written in IDL)

At this moment, the PnP interfaces are proprietary to this PnP approach and thus fall under the category of 'application interfaces' in the Object Management Architecture (OMA). However, It is assumed that these CORBA-based PnP interfaces will be standardized through some SIG in future and the vendors will start manufacturing the components with the built-in support for these interfaces.

Interface and Implementation Repositories

Since all CORBA based invocations (static or dynamic) require IDL interfaces, it is proposed that both the vendor's and enterprise's ORB maintain an interface repository (IFR), which is accessible by other ORBs as well. The IFR at the enterprise ORB stores the interface definitions of network client components, PnP components, and admin component, whereas the vendor's IFR maintains the interface definitions for the vendor component.

An Implementation Repository (IR) is also required on both the described domains for storing and retrieving the object implementations. This repository also provides other features such as the activation and deactivation of objects and their servers.

5.3 Useful CORBA Services

This PnP application benefits heavily from CORBA's object services. Although CORBA specifications describe 16 common services [10] that are used (directly or indirectly) by most of the CORBA applications, this section highlights only those services that provide a key functionality to this PnP application.

Naming and Trading service

The clients of an ORB-based system require a directory service for locating the objects across the network. CORBA provides two directory services -- Naming and Trading. The Naming service is the principal mechanism that provides a mapping from a name to an object reference and is used when clients know with what object they want to communicate. The Naming service, however, does not fulfil all the

requirements in dynamic environments. For instance, a client component looking for a printer service that may print color and double sided at a speed of 10 pages/min requires a sophisticated directory service. In this case, the client only knows what it wants but does not know which object best fits the requirement, therefore, it needs to search by category. To fulfil such requirements, CORBA provides a Trading service. The trading service not only serves the basic function of name resolution; it also allows server objects to register their service offers and clients to find the required services. In other words, if a Naming service works as *White pages*, a Trading service performs the functions of *Yellow Pages* in an ORB.

Since CORBA's trading service fits well into the role of a broker component, it is suggested that each ORB in this PnP application run a trader service. The components first obtain the OR of the trading service through the ORB-interface. Using this OR, a PnP component may advertise its service while the client components may discover (find) the services they require.

Security service

CORBA provides a detailed specification for a Security service, specifying mechanisms that provide end-to-end security in multi-vendor object networks. It also specifies the functional aspects for security service as discussed earlier in the requirements section. The CORBA security service defines appropriate security interfaces for the authentication, access control, recording audit events, and security administration. All these features satisfy the security requirements of this PnP application.

Event service

The proposed PnP approach comprises several scenarios where components require the event and notification services for asynchronous communication. The CORBA framework provides an Event and Notification service that may be useful for this purpose. This event service is designed to provide the 'change notification'. This means that a change in an object generates an event that is propagated to all interested parties. The objects that are changeable and generate events are called the *supplier* objects, while other objects that are interested in the notification of the event are called the *consumers* of the event. The suppliers and consumers of an event communicate using another CORBA object, called the e*vent channel*.

Property service

The participant components of the PnP application require definition of some properties at run time. For example, after initial installation of a network component, the administrator may want to configure the component by supplying the initial configuration parameters (e.g., name, location, owner, and id). This requirement is fulfilled by CORBA's Property service that allows associating the properties (i.e., <name,value> tuples) dynamically with an object using the standard interface calls.

Dynamic Invocations

Although the static method invocation works well for most of the PnP negotiations, an alternative dynamic approach may be chosen in some cases. For example, the clients that have no knowledge of the interface of an object can use the DII, whereas the DSI can be used when we want our object implementation to be changed and upgraded without disturbing or affecting its clients. The use of the dynamic invocation feature provides us with many advantages that are listed in the evaluation section of this paper.

5.4 Scheme Description: Step by Step

Having covered the major building blocks and services required for a PnP approach, this section describes the interactions and scenarios of the CORBA-based PnP scheme. First the scope of CORBA in this approach needs to be emphasized as follows.

The CORBA framework is used only for facilitating the installation and configuration steps of a PnP component in this design. Once the corresponding service modules are installed and activated on a client component, the client component uses the required service directly over the network without any involvement of CORBA framework. As an example, consider a PnP printer. The PnP approach uses the CORBA framework, its transport, and its services to arrange the printer device driver installation on the client components. Now, a client component that wishes to print a document does not use CORBA method invocation but communicates directly with the printer component over the network through the installed printer driver.

Component's layout

Consider a typical heterogeneous network. The network is CORBA-enabled; i.e., the components support the software for the ORB functionality. In this approach, a 'network component' refers to the CORBA object representation of the network device. The components are connected over a common ORB object bus, support the standard PnP interfaces, and have unique object references. The ORB is running the

CORBA object services that are required for this application and these services are accessible to every component through the standard interfaces. These services include naming, trading, security, event, and persistence. Moreover, the other necessary ORB elements like an Interface Repository and Implementation Repository are also available to the components.

The responsibility of maintaining the network lies on the shoulders of a network administrator. Therefore, an administration application is running on enterprise ORB that the administrator may use to configure the network components. A similar administration application, not shown in the actual layout, is also running on the vendor's ORB for managing vendor components.

PnP component's installation

Now assume that the administrator realizes the need for a new service -- say printing -- on the network. The administrator purchases a new service provider PnP component (printer) that has a built-in support for that particular kind of ORB. After its manual installation, the PnP component's software module connects the component to the ORB and finds the object references for essential ORB elements (i.e., Naming, Trading, Implementation Repository, and Interface Repository). The PnP component is assigned a persistent object reference by the ORB that is used by the component for registration with the Trader service. The component also provides the information to the Trader about the services it offers. Furthermore, the PnP component's implementation is registered with the Implementation Repository using a POA, whereas, the component's interfaces are stored in the Interface Repository. The PnP component also supports a configuration interface, which is used by the admin component to assign initial configuration parameters and other pertinent configuration information.

Vendor's setup

The PnP component's vendor also maintains an ORB for its product support. The vendor's ORB runs the same facilities and services that are specified for the enterprise ORB. The central component on the vendor's ORB is the vendor component, which is accessed by the vendor-manufactured PnP components. However, the PnP components require the knowledge of its vendor component for accessing it. Therefore, the vendor hard-codes the IOR (Interoperable Object Reference) and stub of its server component in each of its manufactured devices so that they are able to contact the vendor after being installed. Due to the use of persistent IOR for components, a component's reference remains valid even after its location change. The vendor component also implements the standard interfaces for its clients (PnP components). Instead of using the static skeleton approach, the use of DSI at the vendor component is suggested because it allows a vendor to change the implementation of components dynamically.

PnP component's registration with its vendor

After executing an initialization routine and getting appropriate references of the required CORBA services, the newly added PnP component invokes the registration interface on the vendor component using the pre-configured IOR. This registration interface may be proprietary because both the participants -- the PnP component and the vendor component -- belong to the same vendor. The registration request by the PnP component is invoked using the hard-coded static stub approach[3]. The static request goes through the component's ORB and is forwarded to the vendor ORB, where the vendor component object implementation is contacted through the Implementation Repository and the registration is completed. After registration, the PnP component may invoke a standard interface method to download the latest server-side module.

PnP component service advertisement and client's requirements

After registering successfully, the PnP component is ready to offer its service to potential clients. There are two modes for service provisioning as described previously. In the first mode (Pull), a service provider PnP component allows interested clients to discover its service, whereas the PnP component initiates the service offer and discovers its clients in the second mode (Push). These two modes can be incorporated in CORBA-based approach as follows.

Assuming that the PnP component is in charge of the service provisioning, then the question that arises is "how does a new PnP component know which clients are interested in its service?" It is proposed that each interested client registers with the trading service and specifies its requirement and a PnP component queries the Trading service and makes a list of all the objects (its potential clients) that have shown an interest in its service.

The second mode is more complex. In this case, the service provider component advertises its service with the Trading service, whereas a client component queries this trading service for finding a service. If

[3] The registration interface could also be invoked dynamically.

the service is not found, clients need to wait until such a service registers with the trading service. CORBA's Event service is used to provide this notification mechanism; i.e., a 'service availability' event is defined for each type of service, whose consumers are the client components and supplier is a PnP component that offers the associated service. When the PnP component generates the 'service availability' event after completing its initial configuration routine, the registered client components are notified through the standard interfaces. The event contains the IOR of the PnP component as the event information that is used by the client components for interacting with the PnP component. Since the clients now have discovered their required service provider component, therefore, the clients invoke a standard interface on the PnP component that registers the caller client component. The PnP component waits for a reasonable amount of time (depending on network latency and size of network) after generating the service availability event before assuming that all the interested clients have been registered.

Whether using the first or second mode for service discovery and requirement, the PnP component has knowledge of its potential clients at this stage. The next step is to find the client's platform (i.e., machine, OS, etc.) so that an appropriate service module may be arranged for the client component. The PnP component invokes a method on each registered client to retrieve the client's configuration information. The method returns a structure that contains the information about the client component. Note that clients may also supply this information during their registration routine with the PnP component in first case, thereby saving an extra method call by the PnP component on each client. The PnP component now has the pertinent information about its client components, from which the component determines the requirements of its client and constructs a requirement list.

Service module distribution and activation on clients

At this stage, the clients are ready to receive a service module. Of the two possible modes of module distribution to the client components, this approach prefers the mode in which a vendor is responsible. Therefore, the PnP component invokes an operation on a standard module interface on the vendor component and passes the list of client components (their IORs) along with their required module types. The PnP component may use CORBA's deferred synchronous invocation mode for this operation and hence does not block waiting for the response. The vendor component receives the request from the PnP component, gets the required service modules from its repository, and sends these service modules to the corresponding client component by invoking standard PnP interface calls and using IORs supplied by the PnP component. It is to be noted that what is required here is sending an actual executable service module to the client component and not its reference. One of the shortcomings of the CORBA 2.2 framework is that it only supports the option of passing object "by-reference" and not "by-value" in method invocations[4]. In addition to the code that is needed to use the service, the service module contains the information about its corresponding PnP service component. In cases such as printer provisioning, where the service requires just a client-side counterpart, the service module may just contain executable code, whereas for some other cases, it may contain extra information about the server component so as to serve as a proxy at the client. Ultimately, the client receives this service module (as a byte stream) and activates it on its machine. In this manner, all the interested clients are configured and the vendor component sends a response back to the PnP component. With this, the initial cycle of installation and configuration of the PnP component is completed.

Dynamic configuration

For keeping up with the changing network environment, the PnP components should also configure any client components that are added after this initial configuration cycle.

If the PnP component is responsible for the service provisioning (Push mode), then it should poll the Trader service time to time, check all the client components that are interested in its service and filter out the interested clients that have not yet been configured. Once all new clients are discovered, the module distribution and activation process is repeated for these non-configured components. An alternative solution would be to use the Event and Notification service to notify the PnP component whenever some new interested client registers with the Trader service.

Future configuration support is implemented differently when clients are responsible for discovering their server components (Pull mode). In this case, a newly added client component registers with the Trading service and queries it for any PnP component that fulfils its requirements. If such a PnP component is found, the IOR of the PnP component is retrieved from the Trader service and then interface methods are invoked on the PnP component that would in turn call provisioning routines for the new client component.

[4] CORBA 3.0 incorporates this object by-value (OBV) passing feature.

If no PnP component is found by querying the Trader service, the client component uses an event notification mechanism for registering with the PnP event channel to receive notifications when such a PnP component becomes available.

Service module upgrade

The other major requirement for supporting the PnP approach is related to the upgrade of service modules. Since upgrades may arrive any time, an event notification mechanism is needed. A vendor already has the list of registered PnP components, therefore it notifies them directly whenever a service module upgrade occurs. The notified PnP components in turn call back the service module interface on the vendor to retrieve the upgraded service module. If the module is a server-side module, the module is installed and upgraded at the server PnP component (using the DSI mechanism). Whereas, if the upgrade is related to the client-side, then all the client components that were configured previously for that service module need to be upgraded. The PnP component may query all the affected client components for their interest in upgrade by invoking a standard PnP interface. For all those clients that have shown interest in upgrade, the PnP repeats the steps of module distribution and installation. Note that a vendor may also directly notify client components in case of client-side service module upgrades but this requires the vendor to maintain a list of all the registered clients.

Although this PnP approach has not mentioned security explicitly, it is assumed that all those interactions in this PnP application that require security controls use CORBA's Security service.

5.5 Conformance to the Requirements

This section looks back at the CORBA-based design of the PnP approach and tries to evaluate if the design conforms to the requirements presented earlier.

CORBA allows various components to automate their installation process, where the ORB serves as the glue among PnP participants by providing many important facilities. The components, serving as the CORBA object, immediately discover the network through the help of the ORB. The trading service acts as the broker component and allows PnP components to advertise their services and clients to find their required services. The tasks of service provisioning are facilitated by CORBA's static and dynamic invocation mechanisms. The CORBA-based approach also conforms to the requirement of service upgrades and dynamic configurations by using the notification facility of CORBA. The scheme also benefits from CORBA's services for Security, Persistence and Trading, thus fulfilling requirements put forward in section 3.

6. Evaluation

The following advantages and disadvantages have been observed for the CORBA based PnP application.

Advantages

The CORBA environment suggests an architecture that runs on every platform. Therefore, the CORBA based approach provides a portable, object oriented and interoperable application that is hardware, operating system, network and programming language independent. This approach allows development of dynamic applications that can be easily assembled and modified on the fly by using DII, DSI and Trader facilities. The Trader facilitates dynamic discovery and late binding to the services as new objects can easily be added in the system and operations can be invoked easily using DII. On the other hand, the provisioning of DSI allows modifications (construction) of a dynamic server object at run time. The event and notification service makes the configuration task easier for dynamic environments. The use of DII and DSI also provides benefits of a smaller memory foot print since they only require a ***single*** interface for all the invocations when compared to the static mode, which requires a separate stub and skeleton per interface. The CORBA framework saves a programmer from worrying about the basic services for distributed objects (directory, security, transactions, persistence, concurrency, event) because CORBA provides many object services that are common and useful for most distributed objects and applications. OMA standardizes the component's interfaces that enable development of a PnP environment. The clients see only the server object's interface and not its implementation, allowing the substitutability (easy upgrade) of the server implementation. The client and server components are highly de-coupled, involving three intermediate elements -- IDL stub on the client side, one or more ORBs, and an IDL skeleton on the

server side. This isolation provides a great amount of flexibility. The CORBA framework may also contribute significantly to other Network Management applications [7], [8]. Furthermore, any legacy network management application can easily be ported to the CORBA environment by giving it an IDL interface and wrapping it inside a thin CORBA object. CORBA 3.0 specifications include feature of object mobility (object-by-value) within or between ORBs. It will provide the kind of flexibility that is often found in mobile agent frameworks.

Disadvantages

Since CORBA is a complex and expensive framework, therefore, the requirement of ORB support on each component may not be acceptable for smaller devices. CORBA is not the best solution for real-time, embedded systems due to its large footprint and performance issues. Therefore, if an ORB is to be burnt into a chip, it has to shed much of its functionality. This raises the issue of trade-off between simplicity and power. It is to be noted that CORBA 3.0 specifications are making an attempt to define a Minimal/Embedded CORBA. Though CORBA specifications cover many services, most of the vendor's ORBs support only the major and essential services. Dynamic invocations (DII and DSI) are complex mechanisms and not all ORBs support them. Furthermore, the DII and DSI require more effort and coding on the part of the programmer. This approach still requires some expertise from the administrator for maintaining its ORB. Since the PnP interfaces should be standardized and supported by all components, a separate standards group is needed for the PnP approach. The existence of some proprietary protocols among certain CORBA elements (mostly at the server object side) makes portability of code from one ORB to other doubtful.

7 Conclusions

This paper has proposed a plug-and-play architecture for network services based upon mobile code and CORBA. The proposed architecture has a considerable number of advantages when compared to the Java-centric solutions described in [12], [13] and [14] owing to its language independence and extensive standard specifications. While a security service has been defined for CORBA, it is complex and, as a result, few implementations have been forthcoming. Several of the other services required by a CORBA-based PnP system are also not widely implemented. Naturally, the most important obstacle to deployment of architectures such as reported here is in the area of standardized interfaces; an area that falls outside of the research reported here. It should be stressed that an implementation of this architecture – in its most general form – would require the object-by-value message passing mechanism only available in version 3.0 of the CORBA specification.

The essential plug-and-play ideas reported here are being applied in the area of advanced services for IP telephony [18] using concepts of mobile agents and Jini. Future work will explore the CORBA based approach described here for these services.

Acknowledgements

The research reported in this paper was supported by Communication Information Technology Ontario (CITO) and the National Science and Engineering Council (NSERC).

References

[1] Bieszczad, A., White, T., Pagurek, B., "*Mobile Agents for Network Management*". In IEEE Communications Surveys, September, 1998.

[2] Bieszczad A., Raza S. K., Pagurek B., and White T.; *"Agent-based Schemes for Plug-And-Play Network Components"*; in the proceedings of an International Workshop on Agents in Telecommunications Applications (IATA'98); July 1998; Paris, France; available at URL: http://www.sce.carleton.ca/netmanage/publications.html.

[3] Carzaniga, A., Picco, G. and Vigna, G; *"Designing Distributed Applications with Mobile Code Paradigms"*; in Proceedings of the 19th International Conference on Software Engineering, pages

22-32; May 1997; Boston, USA; available from URL: http://www.elet.polimi.it/~carzanig.

[4] Chess, D., Harrison, C. and Kershenbaum, A.; *"Mobile Agents: Are they a good idea?"*; IBM Research Report; RC 19887 (88465); Intelligent Agents Project at IBM T.J. Watson Research Center; available at URL: http://www.research.ibm.com/iagents/paps/mobile_idea.ps.

[5] Fuggetta, A., Picco, G. and Vigna, G.; *"Understanding Code Mobility"*; in IEEE Transactions on Software Engineering, vol. 24, no. 5, pages 342-361, 1998; available at URL: http://www.polito.it/~picco/.

[6] Halfhill, T.R.; *"Transforming the PC: Plug and Play";* in Byte Magazine issue of September 1994; available at URL: http://www.byte.com/art/9409/sec7/art1.htm.

[7] Mazumdar, S.; *"COSS based Common Management facilities for SNMP"*; a proposal for extensions of CORBA object services, Bell Labs.; January 1998.

[8] Mazumdar, S.; *"Mapping of Common Management Information Service (CMIS) to CORBA Object Services"*; Bell Labs.; September 1996.

[9] Microsoft Corporation; *"Microsoft Windows and the Plug and Play Framework Architecture"*; March 1994; available at URL: http://www.microsoft.com/win32dev/base/pnp.htm.

[10] Object Management Group; *"CORBA services: Common Object Services Specification"*; December 1998; available at URL: http://www.omg.org/library/csindx.html

[11] Object Management Group; *"The Common Object Request Broker: Architecture and Specification"*; CORBA/IIOP 2.2, February 1998; available at URL: http://www.omg.org/library/c2indx.html.

[12] Raza, S. K., Bieszczad, A.; *"Network Configuration with Plug and Play Components"*; to be presented at the Sixth IFIP/IEEE International Symposium on Integrated Network Management, May 1999, Boston, USA; full text available at URL: http://www.sce.carleton.ca/netmanage/publications.html.

[13] Raza, S.K., *"A Plug-and-Play Approach with Distributed Computing Alternatives for Network Configuration Management"*, M.Eng. thesis, Carleton University, April, 1999.

[14] Raza, S.K., White, T., and Pagurek, B., *"Network Service Configuration with Jini: A Plug-and-Play Approach with Distributed Computing Alternatives for "*, Submitted to ASA/MA '99.

[15] Sun Microsystems Inc.; *"Jini Architecture Specification"*; available at URL: http://www.sun.com/jini/specs/.

[16] Sun Microsystems Inc.; *"Jini Architectural Overview"*; technical white paper available at URL: http://www.sun.com/jini/whitepapers/.

[17] Susilo, G., Bieszczad, A. and Pagurek, B.; *"Infrastructure for Advanced Network Management based on Mobile Code"*; in Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98; February 1998; New Orleans, USA; available at: http://www.sce.carleton.ca/netmanage/publications.html.

[18] Tang, J., White, T., and Pagurek, B., *"Advanced Mobile Agent Architectures for H.323 IP Telephony"*, submitted to MATA '99.

[19] Vinoski, S., "*New Features of CORBA 3.0*"; Communications of the ACM, Vol. 41, No. 10, pp. 44-52, October 1998.