PLUG-AND-PLAY NETWORK SERVICE CONFIGURATION USING CORBA

Syed Kamran Raza, Bernard Pagurek, Tony White

*Dept. of Systems & Computer Engineering, Carleton University, Ottawa, Canada
skraza@nortelnetworks.com, {bernie, tony}@sce.carleton.ca*

Key words: Plug-and-Play, Service Provisioning, Configuration Management, CORBA

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 utilisation 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

Configuration of network services is a difficult problem, which requires considerable human involvement. As the installation and configuration of a PC peripheral was made easier through the introduction of Microsoft's and Intel's jointly proposed plug-and-play (PnP) approach [3], [4], the network service configuration problem can also be facilitated by adopting a plug-and-play approach.

The concept of PnP from the PC world, however, requires an acknowledgement of added complexity seen in networks because there is no centralised OS available in networks that could handle the auto-installation

---

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35581-8_35](https://doi.org/10.1007/978-0-387-35581-8_35)

scenario. Furthermore, maintaining a registry (like Windows) of all the network components/services with their interaction schemes is almost impossible. Finally, provisioning a network service is usually more complicated than PC driver installation.

Technologies are only now becoming available that could be used to support a plug-and-play capability in networks. For example, CORBA [6] has introduced naming and trader features [5] that provide for service discovery, thus allowing potential clients of a service to locate a service entity. Furthermore, CORBA 3.0 [13] offers a form of code mobility through exploitation of passing object-by-value that would facilitate the deployment of client-side code on the client device thus enabling service utilisation.

The research reported in this paper is part of an overall effort to evaluate recent emergent technologies and their 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 [10] and mobile agents [7]. Further, [9] presents an evaluation of these distributed computing technologies for network service configuration. A CORBA evaluation is important in that mobile agents and Jini [11] represent (largely) Java-centric solutions; interoperability thus suffers. A CORBA approach therefore promises to provide language independence and interoperability with legacy components.

2. MOTIVATIONS AND OBJECTIVES

The services over a typical network are usually installed on a server and shared among clients. The servers usually provide access to some resource on the network such as a printer, storage disk, or router. Often, there is a need for installation of corresponding software on the client machine for that resource. We term these resources and their associated software as *service* and *service module* respectively.

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 set up and configure in networks.
- To provide continuous supervision of the networks for service provisioning due to networks' dynamic nature.
- To minimise the interactions between a user and a network component during its installation.

- To minimise network administrator intervention in a component's configuration.

We propose a Plug-and-Play 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. Although, a network printer is chosen to represent a service provider component for describing the network PnP framework in forthcoming sections, more general plug-and-play services can easily be accommodated within the framework presented here. Classical object-oriented analysis and design methodology has been adopted for defining our PnP system.

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. On inspection, we discover five major actors that interact with our framework -- *Vendor*, *Network Administrator*, *Service provider network component*, *Client network component*, and *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. Finally, the broker serves as glue between clients and server components and participant components find each other using this broker.

3. REQUIREMENTS AND FACILITIES

3.1 Requirements

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

- Auto installation of client and server components
- Discovery-and-Join of the network by participant components
- Service provisioning by PnP component
- Service discovery and utilisation by client components
- Service upgrades

- Dynamic configuration provision

However, installation and configuration cannot be automated completely; there are tasks that require user intervention. Examples of these tasks include establishment of a physical network connection, supply of initialisation information related to the configuration, set up and security, and monitoring of system messages, error messages and logs, etc.

After determining the major requirements of our system, we organised the requirements by assigning appropriate responsibilities and functionality to each actor [8]. The assignments also led us to discover essential interactions among components.

3.2 Facilities

After performing more analysis, we also determined various facilities that are required to architect our PnP system. These major facilities, in summary, are Naming/Trading, Security, Notification (events), Persistence (storage), Transaction, and Logging.

3.3 Component Interfaces

The framework consists of many components that need to know each other's interfaces to collaborate for achieving the PnP functionality. Therefore, we also defined necessary interfaces (*PnP*, *Client*, and *Vendor* interface) for the main participants. A configuration interface is also defined, which is used by an administrator to configure the component. 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

To solve the problems of application interoperability and portability with increasing heterogeneity in today's networks, the Object Management Group (OMG) has specified a distributed object framework, called the Common Object Request Broker Architecture (CORBA) [6]. The key feature of CORBA is its cross-platform communication architecture, which enables definition, transportation, implementation, and invocation of objects in virtually any programming language. All these features make CORBA a powerful enabling technology for network management purposes.

CORBA specifies various common object services [5] that are very useful for distributed computing. Further, those aspects of CORBA that promote interoperability are also quite important.

A high-level view of the CORBA architecture provides three main components -- a client, an object implementation (server), and an ORB. The *ORB* is the middleware component that acts as a broker and glues clients and servers together. The services provided by an object are expressed in the form of an *interface*, which is defined using the Interface Definition Language (IDL). CORBA defines two modes for the invocation of requests - static and dynamic invocation. Static invocation uses statically defined IDL information, compiled into two parts -- a client side *stub* and a server side *skeleton*. Some applications, for which priori knowledge of IDL (stub/skeleton) is not possible, require a dynamic invocation approach to interact with other applications. Furthermore, it is desirable to discover and use the new objects as soon as they are added to any ORB. CORBA provides these features through its Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI).

5. CORBA BASED PLUG-AND-PLAY APPROACH

The CORBA-based PnP approach is a distributed application that realizes PnP participant components as CORBA objects. These components are distributed over CORBA-enabled networks, where an ORB acts as the underlying communication structure and middleware component, thus providing appropriate transparencies. Essential CORBA object services (Naming/Trading, Security, Events, etc.) and other ORB elements (Interface Repository, Implementation Repository, etc.) are also provided on ORBs.

5.1 Main Building Blocks

Figure 1 illustrates the deployment of the CORBA-based PnP approach. Before describing the PnP scheme, we describe the major building blocks.

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 and communicate using the Internet Inter-ORB Protocol (IIOP) for enabling remote bindings.

Participants as CORBA Objects

Every hardware component (PnP and other network components) has appropriate ORB functionality pre-installed to function as a CORBA object and has a unique Object Reference (OR), which is assigned by its host ORB.

Figure 1. CORBA based PnP Approach

The underlying ORB and its directory service play the role of a broker actor.

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 are used to define the required contract/agreement among the clients and server objects. The OMG-IDL can be used to write the PnP interfaces in CORBA. An example PnP interface, written in IDL, is shown in Figure 2. This interface is implemented by a vendor component, helping the PnP components in (de) registering with their vendors.

```
module PnP {  
    struct ComponentInfo {  
        string name;  
        string owner;  
        string address;  
        ....  
    };  
  
    interface vendor {  
        readonly attribute string vendorName;  
        ....  
        boolean register(in ComponentInfo pnpce) raises SecurityException;  
        void deregister(in long long registrationID);  
    };  
};
```

Figure 2. Example PnP Interface (in IDL)

Interface and Implementation Repositories

Since all CORBA based invocations (static or dynamic) require IDL interfaces, it is required that both the vendor's and enterprise's ORB maintain an interface repository, which is accessible by other ORBs as well. An Implementation Repository 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.

5.2 Useful CORBA Services

This PnP application benefits heavily from CORBA's object services. Although CORBA specifications describe 16 common services [5] 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

CORBA provides two directory services -- Naming and Trading – that can be used by the clients for locating the objects across the ORB or network. The Naming service provides a mapping from a name to an object reference but it does not fulfil all the requirements in dynamic environments. For instance, a client component looking for a printer service that may print colour and double sided at a speed of 10 pages/min requires a sophisticated directory service. 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.

It is required therefore, that both vendors and the enterprise network run a Trading service. The components first obtain the OR of the trading service through the ORB-interface. Using this OR, a PnP component advertises its service, while the client components discover the services they require.

Security service

CORBA provides a detailed specification for a Security service. The service specifies the mechanisms that provide end-to-end security in multi-vendor object networks through appropriate security interfaces for the authentication, access control, recording audit events, and security administration. These features satisfy our PnP security requirements.

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' -- meaning that a change in an object (supplier) generates an event that is propagated to all interested parties (consumers) using another CORBA object, called the *event channel*.

Property service

CORBA's Property service provides mechanism for associating the properties (i.e., <name,value> tuples) dynamically with an object. This feature helps in supplying initial configuration parameters to components.

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.

5.3 Scheme Description: Step by Step

Before describing the interaction steps and scenarios of the PnP scheme, the scope of CORBA needs to be clarified as follows.

The CORBA framework is used only for facilitating the installation and configuration steps of a PnP component. Once the corresponding service modules are installed and activated on a client component, the client components use the required service directly over the network without any involvement of the CORBA framework. For example, the PnP service configuration for a printer uses the CORBA framework, its transport, and its services to arrange the printer device driver on the client components. Later on, the client components communicate directly with the printer component over the network through the installed printer driver for printing.

Although this PnP approach does not mention security explicitly, it is assumed that all the sensitive interactions use CORBA's Security service.

Component's layout

Consider a heterogeneous network, which is CORBA-enabled. The participant components are connected over a common ORB object bus and the ORB is running the CORBA object services that are required for this application. These services are accessible to every component through the standard interfaces and include Naming, Trading, Security, Event, Property, and Persistence. Moreover, the other necessary ORB elements like Interface Repository and Implementation Repository are also available to the components.

Since the responsibility of maintaining the network lies on the shoulder of a network administrator, an administration application is running on the

enterprise network (as well as on the vendor's) that the administrator may use to configure the network components.

PnP component's installation

Now assume that the administrator realises 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. The component registers its service with the Trader, implementation with the Implementation Repository (using a POA), and interfaces with the Interface Repository.

Vendor's set up

The PnP component's vendor also maintains an ORB for its product support. This 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 implements standard interfaces and is the main access node for vendor-manufactured PnP components. A PnP component, however, requires knowledge of its vendor 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 IORs, a component's reference remains valid even after its location changes.

PnP component's registration with its vendor

After executing an initialisation 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 hard-coded information. This registration interface may be proprietary because both the participants belong to the same vendor. The request goes through the component's ORB and is forwarded to the vendor ORB, whereas the vendor component object implementation is contacted through the Implementation Repository and the registration is completed. After registration, the PnP component may also invoke a standard interface method to download the latest server-side module and upgrade itself.

PnP component service advertisement and client's requirements

After the successful registration, the PnP component is ready to offer its service to potential clients. There are two modes for service provisioning -- *Push* mode in which the PnP component initiates the service offer and discovers its clients, and *Pull* mode, where a service provider PnP component allows interested clients to discover its service. These two modes can be incorporated in a CORBA-based approach as follows.

Assuming the Push mode first, a question arises immediately: "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. Then the PnP component queries the Trading service and makes a list of all the objects (i.e., its potential clients) that have shown an interest in its service.

The Pull mode, however, 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 the service is not found, clients need to wait until such a service registers with the trading service. The CORBA's Event service is used to provide a 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. Since the clients now have discovered their required service provider component, they can invoke a standard registration interface on the PnP component. The PnP component waits for a reasonable amount of time (depending on network latency, 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 (or discovered) client to retrieve the client's information. 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 requirements list.

Service module distribution and activation on clients

At this stage, the clients are ready to receive a service module. The PnP component invokes 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. 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 2.0 framework is that it only supports the option of passing object-by-reference and not “by-value” in method invocations, so we need CORBA 3.0 [13] which incorporates this object by-value (OBV) passing feature. 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.

In Push mode, a PnP component should poll the Trader service from 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 on arrival of a new interested client at the Trader service.

Future configuration support is implemented differently for the 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. If no PnP component is found by querying the Trader service, the client uses an event registration mechanism with the PnP event channel to receive notifications on availability of such a PnP component.

Service module upgrade

A major requirement in our PnP approach relates to the upgrade of service modules. Since upgrades may arrive any time, an event notification mechanism is needed. A vendor notifies all the registered PnP components 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

the concerned client components (configured previously) need to be upgraded. The PnP component may query all the affected client components for their interest in the upgrade by invoking a standard PnP interface. For all those clients that have shown interest in the upgrade, the PnP component repeats the steps of module distribution and installation.

6. EVALUATION

6.1 Advantages

The CORBA environment suggests an architecture that runs on every platform. Therefore, the CORBA based approach provides a portable, object oriented and inter-operable 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 to 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) by providing common object services. The OMA standardises 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, which provides a great amount of flexibility. 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. Finally, CORBA 3.0 specifications include the feature of object mobility (Object-By-Value) within or between ORBs that will provide the kind of flexibility that is often found in mobile agent frameworks.

6.2 Disadvantages

Since CORBA is a complex and expensive framework, 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 a trade-off between simplicity and power. CORBA 3.0 specifications are also 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, not all ORBs support them, and they 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 standardised and supported by all components, a separate standards group is needed. The existence of proprietary protocols among certain CORBA elements (mostly at the server object side) makes code portability from one ORB to another questionable.

7 CONCLUSIONS

The proposed CORBA architecture has a considerable number of advantages when compared to the Java-centric solutions described in [7], [8] and [10] 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. The most important obstacle to deployment of architectures such as reported here is in the area of standardised interfaces. 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 [12] 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 Communications and Information Technology Ontario (CITO) and Natural Sciences and Engineering Research Council (NSERC), Canada.

REFERENCES

- [1] Bieszczad, A, White, T., and Pagurek, B., "*Mobile Agents for Network Management*"; IEEE Communications Surveys, Sep. 1998.
- [2] Bieszczad A., Raza S. K., Pagurek B., and White T.; "*Agent-based Schemes for Plug-And-Play Network Components*"; Proceedings of the International Workshop on Agents in Telecommunications Applications (IATA'98); Jul. 1998; Paris, France
- [3] Halfhill, T.R.; "*Transforming the PC: Plug and Play*"; Byte Magazine, Sep. 1994; URL: <http://www.byte.com/art/9409/sec7/art1.htm>.
- [4] Microsoft Corporation; "*Microsoft Windows and the Plug and Play Framework Architecture*"; Mar. 1994; available at URL: <http://www.microsoft.com/win32dev/base/pnp.htm>.
- [5] Object Management Group; "*CORBA services: Common Object Services Specification*"; Dec. 1998; available at URL: <http://www.omg.org/library/csindx.html>
- [6] Object Management Group; "*The Common Object Request Broker: Architecture and Specification*"; CORBA/IIOP 2.2, Feb. 1998; available at URL: <http://www.omg.org/library/c2indx.html>.
- [7] Raza, S. K., and Bieszczad, A.; "*Network Configuration with Plug and Play Components*"; Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99), May 1999, Boston, USA; full text at URL: <http://www.sce.carleton.ca/netmanage/publications.html>.
- [8] Raza, S.K., "*A Plug-and-Play Approach with Distributed Computing Alternatives for Network Configuration Management*", Master of Engineering thesis, Carleton University, April 1999; Ottawa, Canada.
- [9] Raza S. K., Pagurek B., and White T.; "*Distributed Computing for Plug-and-Play Network Service Configuration*"; Submitted to IEEE/IFIP Network Operations and Management Symposium (NOMS'00); April 2000; Honolulu, Hawaii.
- [10] Raza, S.K., White, T., and Pagurek, B., "*Network Service Configuration with Jini: A Plug-and-Play Approach with Distributed Computing Alternatives*" Available at: <http://www.sce.carleton.ca/netmanage/publications.html>.
- [11] Sun Microsystems Inc.; "*Jini Architecture Specification*"; available at URL: <http://www.sun.com/jini/specs/>.
- [12] Tang, J., White, T., and Pagurek, B., "*Advanced Mobile Agent Architectures for H.323 IP Telephony*", To be presented at the workshop on Mobile Agents for Telecomm. Applications (MATA'99); Oct. 1999, Ottawa, Canada.
- [13] Vinoski, S., "*New Features of CORBA 3.0*"; Communications of the ACM, Vol. 41, No. 10, pp. 44-52, Oct. 1998.