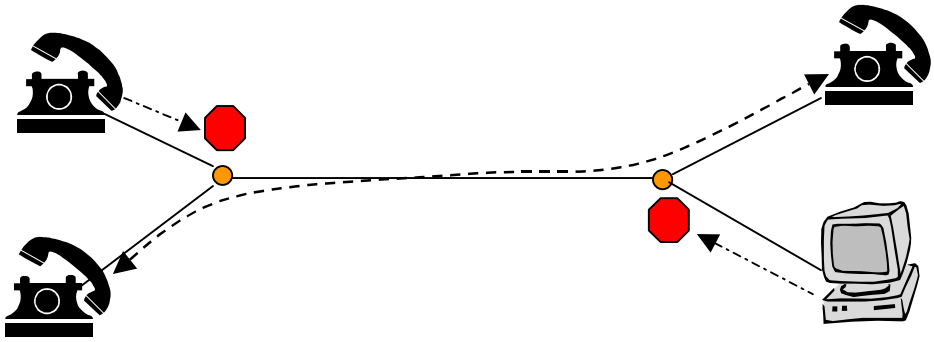
Call Processing

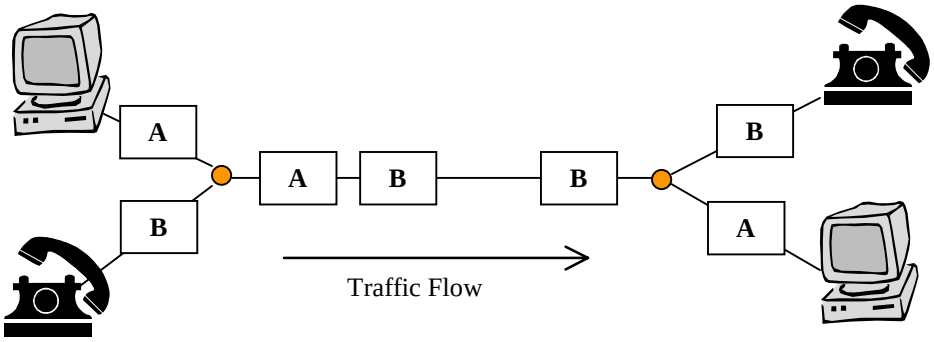
Robert Hanmer
Lucent Technologies
2000 N. Naperville Road
Naperville, IL 60566-7033
hanmer@lucent.com
+1 630 979 4786

The function of telecommunications systems is to connect two (or more) end points so that they can exchange information. There are generally two parts required to accomplish this task. The first is the actual transmission of information. Two methods are used to accomplish this: circuit switching or packet switching. The other part embodies the intelligence to determine which end points to connect and to direct the systems to make these connections. This is the *call processing*.

Traditional telephone service was created on a framework of circuit switching. In this method, a connection for information exchange is established that remains constant and fixed for the duration of the session. This locks resources into the call so that they cannot be used for other calls as long as the first call is in progress.



Modern data communication employs packet switching. The information to be exchanged is broken into small "packets" of information that are sent as they become available from one endpoint to the other. The path between the endpoints is only used when there is a packet to send. The path might change from packet to packet for the same call. Since the path is not in use continuously by the packet transmission the path can be shared by many calls simultaneously.



In either of these two schemes, the system must perform some form of call processing. Call processing includes collecting the session startup information, i.e. the address of the distant endpoint, applying any features[1] to the call, advancing this setup information to the distant endpoint (and any intermediate helpers) and establishing a connection for user information.

As the pattern that follows will show, there are a variety of different architectural configurations that can be used to support call processing. These vary based upon the number of intermediate system objects used to communicate between the parts. HALF CALL describes the most commonly used method. It is employed in many products produced by Lucent Technologies and Nortel Networks [Utas]. Gerard Meszaros published a refinement to solve the problem of "making the difference between one and multiple address spaces transparent" as HALF-OBJECT+PROTOCOL (HOPP) [Mesz].

[1] Features include such things as three-way calling, toll-free services, voice messaging, etc.

1. HALF CALL

WAC Cpl. Alma Bradley operates the switchboard at the "Little White House," the residence of President Harry S. Truman during the Potsdam Conference in Germany. USA National Archives and Records Administration. www.nara.gov

There are two (or more) parties that want to exchange information. This information exchange might be between humans as on a telephone call. It might be between objects that require some sort of message passing. The parties, human or otherwise, are not co-resident so some sort of intermediate service is required to connect them. Postal services connect the parties over vast distances, but is slow but introduce significant temporal drift. Semaphores and smoke signals are better at the temporal issue, but require the parties to be geographically close to each other. Using an electronic means to exchange information in real-time seems plausible overlarge distances. For every party that wishes to exchange information with others, we cold deploy a fully interconnected network at great expense. In the telephone case resources can be conserved by creating a system to identify the destination endpoint, determine how to communicate with it, and make the required requests on the originator's behalf to the destination so that information can flow.

In order to communicate the systems at both ends representing the "calling party" and "called party" must maintain some sort of momento that the endpoints are communicating. Thus the information exchange session, or "call", is really split between the two endpoints. The term *endpoint* will be used to refer to the customers and their external endpoints, and also to the appearances representing those external endpoints that are internal to our system.

One way of looking at the information flow at any of the intermediate systems handling the call is that they take input from one side (the calling party side), process it, and pass it to the other side (the called party side).

❖ ❖ ❖

How should the system's handling a telephone call internally be designed to correctly process requests for the call that span two different endpoints?

Rather than connect all of the endpoints together directly, add a mediating call entity in between the endpoints. At one level of abstraction this entity represents the telephone switching system itself. At a lower level it is the agent for the parties within our system.

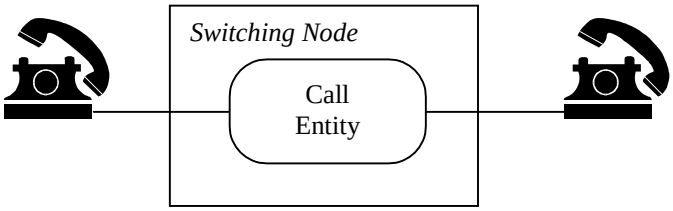
There are several ways to consider adding this node between the parties to the call. You could add one object, two objects, three objects, four objects or more between the parties. Each of these has advantages and disadvantages. Those worth most consideration are the cases of one, two and three objects. The cases of four and more can all be sorted into one of these three cases by careful grouping of the objects and their functions.

The general functions that need to be considered and mapped into the objects include these:

- Interfacing to the protocols external to the node that the endpoints use.
- Actually making decisions about how the call should be processed.
- Returning resources to the common pool at the end of the session.

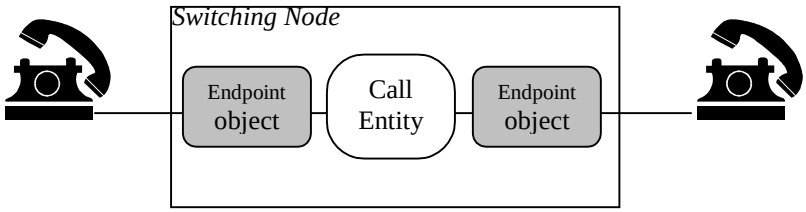
In the alternatives discussed below these responsibilities are sometimes combined into an object.

One way is to have a single call entity within the system. The endpoint interfacing will be included within this one call entity. This entity could be an object in an OO implementation of the system. All the information from the calling party side and the called party side can be used since one object knows it all.



A problem is that as the potential for different internal endpoints occur new classes that incorporate the specifics of all these endpoints are required. These new endpoints might be required because a new signaling system is created or new functionality is required at an endpoint. This greatly complicates the system. This is called a "whole call model".

Another way is to have the two endpoints as separate objects with a third object in between.

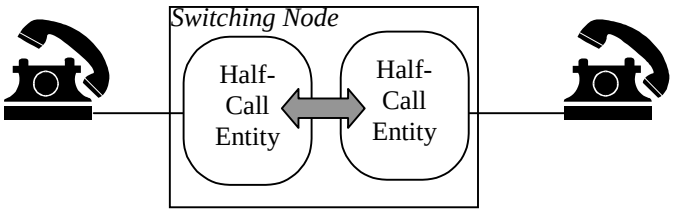


This is a recursive way of conveying the information in that it mirrors the larger system of users and a central switch or network. Objects are used to represent the endpoints, as well as the middle entity. A protocol is required between the objects to ensure that they know how to deal with stimuli from the other object. This is the most direct interpretation of a human telephone operator of days gone by. Patch cords from each endpoint represent the endpoint objects. The operator represents the third dispenser object that will process information requested from an endpoint and create a connection between that endpoint and the other endpoint.

This method has the advantages of being able to easily adapt to many different types or styles of endpoints. The coordination of the two endpoints and the central point adds to the complexity of this solution. The question arises as to whether the central object should be a SINGLETON [GOF], or a class for which there are many instances. A SINGLETON can be argued for because this central object needs to have access to all of the resources within the system. Having a SINGLETON simplifies many issues such as congestion control. A SINGLETON would be a bottleneck, as you would have to synchronize on shared data/attributes/members. Conceptually it is much easier to envision the system with separate objects for each call.

This is called a "hybrid call model". [Copl]

A middle approach is also possible. This is the "half call model". Each internal endpoint is a logical one-half of the total "call" entity object. Each one-half of the call entity also deals with the endpoint functions.



It reduces the number of classes required over the hybrid model by removing the central dispenser. It represents an increase in the number of classes over the whole call model, but adds greatly to the flexibility by allowing the endpoints to be different and customized as necessary.

The half call entities are in direct control of the resources at its end of the system, which simplifies the allocation and release of resources. It also reduces the internal communication bottlenecks. The half call entities can be created to mirror the resource classes.

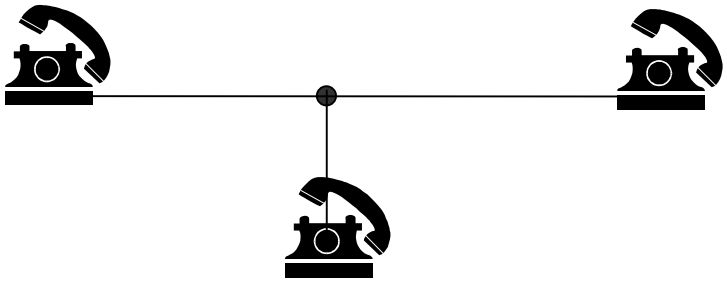
Again a protocol is required to communicate between the two endpoints models. [Mesz][Marq]

Therefore,

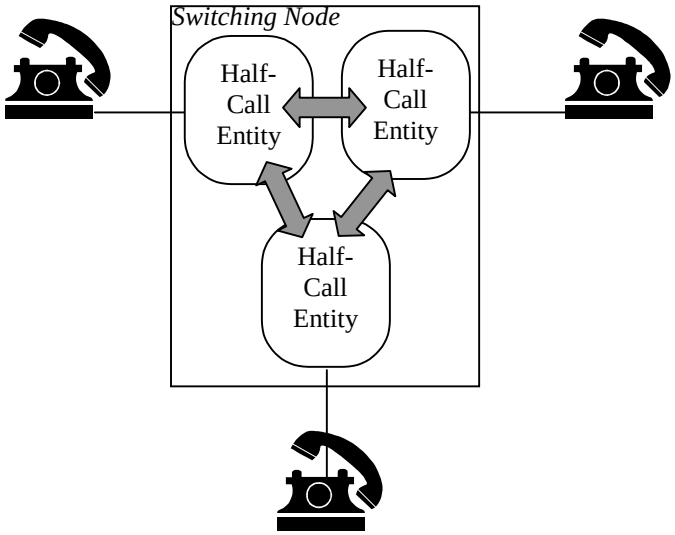
Use a half call model. It offers a balance between the monolithic whole call model and the object complexity of the hybrid model.

❖ ❖ ❖

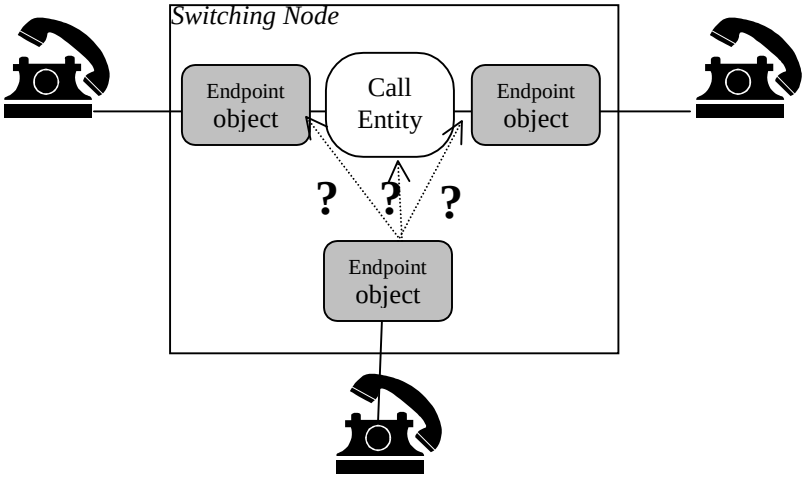
This pattern only addresses simple connections between the endpoints. What about when there are more than two endpoints?



The half call model can handle this most easily.



It seems that internally a whole call model must create a sort of half call to deal with this situation. With a hybrid call model the difficulty arises as to where the new endpoint is to be added: before, after or at the central dispenser. It raises questions about which object is in overall control.



How are features added in each of these systems?

A great number of patterns follow this pattern in the creation of a robust, feature rich call processing system.

As a network of communicating systems are created the parts must talk to each other. Consider using PROTOCOL-FOLLOWS-APPLICATION DESIGN in [Marq] to define the protocols at all levels (network and internal).

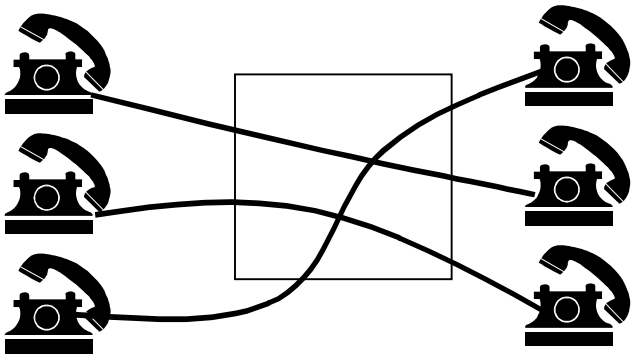
The hybrid call model is used successfully by the Lucent Technologies 4ESS™ toll and tandem switch [CH] and the Nortel DMS-100 and DMS-250 Switches [Utas01].

The Lucent Technologies 5ESS® Switch and switches from Nortel Networks use the half-call model.

2. SWITCH DATA STORE

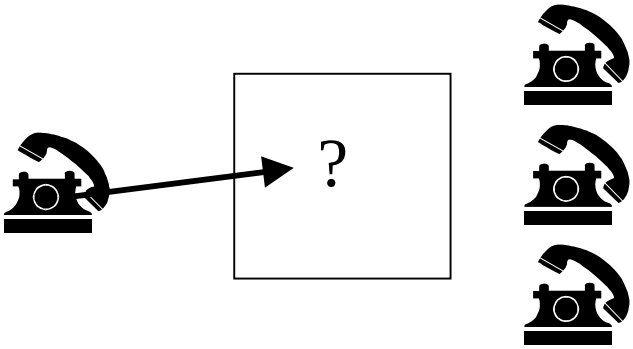
Train passengers use a bank of telephones in the 30th Street Station at Philadelphia, PA.
USA National Archives and Records Administration.
www.nara.gov

In simple (or low in the protocol hierarchy) systems there might be a permanent mapping between endpoints. If a call comes in on one endpoint, it always goes out on another.



But that isn't very useful. It also ties up resources and limits the possibilities.

What would be ideal is if the connections could be reconfigured for each call.



The previous pattern HALF CALL discusses how we can arrange the objects that will participate in a connection between endpoints within our system. It describes the statics of the system's objects, but it doesn't discuss the dynamics of an actual connection request.

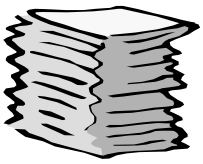
In particular it discusses the structure of the program to handle the connection request, but not the data that is required.

❖ ❖ ❖

How can the system know how to handle connections that aren't known until they are requested?

Once upon a time having an intelligent agent -- a human at the center solved this problem. Calls would arrive from the originating party and ask the human operator to speak with a specific party. The operator would look up or would know from prior experience how to connect the parties.

This prior knowledge might be in the form of a lookup of the form: to connect Frank to Jane, connect ports 3 and 14 together. As the number of possible connections grew, the amount of knowledge that is required grows.



Eventually the amount of information becomes more than the operator can handle, and a better way is needed.

The queries to the system are very consistent:

How do I connect calling party a to called party b?

The calling party might not even be described by name; the name is not important. What is important is the port through which the calling party accesses the system. This changes the query to:

How do I connect the calling party on port A to called party b?

Generally the modern telephone network identifies particular parties with a number. Usually a telephone number, such as 123 4567, or an IP address 123.12.34.567. The actual party at the end of the number may or may not be the desired party B, but the number points to a place that B has been (and is paying for).

So the query is now:

How do I connect the calling party on port A to the called party on port B?

Such a regular query can easily and quickly be put into a simple database.

But what about when customer **b** receives a new, additional telephone number or network address and is now on port **B1** in addition to port **B**? Some way of updating the database is required. Both ports should connect to pretty much the same location.

Early systems hardcoded these port translations. If a call desired number **b** it was switched to port **B** all in hardware. Eventually software replaced this hardware function.

Transactions that are required include the initial population of number to port mappings into the switch database, changing entries and deleting an entry altogether. Few other types of transactions will be required. The data needs are quite simple.

Since this database will be interrogated on every connection within the system it must return its answer quickly. So a lightweight database, perhaps custom coded, should be employed.

If the database is unavailable due to a fault, the system cannot connect calls, so the database system must have few faults. Again this argues in favor of a lightweight database system.

Install a lightweight database system that will be able to quickly and reliably decide how to connect two parties to the call.

❖ ❖ ❖

In many systems, such as at the lower level protocol routers and switches this database might be implemented in hardware. And some protocols might include all the addressing information within the contents of the message, eliminating the need for a custom database.

The database can get much more complicated as additional features such as address translations are required. An example of this is when a telephone call with one of the toll-free area codes such as 800, 877, or 888 are made. In these cases the systems processing the call request must translate the number that was entered (888 123 4567) into the number of a real telephone. [SW]

In all probability the system's database will not be populated through entirely manual actions, nor will it remain constant. The types of transactions are simple, yet the system can benefit through having a PROVISIONING SYSTEM (unwritten) that will administer the changes.

Acknowledgments

Thanks to Monica Sentoff, David DeLano, Mike Wu, Greg Utas and John Letourneau for reviewing early versions of this work. Thanks to my PLoP '01 Shepherd Manfred Lange for his valuable suggestions.

References

[CCRSS] Cieslak, T., L. Croxall, J. Roberts, M. Saad, and J. Scanlon, 1977. "No 4 ESS: Software Organization and Basic Call handling." **Bell System Technical Journal**, vol 56, no. 7, Sept, 1977: 1113-1138.

[CH] Carestia, P., and F. Hudson, 1981. "No. 4ESS: Evolution of the Software Structure." **Bell System Technical Journal**, vol. 60, no. 6, Part 2, July-August, 1981: 1167-1201.

[Copl] Coplien, J. 2001 "Worth a Thousand Words" in **Design patterns in Communications Software**, edited by L. Rising. Cambridge: Cambridge University Press.

[GOF] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. **Design Patterns Elements of Reusable Object Oriented Design**. Reading, MA: Addison-Wesley, 1995.

[Marq] Marquardt, K. 2000. "How to Define a Protocol for Object Transportation" in Proceedings of EuroPLoP 2000 conference.

[Mesz] Meszaros, G. 1995. "Half-Object Plus Protocol" in **Pattern Languages of Program Design**, edited by J. O. Coplien and D. C. Schmidt. Reading, MA: Addison-Wesley Publishing Co.

[SW] Sheinbein, D., and R. P. Weber, 1982. "800 Service Using SPC Network Capability." **Bell System Technical Journal,** vol. 61, No. 7, Part 3, Sept. 1982: 1737-1757.

[Utas] Utas, G. 1999. "A Pattern Language of Call Processing." **IEEE Communications Magazine**, vol. 37, no. 4, April, 1999: 64-69.

[Utas01] Utas, G. Personal Communication, 1 June 2001.