Inter-Office Memorandum

To	Voice Project	Date	August 31, 1981
From	Dan Swinehart	Location	Palo Alto
Subject	Control Protocols for the Etherphone System	Organization	CSL

XEROX

Filed on: [Ivy]<Audio>Docs>Thrush.memo

Introduction

This document represents the protocol design as of mid-August, 1981. Subsequent developments will appear in later versions.

This section describes the conceptual structure of the Etherphone system (the architecture as viewed by the software.) It begins by describing the basic entities in the system and the nature of their interaction. As much as possible, this design is independent of the assignment of these entities to hardware components and the actual communication paths that are employed; we envision a redistribution of responsibilities once we have gained some experience.

Given the structure, we will identify a number of important interfaces, and describe the functionality of the system in terms of the operations that these interfaces supply. Finally, there is a discussion of the intended assignment of functions to machines and of the low-level communications mechanisms that will be used to implement the interfaces.

Goals

A review of some of the goals of this project will help put the design in perspective. An overriding goal is to produce a telephone (etc.) system that we can have full control over, and in the manner to which we have become accustomed. This means that it must be possible for a programmer to write an application that will run on a workstation in our internetwork, with as much access to and control over our voice capabilities as it needs. Of course, we should attempt to present these capabilities to such a programmer in as uncluttered a form as we can manage.

On the other hand, we intend to replace existing telephones with our gadgets. This places availability and reliability constraints on the design that can only be met of if the phone system can operate independently from individual workstations. We have described in other sections the hardware ramifications of this requirement. Both those hardware decisions and the realities that led to them also have an effect on the software architecture.

In summary, then, our system will have *clients* as well as *users*, and we have to deal with that.

Basic Entities

We have found it convenient to think and talk about the software architecture of this system using the object style typified by <Smalltalk, etc.>. In this case, there is a small number of basic kinds of objects, or classes, instances of which communicate with other objects of their own and different classes. There may be but one instance of some classes, but in general there are many. Some of the classes are further specialized into subclasses which deal with variations of whatever basic themes their classes represent. For most purposes, it is reasonable to think of objects as Mesa program instances (global frames) that communicate with other objects by calling procedures in the interfaces supplied by those program instances.

To motivate the choice of object classes in the current design, consider the following, relatively disjoint, kinds of activities and responsibilities:

1. The basic purpose of the system is to provide a variety of telephone and other voice communications facilities. There must be components of the system which know about telephony:

    Name to number to hardware-specific mappings;

    Interpretations of addresses in terms of the network connectivities required to establish conversations;

    The nature of the call placement, connection, and maintenance activity;

    The meaning and implementation of features like call forwarding, conferencing, call recording, retry, etc.;

    The novel negotiation and filtering techniques that we intend to experiment with;

    Traffic and load management and instrumentation.

   These are the kinds of capabilities one would expect to find in any computer model of a telephone network; they can be described, and in some sense implemented, without detailed knowledge of the specific hardware and interfaces that the user sees. These capabilities are also representative of the kinds of facility that the CSL voice project is interested in implementing.

2. The clients of this general telephony knowledge are the specific implementations of the applications and user interfaces (digital and analog) that make up the voice communications system: workstation applications (for individuals and attendants), stand-alone telephone behavior, voice file server implementations, etc. These applications must be provided with an interface to the telephony "model" on the one side, and with the specific hardware on the other. They will determine the external appearance of the system. We will implement some of these facilities as part of the voice project; other people will produce additional applications at this level.

3. We have decided to minimize the size and complexity of programs in the Etherphone processor, assigning to other server processor(s) the responsibility for interpreting its control inputs and deciding its sequence of control outputs. It is therefore necessary to define the interface that will be used by this remote intelligence to communicate with the primitive Etherphone capabilities. Eventually, there may be a number of implementation-dependent interfaces.

Based on this taxonomy and the reasons behind it, we have produced the conceptual system architecture depicted in Figure 1. The labelled items represent objects whose classes are suggested by their outline shapes. The labelled lines, of varying thickness, represent the interfaces that these objects present to their various clients.

Fones

The circular objects are instances of the class *Fone*. Each object represents an individual or other entity that is or could be a party in a voice conversation. The abbreviation below the horizontal bar in each Fone identifies that Fone’s subclass. Thus there is a Fone[IND] for every individual (identified by Grapevine RName) within the system who has an active Etherphone. In addition, there is a Fone[TRK] for each available conventional telephone trunk connecting the Etherphone system to the public switched network, other PBX lines, etc. The figure depicts a third kind of Fone, Fone[REC], representing the voice file system’s involvement in recording a particular conversation.

Net

The triangular object represents a collection of objects and activities that will be further elaborated as the design progresses. It is supposed to represent the capabilities for registering and creating new Fones and other more numerous objects, for managing various name/address databases, for observing and controlling overall network traffic, and in general for whatever other truly shared or centralized concepts need to be represented. In an internet with multiple servers, each server machine will need its own Net object, so even here we must allow for the various kinds of distribution and replication that will result.

Smarts

The square boxes depict objects that implement the actual applications, or "smarts", in the system. This is the trickiest object class to motivate or describe; a substantial portion of the rest of this memo is devoted to doing that. For now it suffices to state that there is a Smarts object, residing somewhere in the system, for every distinct application or user interface that provides components of or interacts with the voice services. Thus, both because there are some voice functions (e.g., audible ringing and providing actual voice converations, stand-alone call placement and reception) that only the Etherphone processor itself can do, there is a Smarts object (Smarts[EP_x] for various x) providing or supporting these functions. But because of our critical goal of allowing workstation participation, there is also a Smarts object (Smarts[WS_y]) to represent the workstation implementation. Similarly, other Smarts objects provide the "intelligence" for such facilities or functions as outside telephone lines (trunks) and voice recording facilities.

Etherphones

The rectangles named EP_1, EP[TRK_1], etc. in the picture, merely represent the actual implementations of the simple programs, residing on the Etherphone processors, that provide the basic hardware control and voice transmission. These EP objects by themselves do not interpret user actions or understand how to participate in telephone activity.

Protocols (Interfaces)

It is hard to ascribe any value to this assignment of functions to objects without an understanding of the interfaces between the objects. In fact, a realization of the various kinds of interfaces that would be suitable for this system preceded this particular choice of object classes. The Fone, Smarts, Net, and EP objects were originally designed to provide reasonable places to put these interfaces. A small number of refinements later, the objects themselves began to make a good deal of sense as a way to factor and structure the system.

The most important requirement was that workstation-based client programs could play an active role in the user’s telephone dealings by, in effect, programming in a very high level telephony language. Short of actually implementing a new language, it is of course simpler to provide a collection of procedures, operating on data types that capture the appropriate level of abstraction. In the Etherphone system this high-level client to phone system interface operates exclusively between Smarts objects, representing the client applications, and Fone objects, representing the telephony model. In the figure, these interfaces are labelled "C".

Similarly, another interface, distinct from the high level is needed to convey user actions from Etherphone processors to their remote intelligence, and vice versa. These communications paths are denoted by the label "P" in the figure.

Communications between the Fone objects and between Fones and the Net are required to set up, take down, and monitor the progress of telephone calls, dictation sessions, etc. The diagram indicates by connecting lines the objects that would have to interact in order for the user of EPhone1 to place and record a call to the user of EPhone2. These interfaces, labelled "F", are private to the Fone/Net implementations; Fone clients do not need to know what they are.

Wherever network communications are required these interfaces must be expressed in terms of protocols that deal properly with the communications problems while expressing the right semantics. It is our current intent to employ the Remote Procedure Call (RPC) methods being developed for Cedar. Thus all of the interfaces can be expressed as procedural interfaces, whether or not they span machine boundaries. This approach will require careful attention to the process structures of the machines that comprise the system. That work remains to be done.

One!Many Mappings

A number of activities require the participation of both the workstation and an associated Etherphone. Examples include:

> Placing calls -- the workstation initiates the activity, but the Etherphone must perform the actual voice transmission. It may also be called upon to generate call progress tones, etc. In addition, it has to keep track of the user's switchhook.
>
> Receiving calls -- the workstation Smarts may choose to be involved in the filtering and information flow that accompanies an incoming call. In particular, for calls forwarded to a central position, the attendant's workstation will perform a crucial role. We will probably discover many other possibilities.
>
> Messages, transcription applications -- the workstation will of course be in control.

This multiple participation is indicated in the figure by the appearance of connections between a given Fone objects and a number of Smarts objects. Rather than anticipate all the interactions that might be desirable, we've settled on a simple scheme for managing this multiplicity. It will be very nice if it works.

In the Smarts!Fone direction there's no problem: a Fone will accept or deny a request based on the current state of the system. Going the other direction, there is an ambiguity about which of the Smarts should handle each activity. The proposed scheme is to register each of the Smarts for a Fone in a list maintained by the Fone, and in a precedence order. Each entry but the last is permitted to handle or pass on each request; the last must be willing to handle all requests (at least by firmly rejecting them.) A Smarts that passes on a given request may still choose to take some application-dependent action based on the request (posting the caller's name on the screen, for instance.)

If two Smarts have the same priority, the requests will be issued to each simultaneously, and the earlier respondent will win. This allows for the appearance of an individual's telephone line in more than one location. Such an individual has one Fone, connected to a Smarts representing each instrument bearing his line (e.g., in his office and in his laboratory).

Possibly there will have to be a different priority ordering for different groups of Fone!Smarts requests; preferably not.

The lab phone situation also introduces the need for a multiplicity of Fone connections from a single Smarts object. Such a telephone may represent a number of different individuals. All of this resembles what happens at more complex attendant locations, and still needs to be worked out.

Mini-Scenario

A later section (to be written) includes detailed scenarios with sample uses of most of the protocol design. Here is a simple, high-level description of how a simple "stand-alone" call might be placed, maintained, and terminated. Assume that the system of Figure 1 has been initialized and contains the objects pictured there:

Cohen, intending to call *Jones*, lifts the handset of her telephone. EP1 detects the action and forwards the offhook indication to Smarts[EP_1]. The Smarts instructs EP1 to issue a dial tone. When *Cohen* pushes "4", EP1 forwards the "4" to its Smarts, which responds by ordering EP1 to silence. After "4977" has been entered and forwarded, EP1's Smarts asks Fone[IND_1] to place the call. The Fone consults the Net to obtain the RName "Jones.PA" matching the phone number "4977", and a handle for Fone[IND_2], the representative for Jones.PA. Negotiations between the Fones determine that Jones is in to callers, and they agree to set up the call. Fone[1], via Smarts[1], instructs EP1 to provide a "ringing" tone to Cohen. Fone[2], via Smarts[2], instructs EP2 to ring its telephone. When *Jones* lifts the receiver, Smarts[2], Fone[2], Fone[1], and Smarts[1] find out about it, in various ways. A set of socket numbers identifying the conversation is distributed to the EP's, and they converse. The Fones register the conversation with the Net, which uses the information to monitor Ethernet traffic.

As the conversation progresses, the Fone objects monitor each others' status. Each will terminate the conversation if it detects any uncorrectable anomaly in the other (e.g., no response to the query.) Normally, though, the conversation will be explicitly terminated when one party hangs up and the change in switchhook state progresses through the system.

Intial Architecture: Thrush and Etherphone 1

It seemed important (at the time) to describe this architecture in the absence of specific assignments of functions to machines. In fact, the architecture should survive a number of reassignments that we contemplate making over time. But we have of course chosen an initial system configuration; Figure 2 is an augmented version of Figure 1, depicting the proposed setup.

The large central box is the Etherphone server (its program is named *Thrush*). The Thrush server provides the entire implementation for Fone and Net objects -- the network model. In addition, it is the current site for the "Smarts" for the Etherphones. These Smarts provide stand-alone Etherphone functions as well as the ultimate interpretation of workstation requests that must be satisfied by the Etherphones.

As we have said, the initial system will not include a separate server connecting to outside telephone lines, or trunks; instead, each Etherphone will have a "back-door" connection to the existing Centrex telephone line for that office. However, to indicate that this connection is in principle entirely separate from the the local telephone instrument and the Ethernet connection, and will in fact eventually be concentrated in separate servers, we have explicitly separated them in the design, by providing independent Smarts and Fone objects for the back door connections. The Etherphone will also deal with them independently.

The Etherphone processors, will implement the EP objects, using RPC communications (in both directions) to obtain the wisdom of their Thrush-based Smarts. We will write Etherphone programs of this kind for both the initial Alto I Etherphones and for the later microprocessor-controlled systems.

The workstation-based smarts, reside, naturally enough, in the workstations. They comprise the realization of customized calling and answering capabilities, powerful attendant features for outside, unanswered calls, voice document annotation systems, and the like.

Provisions for multiple Thrush servers

If we are to have any confidence of achieving our reliability requirements (basic telephone service always available), we will need more than one Thrush server. What sounds best at present is a model similar to the Grapevine server model: there is a Thrush server at each campus (or perhaps on each Ethernet in a large campus), which serves as the primary server for Etherphones in its locale. Another server can provide service to an Etherphone when its primary server is broken (rejecting or not responding.) Handling conversations that take place between Etherphones with different primary servers will require the participation of both, to an extent and in a manner yet to be determined (agent Fones representing the other end in each server? Net to Net communication?

direct Fone to Fone communication via cleverly arranged Remote Procedure Call?)

When a server goes down, we will attempt to avoid terminating the conversations that it was managing. Instead, the Etherphones will frequently reassure themselves that their (Thrush-based) Smarts are still functioning, searching frantically for new ones if they are not. They will provide enough information in the process to allow the new (or resurrected) server to rebuild some sort of model of the ongoing conversation.

Specifics of Protocols

These aren't done. They aren't even right any more. They are representative of the kinds of things going on at each level.

Notation

We have decided to base our control protocols on the Remote Procedure Call (RPC) methods being developed for Cedar. The idea is that of a simple packet exchange, simulating a procedure call and its return. The packet exchange will comprise a pair of packets of the following sort:

```
PUP[code, ID (sequence #), sourceSocket, destSocket, data]
        RESPONSEPUP[matchingCode, same ID, reversedSockets, responseData]
```

In what follows, we will abbreviate this as a procedure call qualified by an indication of the source and destination objects, or by an interface name that implies the source and destination. We will use this notation whether or not network communications are involved. In the case of network communications, the source and destination values should be thought of as socket identifiers that will locate the objects to which the packages are addressed. Sometimes this socket interpretation will be explicitly indicated (e.g., as [net#host#socket#].)

Abbreviation:

```
<Source!Dest>.Code[data] RETURNS [responseData];

        or

Interface.Code[data] RETURNS[responseData]; -- familiar?
```

Smarts$Fone Protocols (Client interface)

<Smarts!Fone> interface abbreviated as ToFone; <Fone!Smarts> interface abbreviated as ToSmarts

```
ToFone.GetStatus[] RETURNS [status: {callInProgress, outOfService, idle, TBD},  filterInfo:
                TBD}];

ToFone.CallByRName[self: Smarts, name: RName, priority: {TBD, includes "normal"}]
                RETURNS [{callInProgress, priorityTooLow, rejected,
                        busy (and not rejected), noAnswer}];
```

Smarts[EP]$EP Protocols ("Hardware" interface)

<Smarts!EP> interface abbreviated as ToEP; <EP!Smarts> interface abbreviated as FromEP (I know, I know)

FromEP.StillHere[!timeout, rejection=>-- time to reregister];

To be issued at intervals by EP, or when there's no response to some other query.

FromEP.RecordEvent[Event];

Event is an enumerated type containing {0, ..., 9, #, *, A, B, C, ..., onHook, offHook, ...};

ToEP.Reset[severity parameters?];

Cancel any tone sequence in progress. Clear the display. Forget about any conversation in progress. Hang up any automatically switched audio devices (Speakerphone, etc.) Forget the name of your Smarts, except when received as the first command after a GetSmarts (?????) Alternative: severity parameters indicate how much to forget. This is a catchall for various kinds of reset or abort functions.

ToEP.Tones[f1, f2, modulation: Hertz, on, off: Milliseconds,
repetitions: CARDINAL, mode: {ring, ringback, transmit}];

f1=f2 implies silence. modulation=0 implies that f1 and f2 specify sine waves to be added. Otherwise tones of the two frequencies alternate at the modulation rate. Tones occur in bursts whose duty cycle is deterined by on and off. Any Tones in effect are cancelled by a Reset[] or Converse[] request. Tones returns immediately. **TBD**: how to achieve a timed sequence of tone behavior (aside from Feep); require EP to queue requests, abortable only by Reset?.
mode:
ring -- tone to office speaker only -- annunciation
ringBack -- tone to office handset speaker only (or speakerphone/headset equiv.) -- call progress
transmit -- tone to office handset and transmission line (both parties) -- signalling

ToEP.Feep[on, off: Milliseconds, mode: {ring, ringback, transmit},
length: CARDINAL, number: PACKED ARRAY [0..0) OF Event];

Equivalent to a complex sequence of Tones requests resulting in the generation of a DTMF sequence, usually in transmit mode. Make/break intervals (on, off) parameterized, since one can usually push the TelCo specs so experimentation will be useful.

ToEP.Display[length: CARDINAL, number: PACKED ARRAY [0..0) OF Event];

EP is assumed to have a one line character display. This specifies the string to be presented there.

ToEP.ConverseWith[yourParty: Socket, otherParty: Socket, protocolType: {interactive, recording}];

The protocolType field will allow us to behave differently towards the file server than towards other conversants. Smarts or its Fone will have to communicate further with the file server to obtain "time and charges" -- information about the call's duration and nature.

ToEP.GetStatus[yourParty: Socket, otherParty: Socket, protocolType: {interactive, recording}];

The protocolType field will allow us to behave differently towards the file server than towards other conversants. Smarts or its Fone will have to communicate further with the file server to obtain "time and charges" -- information about the call's duration and nature.

Fone\$*Fone, Fone*\$*Net Protocols (Internal interfaces)*

(mostly TBD during Etherphone Server design and implementation -- none of these interfaces are visible to the Client (Smarts) or to the Etherphone implementations.)

<Fone$_i$! Fone$_j$> abbreviated as InterFone; <Fone$_i$! Net> abbreviated as ToNet

InterFone.CallRequest[callDescriptor: TBD (includes Fone$_i$ ident), priority: {TBD, includes "normal"}]
RETURNS [{canBeDone, priorityTooLow, rejected, busy (and not rejected)}];

InterFone.ConnectRequest[callDescriptor: TBD] RETURNS [{callInProgress, rejected, timedOut}];

ToNet.RegisterCall[callDescriptor: TBD, $Fone_j$: Fone] RETURNS [<two conversation socket values>];

"Sneak Paths" (Bootstrapping interfaces)

```
<x!Broadcast>.RoutingInfoRqst RETURNS [routingTable];
<x!Broadcast>.NameLookup[serviceName: STRING]
        RETURNS [list of [net#host#rtpSocket] tuples];
```

(interpretation of rtpSocket (rondezvous/termination protocol): a socket allowing a brief sneak path from the caller to the Net object in the Thrush server to generate some smarts)

There are more sophisticated schemes for linking up to services, etc., in the works as part of the RPC effort; we will watch them with abiding interest.

<EP!Net>.GetSmarts RETURNS [Sx: SmartsSocket];

Net is [ThrushHost#rtpSocket] obtained from NameLookup. Sx will be used in the sequel to describe the socket over which the EP and its Smarts communicate.

<Sx!Net>.GetFone[self: Smarts, rName: RName] RETURNS [Fone: FoneHandle];

This is just a procedure call within the server.

<Sx!Net>.GetRName[soc: Socket] RETURNS [rName: STRING];

Finds an RName associated with that host, in local data base. TBD: a Smarts!Fone!Net function to update this and other data bases.

Scenarios

When sufficient pieces of the protocol have been designed, it will be possible to render a detailed scenario of call placement and receipt under a variety of circumstances. We will use these scenarios to convince ourselves that we have enough bases covered to begin programming.

Figure 1. EPhone 1 calls EPhone 2

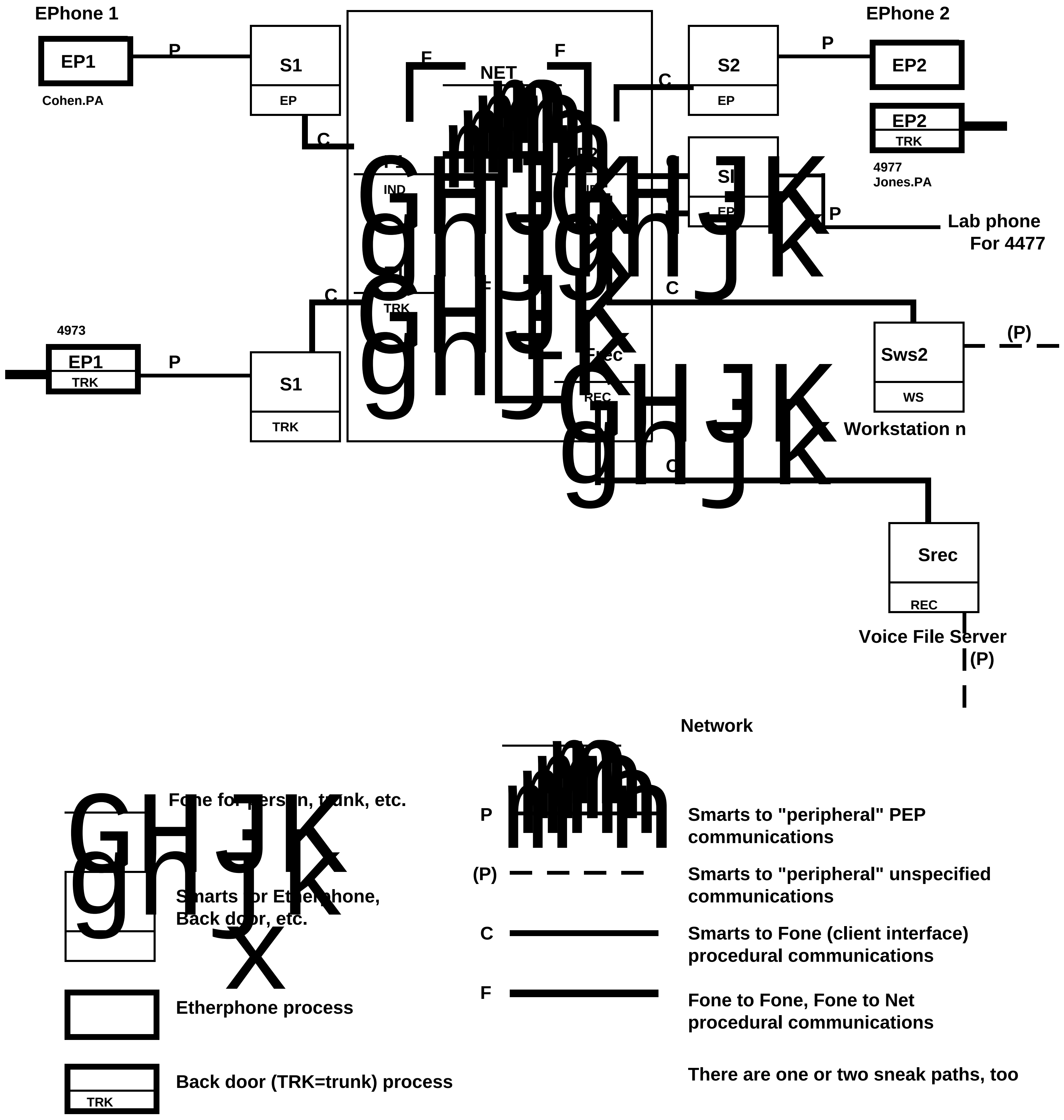
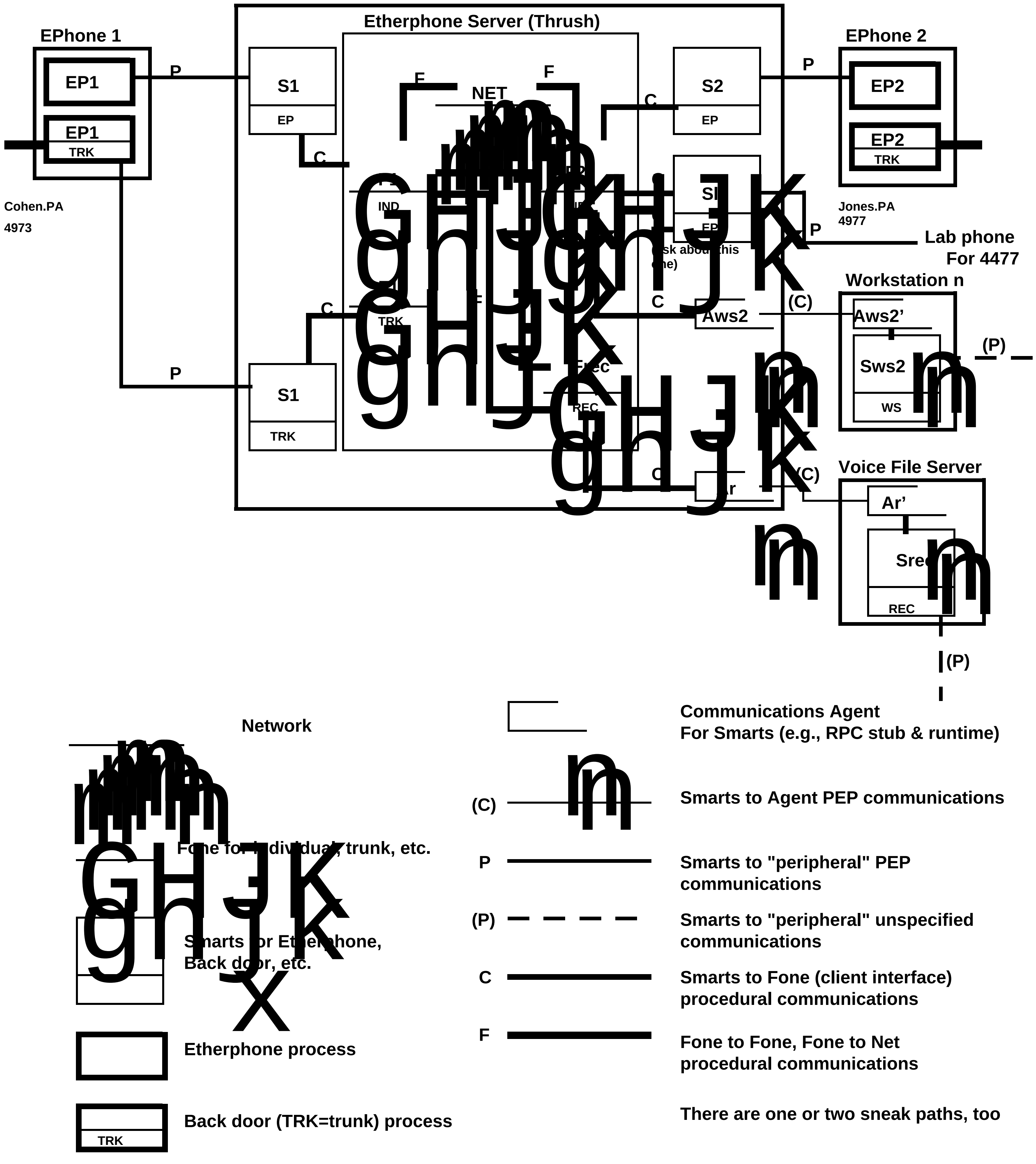

EPhone 1
EP1
Cohen.PA
P
S1
EP
C
F
NET
F
C
S2
EP
P
EPhone 2
EP2
EP2
TRK
4977
Jones.PA
IND
P
Lab phone
For 4477
C
TRK
C
4973
EP1
TRK
P
S1
TRK
Sws2
WS
(P)
Frec
REC
Workstation n
C
Srec
REC
Voice File Server
(P)
Network
Fone for person, trunk, etc.
Smarts for Etherphone,
Back door, etc.
Etherphone process
Back door (TRK=trunk) process
TRK
P
Smarts to "peripheral" PEP communications
(P)
Smarts to "peripheral" unspecified communications
C
Smarts to Fone (client interface) procedural communications
F
Fone to Fone, Fone to Net procedural communications
There are one or two sneak paths, too
T
POTS protocols

Figure 1. EPhone 1 calls EPhone 2

Figure 1. EPhone 1 calls EPhone 2



Etherphone Server (Thrush)
EPhone 1
EP1
EP1
TRK
P
S1
EP
C
S1
TRK
C
P
Cohen.PA
4973
F
F
NET
IND
TRK
Frec
REC
C
S2
EP
EP
(ask about this one)
P
EPhone 2
EP2
EP2
TRK
Jones.PA
4977
Lab phone
For 4477
Workstation n
C
Aws2
(C)
Aws2'
Sws2
WS
(P)
C
Ar
(C)
Voice File Server
Ar'
Srec
REC
(P)
Network
Fone for individual, trunk, etc.
Smarts for Etherphone,
Back door, etc.
Etherphone process
TRK
Back door (TRK=trunk) process
Communications Agent
For Smarts (e.g., RPC stub & runtime)
(C)
Smarts to Agent PEP communications
P
Smarts to "peripheral" PEP communications
(P)
Smarts to "peripheral" unspecified communications
C
Smarts to Fone (client interface) procedural communications
F
Fone to Fone, Fone to Net procedural communications
There are one or two sneak paths, too
T
POTS protocols

Figure 2. Assignment of functions to machines