Call Management in the Open Distributed Office

Mike Rizzo, Peter F. Linington and Ian A. Utting
Computing Laboratory, University of Kent at Canterbury,
Canterbury, KENT CT2 7NF, UK

November 1994

1 Introduction

Recent developments in telecommunications networks represent an opportunity to develop a whole new range of functions and services. More specifically, computing technology can be integrated with telephony to provide new functionality that extends far beyond the capabilities of traditional telephone networks, which were essentially centred around the straightforward connection and disconnection of pairs of named end-points.

Thus far, efforts to make use of this technological opportunity have concentrated on the provision of very basic stand-alone services, such as call forwarding, which exhibit very little integration with other data services, if at all.

This paper describes an agent-based model which takes advantage of the integrated computing-telephony environment to provide effective management of voice calls. In this model, agents manage calls on behalf of users, who influence the behaviour of their agents by means of policy specifications. Call setup involves a negotiation process whereby agents attempt to agree upon some course of action to take. Agents can also exercise control over a call in progress.

The model arose out of the Open Distributed Office (ODO) project [RLU94b] at the UKC Computing Laboratory. This project sought to investigate organisational, user interfacing and programming techniques to exploit integration of voice and data services at the application level, taking the office environment as a test case. Reference to this project will be made for illustrative purposes. However, we believe that the model is general enough to be applied in other areas, including public telephony services.

The rest of the paper is structured as follows: section 2 gives an overview of the ODO project, section 3 outlines the low-level switching interface to the ODO voice network (known as the connection level or binding level), section 4 introduces the ODO service level (where negotiating agents come into play), section 5 describes the negotiating agents model, section 6 illustrates how the model works using an example, and finally section 7 rounds up with conclusions and an indication of future directions.

2 The Open Distributed Office

ODO focusses on a number of services which were deemed interesting with respect to speech and data integration in the office. Figure 1 lists these services, illustrates the kinds

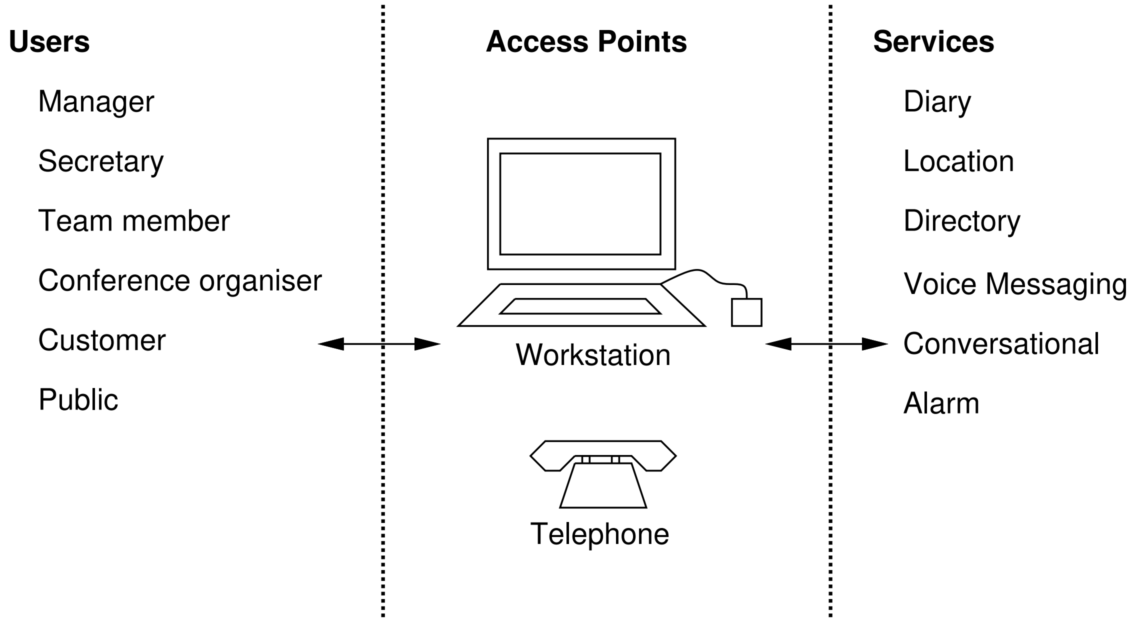


Figure 1: The Open Distributed Office

of devices used to access them, and gives examples of the sort of people that would be expected to use them.

For example, callers can make appointments via phone-based 'automated secretary' services, and users can browse their calendar of events via the phone. A user's calendar data might also be used as a source of location information to automatically route incoming calls to the nearest access point to that user.

Another feature of ODO is that users are able to offer alternative fallback services when they are not able to answer calls. For example, if a site's postmaster was going to be away for a few days, he might specify that all calls related to electronic mail issues should be forwarded to the deputy postmaster, whereas all other calls should be re-directed to his voice message system. Consequently callers would be greeted by a menu, asking them to select which of the two options they would like to take.

The ODO architecture comprises two levels: the *binding* (or *connection*) and the *service* level. The binding level corresponds to the voice switching network, whilst the service level comprises the services and agents that make use of the binding level. Call management therefore belongs to the service level, but before proceeding to describe it, an understanding of the binding level services is necessary. This is the topic of the next section.

3 The Binding Level

The binding level is based on the ODP streams model[Inta]. In this model, a continuous stream flows amongst two or more *stream interfaces* under the control of a *binding object*. In ODO, we refer to stream interfaces as *end-points* and a binding is currently limited to an association of exactly two such end-points. However, in the long term, the intention is to support multi-party bindings for an arbitrary number of end-points (within reasonable

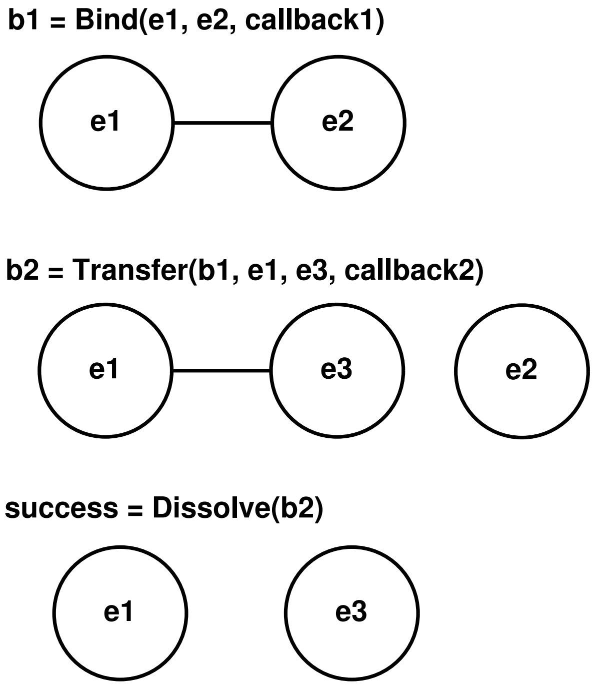


Figure 2: Binding operations

bounds) to allow conference calls.

An end-point is an abstraction of any device which is capable of

- transmitting and/or receiving an audio stream, and
- optionally generating and/or handling keypad events.

When two end-points are paired in a binding, information exchange commences so that audio and keypad events produced by one end-point are consumed by the other and vice-versa. This information exchange continues throughout the lifetime of the binding, which lasts until the binding is dissolved or transferred via the binding's *control interface*. As implied, this interface contains two operations: a *dissolve* operation can be used to terminate the binding, whilst a a *transfer* operation can be used to replace one end-point by another to produce a new binding (see figure 2).

Three kinds of end-points were implemented in the ODO prototype: phone end-points, workstation end-points, and service end-points. Phone and workstation end-points are collectively referred to as user end-points because they allow human users to communicate with the system. Service end-points, in contrast, are driven by service user interfaces. All bindings must involve at least one user end-point; it does not make sense for service end-points to be bound together.

Creation of and communication with bindings is done via a *connection manager*. This composite object may be likened to a telephone exchange, with the difference that it is not responsible for handling requests or notifications to or from end-points. It is essentially a 'pure' switch which allows connections to be created, manipulated, and destroyed. In order to connect two end-points, an end-point *capability* for each of them must be presented to the connection manager.

4 The Service Level

Above the binding level, the service level is responsible for setting up, manipulating, and dissolving bindings on behalf of its clients. It consists of a number of *services* and a collection of *negotiating agents* (NGAs), each associated with a particular *client*. Several different kinds of client are possible, including end-points, end-point groups, users, user groups, organisations, and services.

NGAs are responsible for call management and they may make use of available services to assist them in completing some task. For example an NGA representing a user (henceforth referred to as a user NGA) might use a location service [RLU94a] in order to find a suitable end-point so that communication with the associated user may be established. NGAs may also act on behalf of interactive services, such as voice messaging, so that calls are handled in a uniform manner regardless of whether a caller is talking to a human or an interactive service.

A call is initiated whenever a client passes a request to its NGA. Several NGAs may be involved in processing the request. For example, a user Joe might request that he would like to speak to another user Jane. Joe's NGA contacts Jane's NGA which responds by locating Jane and contacting the NGA for the end-point nearest to her. If the end-point is available, its NGA might 'lock' the end-point (with a time-out on the lock) and pass a capability back to Joe's NGA via Jane's NGA in order that Joe's NGA might set up the call.

Each NGA is guided by a *call management policy* which may impose a number of constraints on the NGA's permissible behaviour, as well as endow it with a number of empowerments. Typically, a policy is defined in terms of a mixture of (i) static rules that are hard-wired into the NGA and cannot be modified, as well as (ii) dynamic rules which are supplied by an *administrator* responsible for the behaviour of the NGA, and which may may be changed several times in the NGA's lifetime. The administrator may choose to (partially) delegate responsibility for an NGA's dynamic policy to other objects in the system, including possibly the client of the NGA itself.

The processing of a request may be affected by the policies of any of the NGAs involved in a call. In the previous example, Joe's NGA might be instructed to set up calls only if the identity of the target end-point is revealed. Jane's NGA's policy might dicate that end-point identity (which is indicative of her location) should only be revealed if she is within the confines of her company's building, but not anywhere else. And an end-point's policy might require that the identity of the caller be known in order for it to grant a request.

In the event that some constraint cannot be satisfied, the request cannot be processed further. The initiating NGA may choose to follow some alternative course of action. If the unsatisfied constraint is due to an NGA other than the initiator, the initiator might turn to one of its peer NGAs (that is, one that it is negotiating with) for suggestions as to what try next. It might then decide to follow up such a suggestion, follow some other course of action specified in the original request, or simply return failure back to its client.

Expanding on the previous example, suppose Joe's policy did not reveal his identity to certain users, Jane being one of them. This means that any end-point chosen by Jane's NGA would not be handed the identity of the caller. If the chosen end-point NGA turned out to be fussy about caller identity, then this might return failure in repsonse to Jane's NGA's request. Jane's NGA might then try to find another suitable end-point. If unsuccessful, Jane's NGA could inform Joe's NGA that the request could not be satisfied, but that it would be able to try an alternative course of action if desired. Joe's NGA might want to try to do something completely different first, and then come back to Jane's NGA's alternative course of action later.

5 The Negotiating Agents Model

An NGA is an object typically characterized by the following operational interfaces:

- a pair of *client interfaces*, namely a *client request interface* (CRI) and a *client notification interface* (CNI), via which the NGA interacts with its client to obtain instructions on set up and manipulation of calls,

- four *negotiation interfaces*, namely a *negotiation initiator request interface* (NIRI), a *negotiation initiator notification interface* (NINI), a *negotiation responder request interface* (NRRI), and a *negotiation responder notification interface* (NRNI), via which the NGA communicates with other NGAs,

- a *configuration interface* via which dynamic call management policy may be specified.

Additionally, an NGA may also provide interfaces to other objects that it needs to communicate with in the course of a negotiation. Two examples of such interfaces are for access to connection management and location services. The nature of such interfaces depends very much on the type of client that an NGA represents, as well as the kind of functionality that the NGA offers.

5.1 Configuration interface

NGAs may classified according to the type of client they represent. For example, user NGAs perform tasks on behalf of users, whereas end-point NGAs act on behalf of end-points. For each NGA type, there may be specific issues that are relevant to specification of policies for NGAs of that particular type.

This classification can be broken down further based on the functionality that NGAs provide. For example, one kind of user NGA might act upon explicit instruction, while another might try to act autonomously by learning from the user's past behaviour (an 'intelligent' agent). Depending on the type of its client and the kind of functionality it offers, the NGA's configuration interface provides a set of *configuration operations* which can be used to guide and constrain the agent's behaviour as required by the administrator, and possibly also to query the status of the agent and obtain performance indicators.

Dynamic call management policy is specified by means of a *policy specification language* (PSL). A *policy specification interpreter* (PSI) is used to parse such specifications and configure the NGA via its configuration interface. The interpreter is therefore the NGA's peer across this interface, and it must generate configuration operation invocations that correspond to the PSL specification being processed.

A PSL specification is supplied via a user interface provided by the PSI. The nature of this interface depends very much on the characteristics and complexity of the PSL, which could very well be a visual language as opposed to a text-based one.

Several PSLs may be defined for use with any one configuration interface, as long as a sensible mapping to the associated configuration operations exists for each choice of PSL. An appropriate PSI that implements the mapping must also be provided. Support for multiple PSLs makes it possible to target different classes of administrator requirements.

One of the design considerations that must be taken into account when developing a PSL is the trade-off between flexibility on one hand, and ease-of-use on the other. A PSL may be designed around a configuration interface so that the corresponding mapping is 'one-to-one'. The resulting PSL is maximally flexible in the context of its associated configuration interface, but may be rather complicated and difficult to use. At the other extreme, a PSL might allow specification of values for a limited number of parameters, so that the nature of the policy essentially always remains the same, changing only with respect to the said parameters. In this case, ease-of-use is maximised, but there is very little flexibility. In practice, PSLs are likely to fall somehwere in between these two extremes.

An NGA may support multi-tier policy specification whereby, for example, organisation-level policy requirements can be specified independently of user-level requirements, with the proviso that organisation policy has priority over user policy whenever the two clash. Different PSLs may be provided for each tier.

The same interpreter object may be used with several NGAs at once. This can be useful in managing groups of agents. For example, an organisation might employ an administration service which maintains identical policies for groups of agents and updates them simultaneously whenever changes are made.

The configuration interface may optionally include operations to query the status of the NGA, and to access a history of its behaviour. Once again, the operations are NGA-specific. The data obtained through these operations can be used to evaluate the performance of the NGA and to identify any shortcomings with its current policy.

5.2 Client interfaces

Communication between an NGA and its client takes place for one of two reasons:

- client-initiated: the client may wish to make a request to place, manipulate, or hang up a call;
- agent-initiated: in the course of negotiation, the NGA might (on the basis of its policy) need to interact with its client in order to take a decision.

These two kinds of communication involve operation invocations in opposite directions and consequently a separate operational interface is required for each. From the point of view of an NGA, the CRI plays a serving role and handles client-initiated communication, whilst the CNI plays a calling role and handles agent-initiated communication. Associated with these interfaces are a *client request protocol* (CRP) and a *client notification protocol* (CNP) respectively. Figure 3 depicts a typical arrangement around a user NGA's client interfaces.

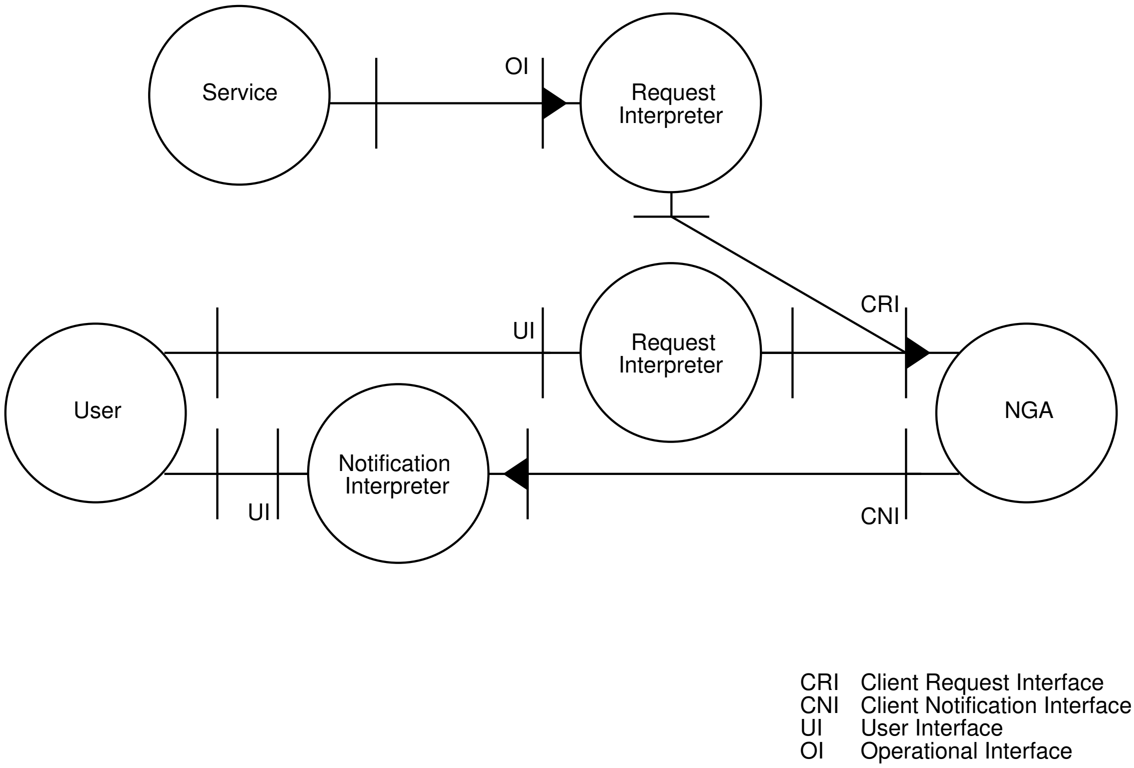


Figure 3: User NGA and client environment.

Client-initiated communication

Client-initiated communication typically (though not necessarily) originates from NGAs whose clients are users. In such cases, client requests are expressed in a *request specification language* (RSL), and are passed (via an appropriate interface) to a *request specification interpreter* (RSI) that generates corresponding CRP messages for communication with the NGA. An RSL specification effectively extends the NGA's call management policy for the processing of the one particular request that it represents. As is the case with PSLs, several RSLs may be defined for use with a CRP, each possibly offering different levels of flexibility and ease-of-use characteristics.

RSL specifications may be passed to an RSI via a user-oriented or service-oriented interface. Thus a request to an NGA may originate directly from a user interface or from another service (via an operational interface). An example of the latter is the ODO voice message service described in [RLU94c], which may generate a request when a user decides to reply to a message he has just listened to. Such a request must identify an RSI and provide a specification in the corresponding RSL. Again, as for PSLs, an RSL need not necessarily be textual. In particular it can also be graphical, or phone/voice-based.

Client-initiated communication may also originate from non-user NGAs. For example, an NGA representing a market survey company might automatically attempt to set up calls to conduct market research questionnaires over the phone. The concepts of an RSL and RSI are still relevant in this case, allowing specification of a list of telephone numbers to try.

Agent-initiated communication

The model also supports client/NGA interaction arising in the course of the negotiation process. For example, a user interface for a user NGA might draw the attention of the user to an incoming call, asking whether the user would like to accept or reject the call. This kind of communication is initiated by the NGA, which notifies the client when it requires its intervention (hence the term client *notification* protocol).

In order that notification messages may be received by a client, the latter must supply a server for a notification interface and have a reference to it registered with its NGA. Such references effectively form part of the NGA's policy and should therefore be specified via the configuration interface. Alternatively, it should be possible for the NGA to obtain a notification interface reference through other methods, for example by using a locator system.

A notification message normally warrants some action on the part of the NGA's client, and often requires a reply. The nature of this action depends on the type of the client. For example, an end-point NGA may ask its client whether it is in a position to accept an incoming call, in which case the end-point can return a reply based on its internal state and policy. In the case, of user NGAs, such interaction with the client i.e. a user, must be done via an appropriate user interface.

The concept of an interpreter is still useful in agent-initiated communication wherever the NGA represents a user. When user interaction is required, the notification server assumes the role of a *notification interpreter* which communicates with the user via one or more of the user interfaces available. The latter are likely to change over time as the user moves from one location to another. Thus, as before, several notification interpreters may be used with the same NGA, each interpreter supporting a different form of interaction. This allows interaction to take place over different kinds of devices. For example, notification of an incoming call may be done by ringing a phone's bell if the user it sitting next to a phone, or via a workstation-based graphical interface if the user is knwon to be working on a particular workstation. The possibility of the notification being done using both mechanisms simultaneously is not excluded either.

Determining which device/interpreter pair to use is the task of the NGA, drawing on its policy for guidance. A simple approach would be to specify a fixed device/interpreter pair in the NGA's policy. This works as long as the client is immobile, but for user-NGAs where the user's location changes over time, a more sophisticated approach is required.

One possibility involves distinguishing between two kinds of agent-initiated communication. Such communication may arise out of a negotiation that was initiated as a consequence of (i) a request by the client under consideration (as is the case when placing a call), or (ii) a request by some other client (as is the case when accepting or rejecting a call). In (i), an interpreter can be nominated as part of the request, as the location of the corresponding user is known by virtue of the device from which the request is made. For example, if a user picks up a phone and dials a number, it would be reasonable to expect that this user should be notified whether the target is available or busy via that same phone. In (ii), a device/interpreter pair must be determined by the NGA. Typically, this would make use of a location service to find an appropriate device. If a device is tried without succcess, it may be possible to try another. It may also possible to try several devices simultaneously,

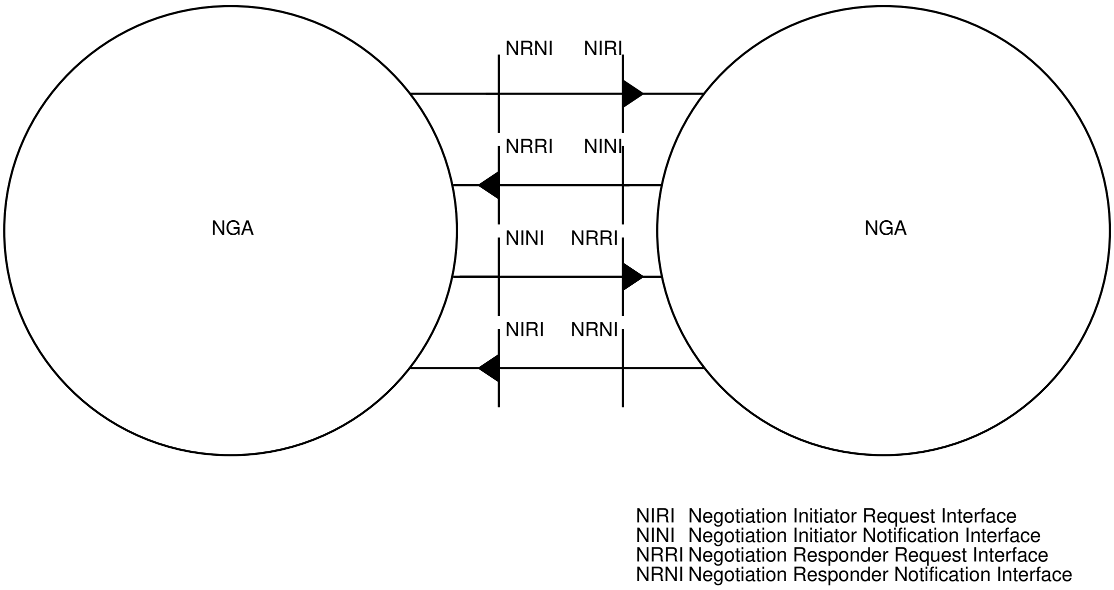


Figure 4: NGA interfaces.

taking care to ensure that once a user has responded through one device, other devices are notified so that they may react accordingly.

5.3 Negotiation interfaces

Evey negotiation process involves exactly two NGAs, one of which assumes the role of *initiator*, the other of *responder*. Either role is capable of acting both as server and caller with respect to operation invocations, hence the need for four interfaces. The protocols used on these interfaces are interdependent and are collectively referred to as the agent negotiation protocol (ANP). The relationships between NGA interfaces is shown in figure 4.

The NIRI interface allows an initiator to pass a request to a responder. A typical example is a request to obtain an end-point capability. Such a request might return an indication that the phone at the responder's end is now ringing. The responder might then notify the initiator (via the initiator's NINI) when the phone has been picked up. Of course these two examples of operation invocation also involve the NRRI and the NRNI of the responder as these interfaces are complementary to the NINI and NIRI respectively.

Unlike the cases for the configuration and client interfaces, all NGAs must speak the same ANP for communication amongst themslves. Additionally, the choice of protocol determines the flexibility with which agents can process requests, and consequently the design choices available for any NGA's configuration interface, CRI and CNI are limited by the expressive power of the common ANP.

For example, suppose a user wishes to place a call directed at a specific end-point, specifying that should the latter be found to be unavailable then the call should be re-attempted as soon as it becomes available. It is not enough for the PSL to provide suitable language constructs; the user's NGA must also be able to request notification of availability from the end-point's agent via the ANP. The ANP must also allow such a notification to be made.

Clearly then, the choice of an ANP is one of the most important design decisions to be taken in such an arrangement of agents. It is the only characteristic that is common to all agents and it determines how closely they are able to co-operate amongst each other. A good ANP should be able to maximize inter-NGA co-operation while at the same time be general purpose enough to allow for the fact that NGAs do different things for different kinds of clients and handle different information types in the process. It should also be easily extensible, to accomodate new requirements after a system has already been deployed.

5.4 Connection management interface

NGAs capable of setting up connections must provide a caller interface for communication with the connection manager described in section 3. In order to set up a binding between two end points, the connection manager requires a capability for each of the end-points to be bound. Such capabilities are obtained through the negotiation process.

Typically, a request to set up a call originates from a user NGA which attempts to obtain an end-point capability for its own user and for the destination. Once this has been the done, the NGA instructs the connection manager to create a binding between these end-points.

6 Illustration

Choices for the interfaces and langauges discussed thus far are currently still an area of experimentation. As pointed out earlier, the choice of ANP is a critical one, whereas choices for other interfaces and languages are less so because they can evolve without causing too much disruption. Introduction of new choices for the latter can be made as and when changing enterprise requirements dictate. Ideally, evolution of the ANP should also be supported, but any additions or modifications must preserve backward compatibility.

Our current approach uses a two-tier ANP in which the lower *session tier* establishes a number of primitives for communicating items of information (called messages) and the upper *information tier* establishes a number of information types which may be recognised by NGAs. New information types may be introduced after a system is deployed, allowing for the introduction of new agent types that handle new kinds of information, and consequently supporting evolution of the system in general. Examples of the kinds of information that may be exchanged include information requests, end-point capabilities, user identitifiers, location identifiers, and service type identifiers.

In the rest of this section we illustrate how the agent model might work with respect to a particular example, highlighting the mappings between policies and the ANP. Choices for policies and the ANP are made purely for illustrative purposes, and are by no means definitive. Further research is required to establish a general-purpose ANP that will satisfy most requirements.

6.1 Example

Our example involves five agents A, B, C, V, and W. A, B, and C represent users, V represents a a voice message service for the user represented by B (henceforth referred to as user B), and W represents a workstation end-point associated with the audio-equipped workstation at which the user A is currently seated. The scenario described in this example arises as a consequence of user A attempting to place a call to user B.

A utilises a simple PSL which allows her to specify a list of agents that represent users to whom she would *not* like to be connected to. This is similar in concept to *Terminating Call Screening (TCS)* defined for Intelligent Networks CS-1 [Intb], except that here the policy explicitly forbids connection to clients of the indicated agents, no matter what the circumstances[1]. A's dynamic policy includes C in its list of screened agents. Additionally, A's static policy dictates that whenever an attempt to obtain service from another agent fails, then any alternatives that this may propose must be tried, as long as other policy rules are not violated. Such alternatives are tried until either one succeeds, or all suggested alternatives are exhausted.

B makes use of a more complex PSL with structuring constructs, including conditional statements. B's policy states that in the event that user B is busy, does not accept the call, or is currently in the coffee room, his calls should be diverted to C. Failing that, calls should be routed to his voice message service.

The policy for V allows any incoming call to be connected to user B's voice message service, provided that it is not connected to any other call at the time.

W's policy is to supply an end-point capability whenever it is asked for one. W is known only to A, so that requests for end-point capabilities can only be made via A.

Details of C's policy are not needed for this example. The only aspect of C's policy that we need describe is that C has no objection in revealing its identity to any other agent.

The sequence of events is described informally below:

1. user A instructs her agent to set up a call with user B;
2. A asks W for an end-point capability
3. W sets up the workstation audio hardware so that it is ready to participate in a connection, and returns a capability to A;
4. A obtains a reference to B via a directory service;
5. A initiates a negotiation with B and asks for an end-point capability to communicate with user B;
6. B attempts to locate user B, and (say, by means of an active badge location system [WHFG92]) discovers that user B is in the coffee room;
7. B informs A that it is not possible to satisfy user A's request;
8. A asks B whether it can suggest an alternative agent;
9. B replies in the affirmative;
10. A asks B for the identity of the agent that it has in mind;
11. As a matter of courtesy, B asks C whether it would be acceptable to disclose C's identity;

[1]In CS-1, it is not clear whether a call diverted to a number in the caller's TCS list should be screened or not.

12. C replies in the affirmative;
13. B returns the identity of C;
14. A asks B whether it can suggest yet another alternative agent;
15. B replies in the affirmative;
16. A asks B for the identity of the agent that it has in mind;
17. B returns the identity of V;
18. A asks B for an end-point capability for this service;
19. B passes this request on to V;
20. user B's voice message service is busy at the time, and V therefore informs B accordingly;
21. B passes on this message to A;
22. A receives the message and waits;
23. user B's voice message service becomes free and consequently V prepares its voice message service end-point for connection and returns a capability to B;
24. B passes the received capability to A;
25. A instructs the connection manager to bind the two end-points for which it has obtained capabilities;
26. user A interacts with user B's voice message service, which instructs user A to hang up when she is ready;
27. user A leaves a message and hangs up;
28. A sends a hang up request to B;
29. B passes on the hang up request to V;
30. V notifies B that it approves the hang up request;
31. B passes V's approval on to A;
32. A instructs the connection manager to dissolve the binding;
33. A informs B that the call has terminated;
34. B informs V that the call has terminated;
35. V frees all the resources that were involved in the call;
36. B frees all the resources that were involved in the call;
37. A informs W that the call has terminated;
38. W frees all the resources that were involved in the call;

Initiator to Responder (NIRI/NRNI)	
`open(ihnd) -> rhnd`	Open a new negotiation session, passing an initiator handle `ihnd` and receiving a responder handle `rhnd`.
`close(rhnd)`	Close the session indicated by `rhnd`.
`request(rhnd, ilist, itype)`	Initiate an interrogation within the negotiation session identified by `rhnd`, passing an item list `ilist`, and the type of the requested item `itype`.

Responder to Initiator (NINI/NRRI)	
`announce(ihnd, item)`	Return information `item` in the course of processing the current interrogation within the negotiation session `ihnd`.
`reply(ihnd, item)`	Reply with information `item` to the current interrogation within the negotiation session `ihnd`.
`fail(ihnd)`	Indicate that the current interrogation within the negotiation session `ihnd` can never be replied to.

Table 1: Session tier primitives.

6.2 Agent negotiation protocol

This section outlines one possible ANP that supports the example scenario described above. The ANP consists of two tiers as described earlier.

Session tier

The lower session tier of the ANP makes use of six primitives, three of which are directed from the initiator to the responder (NIRI to NRNI), and three of which are directed from the responder to the initiator (NRRI to NINI). These primitives are summarised in table 1.

The session tier centres around the notion of a negotiation session between a negotiation initiator and a negotiator responder. Within a session, the initiator may invoke a sequence of interrogations to the responder. The purpose of an interrogation is for the initiator to obtain an item of information from the responder. The initiator may itself supply a number of information items to the responder; the latter can use this information in determining what information it should return in its reply. Once an interrogation request has been passed to the responder, the latter may take as much time as it requires to come up with an interrogation response or indication of failure. During this time it may periodically return announcements to the initiator to give it some indication of the progress it is making with respect to the interrogation. Only one interrogation at a time is allowed within a negotiation session.

Information tier

The upper session of the ANP defines four information types as described in table 2.

Type	Description
bool	Truth value: `true` or `false`.
req	Used to indicate what is required of the responder in an interrogation. Possible values are: `connect`, `hangup`, `fallback`, `reveal_id`.
status	Used to announce the status of an NGA's client. Possible values: `unavailable`, `busy`.
epcap	End-point capability.

Table 2: Information tier item types.

Values of the request type `req` are used to indicate what information is required by the initiator in an interrogation. `connect` signifies that an end-point capability for connection is required, `hangup` is used to ask the responder if it agrees to a connection being dissolved, `fallback` is used to ask the responder whether it has an alternative course of action to offer, and finally `reveal_id` is used to ask the responder whether it is willing to disclose its identity.

Status values are used to communicate the progress of a `connect` request; `unavailable` denotes that the client is not in a position to accept the call, and `busy` denotes that the client is currently occupied, but should be able to participate in a new connection shortly.

6.3 Behaviour of ANP in context of example

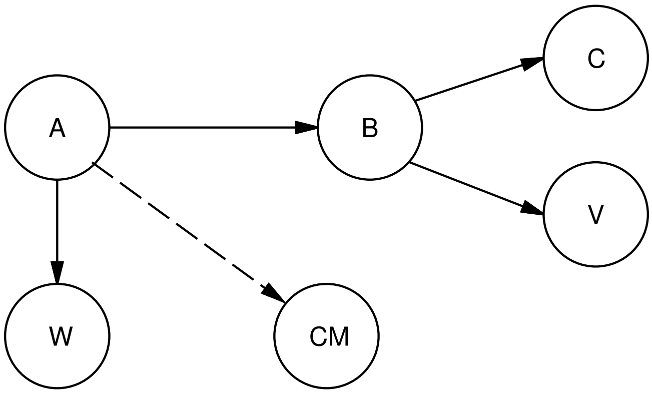
Figure 5 shows the negotiation sessions that arise in our example scenario and details the interactions between NGAs using the ANP we have just defined. Interactions between A and the connection manager (CM) are also included.

6.4 Discussion

The choice of ANP for this example is a minimalist one: it only defines the essential session primitives and information types needed to support the scenario under consideration. This section discusses some additions which would make the ANP more flexible. These additions can be made to the ANP described thus far without sacrificing backward compatibility. New types and additional new values for existing types can be introduced without requiring all agents to be upgraded simultaneously, thereby demonstrating how evolution of the ANP can work. The discussion also throws some light on how the ANP can be used to achieve various effects.

With the existing ANP, there is no way for the initiator to cancel an interrogation for information which is no longer required. A `cancel(rhnd)` primitive can be added so as to allow this. Consider the case where an interrogation results in the target's telephone ringing. After waiting for a number of rings, the intiator may wish to abandon the interrogation. This is analogous to hanging up an outbound call before it has been answered in POTS.

Additional values for the types `request` and `status` may be useful. An agent might want to ask for the current location of a user, or perhaps request notification as to when a user has become available. A status value may also be needed to indicate that there is a good chance that a client is available but some time is required before this can be



1.	A → W	open(awh) → wah
2.	A → W	request(wah, (”req”,”connect”), ”epcap”)
3.	W → A	reply(awh, <wepcap>)
4.	A → B	open(abh) → bah
5.	A → B	request(bah, (”req”,”connect”), ”epcap”)
6.	B → A	announce(abh, (”status”, ”unavailable”))
7.	B → A	fail(abh)
8.	A → B	request(bah, (”req”,”fallback”), ”bool”)
9.	B → A	reply(abh, ”true”)
10.	A → B	request(bah, (”req”,”id”), ”id”)
11.	B → C	open(bch) → cbh
12.	B → C	request(cbh, (”req”,”reveal_id”), ”bool”)
13.	C → B	reply(bch, ”true”)
14.	B → A	reply(abh, ”C”)
15.	A → B	request(bah, (”req”,”fallback”), ”bool”)
16.	B → C	close(cbh)
17.	B → A	reply(abh, ”true”)
18.	A → B	request(bah, (”req”,”id”), ”id”)
19.	B → A	reply(abh, ”B-vmsg”)
20.	A → B	request(bah, (”req”,”connect”), ”epcap”)
21.	B → V	open(bvh) → vbh
22.	B → V	request(vbh, (”req”,”connect”), ”epcap”)
23.	V → B	announce(bvh, (”status”, ”busy”)
24.	B → A	announce(abh, (”status”, ”busy”)
25.	V → B	reply(bvh, <vepcap>)
26.	B → A	reply(abh, <vepcap>)
27.	A → CM	connect(<wepcap>, <vepcap>) → avbh
28.	A → B	request(bah, (”req”,”hangup”), ”bool”)
29.	B → V	request(bvh, (”req”,”hangup”), ”bool”)
30.	V → B	reply(abh, ”true”)
31.	B → A	reply(abh, ”true”)
32.	A → CM	dissolve(avbh)
33.	A → B	close(bah)
33.	B → V	close(vbh)
34.	A → W	close(wah)

Figure 5: Sequence of interactions in example scenario.

ascertained. This value can be used to indicate that a phone at the target end is ringing but has not yet been picked up.

Introduction of new types and values does not require all NGAs in the system to be upgraded simultaneously. This means, however, that it is quite possible for an NGA to receive references to types and values which it does not understand. An NGA's policy defines what should happen in such circumstances. For example, an NGA that receives an interrogation request which it does not understand may simply fail the interrogation. In the case of user NGAs, however, one interesting possibility is to ask the peer NGA for an informal explanation of the meaning of the received information and present it to the user in order that he or she can take a decision. This would require the addition of some kind of `help_request` type.

The example involved only one negotiation session between any pair of NGAs. Sometimes two or more sessions may be required simultaneously, and it is possible that the same NGA might assume the role of initiator with respect to one session, and the role of responder with respect to another. Extending our example, suppose that the voice message service allowed the caller to indicate the end of the message by pressing a key, and then hung up the call itself. This would require V to initiate the hangup process, the sequence of interactions being roughly as follows: V opens a session with B, which in turn opens a session with A. V asks B whether it would be acceptable to hang up. B passes the request on to A, and A responds in the affirmative, dissolves the binding, and closes the original session. Simultaneously, B passes A's reply to V which closes the session it initiated.

Note that the reason for asking an NGA whether it is acceptable to hang up is that in some cases, emergency calls being one example, it is desirable to restrict control over hang up to one end of the call, or possibly even to a third party. In a multi-tier policy system, the tier with highest priority would typically stipulate that an NGA should always reply in the affirmative in response to a hangup request, unless the NGA represented a special case such as an emergency service.

In the example, A essentially negotiates with V through B. This indirect negotiation approach doubles the load on the control network in this case, but on the other hand it can also be useful because B retains control over the call which, after all, was directed to it in the first place. If user B leaves the coffee room and returns to his office while user A is still recording her message, user B is able to intervene by cancelling the message recording and picking up the call. This requires B to open a session with A and ask it whether it would be acceptable for it to have the call transferred, supplying a suitable end-point capability for user B in the interrogation request. Again, if A does not understand the transfer request, it can simply fail B's interrogation and the voice message recording continues.

7 Conclusions and Future Directions

In this paper we described a negotiating agents model for call management in an open distributed office system. A number of objects and interfaces were identified, but specific choices for these have not been made because it is intended that such choices will evolve. The only choice that needs to be established for any system based on the model is the agent negotiation protocol, which should also be designed with evolution in mind. For illustrative purposes, an example negotiation protocol was described and applied to one specific scenario.

We are currently experimenting with a variety of choices for each of the identified interfaces. Such choices are evaluated for flexibility with respect to a number of scenarios (also known as use-cases) such as the example given in the previous section. Languages at user interfaces will also require evaluation for usability by means of field trials.

In the context of the ODO project, the negotiating agents model is useful for bringing voice and data services together, offers unlimited flexibility, supports distributed evolution, and most importantly provides a mechanism whereby an acceptable solution to all parties involved in a call can be found.

The kind of negotiation session described in this paper is limited to two NGAs at a time. Involvement of more than two NGAs in the processing of a client request can therefore only be done by means of multiple negotiation sessions. This has been shown to be quite appropriate for two-party calls. However, we have not yet addressed multi-party (conference) calls, where it may be better to involve more than two NGAs in the same session. The concept of a multi-NGA session might also be useful if quality-of-service requirements for connections are introduced into the negotiation, in which case it would make sense to represent the connection manager by an NGA.

The negotiating agents model is general enough to be applied in other areas, beyond the scope of the ODO project. Recently we have started to apply the model to the provision of advanced telecommunications services in public telephone networks [RU95]. In this context, the model is useful in two respects: it gives subscribers more flexibility and provides a better infrastructure for dealing with feature interaction. We believe that the model will also prove useful in interactive multi-media applications.

References

[Inta] International Standards Organisation. *ISO/IEC 10746 (ITU-T Recs X.901-904) Reference Model for Open Distributed Processing.*

[Intb] International Telecommunication Union - Telecommunication Standardization Sector. *ITU-T Rec Q.1211 – Introduction to intelligent network capability set 1.*

[RLU94a] Mike Rizzo, Peter F. Linington, and Ian A. Utting. Integration of location services in the Open Distributed Office. Technical Report 14-94, Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent CT2 7NF, United Kingdom, August 1994.

[RLU94b] Mike Rizzo, Peter F. Linington, and Ian A. Utting. The ODO project: a case study in integration of multimedia services. Technical Report 12-94, Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent CT2 7NF, United Kingdom, August 1994.

[RLU94c] Mike Rizzo, Peter F. Linington, and Ian A. Utting. VitKit: a voice interaction toolkit. Technical Report 13-94, Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent CT2 7NF, United Kingdom, August 1994.

[RU95] Mike Rizzo and Ian A. Utting. An agent-based model for the provision of advanced telecommunications services. In *Proceedings of TINA '95*, pages 205–218. IEEE Communications Society, 1995.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Johnathon Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.