A Negotiating Agents Model for the Provision of Flexible Telephony Services

M. Rizzo
Dept. of Computer Science & A.I.
University of Malta
Msida MSD 13
Malta

I. A. Utting
Computing Laboratory
University of Kent
Canterbury, Kent CT2 7NF
England

Abstract

Current telephone systems suffer from a service interface bottleneck problem, wherein network resources are under-utilised and customer requirements are often not met, in spite of these resources' ability to satisfy such requirements. This bottleneck is primarily due to a coarse-grain service interface, coupled with the inability to support arbitrary terminal types, and the inability to inter-operate with other systems.

This paper outlines a new model for telephony services, based on the concept of negotiating agents. In this model, functionality is not made available to users in the form of services, as has been the case traditionally. Instead, users specify policies that describe how they wish their calls to be handled. These policies are used to guide agents appointed to act on behalf of users. The paper also describes a prototype that was built to demonstrate the capabilities of the model.

1 Introduction

Developments in telecommunications technology have, in recent years, enabled network operators to introduce a range of supplementary services. These first appeared as extensions to the traditional Public Switched Telephone Network (PSTN), then in the Integrated Services Digital Network (ISDN), and more recently in the Intelligent Network (IN).

Supplementary services modify the normal call setup process to provide some useful function. Popular examples are *Freephone*, *Call Forwarding on Busy*, and *Call Waiting*. In Call Forwarding on Busy, for instance, call setup is modified so that if the destination number is busy, the call is routed to another number.

Many supplementary services allow some degree of customisation, thereby enabling subscribers to tailor services to their specific needs. For example, Call Forwarding on Busy allows subscribers to specify a forwarding number. However, the extent to which such services may be customised is generally rather limited, offering little expressive power to the subscriber, who is only allowed to specify values to service parameters and turn services on and off. Consequently, these services cannot always meet customer demands, even if network resources could, in principle, be used to provide the required functionality. This problem is termed the *service interface bottleneck*.

This paper describes NAT (Negotiating Agents for Telephony), an alternative model for the provision of telephony services which aims to deliver increased functionality to end users. Rather than package functionality into a set of rigid, inflexible services, NAT allows users to express policies which describe how they would like their calls to be managed. These policies are used to guide users' agents, which negotiate and take action on behalf of their users. The aim of negotiation is to find a suitable course of action that is acceptable to all users concerned.

The paper is structured as follows: section 2 describes the service interface bottleneck, showing how it is related to the well-known problem of *feature interaction* [1]; section 3 reviews relevant research activities; section 4 introduces the NAT model; section 5 describes a prototype system that was built to demonstrate the capabilities of the model; section 6 concludes with an evaluation of the NAT approach.

2 The service interface bottleneck

Consider a telephone user X who wishes to have all his/her calls forwarded to a user Y if Y is not busy at the time of the call, or to another user Z otherwise. Clearly, a stored program control exchange is capable of doing this, and any necessary control messages could easily be carried over a modern digital transmission signalling network. Yet, such a service is not defined for PSTN and ISDN supplementary services, nor is it listed in existing IN capability sets.

Unfortunately, the problem highlighted by this simple example is characteristic of today's telephone sys-

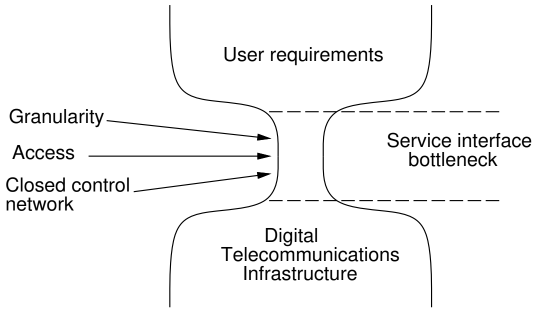

Figure 1: The service interface bottleneck

tems: the service software that sits between the transport network and the user represents a *service interface bottleneck* that often results in the under-utilisation of network resources and the inability to meet subscriber's needs, in spite of the fact that the resources *per se* are able to meet these needs. This is depicted graphically in figure 1.

Three principal causes of this bottleneck are:

- *service interface granularity*: the service interface offered to users is too coarse-grained, offering a limited number of highly specialised, inflexible services which suffer from feature interaction;

- *service interface access*: user terminal equipment is rather restrictive, and support for introduction of new terminal types is cumbersome;

- *closed control network*: service logic must execute on network operator equipment and cannot utilise sub-systems that are available on customer premises equipment.

NAT was designed with all the above causes in mind. However, this paper will focus primarily on the interface granularity aspect, as this turned out to be more interesting than the others. For further discussion of the service interface bottleneck problem see [2]. Essentially, the interface granularity problem prevents users from indicating how they would like their telephone calls to be managed. The interface made available to them is not flexible enough to allow them to specify their requirements. Furthermore, evolution of this interface is controlled by network operators and is made difficult because of feature interaction.

The remainder of this section illustrates the interface granularity problem (2.1), and discusses how it affects service deployment and scalability (2.2).

2.1 Illustration

The interface granularity problem will be illustrated using the Call Forwarding on Busy/No Answer (CFC) service feature as described for Capability Set 1 (CS-1) in clause B.2.8 of ITU-T Q.1211:

> *Call forwarding on busy/no answer* (CFC): This service feature allows the called user to forward particular calls if the called user is busy or does not answer within a specified number of rings.

Throughout this illustration, *scenarios* will be used to illustrate problems and concepts. A scenario is described in terms of a list of participating subscribers, a description of initial conditions, and a sequence of events relating to a particular call. Our first scenario illustrates the basic operation of CFC:

Scenario 1: Basic operation of CFC
Parties: A, B, C.
Setting: B has enabled CFC to C.
Sequence: A calls B; B is busy.

The outcome of this scenario is fully determined by the status of C. In the event that C is busy then A hears busy tone. Otherwise A hears ring tone until either C picks up the phone, in which case A is put through to C, or A gives up and replaces the handset.

Now consider the following scenario:

Scenario 2: Feature Interaction in CFC
Parties: A, B, C, D.
Setting: B has enabled CFC to C. C has enabled CFC to D.
Sequence: A calls B; B is busy.

In this scenario the outcome is not necessarily determined by the status of C. If C is not busy, then A is put through to C, but if C is busy then there are two possible outcomes: either A hears busy tone (as in scenario 1), or C forwards the call to D. As things stand, this choice is non-deterministic because the description of CFC given earlier is incomplete: it makes no explicit statements about what should happen if the user to whom the call is forwarded is also busy, or does not answer within a specified number of rings.

Thus, two possible interpretations of the description of CFC in the context of this scenario are: (i) that the attempt should fail and the calling user should hear busy tone, and (ii) that the user to whom the call was forwarded should be viewed as the new called user, so that CFC may be invoked again if enabled by the said user.

In fact in CS-1, CFC is actually a feature of an IN type A service (Q.1211,clause 5.1), which effectively dictates that the correct interpretation of the description of CFC is (ii). However, this is not always necessarily what is required by subscribers: some subscribers may require that their incoming calls are not forwarded to any numbers other than those which are explicitly named, in which case the behaviour described in interpretation (i) is desirable. This exemplifies how the use of *static resolution rules* to combat feature interaction contributes to the service interface bottleneck.

Next, consider a subscriber requiring the ability to specify two forwarding numbers for CFC. This might be provided in the form of a 'slightly' generalised form of CFC, say *Double Call Forwarding on Busy/No Answer* (DCFC). The following scenario shows how DCFC might be used:

Scenario 3: Double call forwarding with DCFC
Parties: A, B, C, D.
Setting: B has enabled DCFC to C and D in that order.
Sequence: A calls B; B is busy; C is busy.

The outcome of this particular scenario is unambiguous: B attempts to forward the call to C but fails and so re-attempts to forward the call to D, which accepts the call. However, along with the introduction of this new feature comes at least one new feature interaction type, which is exemplified by the next scenario:

Scenario 4: DCFC interacting with CFC
Parties: A, B, C, D, E.
Setting: B has enabled DCFC to C and D in that order. C has enabled CFC to E.
Sequence: A calls B; B is busy; C is busy.

In this example there are two possible courses of action once the call has reached C. The call might be forwarded to either D or E, depending on whether B's or C's feature is invoked! Either behaviour might be acceptable in the right circumstances; it would be wrong for the definition of DCFC to stipulate that paths offered by the first forwarding number should or should not be tried before trying the second DCFC forwarding number. Ideally, the decision should take into account the preferences of all the parties involved, with those of the caller A being given priority.

On to the next example! Along comes another subscriber asking for CFC with three forwarding numbers, and yet another asking for five. The CFC service provider decides to extend the coarse-grain service interface by providing *Multiple Call Forwarding on Busy/No Answer* (MCFC) with n numbers where n is a number deemed sufficiently large to satisfy all subscribers' demands.

Scenario 5: Multiple call forwarding with MCFC
Parties: $A, B, C_1, C_2, \ldots, C_n$.
Setting: B has enabled MCFC to C_1, C_2, …, C_n in that order.
Sequence: A calls B; B is busy ; C_1, C_2, …, C_j $(1 \leq j \leq n)$ are busy.

If $j < n$, then the outcome can be described as follows: B cannot accept the call and forwards the call to C_1; C_1 cannot accept the call so B forwards the call to C_2; …; C_j cannot accept the call so B forwards the call to C_{j+1}; C_{j+1} accepts the call. The remaining case i.e. $j = n$, results in A hearing busy tone.

MCFC represents a generalisation of CFC that is somewhat better able to meet customer requirements in the sense that the range of possibilities it offers is slightly larger. But it must be stressed that it is only *slightly* larger, because the improvement is not that significant in the context of the range of call management requirements that subscribers might have! Consider a subscriber requiring a combination of MCFC and *Call Waiting* (CW) which behaves such that if all MCFC forwarding numbers are busy, CW is invoked. The service provider can provide yet another new service but clearly there are limits to this approach, because of the infinite number of ways in which service features can be combined[1]

2.2 Service deployment and scalability

The illustration above highlights the principal problem with current approaches to service provision, namely that the granularity of the service interface provided to subscribers is too coarse. Functionality is packaged into simple features designed around what the service provider believes that subscribers need, offering very limited expressive power to the subscriber.

The service interface granularity problem cannot simply be solved by adding more services, for at least two reasons. Firstly, it does not make sense to deploy network-wide services which, because they are so specialised, will only be used by a small number of people. This problem can be avoided by the IN architecture to some extent, but such services still have to be subjected to regulation by the network provider. Even if this heavyweight service acceptance procedure was removed to allow unconstrained addition of new IN services by third parties, this would easily overwhelm the network's resources: it would be better to utilise third-party equipment to handle such specialist services. A second reason is that with each service that is added, the feature interaction problem is compounded. The

[1]The notion of creating new services by combining others does exist in CS-1 e.g. *Selective Call Forwarding on Busy/No Answer* combines CFC with *Terminating Call Screening*.

more services that are added, the greater the potential for interaction and subsequent undesirable behaviour.

The service interface granularity problem can only be overcome by providing a lower-level service interface to subscribers, giving them more expressive power that allows them to specify call management requirements in terms of finer-grain primitives than service features. But doing this requires careful thought and consideration. Giving subscribers more flexibility could lead to abuse or incorrect use of this flexibility. Moreover, a significant increase in expressive power could very well overwhelm subscribers with simpler requirements and limited know-how.

3 Related work

This section briefly reviews recent and ongoing initiatives which are relvant to the issues highlighted in the previous section.

3.1 The IN architecture

The IN architecture offers more service control flexibility than ISDN and its predecessors by providing a wider range of supplementary services. However, as we noted earlier, a finite number of pre-defined services can never be enough to satisfy all users' requirements, regardless of the number of services available.

The IN architecture's main improvement is that it makes service creation by third-party service providers possible. Unfortunately the approach is cumbersome and requires updating of components that fall under the auspices of the network operator. This does not permit lightweight deployment of behaviour tailored to one user's specific needs.

Another problem with IN is that the all-important Basic Call Processing (BCP) module still represents traditional circuit-related control. IN service invocation can only take place after the BCP has started, with the result that the IN services appear as an extension of the traditional telephone service instead of services in their own right.

Feature interaction continues to be the dominating problem within the IN community. Several solutions have been proposed, but none seems to be general enough to capture the problem in its entirety.

3.2 Velthuijsen & Griffeth model

Velthuijsen and Griffeth [3] propose a model which aims to avoid the need to study the behaviour of features taken in combination each time a new service is introduced. Users, terminals, and the bearer network are all represented by agents which negotiate with each other in order to establish a course of action which will achieve some specified goal, the assumption being that there may be several ways to satisfy a goal, not all of which may be acceptable to the parties concerned.

This approach is useful in resolving feature interactions where a pre-defined set of service features is deployed across an entire network. However, because agents must share a common goal hierarchy, which essentially determines the service features available, it is not possible to support lightweight introduction of new services, meaning that the model still suffers from the interface granularity problem.

Nevertheless, this negotiation model certainly represents a step in the right direction with respect to the feature interaction problem, because it avoids the need to check all possible combinations of features. The NAT model we will be describing attempts to go a step further by avoiding the need to have standardised, network-wide features.

3.3 ODP and TINA

There has recently been much activity in the area of *Open Distributed Processing* (ODP), including standardisation of a reference model (RM-ODP) [4], which addresses issues relating to the development of heterogeneous, large-scale distributed systems. RM-ODP centres around an object-based paradigm and makes extensive use of the principles of encapsulation and abstraction. Two fundamental concepts in RM-ODP are: *viewpoints* for separation of concerns, and stream bindings to support communication of continuous media. Both are used extensively in the NAT model.

One of the first applications to which ODP technology is being put is that of telecommunication services over broadband networks, the main initiative here being the *Telecommunications Information Networking Architecture* (TINA) [5]. TINA has no notion of a basic call process as in IN and its predecessors. Telephony is simply one kind of application, for which several access interfaces may be provided. Clearly there is a lot of potential for reducing the granularity aspect of the service interface bottleneck [6]. However, because TINA is a general architecture, it does not prescribe too much detail relating to the structure of services, and consequently it does not provide any insight as to how the granularity problem should be tackled.

4 The NAT model

NAT is a framework for the provision of flexible telephone services. It is described in terms of concepts and notations of RM-ODP.

NAT provides modelling concepts for the design of telephone systems, and is particularly geared towards eliminating, or at least widening, the service-interface bottleneck by (i) making a finer-grain interface accessible to end-users, (ii) facilitating the connection and management of arbitrary third-party terminals, and (iii) supporting integration with other systems. Thus

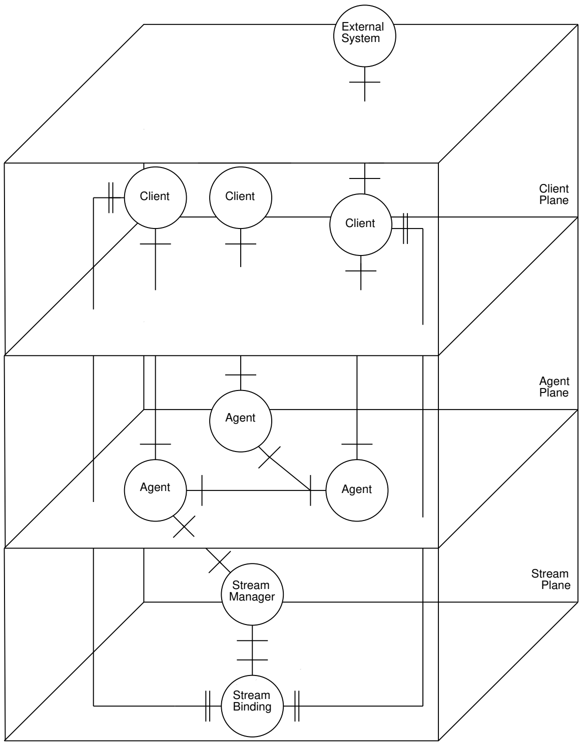

Figure 2: The NAT computational model

well-designed telephone systems based on NAT are better able to meet customer requirements, and can make use of network resources more efficiently.

NAT is not a design for any one particular telephone system, but is a framework that provides concepts which may be used to construct several such designs. In a particular NAT-based system, the degree to which customer requirements are met and network resources are utilised ultimately rests on the design of the system. Thus simply basing a design on NAT does not necessarily constitute a guarantee for customer satisfaction and efficient resource utilisation.

NAT centres around the concept of a *negotiating agent*. A negotiating agent is an object that engages in and conducts negotiations with other such agents on behalf of some *client* for which it has been appointed. Several kinds of client may be represented by a negotiating agent, including users, terminals, teleservices, administrators, user groups, and organisations.

When a client wishes to pursue some task that necessitates participation of other clients, its agent negotiates with these clients' agents in order to find a course of action that is acceptable to all involved. In the course of the negotiation process, agents may exchange questions, answers, proposals and counter-proposals until some consensus is reached, or until it is established that no consensus can be reached. New agents may be brought into, and others may be withdrawn from, the process at any time before the negotiation is completed.

Once a consensus is reached, transmission paths may be set up over an underlying transport network. The transport network may actually be capable of carrying several types of multimedia traffic, and is not necessarily restricted to voice.

The computational viewpoint for NAT-based systems may be specified in terms of three planes, namely the stream plane, the agent plane, and the client plane. Each of these has a static aspect and a dynamic aspect. The static aspect defines relationships between object and interface types, whereas the dynamic aspect describes relationships between instances of such types. Figure 2 depicts a particular configuration of the dynamic aspects of all three planes of a NAT system at some point in time.

The stream plane defines a multimedia transport service in terms of a set I of stream interface types, and a set B of stream binding types. I determines the kinds of subscriber terminals and teleservices supported by the network with respect to the information flows that may be carried over the lifetime of a call. It does not, however, have any say as to how such terminals or teleservices should be controlled. B determines the kinds of information communication that can take place over the course of a call. However, creation of and subsequent control over bindings is not the responsibility of the stream plane.

The agent plane contains agents which act on behalf of clients in the client plane. Each agent communicates with its client via a *client interface*. The static aspect of the agent plane defines a set of agent types and a set of client interface types. In the dynamic aspect of the agent plane, each client interface instance provided by a client is paired with exactly one agent instance and vice-versa. Agents also host a negotiation interface via which they negotiate amongst each other according to some *negotiation protocol*.

The client plane defines a number of client types, each of which supports exactly one client interface type. The same client interface type may be supported by more than one client type. For example, for a given client interface, one client may provide an implementation based around the standard telephone, whilst another might provide an implementation based on a multimedia workstation.

Further details on the NAT model can be found

`strmgr`	Stream manager: manages bi-directional bindings between audio stream interfaces.
`usterm`	User terminal: provides an audio stream interface, a user interface, and a client interface for interacting with a user agent.
`usagent`	User agent: provides a client interface for interaction with a user terminal, a negotiation interface for negotiating with other agents, and a policy specification language for expressing user requirements.
`stardev`	PC-based component which bridges the workstation network with the PABX network to enable exchange of voice across the two (uses StarT32 card from Staria [7]).
`starman`	Manages a pool of `stardev` components.
`stagent`	Gateway agent: translates between negotiation messages in the workstation domain and signalling messages in the PABX domain.

Table 1: NEMO-2 components

in [2]. The next section gives a clearer indication of NAT's capabilities by describing a more concrete demonstration system.

5 NAT demonstration system

NEMO-2 (NAT dEMO 2) is a NAT-based system that makes use of various resources available at the University of Kent at Canterbury (UKC) including the Computer Laboratory's computer network, an active badge network [8], and the university's internal telephone network. The system was implemented using ANSAware [9] as an engineering platform.

In NEMO-2, users communicate via workstation-based terminals attached to the computer network, and via telephones connected to UKC's internal telephone network. Each user is represented by an agent, and a policy specification language is defined to allow expression of call management policies.

Table 1 lists the components that make up NEMO-2. Figure 3 illustrates how these components would interact in a call between a workstation terminal and a telephone.

5.1 Negotiation protocol

The negotiation protocol used in NEMO-2 centres around the notion of a *negotiation session* between an initiator agent and a responder agent. Within a session, agents may ask each other questions about their intentions. At some point, the initiator may issue a request for connection. The responder can react by meeting the request or by offering an alternative course of action, referred to as a *fallback*. In the latter case, the initiator is allowed to start another exchange

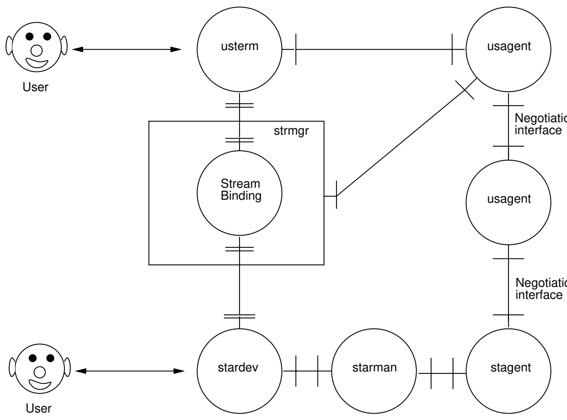

Figure 3: Call between workstation terminal and telephone

Figure 4: Workstation-based terminal for user agents

of questions and answers in order to discover information about the proposed fallback. The initiator might then choose to pursue the fallback or check whether the responder has other fallbacks to offer.

5.2 User agent terminal

The user agent terminal makes use of workstation audio hardware and a graphical user interface. The audio hardware is used to provide a bi-directional audio stream interface, and the graphical interface enables users to communicate with their agents.

Figure 4 shows the graphical user interface presented by the user agent terminal, comprising a message display, an input window, sliders for controlling input and output volume, and a number of control buttons. In order to use a user agent terminal, a user must first log in. Subsequently the user may use the terminal to place outbound calls and to receive incoming calls. The figure depicts an incoming call from a user *peter* for the logged user *mike*. At this point, user *mike* can press either of the `Accept` or `Reject` buttons to accept or reject the call respectively.

```
String coffeeroom := "SW101"
String lab := "SW103"
StringSet donotdisturb := [coffeeroom,lab,"S120"]
StringSet pricallers := ["ian","peter","john"]
BhvExpr dcf := forward("peter") {
    nolocate,noanswer,busy,reject,
    fallback,other -> forward("ian") {
        nolocate,noanswer,busy,reject,
        fallback,other -> giveup
    }
}
BhvExpr con := connect() {
    nolocate,noanswer,busy,reject,other -> dcf
}
BhvExpr incoming := callerid(pricallers) {
    true -> con
    false,dontknow -> timein(09:00,17:00) {
        true -> location(donotdisturb) {
            true -> dcf
            false,dontknow -> con
        }
        false -> dcf
    }
}
InitiatorForwApproval(on)
ResponderPolicy(incoming)
```

Figure 5: Policy specification for user Mike

5.3 Policy specification language

The policy specification language is particularly geared towards supporting flexible call forwarding policies, which may make use of users' identities, the time of day, and users' locations (obtained via the UKC active badge system).

A policy consists of a list of *constant definitions* and *policy statements*. Constant types available include: strings, string sets, booleans, and *behaviour expressions*. The latter can be used to descibe a *behaviour tree* which may then be used to guide an agent.

Figure 5 shows the policy for a user Mike, which consists of two policy statements. The first enables a feature of the NEMO-2 agent called `InitiatorForwApproval`. This causes Mike's agent to ask for Mike's approval before any of his calls are forwarded. The second describes how Mike's incoming calls are to be handled in terms of the behaviour expression `incoming`. This specifies that if the caller is in the list of priority callers, then a connection attempt should be made, otherwise the action to be taken depends on the time of day.

```
BhvExpr incoming := timein(9:00,17:00) {
    true -> connect() {
        nolocate,noanswer,busy,
        reject,other -> giveup
    }
    false -> giveup
}
InitiatorForwApproval(on)
InitiatorForwScreen(["peter"])
ResponderPolicy(incoming)
```

Figure 6: Policy specification for user Dave

5.4 Example scenario

Consider a user Dave, whose policy is shown in figure 6, calling Mike at 10:00 whilst Mike is in the coffee room. Dave's policy makes use of another agent feature, namely `InitiatorForwScreen`, which indicates that Dave's calls should *never* be forwarded to Peter. Dave also has `InitiatorForwApproval` enabled, so that forwarding attempts other than those to Peter may be approved interactively. The ensuing sequence of events is described in figure 7 (D denotes Dave's agent, M denotes Mike's agent).

6 Discussion

The contribution of this paper is twofold: (i) it views current approaches to telephone networks from a new perspective, exposing problems relating to satisfaction of customer requirements and efficient use of network resources, as well as throwing some light on the problem of feature interaction, and (ii) it suggests an alternative approach to telephone networks, emphasizing the need for de-centralised control and increased communication amongst the individual components that make up a system.

It is clear that the current philosophy of making functionality available as a tightly-controlled set of rigid, inflexible service units is not a good long-term paradigm for the telephony services of the future. This raises serious questions about whether it is worth spending so much time and effort investigating the feature interaction problem, when this problem arises only as a consequence of a particular choice of service provision paradigm. Indeed the feature interaction problem may be unsolvable, and the only way around it might be to make it disappear by adopting an altogether different approach to service provision.

The NAT model represents one such alternative approach. It allows users to express their requirements in terms of policies rather then service features, making

1. D requests a stream interface reference from M.
2. M asks for the identity of the caller.
3. D obliges.
4. M determines that the caller is not in the list of priority callers.
5. M determines that the time of day is between and 09:00 and 17:00.
6. M determines that Mike is in the coffee room.
7. M rejects D's request.
8. D asks M whether it has any fallbacks to offer.
9. M indicates that it is willing to forward the call to another user.
10. In view of Dave's policy statements on call forwarding, D asks M for the identity of this user.
11. M obliges, supplying the identity *peter*.
12. D immediately informs M that it is not interested in this course of action, and asks whether another fallback is available.
13. Again M indicates that it is willing to forward the call to yet another user.
14. Again D asks M for the identity of this user.
15. M obliges, supplying the identity *ian*.
16. D displays a message on Dave's terminal, asking Dave to approve the forwarding attempt.
17. Dave presses the `Accept` button to approve the forwarding attempt.
18. D informs M of Dave's decision, and M then tries to set up a call with Ian.

Figure 7: Sequence of events in example scenario

it possible for users to specify a much wider range of requirements. Moreover, the negotiation mechanism enables *all* participants in a call to have a say in the way the call is processed, and conflicts are resolved dynamically by taking all policies into consideration, rather than statically by a central service provider.

Of course the approach suggested by NAT also introduces a number of new problems, many of which require further research before the approach can be adopted in large-scale systems. For example, new approaches to charging and marketing are required, because functionality is no longer packeged into convenient tarriffable, easily-understandable units. The choice of negotiation mechanism is also extremely important, and should be designed to allow evolution and interworking with legacy systems or systems utilising different negotiation mechanisms. Care is needed to ensure that the increased flexibility offered by the approach does not compromise safety and reliability. Last but not least, dimensioning to meet performance guarantees is made harder by the proposed approach, as it is not possible to predict utilisation of the network in advance of negotiation.

The importance of each of these problems may vary, depending on the applications to which the model is put: whilst charging may be an important concerns in a public network, it is not likely to be an issue in a localised office communications system.

The NAT model is seen to be very much in the spirit of TINA. Work is currently underway to integrate NAT concepts into the TINA framework.

Acknowledgment

The authors are grateful to Olivetti Research Limited (UK) for donating an Active Badge system.

References

[1] E. J. Cameron *et al.*, "A feature-interaction benchmark for IN and beyond," *IEEE Communications Magazine*, vol. 11, pp. 64–69, Mar. 1993.

[2] M. Rizzo, *A Model for Flexible Telephony Services based on Negotiating Agents*. PhD thesis, Computing Laboratory, University of Kent at Canterbury, UK, 1996.

[3] N. D. Griffeth and H. Velthuijsen, "The negotiating agents approach to runtime feature interaction resolution," in *Proc. of 2nd Int. Workshop on Feature Interactions in Telecommunications Systems*, pp. 217–235, IOS Press, 1994.

[4] P. F. Linington, "Introduction to the basic reference model of open distributed processing," *IFIP Transactions C, Special Issue on Open Distributed Processing*, vol. C-1, pp. 3–13, 1992.

[5] M. Chapman and S. Montesi, *Overall Concepts and Principles of TINA (TB_MDC.018_1.0_94)*. TINA Consortium, 331 Newman Springs Road, Redbank, NJ 07701, USA, Feb. 1995.

[6] R. Minerva, "TINA service architecture: some issues in service control," in *Proc. of TINA '95, Melbourne, Australia*, pp. 97–112, Feb. 1995.

[7] Staria Ltd, 13 Market Place, Ross-on-Wye, Herefordshire, UK, *StarT32 Programmer's Manual*.

[8] A. Harter and A. Hopper, "A distributed location system for the active office," *IEEE Network*, vol. 8, pp. 62–70, Jan. 1994.

[9] Architecture Projects Management Ltd, Poseidon House, Castle Park, Cambridge, UK, *RM.099.02: An Overview of ANSAware 4.1*, May 1992.