The Negotiating Agents Approach
to Runtime Feature Interaction Resolution

Nancy D. Griffeth  
*Bellcore, 445 South Street, Morristown, NJ 07962-1910, USA*

Hugo Velthuijsen  
*PTT Research, St. Paulusstraat 4, 2264 XZ Leidschendam, The Netherlands*

**Abstract.** This article describes how to use the Negotiating Agents approach on a telecommunications platform. Negotiation is used in this approach to resolve conflicts between features of one user and of different users. The theory behind the approach is discussed briefly. Methods for implementing the approach are given along with the methods for defining IN features in terms of the Negotiating Agents approach in order to resolve conflicts between these features.

1 Introduction

Rapid change in the telecommunications industry increases the complexity not only of building but also of using telecommunications services. Much of the complexity arises from the feature interaction problem. When features interact, a user must understand the behavior of features in combination – even how features of other users may affect the behavior of her features. Similarly, a service provider must determine how combinations of features will behave, including combinations of its own features with other providers’ features. The need to understand features in combination, rather than individually, limits the ability of users to use the network and the ability of providers to add to it. In this paper, we propose an approach to one class of features and feature interactions that avoids the need to understand features in combination. This approach also provides an elegant mechanism for detecting and resolving feature interactions.

We assume that there is a standard platform underlying all services on a telecommunications network and offering a collection of operations to the users of the network. Even though this is not the case with existing networks, various groups are working to
standardize such a platform [1, 6]. Given such a platform, we define two kinds of features. One kind of feature is a technology feature, which is an individual operation that the platform provides. A new technology feature is created by creating a new operation on the platform. We imagine that these features will usually correspond to new technology or new equipment provided in the network—for example, voice activated dialing or voice recognition technology. The second kind of feature is a policy feature, which is a constraint on the set of operations that a user or provider is willing to perform in initiating or modifying a call. For example, an end-user may say that his customer equipment should never be connected to a particular number, or a service provider may say that its service should be provided only to an enumerated list of subscribers.

The problem of interactions between technology features is quite different from the problem of interactions between policy features. We address the problem of interactions between policy features only, and henceforth in this paper the term “feature” refers to policy feature. A single-party feature interaction occurs when it is not possible to satisfy all of the constraints of a single subscriber. A multi-party feature interaction occurs when it is not possible to satisfy all of the constraints of all parties to an attempted call.

Today’s features modify the process of call setup. The correct functioning of any such feature depends on assumptions about how this process works. But since what all features do is to modify this process, interaction between features is inevitable. The interaction is sometimes desirable (as when Block Calling Number Delivery prevents Calling Number Delivery from providing the originator’s number to the terminating party) and sometimes undesirable (as when Call Forwarding allows calls to be set up that Call Screening features would prevent). In any approach to the problem of feature interactions, it is necessary to describe the intended effect of a feature on a call in order to distinguish between desirable and undesirable interactions.

Our approach is to enable users and providers involved in a call to specify directly what are the constraints on the calls they will accept, instead of having to embody these constraints in features that are hard to understand and hard to use. We assume that constraints are expressed in terms of operations that a user or provider is willing or unwilling to perform. To resolve conflicts between the constraints of different users, we provide a mechanism for them to negotiate with each other to determine what set of operations will be used to initiate or modify a call. Such a negotiation process provides an automated method for detecting and resolving policy feature interactions at run time. This negotiation method guarantees that the policies or intentions of the various entities are respected, while calls are set up whenever possible. An important consequence of the use of negotiation is that the autonomy of different users and providers can be preserved.

We claim that negotiation provides the following additional advantages to end users and to service providers. The body of the paper substantiates these claims.

• Single-party feature interactions are automatically detected. Resolution of these
interactions is up to the user or service provider.

- Multi-party feature interactions are automatically detected and resolved at runtime.
- Each user and subscriber can specify an individual policy involving calls.
- All calls that can be set up without violating someone’s policy will be set up.
- Unanticipated feature interactions (even those due to faulty implementation or to system failures not involving the negotiation mechanism) will be detected if they result in a violation of someone’s policy.
- No one need be aware of any policy features belonging to anyone else, and in particular there is no need to know about the combinations of policy features used in setting up a particular call.

In section 2 of this paper, we present examples of features and their interactions for which more conventional methods of automatic feature interaction detection are problematic. In section 3, we describe how to incorporate our paradigm for negotiation in an arbitrary extensible, object-oriented platform for telecommunications. Our negotiation mechanism has been presented in detail elsewhere [4, 5], but we review it in section 4. Then we describe how negotiation has been implemented on one platform, the Touring Machine™ platform [1]. In section 6, we discuss how some generic AIN and IN CS-1 features would be designed using our paradigm. Finally, in section 7, we summarize the advantages and discuss some remaining research issues.

2 Example Feature Interactions

We present features in this section which subvert the intention of another user’s features. This kind of interference appears to require that each user’s service “know”, in some sense, about all features that may be encountered. Furthermore, it appears that in order to resolve the interaction, we must favor one feature over another. Our work offers an alternative to knowing about other features and also to favoring one over another. We use two examples to illustrate the basic idea behind our approach.

One example of a feature which can subvert the intended purpose of another feature is calling number delivery, which provides the number of the originating subscriber to the terminating subscriber. This feature can interact with the unlisted number feature. Whether it does or not depends on the subscriber’s reason for subscribing to unlisted number. If the subscriber doesn’t want her number to be known, calling number delivery subverts her intentions. This example and our resolution of it is discussed in detail in

™Touring Machine is a trademark of Bellcore.
We assume that the subscriber to unlisted number is willing to identify herself, but not to provide her number, and that the subscriber to calling number delivery wants only to know the identity of the calling party. Our proposal to satisfy the intentions of both parties to the call is to deliver the name instead of the number.

In order to illustrate the application of negotiation more fully, we present a second example here which raises similar issues. Suppose that one user has the Terminating Key Code Protection (TKCP) feature, which protects the line by requiring any caller to enter a user-defined key before the call is offered to the line.

Consider the interaction between TKCP and Call Forwarding (CF). First, suppose that the subscriber with Call Forwarding (called F from now on) has forwarded all calls to the line (called L) of the subscriber (called K) with terminating key code protection. Any call to F will be forwarded to L and the caller will be asked to enter a key, which he may not know. Even if he does know the key, he cannot use it unless he has been told which line the call has been forwarded to.

Appropriate resolution of the interaction requires knowing the intentions of the subscribers when they activated the features. The services mentioned above can be used for a number of different purposes. We select just two of these purposes for each feature for the sake of this example.

One purpose of call forwarding is to make sure that calls to a subscriber’s number (e.g., her home phone) still reach her when she is elsewhere. A second is to redirect calls appropriately, as to a secretary or voice-mail.

Similarly, one purpose of Terminating Key Code Protection is to make sure that only authorized users are put through to the line. In other words, TKCP protects the line from unauthorized callers. Another purpose is to permit the subscriber to select which callers he will talk to. In this case, TKCP protects the subscriber from random callers.

Considering these purposes, let’s suppose that a friend is calling F and F has forwarded calls to L because that’s where F is. We argue that the correct resolution of the interaction is to require the key if the purpose of the key is to protect the line. But if the purpose of the key is to protect the subscriber, then the key should not be required when a different subscriber is the object of the call.

On the other hand, if F has forwarded calls to L because F wants K to handle the calls, then the key should be required whether TKCP protects the subscriber or the line.

In most cases the key will still be required, but we see that in one case it is possible to satisfy the goals of both subscribers and complete the call without requiring the key.

This approach is based on the following observation. Typically, the particular implementation of a feature does not represent the ultimate intention of a user, but merely one way to achieve a user’s intentions. There might exist alternative ways to achieve these intentions as well. Such alternatives provide room for negotiation. To explore these alternatives, it is necessary for a user agent — after receiving a request to set up
a call that is unacceptable to that user — to recognize what intention might be behind the request, and to derive from that intention alternative (possibly acceptable) ways to achieve it. Even when a user is not informed explicitly about the intentions of other users, the user may be able to speculate about the intention, based on the information he or she does have, i.e., the received request.

We automate the process of recognizing intentions by building a hierarchy of possible goals on top of primitive system operations. If a subscriber is not willing to agree to a proposed set of operations to initiate or modify a call, an agent of the subscriber uses the hierarchy to infer which goal was intended by the caller, and then checks to see if there is an alternative way to achieve the same goal. If there is an alternative, then instead of rejecting the call outright the subscriber can offer the alternative as a counterproposal.

3 Implementing Negotiation as Part of a Telecommunications System

A negotiating system negotiates about what collection of operations to perform on a given platform. We describe three levels in a telecommunications system that uses negotiation: the platform itself, the negotiating objects, and the user interface.

3.1 The platform

Our mechanism for negotiating assumes that call initiation and modification are described by a collection of operations that determine the form of the call. Policy features and technology features are both related to these operations: the technology features determine which operations are available, and the policy features determine which operations a system user is willing to use. In order to add features rapidly and easily, including new operations, the operations should be provided by an extensible, object-oriented platform for telecommunications [1, 6]. We believe that the issues of interaction between technology features can be addressed by using such platforms.

Policy features determine which operations a system user is willing to execute, but given a collection of operations that have been proposed by one system user, how can we decide which other system users or providers should be asked to authorize the operations? To do this, we require that for each instantiation of each operation's arguments, we can determine which system users or providers should authorize the operation. A user may want to authorize any operation that affects the user, for example, by connecting equipment she owns to other network resources or by billing her for use of network resources. Also, a provider may want to authorize any operations making use of the providers resources. We require that the collection of system users or providers that must authorize a collection of operations be the same as the union of the sets of
Figure 1: The logical structure of a negotiating system for telecommunications.

authorizing users or providers for the individual operations.

A telecommunications system user creates a proposal for initiating or modifying a call as a partially ordered set of the operations that are provided by the platform. (The partial order refers to the order of execution of the operations.) The set of operations is sent to the negotiating system, which returns a possibly different set of operations to be executed. Each of the system users and providers required to do so will have approved the operations in the returned collection of operations.

3.2 The agents and negotiator
A negotiating system works on behalf of entities, which have policies restricting what collections of operations they are willing to perform. It includes agent objects, which represent the various entities in the system and try to carry out their policies, and negotiators, which help the agents reach agreement. Figure 1 illustrates the interrelationships in a negotiating system for telecommunications. In the figure, the entities include system users, local equipment (telephones, videophones, workstations, etc.), and the public network itself. Other potential entities would include information providers, software service providers, and various communications providers.

An agent object is assigned to each entity. The functions of an agent are to produce proposals to initiate or modify a call and to evaluate proposals from other agents, possibly generating counter-proposals. Proposals specify the desired operations on calls. If a proposal is not acceptable to an entity, the agent constructs a counter-proposal. In the following section, we describe a mechanism for evaluating and generating (counter-)proposals.
We distinguish between three quite different organisations for negotiation:

**Direct negotiation.** In this case, agents negotiate directly with each other without the assistance of a mediator.

**Indirect negotiation.** Here a dedicated entity is used to recognize which agents have to approve a proposal and to route proposals and counter-proposals to the appropriate agents. We call such an entity a **negotiator**. This entity could also be used to monitor the progress of a negotiation session, determine when agreement is reached, and even enter proposals into the negotiation process that have proven to be widely acceptable in previous sessions.

**Arbitrated negotiation.** An **arbitrator** takes the complete script of each agent and has sole responsibility for finding a resolution of a conflict. Thus, in arbitrated negotiation the agents don’t need to generate and evaluate proposals.

We use a negotiator object for indirect negotiation between agents. There are several reasons to opt for this organisation. The advantages of indirect negotiation (and also arbitration) over direct negotiation are the following.

1. In contrast to direct negotiation, a separate negotiator can be used to monitor the negotiation process and to make sure that actual progress is being made (otherwise a non-terminating sequence of proposals and counter-proposals might be created).

2. The use of a mediator (negotiator or arbitrator) centralizes communication and makes it easier to include new agents and to communicate with those new agents.

3. The negotiator can use knowledge acquired by monitoring many previous negotiation sessions to propose solutions.

4. Different mediators can be used for different situations, providing benefits of specialization. For instance one can envision ‘smart’ and ‘dumb’ negotiators which provide more or less support and hence could be more or less expensive.

5. The mediator can act as an “honest broker,” in that it prepares, based on proposals it has handled before, an agreement that is about equally good for all parties.

The primary advantage of indirect negotiation over arbitration is that it is more generally applicable. In many cases, including telecommunications systems, the parties to a negotiation may not want to share their preferences with any third party. It used to be that a phone company was seen as a trusted party, but with the current increase in competition between the phone companies makes them less likely candidates for this role.
Another reason to prefer a negotiator over an arbitrator is that a negotiator makes it possible for agents to distinguish themselves from other agents and to improve their results by being smarter than other parties. Actually, this may seen as both a good thing and a bad thing. In general, the problem of malevolent agents trying to take advantage of other system entities must be addressed.

It appears that arbitrated negotiation is computationally less expensive than indirect negotiation since all relevant information is locally available in the arbitrator. However, the arbitrated-negotiation approach requires that all information is sent to the arbitrator, while in case of indirect negotiation information will be sent to some other entity only when it appears to be pertinent to resolving a particular conflict. This suggests that arbitrated negotiation is more expensive in terms of communication overhead. In future work, we may address the benefits and drawbacks of these different methods in more detail. For now, we concentrate on the case of indirect negotiation.

3.3 The user interface

System users and providers need access to two kinds of functionality: one for expressing constraints on the sets of operations they will agree to and the other for submitting proposals. The first functionality is like provisioning. This is how a user defines the policy part of her service. The second functionality initiates call set-up or modification.

A constraint language must be provided to describe which collections of operations can be allowed in which states. Such constraints can be expressed in any logic language, but to avoid intractability we have chosen a simple language based on a small collection of user goals. The user specifies the acceptability (or unacceptability) of each of these goals. In the next section, we describe how the acceptability of a proposal can be inferred from the acceptability of goals.

In this language, an entity must be able to specify a finite set of states that it can enter, enumerate the instantiated goals that are acceptable or unacceptable in each state, and specify the operations that trigger a transition from one state to another. For example, suppose that a platform offers two operations, \texttt{connect}(x,y) and \texttt{hangup}(x,y). A user might define two states, \texttt{busy} and \texttt{idle}, and say that \texttt{connect}(x,y) is acceptable in the idle state and not otherwise, while \texttt{hangup}(x,y) is acceptable in the busy state and not otherwise. The operation \texttt{connect}(x,y) triggers a transition from idle to busy state and \texttt{hangup}(x,y) triggers a transition from busy state to idle state.

When a user wants to initiate or modify a call, he simply needs to select a goal. The negotiating system will do what it can to find a way of meeting the goal.

4 Negotiation mechanism

We now present a description of our negotiation mechanism. The description of the negotiation mechanism uses the example of Terminating Key Call Protection and Call
Forwarding from the preceding section. We use this example to show how the process of generating and evaluating proposals and counter-proposals can be formalized and automated. The process uses a goal hierarchy, a definition of what acceptable proposals are, and algorithms for determining acceptability of proposals and generating counter-proposals. For a more detailed description of the mechanism we refer the reader to [4, 5].

4.1 The goal hierarchy

The basis of negotiation is a goal hierarchy whose lowest-level (basic) goals are operations on a given platform. Higher-level goals correspond to possible goals or intentions of a user. Examples of basic goals in figure 2, containing the goal hierarchy for our example, are get-key(t, u) and connect-station(u, s, v, t). The operation get-key(t, u) asks user u to enter the key for station t. It succeeds if u responds with the correct key. The operation connect-station(u, s, v, t) connects stations s and t on behalf of users u and v.

New goals can be formed by combining other goals in one of two ways. The goal call(u, v) is an abstraction of the goals connect-station(u, s, v, t) and key-connect(u, s, v, t). The goal call(u, v) denotes a call between two users u and v. connect-station(u, s, v, t) denotes a connection between two stations s and t, where user u is at station s and user v at station t. The abstraction relation between connect-station(u, s, v, t) and connect-user(u, v) with uninstantiated parameters is actually short hand and should be interpreted as follows: given an assignment of values, say U and V, to the parameters u and v, the goal connect-user(U, V) is an abstraction of the goals connect-station(U, S, V, T) for any assignment of values S and T to parameters s and t.

Thus, returning to the goal hierarchy of figure 2, the goal call(u, v) is achieved if either connect-station(u, s, v, t) or key-station(u, s, v, t) is achieved for some value of s and t. When one goal is an abstraction of a second goal, then we say that the second goal is a specialization of the first. Achieving a specialization of a goal implies achieving the goal itself.

We also define a composition relation between goals. In our example, the goal key-connect(u, s, v, t) is composed of the two goals get-key(t, u) and connect-station(u, s, v, t). A composite goal is achieved if all its component goals (or subgoals) are achieved. Based on these two types of relationships, we can build a hierarchy, as illustrated in figure 2. The nodes represent goals, the broken lines represent abstraction relationships, and the solid lines represent composition relationships.

Whenever a goal needs to be achieved, that goal can also be achieved by achieving any of the specializations of that goal or by achieving all of its composite subgoals. Thus, the abstraction and composition relations define the different ways that a goal can be achieved. We use these notions of abstraction and composition to define
a specification of a goal. A specification of a goal is derived by recursively replacing an abstract goal by one of its specializations and a composite goal by the set of its components. Let \( \text{home}(x) \) stand for the location of the home (or usual) station for user \( x \). In the example of figure 2, \{\text{connect-station}(A, \text{home}(A), B, \text{home}(B))\} is a specification for the goal \text{call}(A, B). This means that one way for A to call B is to set up a connection between the home station for A and the home station for B. Another specification for the goal \text{call}(A, B) is \{\text{connect-station}(A, S, B, T)\}. This means that another way for A to call B is to connect stations S and T. This will work as intended only if A is at station S and B is at station T. We will discuss how to handle this in section 6. A third specification of the goal \text{call}(A, B) is \{\text{get-key}(A, \text{home}(B)), \text{connect-station}(A, \text{home}(A), B, \text{home}(B))\}. If this specification is used to set up the call, A will be required to produce the key protecting station \text{home}(B) before the connection will be made. By definition of a specification, whenever all goals in the specification of a goal are achieved, then that goal itself is achieved.

We use specifications as the proposals and counter-proposals in our negotiation mechanism. Whenever a proposal is received, the hierarchy can be used to infer what goals it is trying to achieve (for a detailed description of the algorithms see [5]). The following section defines what an acceptable proposal is.

### 4.2 Acceptability

A specification (and thus a proposal) is acceptable for an agent if the entity that the agent represents (e.g., a subscriber) would agree to it. A specification is unacceptable if the entity would not agree to it. Each specification is either acceptable or unacceptable to an agent. However, there are possibly many specifications for each goal, so a subscriber is required only to record the acceptability of goals (a subset of all specifications), not the acceptability of specifications. A goal can be marked either acceptable or unacceptable; or it can be left unmarked. When a goal is unmarked, we call that goal
indeterminate. The acceptability of specifications can be derived from the acceptability of goals they achieve by using a number of rules, as is described below.

All specifications of an acceptable goal are acceptable. This means that by marking a goal as acceptable, a subscriber indicates that no matter how a goal is achieved, it will be acceptable. Analogously, no specification of an unacceptable goal is acceptable. A goal with no marking may have both acceptable and unacceptable specifications.

It can happen that a goal is unacceptable for an agent, while all its component goals are acceptable. For instance, a subscriber may agree to talk to either one of two other subscribers, but not to both at the same time. Likewise, a goal may be acceptable for an agent, while one of the component goals is unacceptable. For instance, a subscriber may not agree to accept a call from a particular person unless a lawyer is also included in the call. Thus, we cannot use the definition of composition to infer formally acceptability of goals from the acceptability of composite goals. However, in such situations we use heuristic rules to make assumptions about acceptability in the absence of explicit assignments for goals. One of these rules states that when a composite goal is neither acceptable nor unacceptable and all component goals are acceptable, then we assume that the composite goal is acceptable. Similarly, a second heuristic rule states that when a composite goal is neither acceptable nor unacceptable and at least one component goal is unacceptable, then we assume that the composite goal is unacceptable.

An acceptability marking of a goal hierarchy is made complete by repeatedly using these inference rules and heuristic rules, until application of the rules will not result in any more changes of markings. Completing the marking will, in general, diminish the searching needed for determining applicability of received proposals.

4.3 Outline of the negotiation process

A negotiation process consists of a number of separate tasks: specification of policies, generating proposals, determining acceptability of proposals, and generating counter-proposals. We illustrate using the example how negotiation would proceed using the definitions of the previous section.

Specification of policies. The task of the agents in our negotiation mechanism is to reach agreements that achieve short-term goals and meet long-term constraints, or policies. In our example, the short-term goal is to set up a call. Long-term constraints are specified by policies. A policy for K (the user that subscribes to TKCP) is that her line cannot be accessed without entering the correct key. Policies can be specified separately from any particular negotiation process by marking an agent’s goal hierarchy with the labels acceptable and unacceptable. Figure 3 illustrates such markings for three subscribers: O, the originator of a call, who will accept any call at all; F, a subscriber to Call Forwarding, who has forwarded calls to line L belonging to subscriber K; and subscriber K, a subscriber
Figure 3: Acceptability markings.
to Terminating Key Code Protection, who has set a key to screen calls from anyone not knowing the key.

Generating a proposal. The negotiation process is initiated when a goal is identified that needs to be achieved (e.g., when a subscriber indicates that she wants to make a call). The goal itself may not be acceptable, but, assuming that it is also not unacceptable, an acceptable specification of that goal could exist. The hierarchy is searched to find an acceptable specification of the goal. For example, any specification of \( \text{call}(O,F) \) is acceptable to \( O \). Suppose that \( O \)'s agent sends \( \{\text{call}(O,F)\} \) as the initial proposal.

Determining acceptability. When a proposal or counter-proposal is received by an agent, it has to decide whether it is acceptable. If it is, the agent can agree to it; if not, an alternative that is acceptable to the receiving agent needs to be generated. Acceptability is determined by searching the hierarchy. If the proposal is a specification of at least one acceptable goal, the proposal is acceptable. If it is the specification of at least one unacceptable goal, the proposal is clearly unacceptable. If none of the goals achieved by the proposal are acceptable or unacceptable, the proposal itself can still be acceptable. Intuitively, if a proposal can be subdivided into parts each of which achieves an acceptable goal, then the proposal is acceptable. Whenever a proposal is not acceptable, a counter-proposal is generated if there are any acceptable ones that have not already been explored. The following illustrates when proposals are acceptable.

1. The proposal \( \{\text{call}(O,F)\} \) is neither acceptable nor unacceptable to \( F \). However, one of its specifications, namely \( \{\text{connect-station}(0,\text{home}(O),F,\text{home}(K))\} \), is acceptable to \( F \). Suppose that \( F \) sends this specification as a counter-proposal.

2. The counter-proposal \( \text{connect-station}(0,\text{home}(O),F,\text{home}(K)) \) does not achieve a goal that is acceptable to \( K \). But it achieves the goal \( \{\text{call}(O,F)\} \), which is neither acceptable nor unacceptable to \( K \) (some of its specifications are acceptable, and some are not). \( K \)'s agent can look for an alternative specification of this goal that will be acceptable to \( K \).

3. The proposal \( \{\text{get-key}(O,\text{home}(K)),\text{connect-station}(0,\text{home}(O),F,\text{home}(K))\} \) is acceptable to \( K \). It is also acceptable to \( O \).

Generating a counter-proposal. When a proposal is not acceptable, we need to find an alternative. We distinguish three cases:

1. The proposal itself may not be acceptable, but sometimes a more detailed specification can be found that is. For instance, the proposal \( \{\text{call}(O,F)\} \) is not acceptable to \( F \), but \( \{\text{connect}(O,\text{home}(O),F,\text{home}(K))\} \) is.
2. The proposal may achieve a goal that is unacceptable, but possibly also another, not unacceptable goal (e.g., an abstraction of that unacceptable goal) for which an acceptable specification can be found. The goal \texttt{connect-station}(0,\texttt{home}(0),F,\texttt{home}(K)) is unacceptable to K, but its abstraction \texttt{call}(0,F) does have an acceptable specification.

3. If neither of these cases hold, we cannot find a solution without changing the acceptability markings in at least one of the agents. This would require relaxing the constraints of at least one of the involved agents. How to do this is an interesting question requiring further research.

The iterative generation of proposals and counter-proposals constitutes a search process through the goal hierarchy to find a specification that achieves the original goal of the agent that initiated the negotiation and is acceptable for all involved parties. The negotiation process terminates when either such a proposal is found, or one of the parties has exhausted all possibilities for generating counter-proposals.

5 Using negotiation on the Touring Machine platform

A negotiating system has been implemented on the Touring Machine platform [1]. This implementation depends on the operations provided by the Touring Machine platform, so we begin with a brief discussion of these operations.

These operations are applied to a Touring Machine structure called a \textit{session}, which corresponds to the idea of a call in today's telecommunications systems. The basic structure used to define a session is called a \textit{connector}. A connector contains a set of \textit{sources}, which send information, and \textit{sinks}, which receive information. Every sink receives from every source. The method used to combine signals from two or more different sources depends on the \textit{type} of the connector. For example, audio signals are combined into one by a weighted addition. Video signals are combined into one by dividing the monitor screen into segments and displaying each signal in its own segment of the screen.

The operations provided by the Touring Machine platform for operating on sessions are:

- \texttt{addClient}(x,y) \hspace{1cm} \text{add clients } x \text{ and } y \text{ to a session}
- \texttt{addCon}(x,t) \hspace{1cm} \text{add a connector named } x \text{ of type } t
- \texttt{delCon}(x) \hspace{1cm} \text{delete a connector named } x
- \texttt{addSink}(x,w,p) \hspace{1cm} \text{add a sink to connector } x \text{, for user } w \text{, using logical device } p
- \texttt{addSource}(x,w,p) \hspace{1cm} \text{add a source to connector } x \text{, for user } w \text{, using logical device } p
- \texttt{setPrivacy}(v) \hspace{1cm} \text{set privacy mode } v \text{ (who can know about the call) } v
- \texttt{setPermission}(v) \hspace{1cm} \text{set permission mode } v \text{ (who can operate on the call)}
To initiate a two-party audio/video call between subscribers a and b, the following collection of operations would be used:

addClient(a,b) specify the parties to the call
addCon(w, ”audio”) addSink (w, a, ”speaker”) addSource (w, b, ”mic”)
addCon(x, ”audio”) addSink (x, b, ”speaker”) addSource (x, a, ”mic”)
addCon (y, ”video”) addSink (y, a, ”monitor”) addSource (y, b, ”camera”)
addCon(z, ”video”) addSink (z, b, ”monitor”) addSource (z, a, ”camera”)

setPrivacy (”group”) describe who can be aware of the call
setPermission (”public”) describe who can join the call

In figure 4, we give the part of the goal hierarchy for the Touring Machine having to do with two-party calls. In this hierarchy, we provide for Call Forwarding and Voice Mail in addition to normal call setup.

To define a goal hierarchy, a provider uses a Prolog-like language. The definition contains a collection of rules, one for each composite goal and one for each specialization of an abstract goal. The rule for a composite goal has the form:

\[ C \rightarrow S_1, \ldots, S_n \]

where C is the composite goal and \( S_1, \ldots, S_n \) are its subgoals. The rule for a specialization of an abstract goal has the form

\[ A \rightarrow S \]

where A is the abstract goal and S is its specialization.

A goal hierarchy will be most useful if the same one is used by all entities involved in the telecommunications system. Otherwise, an entity may receive a proposal involving goals that it doesn’t recognize. If it always accepts such a proposal, it gives up the possibility of constraining the calls it receives. But if it always rejects such a proposal,
Figure 4: The goal hierarchy for two-party Touring Machine calls. Space limitations require the subgoals of the composite goals `connect-station`, `forward`, and `voicemail` to be written in a vertical column.
it will be refusing calls that it might be willing to accept. In a more sophisticated system, it might be possible to query the originator of a proposal as to what it means, and if all subscribers recognize the same base set of operations on the platform, the answer could be given in the form of a new subhierarchy to be added to the original hierarchy. The subscriber would then have to supply the constraints for this new subhierarchy. At present, however, we do not do this, and for this reason, all Touring Machine subscribers use the same goal hierarchy.

To define constraints, a user or provider gives a statement about each possible state that he could be in. This statement specifies which instantiated goals are acceptable and which are not. It has the form:

\[
\text{when } S \text{ acceptable } G_1, \ldots, G_n \text{ unacceptable } U_1, \ldots, U_n
\]

where \( S \) is the state name and the \( G_i \) and \( U_i \) are instantiated goals. (An instantiated goal is a goal whose arguments are all constants or the symbol \( \text{ANY} \), meaning any value). Each entity prepares an individual collection of constraints, representing its own policy, based on the goals in the goal hierarchy.

The system entities currently represented by agents in the Touring Machine system are users and stations, or locations. At present, the system itself doesn’t have an agent, because any proposal is acceptable to it. However, if there were restrictions on resource usage or if billing were included in the system an agent for the system would be useful.

Some issues that a designer of a telecommunications system incorporating a negotiation system must face are:

**What are the user goals and how are they related?** We have found it most intuitive to make \( \text{call}(a_1, \ldots, a_n) \) a high-level goal (initiating a call among the specified parties), and then define as specializations various ways of handling the call, including forwarding, voice-mail, postponement, and break-in. However, we can also define even higher level goals than call, such as the goal to set up a set of calls all delivering the same message. This would be a composite goal using call repeatedly as a subgoal.

**Which agent should authorize which operations?** A variety of issues may arise in connection with this. For example, would a company agent authorize all calls to or from any employee agent, or would the employee agents have individual policies? Or would both the company and the employee agent be involved in each call to an employee?

**How will the user interface be designed?** One possibility is to provide a hierarchical system of menus, based on the goal hierarchy itself. A generic interface could be built that would work with any goal hierarchy. Another possibility is to use an existing interface, and relate requests from it to goals in the goal hierarchy to initiate
negotiation. The Cruiser interface to the Touring Machine system has been used in our work [3].

6 Examples of AIN or CS-1 features

The description of the negotiation mechanism in section 4 contained examples of how the features Call Forwarding and Terminating Call Key Protection can be provided by our mechanism. Call screening features are also easily provided by defining appropriate acceptability predicates for parties to a call.

In this section, we illustrate how our mechanism can be used to provide several other complex features on the Touring Machine platform. These features are closely related to the IN features defined in the CCITT Q1200 IN standard [2, Q1211, pages 29-45], and the implementations described here can be used as a guide for how those IN features would be implemented using negotiation.

Some IN-like features have been implemented on top of the Touring Machine system, but not all IN features are meaningful in that environment because of the difference in functionality provided by the current public network or the Touring Machine system. For instance, Abbreviated Dialing is not meaningful in the Touring Machine system, since the ‘dialing’ interface is menu-based.

6.1 Re-routing features

We describe in detail the Touring Machine feature Call Following. Although Call Following is not an IN feature, IN features such as Follow-Me Diversion and Destination Call Routing can be implemented similarly. This feature allows calls to a user to be redirected automatically to the station nearest to where the user is at that moment. The user’s location is tracked by the Touring Machine name-server, which is notified whenever the user’s location changes. As long as a user wears the badge, her location will be known to the system.

The goal hierarchy that is used to implement this feature is similar to the one used for Call Forwarding (see Figure 5). The goal connect-station(B,S,V,T) is acceptable to user B in this example if and only if user B is at station S. Similarly, the goal connect-station(U,S,B,T) is acceptable to user B if and only if user B is at station T. The predicate at(u,s) asserts that user u is at station s. We assume that a user’s agent knows his or her location. If necessary, the agent can determine the location by querying the Touring Machine name-server.

Imagine the following situation. User A wants to call user B, who has Call Following (e.g., the hierarchy of Figure 5) and is currently visiting user C. If A’s agent sends the proposal call(A,B), then B’s agent will decide that the proposal is indeterminate and look for an acceptable specialization, i.e., a goal connect-station(A,S,B,T) such that at(B,T) is true. The agent can find such a station T by querying the name-server. This
Figure 5: Goal hierarchy for Call Following. The touring machine parameters that specify connector names have been omitted.

results in the counterproposal \texttt{connect-station}(A,S,B,T) where user B is at station T.

According to B’s marking of the hierarchy, it doesn’t matter what station is used for A. This may not be precisely true, but it is not reasonable to expect user B’s agent to know where user A wants the call directed. Thus B’s agent relies on A’s agent to supply this information. A’s agent may respond with a further counterproposal with yet another station substituted for S.

Alternatively, if A’s agent sends the proposal \texttt{connect-station}(A, \texttt{home}(A), B, \texttt{home}(B)) (where \texttt{home}(A) and \texttt{home}(B) are A’s and B’s default stations), then B’s agent will substitute the station at which B is currently located for \texttt{home}(B).

Follow-Me Diversion and Destination Call Routing can be implemented similarly. However, for these features an agent wouldn’t need to query the name-server. Instead, its own data would indicate where to find the user.

6.2 Features that deal with busy states

Several features deal with busy states: Call Waiting (CW), Call Forwarding on Busy Line (CFBL), Completion of Calls to Busy Subscriber (CCBS), etc. Each of these features provides an alternative way of handling a call when the callee is busy and can be represented as different specializations of making a call. We show how such features can be implemented in our mechanism.

Figure 6 shows goal hierarchies for users A and B. As described in section 5, B has specified a marking of the goal hierarchy for the busy state, which is the marking shown here. The goals \texttt{call}(u,v) and \texttt{connect-station}(u,s,v,t) are defined as in the previous examples. The goal \texttt{retry}(u,v) denotes that call set up will retried...
Figure 6: Goal hierarchies for dealing with busy states.
later, either by the caller (retry-A) or by the callee (retry-B). The goal cfbl(u,v) implements CFBL.

In this example, we assume that user A tries to call B and that B is busy. User A is subscribed to the CCBS feature. B is subscribed to CFBL. Calls that are coming in for B when B is busy are re-routed to the station of user C.

A's agent sends call(A,B) as a first proposal. Since B is in the busy state, B's agent uses the hierarchy of figure 6 and finds that this proposal is indeterminate. Looking at specializations of call(A,B), the agent also finds that no specialization of connect-station(A,s,B,t) is acceptable to B. There are two possible counter-proposals: connect-station(A,s,B,home(C)) and retry(A,B). The first of these is acceptable to A and forward the call to C's station. The second counter-proposal will result in a new counter-proposal by A (retry-A(A,B)), which is acceptable to B.

A call retry can be implemented for the Touring Machine system by using database triggers. A user can define a temporary trigger in his agent that catches the event that a call is terminated. The agent then performs a predefined action when the event occurs. The goal retry-A(A,B) is implemented by defining a trigger on the termination of B's current call and specifying a corresponding action of setting up call(A,B).

7 Conclusions

We claim that negotiation is more practical than conventional methods for solving the problem of policy feature interactions because it avoids the need for service providers or users to know what policy features other service providers or users have. This claim is based on the fact that conflicts between policies are settled at run-time by an automatic negotiation system that detects and resolves the conflicts (or interactions) at run-time. More conventional methods would examine pairs of features individually, and manually determine the resolution policy for each pair. Furthermore, negotiation considers the actual intentions of the user at run-time, so that individual and even changing policies are possible.

We have already established in earlier work that this kind of negotiation can be automated [4, 5]. This required defining an architecture for negotiation and developing algorithms to do the negotiation. These algorithms detect and resolve multi-party interactions at run-time. A single-party interaction can also be detected automatically, although the resolution is up to the user. An algorithm for completing an initial marking of goals in a goal hierarchy is described at the end of section 3 of [5]. This algorithm will also detect inconsistencies between constraints (an inconsistency is found if some goal is marked both acceptable and unacceptable by the algorithm).

In the present paper we have shown how to add a negotiating mechanism to a telecommunications platform. As long as the platform offers a well-defined set of operations on its interface, this mechanism will work. Furthermore, the properties of the
negotiating system guarantee that some agreement will be found by the system if any
exists that achieves the goal of the initiating subscriber [5].

Challenges that remain to be addressed include:

- How expressive can the language defining constraints be, before it becomes in-
  tractable to detect single-user feature interactions?

- How efficient is negotiation?

- Can constraints be used to guarantee safety properties?

- Can an automatic mechanism be devised for relaxing user constraints when no
  agreement can be reached otherwise?

References

V. Mak, L. Ruston, M. Segal, M. Vecchi, A. Weimib, and S.-Y. Wu. Touring
Machine: a software infrastructure to support multimedia communications. In
MULTIMEDIA ’92, 4th IEEE COMSOC International Workshop on Multimedia

[2] CCITT. New Recommendations Q1200 – Q series: Intelligent Network Recom-


M. Huhns, M.P. Papazoglou, and G. Schlageter, editors, Proceedings Interna-
tional Conference on Intelligent Cooperating Information Systems (ICICIS-93), pages

In Proceedings 12th International Workshop on Distributed Artificial Intelli-