Tingxue Huang and Kenneth J. Turner. Policy Support for H.323 Call Handling, Computer Standards and Interfaces (pre-publication version), January 2005.

Policy Support for H.323 Call Handling

Tingxue Huang and Kenneth J. Turner

Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, UK

Abstract

The need for policies to control calls is justified by the changing face of communications. An overview is given of a general architecture and language for policies. It is then shown how these are adapted for control of calls using the H.323 multimedia communications standard. Policy support for H.323 was created by extending an open-source gatekeeper. The core policy language has been specialised to deal with call control in general, and for H.323 in particular. Examples are given of policies for H.323, illustrating how traditional features can be made to work more flexibly through use of policies. Examples are also provided of policies specific to H.323, and policies that can take advantage of other information such as the context of a call.

Keywords: H.323, Internet Telephony, Policy

1 Introduction

1.1 The Changing Face of Communications

Communication has become increasingly pervasive and intrusive. Calls may be received at work or at home, on fixed-line or mobile telephones. Anyone may call at any time about any subject. Calls may be placed using traditional or Internet telephony. Voice may be supplemented by video, data or other media. Call devices may include conventional telephones, mobile telephones, softphones, PDAs, voicemail, email message transfer agents, and web browsers. As a consequence of these factors, there is an urgent need to enable users and organisations to control their calls.

Traditionally, call control has been supported by network services normally called features. For example, Call Forward Busy Line allows the user to divert calls when busy, or Call Waiting allows the user to hold callers instead. However call features are a somewhat dated approach, and suffer from several disadvantages.

Features stem from a network-centric era in which call control was performed entirely by network operators. This was beneficial in that features were defined by and under the control of

Email addresses: thuang@site.uottawa.ca (Tingxue Huang), kjt@cs.stir.ac.uk (Kenneth J. Turner).

a single network operator. Services are increasingly being deployed at the edge of networks. These may be provided by third parties for other users (e.g. Parlay/OSA, *www.parlay.org*), or may be defined by end-users and their organisations.

Features tend to be low-level, inflexible, implementation-oriented and imperative. Some parameterisation is possible (e.g. the choice of forwarding number) but is very limited. In contrast, many researchers favour the use of policies for call control [6]. Policies tend to be high-level, flexible, goal-oriented and declarative. Policy support has grown out of areas such as distributed systems, access control, and Quality of Service management. This paper reports a new application for policies: call handling.

Internet-based calling presents a striking difference from conventional telephony. The Internet philosophy is to have a simple and efficient core network, with complex facilities provided in the hosts and terminals. Thus in Internet telephony, the approach has been to support advanced call processing in the endpoints. In contrast, conventional telephony emphasises the central role of the network in providing services to simple terminals.

H.323 [16] is a widely adopted set of standards for multimedia communication. The focus of this paper is on policy-based control of H.323 calls. A number of supplementary services have been defined for H.323. As will be seen, policies can emulate these services but do much more and in a more flexible manner.

1.2 Policy Support for Calls

Policies promise to be the replacement for features in Next Generation Networks, which are likely to be based on Internet standards and to support services at the edge of the network. The following examples illustrate what can be achieved with policies, and some of the issues that arise from their use.

When I am busy, calls from customers should be forwarded to a team member, calls from colleagues should have my current schedule spoken to them, and calls from friends should be passed to voicemail. A conventional feature is restricted to a simple concept of busy, to fixed forwarding numbers, and to voice media. In a policy, the concept of *busy* is flexible: it is not simply that my telephone is off-hook. I may be busy, for example, when meeting with my manager but not with a fellow team-member. Policies can use general concepts such as *team member* or *friend*, instantiating these according to the context. Policies can also use different media such as voice, video or web services (according to call capabilities).

My manager must be conferenced into any calls from the Press. This policy is goal-oriented, and uses the general concepts of *manager* and *Press*. A more abstract form of this policy might be: *Employees are not allowed to speak to the Press on their own*. Policies may range from low level (when they are equivalent to features) to high level. Because policies may sometimes equate to features, some workers do not distinguish them. However the authors contend that a useful distinction can be made between them.

Employees may not make personal calls unless it is an emergency. Policies can be defined at any level of an organisation. This policy is organisation-wide, and encompasses broad issues such as what constitutes a *personal call* or an *emergency*. In general, policies apply to domains. These may be hierarchical (e.g. user, team, department, organisation). However a user may belong to complex, overlapping domains. A user at work may thus be subject to policies from several domains such as her own personal one, the ‘director’ category, the work social club, and her company. Furthermore, the policies that apply may depend on the context and the roles of the participants. For example, policies may permit two employees in different companies to call each other about work, but not about a personal matter.

I prefer to speak to Anne if Barry is unavailable. This personal policy might conflict with Barry’s policies. For example, when Barry is out of the office he might wish to forward his calls to Colin. Policy conflicts can arise among user policies, organisation policies and network provider policies. Policy conflict is the analogue of feature interaction [3], whereby independently designed features may interfere with each other. In the traditional approach, features are under the control of one network operator so it is relatively easy to determine and resolve feature interactions. However policies for calls may be deployed anywhere by anyone, so policy conflict is much more difficult to handle. Policies also reflect user intentions, so the possibilities for conflict are much larger. Fortunately, the same information about user intentions provides a much richer basis for resolving conflicts. One of the major problems in handling feature interactions is knowing what the user really wanted to do. In fact feature interactions are often resolved using fixed priorities, instead of considering the call context.

As has been seen, policies offer much more flexibility than features. However, considerations of complexity and call processing need to be taken into account. Although policies appear to be more complex than features, the feature logic embedded in typical telephony systems is in fact very elaborate. For example, the AIN/IN (Advanced/Intelligent Network [14]) defines an intricate architecture for call processing: Service Switching Point, Service Control Point, Service Plane, Global Service Logic, Service-Independent Building Block, etc. A comparatively simple policy language can embody the majority of features, and do much more besides. Because policies are typically higher level, end users could find them more difficult to formulate. It is therefore vital to provide a friendly user interface to a policy system.

It might appear that adding policy support would considerably increase call processing time. In fact, the extent of call processing undertaken by the AIN/IN is comparable to that undertaken by a policy-based system. However if the policy system undertakes to detect and resolve conflicts among distributed policies, this would indeed add significant overhead. In fact a trade-off can be made between the sophistication of conflict handling and the time taken. If simple priorities are enforced as in the AIN/IN, conflict handling is not time-consuming. But if users require advanced handling of conflicts among arbitrary policies, they must be prepared to accept the delays this will introduce.

CPL (Call Processing Language [18]) allows users to define how they wish calls to be handled. However CPL is limited in a number of ways that make it unsuitable for general call control. For example, it is restricted in its network bindings (currently H.323 and SIP). More seriously, CPL gives very limited control over calls, specifically just call setup. It is also desirable to handle mid-call events (e.g. when a new party is added to a call) and end-call events (i.e. when a call is disconnected).

Policies have been used in many kinds of management tasks such as admission control, health-care, network management, quality of service, and security. Policy language developments in industry have largely focused on network management and QoS (Quality of Service). For example, Cisco have developed policy support for control of security and QoS in routers. Lucent and Bell Labs developed PDL (Policy Description Language) for network management. Hewlett-Packard's PolicyXpert was also focused on network management. The IETF standard for COPS (Common Open Policy Service) is intended as a protocol for managing QoS. None of these efforts is of direct relevance to call control.

In the context of this paper, policies are interpreted as the rules for how calls should be handled. Policies lend themselves well to networked applications, where the very distribution demands careful management. Many researchers see policies as important in future call handling [6]. Despite this, call handling systems have attracted little policy support. [1] uses fuzzy policies as a means of resolving feature interactions. [19] discusses the kinds of policies needed in call control. [19] presents a detailed evaluation for call control of the well-known Ponder language [5]. It was found that Ponder was only partly suitable for this purpose. The authors and co-workers have therefore defined a policy language for use with call control systems. A number of considerations required the definition of a specialised policy language for this field.

Call control places different demands on a policy system, and of course it requires specialised support in a communications setting. The events, conditions and actions that arise in call control are completely different from, say, those required in network management. Ideally a policy language should be capable of specialisation for various application domains. The authors' team has therefore defined a core policy language, clearly separated from its use for call control. Only some existing policy languages are capable of this kind of adaptability.

Policy languages for network or systems management often exploit the distinction between the subject of a policy (that performs an action) and the target of a policy (that is acted upon). In call control, it becomes unclear how to designate the subject (caller, call, network?) and the target (callee, call, network?). As a result, policy languages for management can be difficult to map onto call control [19]. Call control also deals with a dynamic and unpredictable configuration of communicating entities and policies. This is quite different from the domains in which policy languages are typically used.

Call control policies need to be comprehensible to ordinary subscribers. The policy language and its supporting system must therefore be designed in a user-friendly fashion. In contrast most policy languages require specialised technical expertise, being designed for developers or administrators. Communication is global, so policy support must be international (i.e. multilingual). With end users defining policies for calls, there is much greater scope for conflict. An effective technique is therefore required to detect and resolve conflicts in a way that end users can relate to.

Distributed definition of policies can lead to incompatibilities among them. Policy conflict resembles the extensively studied feature interaction problem. A general discussion of this problem appears in [2–4]. It is argued in [20] that some techniques from feature interaction can be adapted for detection and resolution of policy conflicts. Nonetheless, conflict handling remains a challenging task. Handling policy conflicts in call control is the subject of a future paper by the authors, so the topic is only touched on in this paper.

1.4 Structure of the Paper

Section 2 gives an overview of the policy system. The architecture is presented, along with a summary of the core policy language. Section 3 then shows how the policy system was specialised for H.323, using open-source gatekeeper code. This is linked into a variant of the policy language that handles (H.323) call control. Policy examples are given as concrete illustrations of the approach. A policy wizard is introduced as a user-friendly means to create policies.

2 The Policy System in General

This section provides an overview of the policy architecture and policy language. The detailed implementation strategy is described in [22,23,26,27]. The policy system architecture is shown in figure 1. The arrows in this figure represent socket interfaces, so the approach is truly distributed. In practice, however, several of the logical entities may reside on the same physical system. The architecture is divided into three separate layers in order to isolate the various aspects of policy support.

2.1 Communications System Layer

The communications system layer is likely to be a complex subsystem in its own right. For example, the authors' team has developed policy support for H.323 multimedia communications, SIP (Session Initiation Protocol [24]), and PBXs. In principle, any communications mechanism may be used. An important goal has to been to make policy support as independent as possible of the underlying communications.

The main (and plausible) assumption is that the communications layer contains servers where call messages can be intercepted. In H.323 the call messages are mostly handled by gatekeepers, though possibly also by terminals. Use with SIP mainly involves proxy servers, though call

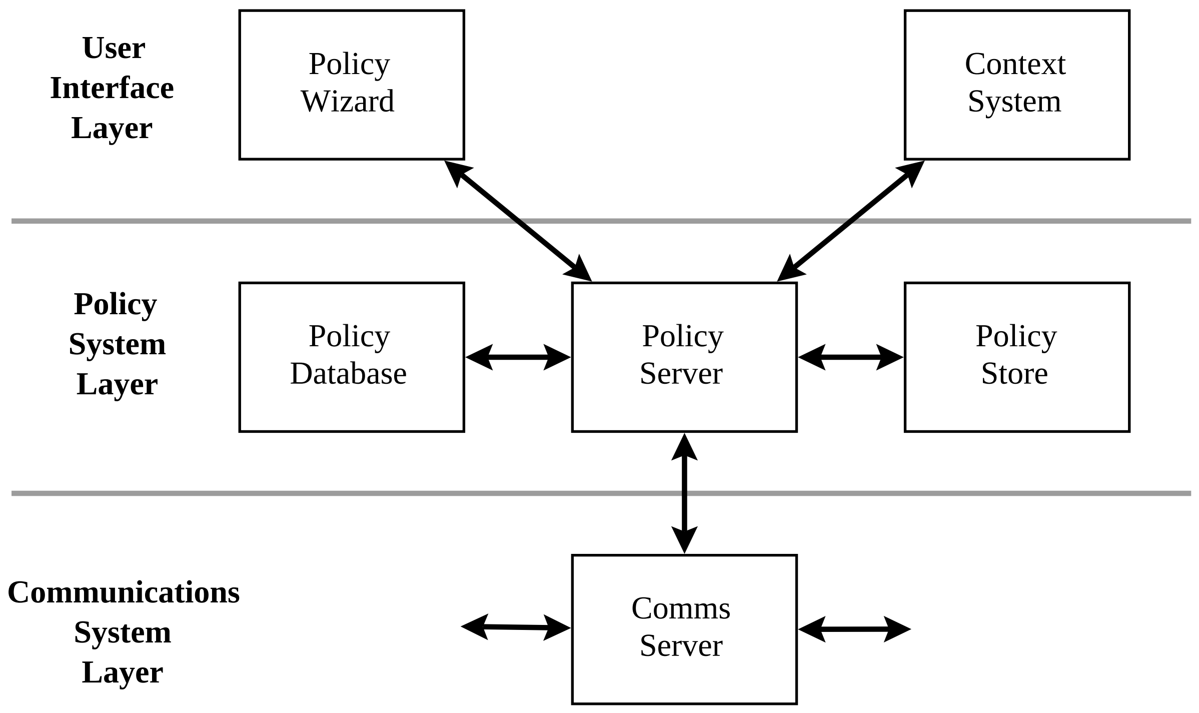


Fig. 1. Policy System Architecture

messages could also be intercepted in user agents or in redirect servers. In the AIN/IN, the communications servers would be Service Switching Points.

Policies need to be aware of call events. Most obviously these are the signals that establish calls. However the approach also allows for mid-call or end-call policies. Policies may also be invoked when users register or are authenticated. The choice of call messages to be intercepted depends on the particular communications subsystem.

It must therefore be possible to intercept call messages in the communications server. This is straightforward if it is open source, such as GnuGk (GNU Gatekeeper) for H.323, or SER (SIP Express Router) for SIP. However even for proprietary products, there are likely to be hooks for third-party call processing. For example, this is true for the Mitel 7000 PBX which has been integrated by the authors' team into the policy system.

To adapt a communications server for use with policies it is necessary to add message interception code. So as not to expose the details of the communications system to the policy system, the policy interface is protocol-independent. Call events are notified in a simple and neutral format: a list of parameter-value pairs, sent from a communications server to a policy server. The response from the policy server is also protocol-independent, specifying what actions should be taken by the communications layer.

A mapping table defines how protocol information relates to policy information. For example, *Setup* in H.323 or *Invite* in SIP corresponds to the policy trigger for a call attempt. Conversely, the policy action to forward a call is mapped to *Divert* in H.323 or a *3XX* response in SIP.

2.2 Policy System Layer

The policy system layer [21] comprises the heart of the system. A policy server is responsible for activating policies in response to call events (e.g. a call attempt or a call hang-up). The policy server is essentially independent of the underlying communications layer. Policy triggers are generic events such as registration, call attempt or disconnection. Policy conditions refer to generic parameters such as caller, call type or role. Policy actions refer to generic instructions such as forwarding, interrupting or rejecting a call.

A communications server contacts a policy server using a socket interface. This means that the policy server may be located anywhere in the network, and that the relationship between communications servers and policy servers can be whatever is required. An organisation might decide to have just one policy server, or might decide to have one per department. The policy server code can even run on the same system as the communications server. Private users might make use of public policy servers, or these might be operated by their service provider. There may be one policy server per domain, or these may be replicated for load sharing or resilience. It is sufficient that each communications server knows which policy server to contact.

A policy database is a conventional relational database (MySQL), containing static information required by a policy server. For example, the policy server aims to be independent of the underlying communications layer. This is achieved through a database table that maps between protocol terms and policy terms.

A policy store contains dynamic information required by a policy server. For example, the details of current policies and user contexts are stored there. Although the policy store could be the same kind of system as the policy database, the demands made on it are rather different. Policies are represented in XML form. Good support is therefore needed in the policy store for XML-based queries. IBM TSpaces (tuple space server, *www.alphaworks.ibm.com/tech/tspaces*) is a convenient solution for this need.

Policy storage and policy enforcement are deliberately separated, allowing policy servers to share policy databases and policy stores. The link between a policy server and a policy database or store is socket-based, and so allows for easy distribution and load sharing. However these logical entities may also reside on the same physical system. The choice is made by each organisation with regard to economy, performance and robustness.

A policy server queries a policy store for policies that might apply to a call. For example when a call is initiated, policies for the caller and/or the callee are retrieved. In fact all applicable policies are retrieved, including those from other domains to which the user belongs (such as organisational ones). The policy server then filters the policies according to the contextual information. The time or subject of a call, for example, might permit certain policies and disallow others. It is then necessary to check these policies for consistency. Some policy conflicts are obvious, e.g. if there are contradictory actions such as forwarding the call to different addresses. Other conflicts may be more subtle, e.g. if the caller prefers one action but the callee strongly wishes another.

A taxonomy of policy conflicts is presented in [23]. This identifies five dimensions in which policy conflicts can arise: the policy types, their domains of application, the call party roles, policy attributes such as their time-frames, and policy modalities such as ‘must not’ or ‘prefer’. A broad strategy has been developed for detecting and resolving policy conflicts [27]. This is too detailed to report here, and is in any case work in progress. Briefly, all the policies relevant to a call are collected on a ‘blackboard’ (a temporary network storage area). The policies are then checked for conflicts along the five dimensions. In the case of incompatibility, special resolution policies define how the conflicts can be removed. Implicit priorities may be used. For example an organisational policy normally overrides a personal policy, or a ‘must not’ policy normally has precedence over a ‘prefer’ policy. Policies may also be relaxed along their dimensions to remove conflicts. For example, timeframe differences may be reconciled, or media inconsistencies might be unified.

2.3 User Interface Layer

The user interface layer allows end users to interact with the policy system. The policy wizard [26] has the important task of allowing users to formulate and edit policies. Considerable effort has gone into making the wizard user-friendly. For example, policies are presented in structured natural language (of the user’s choice). The policy wizard uses a familiar web interface, and provides extensive online help.

The context system embodies information about the context of a call. For example, it provides information about presence and availability to the policy system. This might be given by the user, say, or by an active badge system. As an example, a context system has been implemented to derive presence and availability from a user’s diary in Microsoft Outlook form. The user can also define this information manually. Other information that can be provided by a context system includes the roles of the call parties (e.g. derived from an organisation chart) or their capabilities (e.g. which languages they speak).

2.4 The Core Policy Language

The policy language is called APPEL (the ACCENT Project Policy Environment/Language, a play on the French word for ‘call’). This is cleanly separated into a core language and its specialisation for different purposes such as call control or conflict resolution [22]. The language is XML-based, and is defined by an XML schema. Its chief features are as follows:

Generic Policies: The language supports parameterised policies that are instantiated with particular values for policy variables. This is useful, for example, in template policies for non-technical users.

Domains: Individual policies apply to whoever defines them. However policies may be defined for domains, i.e. sets of users such as all those in some organisation.

Modality: A policy may define a preference (e.g. ‘must’, ‘should’ or ‘prefer’, along with negative versions of these). This information implies a weight for the policy that is taken into account if conflicts have to be resolved.

Rule Combinations: Policies comprise rules that may be combined in various ways, e.g. subject to some condition,tried in sequence, or executed in parallel.

Rules: Policies are in ECA form (Event-Condition-Action). An optional trigger specifies the external event that may activate a policy. Triggers may be combined with ‘and’ and ‘or’. An optional condition defines the circumstances in which a policy may apply. Conditions rely on information established by triggers, such the caller, the time, or the topic of a call. Conditions may be combined with boolean operators. An action gives the effect of a rule. Actions may be combined with various operators such as ‘and’ and ‘or’.

The core policy language defines the structure for triggers, conditions and actions. However it does not define the specifics of these because they are application-dependent. Section 3.3 explains how the core language is specialised for use in H.323 call control, while section 3.4 illustrates the language through policy examples for H.323.

3 The Policy System for H.323

H.323 is a complex set of standards for real-time multimedia communications. This section briefly presents H.323, and then explains how the policy system was specialised for use in this context.

3.1 H.323 Architecture

H.323 defines a protocol stack for real-time multimedia communications, including telephony. A tutorial on H.323 can be found online at *www.iec.org/online/tutorials/h323*. H.323 operates over packet-based networks that may use IP or IPX over LANs and WANs. An H.323 communications network is subdivided into areas called zones that mainly consist of terminals. An H.323 zone is independent of the network topology, and may comprise multiple network segments. A gatekeeper manages an H.323 zone, and provides services such as addressing, authorisation and bandwidth control.

The H.323 protocol stack in figure 2 shows the media and signalling protocols. H.323 is an umbrella for an elaborate collection of standards, shown as shaded boxes in the figure:

H.323 [16] specifies the overall system architecture.

H.225 [12] supports RAS (Registration, Admission and Status) between endpoints and gatekeepers. A channel is created for RAS prior to the establishment of any other channels. H.225 also adapts the Q.931 standard [10] for general call signalling. An H.225 call signalling channel is established between two H.323 endpoints, or between an endpoint and a gatekeeper.

H.245 [13] supports end-to-end control signalling for functions such as exchange of capabilities, opening and closing logical channels, and flow control.

H.450 [9,11,15,17] defines call services for H.323.

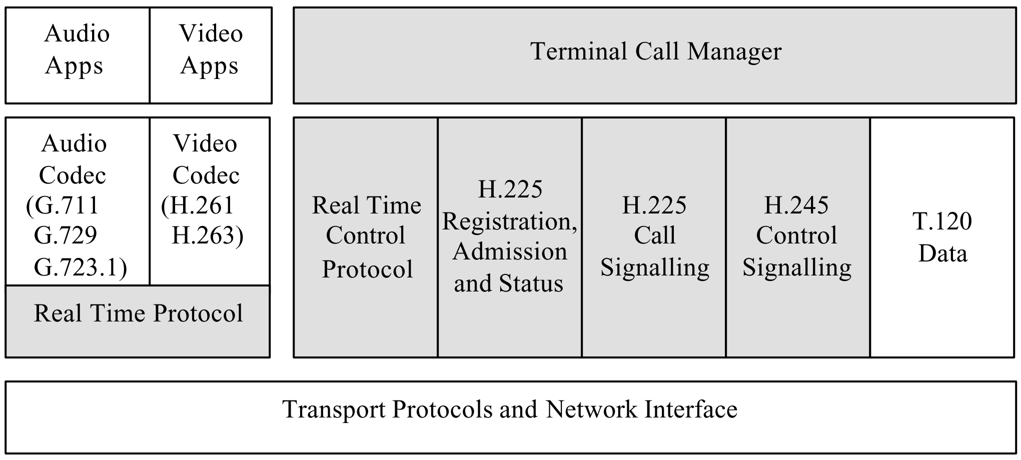

Audio Apps
Video Apps
Terminal Call Manager
Audio Codec (G.711 G.729 G.723.1)
Video Codec (H.261 H.263)
Real Time Protocol
Real Time Control Protocol
H.225 Registration, Admission and Status
H.225 Call Signalling
H.245 Control Signalling
T.120 Data
Transport Protocols and Network Interface

Fig. 2. H.323 Protocol Stack

T.120 [8] supports the transmission of data in an H.323 call.

RTP/RTCP (Real-Time Protocol/Real-Time Control Protocol [25]) is used for end-to-end delivery and control of real-time audio and video. All H.323 terminals must support at least the G.711 audio codec; other audio codecs and all video codecs are optional.

A full H.323 call has three connection stages: call admission, call signalling and call control. Initially the caller is authorised to contact the callee. The endpoints then exchange information about their system capabilities, and decide their role as master or as slave. Finally the endpoints open media channels for audio, and optionally video. Call tear-down closes the media, signalling and admission channels.

Two H.323 terminals can communicate directly without a gatekeeper. However, the use of a gatekeeper is normal in a managed network. For maximum control, all messages except for the media and data streams should pass through the gatekeeper.

3.2 Communications Server Support for H.323

As discussed in section 2.1, each communications layer needs to be tied into the policy system. For H.323, this requires writing a dedicated module for a gatekeeper. As an open-source project, GnuGK (GNU Gatekeeper, *www.gnugk.org*) was ideal for this purpose. GnuGK has become increasingly popular in the H.323 community. GnuGK consists of two main modules: a RAS server/client that deals with Registration, Admission and Status; and a proxy server that deals with call signalling, control signalling and media streams. The two modules are separate, but work in harmony through shared information such as the registration table, the routing table and the call table.

[7] explains in detail how GnuGK was adapted for use with policies. Because policies affect only the signalling procedures, policy support requires additions to the code for RAS, call signalling and call control. Extended versions of these modules were developed using inheritance techniques. The modified classes intercept H.323 messages like *Admission Request, Registration Request* and *Setup*. A new class was added to GnuGK to convey information in these

Element	**Examples**
Trigger	bandwidth_request, connect, disconnect, no_answer(period), register
Condition	active_content, bandwidth, callee, caller, capability_set, destination_address, signalling_address, source_address, traffic_load
Action	add_caller(method), add_medium(medium), confirm_bandwidth, connect_to(address), fork_to(address), forward_to(address), reject_bandwidth(limit), reject_call(reason), remove_medium(medium)

Fig. 3. Example Triggers, Condition Parameters and Actions for H.323

messages to the policy server. This class also implements any actions that the policy server dictates. Checking for no answer to a call requires a new class that runs a timer in a separate thread. The gatekeeper module uses the technique described in section 2.1 to map between H.323 terms (e.g. *Admission Request* and *Registration Request*) and policy terms (e.g. *connect* and *register*).

3.3 The Policy Language for H.323 Call Control

The core policy language in section 2.4 was specialised for call control in general, and for H.323 in particular. This requires specific triggers, conditions and actions to be defined. Figure 3 summarises the additions for H.323. Some of the language elements are unique to H.323 (e.g. *bandwidth_request* and *signalling_address*), but most are useful for control of any communications layer. Many of these generic elements (e.g. availability triggers and call attribute conditions) have been omitted here.

3.4 Policy Examples for H.323

The following illustrates how H.323 can be controlled more effectively through policies. The earlier examples demonstrate that policies can represent standard call features. The later examples show that policies can be used to control a much wider range of H.323 aspects. For brevity, the examples omit some XML ‘red tape’; the obvious XML closing tags are also omitted.

3.4.1 Call Forwarding

Policy support is provided for forwarding under various conditions: unconditionally, on busy, or on no answer. This is also possible with the standard H.323 call diversion service [11], but policies give considerable additional flexibility. An efficiency advantage is that the forwarded-to terminal need be found only once, whereas H.323 must look for the forwarded-to terminal several times. Another advantage is that the policy system can detect and resolve forwarding conflicts. Suppose the callee prefers to forward calls when busy, but the caller insists on speaking to this individual. The two policies are in conflict, perhaps being resolved by choosing the stronger policy. As a further policy example, different callers might be forwarded to different numbers. A standard call forwarding feature lacks this kind of flexibility.

The following is a simple forwarding policy. Anne is the policy owner, and is also the person to whom the policy applies. Policies are normally enabled, but can be temporarily disabled. The date a policy was last changed is recorded in XML date and time format. This policy could be extended in various ways. For example, the policy is currently neutral about the desirability of forwarding. The preference 'should' could be added to indicate that Anne does not insist on forwarding. A range of dates could be defined for when the policy applies, for example when Anne is on holiday.

In the policy below, Anne's incoming calls (*connect_incoming*) are forwarded to Barry. Arguments to policy elements, e.g. *forward_to(arg1)*, are given as XML attributes.

```
<policy owner="anne@acme.com" applies_to="anne@acme.com"
 id="Forward incoming calls" enabled="true" changed="2004-08-12T11:33:00">
   <policy_rule>
      <trigger>connect_incoming
      <action arg1="barry@acme.com">forward_to(arg1)
```

Policies may prohibit certain actions. In the following policy, the administrator for Acme has decided that calls must not be forwarded to junior members of staff. The policy applies to both incoming and outgoing calls (*connect*). Domain names are prefixed with '@' by analogy with email addresses. In this example, the domains are *acme.com* (everyone in Acme) and *juniors.acme.com* (all junior staff in Acme). These domains are defined separately in the policy system as lists of specific individuals.

```
<policy owner="admin@acme.com" applies_to="@acme.com"
 id="Never forward to juniors" enabled="true" changed="2004-08-24T11:43:00">
   <preference>must_not
   <policy_rule>
      <trigger>connect
      <action arg1="@juniors.acme.com">forward_to(arg1)
```

In the case of a prohibition, the argument of an action may be omitted to mean any value. For example, a policy might state that emergency calls must not be rejected for any reason. An emergency call could be explicitly indicated by the caller, or could be identified by the use of a particular address (equivalent to 911 or 999).

In a variant of forwarding, a call attempt may be forked – sent to multiple addresses. Suppose that Barry is regularly away from his office. When someone calls him there, a call attempt should also be made to his cellphone (reached via a PSTN gateway). Whichever device answers first decides which call leg is actually connected.

```
<policy owner="barry@acme.com" applies_to="barry@acme.com"
 id="Try office and cellphone" enabled="true" changed="2004-08-20T11:41:00">
   <policy_rule>
      <trigger>connect_incoming
      <action arg1="6781234567@pstn.com">fork_to(arg1)
```

3.4.2 Registration and Admission Control

A registration policy is enforced during H.323 authorisation and authentication. Each endpoint in an H.323 zone must register with its gatekeeper. Without policies, the gatekeeper can make only a simple registration check: whether the proposed H.323 alias (telephone number) conflicts with an existing one. Policies allow much more flexibility. For example, organisations will normally allow only designated H.323 terminals to register. Some terminals might be forbidden from registering, and some might be allowed more than one H.323 alias.

A University, for example, might allow only staff terminals to register with their H.323 telephone system. A registration attempt by a student terminal should be rejected for this reason. The following policy is parameterised by *STAFF* as a policy variable that is instantiated to a specific list of terminals or to a domain name. Policy parameters are prefixed by ':', and are conventionally given names in upper case.

```
<policy owner="admin@univ.edu" applies_to="@univ.edu"
 id="Registration control" enabled="true" changed="2004-08-23T17:20:15">
   <policy_rule>
      <trigger>register
      <condition>
         <parameter>signalling_address
         <operator>ne
         <value>:STAFF
      <action arg1="only staff may register">reject_call(arg1)
```

Control can similarly be exercised over H.323 admission. A University might decide that H.323 terminals used for enquiries should accept only incoming calls. Outgoing calls from terminals in the *enquiries.univ.edu* domain are therefore rejected.

```
<policy owner="admin@univ.edu" applies_to="@univ.edu"
 id="Call control" enabled="true" changed="2004-08-19T17:10:37">
   <policy_rule>
      <trigger>connect_outgoing
      <condition>
         <parameter>caller
         <operator>eq
         <value>@enquiries.univ.edu
      <action arg1="outgoing calls forbidden">reject_call(arg1)
```

When an H.323 terminal requests admission from its gatekeeper, it can ask for bandwidth. Bandwidth can also be changed while a call is in progress. The gatekeeper manages bandwidth on behalf of its zone, allocating bandwidth within limits that it determines.

Most operators in the policy language are in binary prefix form. In the policy below, 'or' combines the two following conditions. An 'else' governs two actions subject to the preceding condition.

The following policy rejects a bandwidth request if it exceeds 128 Kbps unless it is an emergency call; other requests are accepted. If the entire condition holds, the first action is taken (*confirm_bandwidth*), otherwise the second (*reject_bandwidth*). When a bandwidth rejection is issued, its argument is the acceptable bandwidth limit.

```
<policy owner=″admin@univ.edu″ applies_to=″@univ.edu″
 id=″Limit bandwidth″ enabled=″true″changed=″2004-08-16T09:30:55″>
  <policy_rule>
    <trigger>bandwidth_request
    <conditions>
      <or/>
      <condition>
        <parameter>bandwidth
        <operator>le
        <value>128
      <condition>
        <parameter>call_type
        <operator>eq
        <value>emergency
    <actions>
      <else/>
      <action>confirm_bandwidth
      <action arg1=″128″>reject_bandwidth(arg1)
```

It is also possible to make bandwidth policies depend on the current traffic load (supplied by the gatekeeper to the policy system). Large bandwidth requests can be rejected if the traffic load is high.

3.4.3 Supplementary Services

H.323 defines a call intrusion service [17] that enables a served user *A* to communicate with a busy user *B*, breaking into an established call between *B* and a third user *C*. Call intrusion resembles operator barge-in, whereby a telephone operator can break into an existing call. The enforcement of a call intrusion policy is relatively complicated because the gatekeeper needs to manage the media streams. There are several ways to realise this service:

Conference Call: users *A*, *B* and *C* might be merged into an *ad hoc* conference.
Call Hold: the unwanted user *C* might be split from *B* by automatically invoking call hold.
Silent Monitoring: *A* might be allowed to just listen to the established call.
Forced Release: the established call to *C* might be forcibly released so that *A* can then communicate with *B*.
Wait On Busy: *A* may be required wait on *B* becoming free.

Because there are so many ways to implement call intrusion, it is difficult for an H.323 system to support this service. The call intrusion method should also depend on the circumstances. However, no H.323 system currently on the market supports call intrusion using all these methods.

The action *add_caller(method)* indicates how call intrusion should be performed. The following example combines two rules in sequence. If the first does not apply, the second is tried. Anne allows Barry to silently monitor calls. If an incoming call is urgent (as specified by the caller), the forced release method is used. If the call is not urgent, the caller is directed to wait on busy.

```
<policy owner="anne@acme.com" applies_to="anne@acme.com"
 id="Intrusion control" enabled="true" changed="2004-08-16T10:51:13>
   <policy_rules>
      <sequential/>
      <policy_rule>
         <trigger>connect_incoming
         <condition>
            <parameter>caller
            <operator>eq
            <value>barry@acme.com
         <action arg1="monitor">add_caller(arg1)
      <policy_rule>
         <trigger>connect_incoming
         <condition>
            <parameter>call_type
            <operator>eq
            <value>urgent
         <actions>
            <else/>
            <action arg1="release">add_caller(arg1)
            <action arg1="wait">add_caller(arg1)
```

Policy support has also been designed for other kinds of H.323 supplementary services. Call hold [9] allows either party to suspend the call. A user's policy can decide whether to accept call hold. If it is permitted, either the local or the remote gatekeeper can be directed play a media clip while the call is on hold. Name identification [15] allows the name of the caller or callee to be provided or withheld. In a conventional H.323 environment, this is achieved using a fixed configuration. A policy allows the user to be more selective, for example providing a name only to friends (as defined by an explicit list of addresses).

3.4.4 Context Support

The context system described in section 2.3 allows H.323 calls to be managed in a much richer setting. For example, policies to handle H.323 calls may depend on presence, availability, role, capability, call type and call content.

Suppose Anne wishes to discuss budgetary issues with Barry between 09.00 and 10.00 each day. Barry may have other engagements during this period. Even if he is available, he may not wish to talk about budgets. Anne therefore defines the following policy. She is connected to Barry when he announces his availability to discuss budgets, and the time falls within her preferred period. The operator 'ge' means that the *topic* of availability is exactly 'budget' or contains this word.

```
<policy owner="anne@acme.com" applies_to="anne@acme.com"
 id="Budget discussion" enabled="true" changed="2004-09-01T16:57:49">
   <policy_rule>
```

```
    <trigger>available(barry@acme.com)
    <conditions>
      <and/>
      <condition>
        <parameter>topic
        <operator>ge
        <value>budget
      <condition>
        <parameter>time
        <operator>in
        <value>09:00..10:00
    <action arg1=″barry@acme.com″>connect_to(arg1)
```

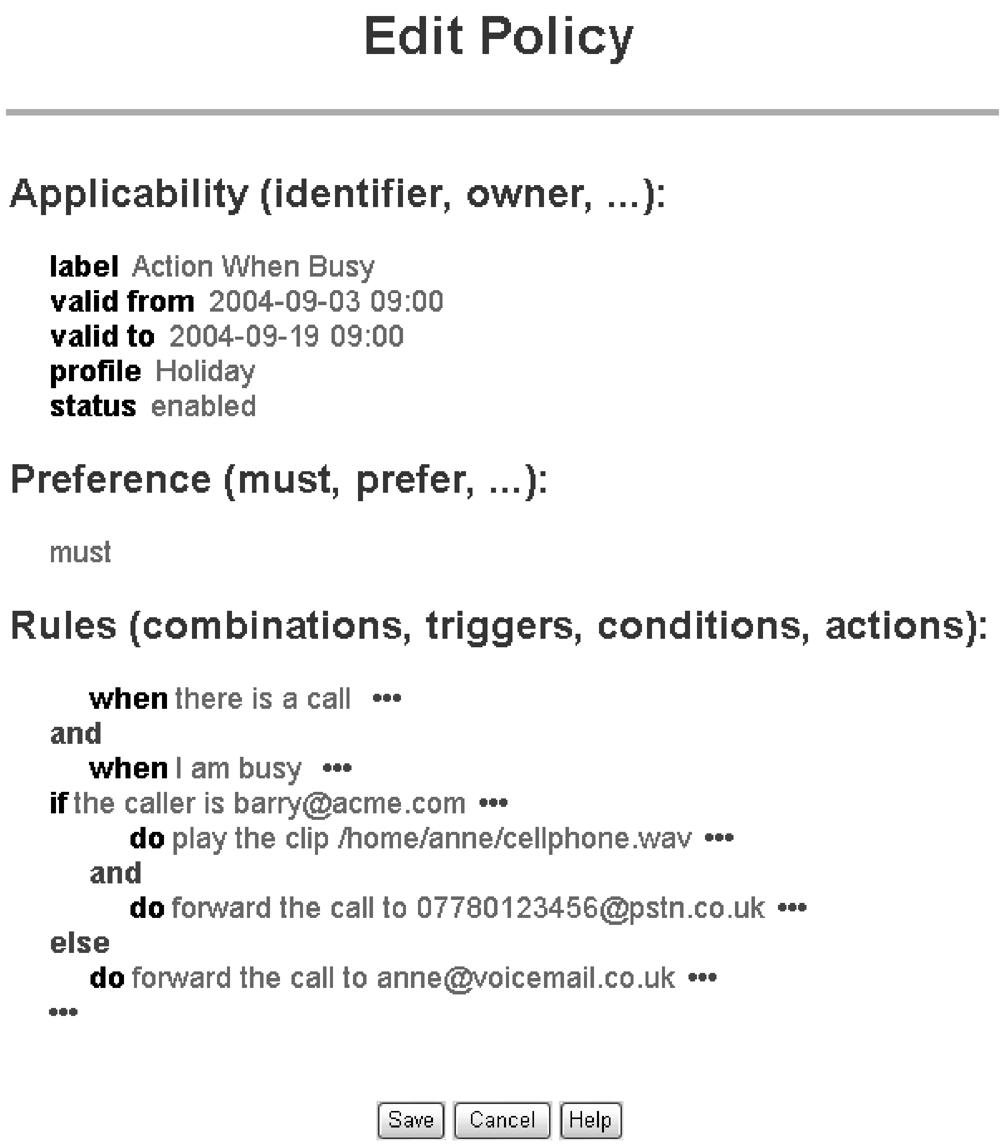
The context system can supply additional information about the capabilities of the caller. These might be explicitly provided by the caller, or might be derived from an organisational database. In the following policy, Anne arranges for calls from French speakers to be forwarded to François, while other calls are sent to Gerry.

```
<policy owner=″anne@acme.com″ applies_to=″anne@acme.com″
 id=″French speaker″ enabled=″true″ changed=″2004-09-03T08:37:01″>
  <policy_rule>
    <trigger>connect_incoming
    <condition>
      <parameter>capability
      <operator>eq
      <value>francophone
    <actions>
      <else/>
      <action arg1=″francois@acme.com″>forward_to(arg1)
      <action arg1=″gerry@acme.com″>forward_to(arg1)
```

3.5 Policy Wizard Support for H.323

As has been seen, policies are represented in XML. This would hardly be appropriate for end users to write. Instead, the policy wizard allows ordinary subscribers to create and edit policies in a user-friendly manner. The wizard is web-based for easy use. It provides many convenience features for the non-technical user, such as template policies, different skill levels, online help and tool tips. Most importantly, policies are created using structured natural language. The policy wizard is multilingual, currently supporting English, French and German. The design of the wizard makes it relatively straightforward to support many other languages (but of course not all).

Figure 4 shows Anne using the wizard to create a policy. The applicability defines a label for a policy, and optionally the timeframe in which it applies (here, from 3rd to 19th September). Policies may be collected into profiles so that they can easily be selected as a group. The policy in the figure is part of Anne's 'on holiday' profile. The 'must' preference indicates that the policy is strongly required.


Edit Policy

Applicability (identifier, owner, ...):

label Action When Busy
valid from 2004-09-03 09:00
valid to 2004-09-19 09:00
profile Holiday
status enabled

Preference (must, prefer, ...):

must

Rules (combinations, triggers, conditions, actions):

```
    when there is a call •••
and
    when I am busy •••
if the caller is barry@acme.com •••
        do play the clip /home/anne/cellphone.wav •••
    and
        do forward the call to 07780123456@pstn.co.uk •••
else
    do forward the call to anne@voicemail.co.uk •••
•••
```

Fig. 4. Editing a Policy

The policy states that when Anne receives an incoming call and is busy, the caller should be checked. If it is Barry, a recorded announcement (/home/anne/cellphone.wav) is played to tell him that his call will be forwarded to Anne's cellphone number (07780123456). The call is then forwarded via a PSTN gateway. If the caller is not Barry, the call is forwarded to Anne's voicemail (anne@voicemail.co.uk).

4 Conclusion

The need for policies has been justified in view of the changing face of communications. In particular, policies have the promise of replacing features in Next Generation Networks. A policy architecture and a policy language for call control in H.323 have been introduced. These allow users and their organisations to define policies that are more flexible than traditional features. Policy support has been developed for H.323 (GnuGK), SIP (SIP Express Router) and PBX (Mitel 7000).

The policy architecture consists of three layers: the communications system, the policy system, and the user interface. Policy support is largely independent of the communications subsystem. A communications server supplies call information to a policy server. The policy server in turn retrieves and filter policies held in policy stores to determine appropriate communications actions. The policy language is based on XML, and so is intended only for specialised use. However, a policy wizard acts as a user-friendly front-end to the policy system.

H.323 support is focused on gatekeepers. This is sensible since much of the management of an H.323 network is vested in them. However it would be possible to consider supporting policies directly in H.323 terminals. A variety of policies has been given to illustrate how H.323 calls can be managed more effectively. There is considerable scope to extend the range and applicability of policies for new H.323 services, as well as to other types of communications systems.

One general aspect that needs further development is security. For now, the policy system is being deployed in private networks. For more public use, the policy system will perform tighter authentication using standard solutions for this kind of issue.

Acknowledgements

The ACCENT project provided the basis for the work reported in this paper. Funding for this project was provided by EPSRC (the UK Engineering and Physical Sciences Research Council, grant R31263) and by Mitel Networks.

Stephan Reiff-Marganiec (originally at Stirling, now at Leicester) undertook most of the design and implementation of the generic policy system. The specialisation of policy support for H.323 was undertaken by Tingxue Huang, who gratefully acknowledges funding from the Sino-British Fellowship Trust while a postdoctoral fellow at Stirling.

The authors thank their colleagues (Lynne Blair and Jianxiong Pang at Stirling, Tom Gray and Peter Perry at Mitel Networks, Joe Ireland at MKC Networks) for insightful discussions about policies.

References

[1] M. Amer, A. Karmouch, T. Gray, and S. Mankovskii. Feature interaction resolution using fuzzy policies. In M. H. Calder and E. H. Magill, editors, *Proc. 6th. Feature Interactions in Telecommunications and Software Systems*, pages 94–112. IOS Press, Amsterdam, Netherlands, May 2000.

[2] M. H. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks*, 41:115–141, Jan. 2003.

[3] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen. A feature-interaction benchmark for IN and beyond. *IEEE Communications Magazine*, pages 64–69, Mar. 1993.

[4] E. J. Cameron and H. Velthuijsen. Feature interactions in telecommunications systems. *IEEE Communications Magazine*, pages 18–23, Aug. 1993.

[5] N. Damianou, N. Dulay, E. C. Lupu, and M. Sloman. Ponder: A language specifying security and managements policies for distributed systems. Technical report, Imperial College, London, 2000.

[6] P. Dini, A. Clemm, T. Gray, F. J. Lin, L. Logrippo, and S. Reiff-Marganiec. Policy-enabled mechanisms for feature interactions: Reality, expectations, challenges. *Computer Networks*, 45(5):585–603, Aug. 2004.

[7] T. Huang. Policies for H.323 internet telephony. Technical Report CSM-165, Department of Computing Science and Mathematics, University of Stirling, UK, May 2005.

[8] ITU. *Data Protocols for Multimedia Conferencing*. ITU-T T.120. International Telecommunications Union, Geneva, Switzerland, 1996.

[9] ITU. *Call Hold Supplementary Service for H.323*. ITU-T H.450.4. International Telecommunications Union, Geneva, Switzerland, Sept. 1998.

[10] ITU. *ISDN User-Network Interface – Layer 3 Specification for Basic Call Control*. ITU-T Q.931. International Telecommunications Union, Geneva, Switzerland, May 1998.

[11] ITU. *Call Diversion Supplementary Service for H.323*. ITU-T H.450.3. International Telecommunications Union, Geneva, Switzerland, May 2000.

[12] ITU. *Call Signalling Protocols and Media Stream Packetization for Packet-Based Multimedia Communication Systems*. ITU-T H.225. International Telecommunications Union, Geneva, Switzerland, Nov. 2000.

[13] ITU. *Control Protocol for Multimedia Communication*. ITU-T H.245. International Telecommunications Union, Geneva, Switzerland, Nov. 2000.

[14] ITU. *Intelligent Network – Q.120x Series Intelligent Network Recommendation Structure*. ITU-T Q.1200 Series. International Telecommunications Union, Geneva, Switzerland, 2000.

[15] ITU. *Name Identification Supplementary Service for H.323*. ITU-T H.450.8. International Telecommunications Union, Geneva, Switzerland, May 2000.

[16] ITU. *Packet-Based Multimedia Communication Systems*. ITU-T H.323. International Telecommunications Union, Geneva, Switzerland, Nov. 2000.

[17] ITU. *Call Intrusion Supplementary Services*. ITU-T H.450.11. International Telecommunications Union, Geneva, Switzerland, Mar. 2001.

[18] J. Lennox and H. Schulzrinne, editors. *Call Processing Language Framework and Requirements*. Internet Draft CPL-Framework-02. The Internet Society, New York, USA, Jan. 2000.

[19] S. Reiff-Marganiec and K. J. Turner. Use of logic to describe enhanced communications services. In D. A. Peled and M. Y. Vardi, editors, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE XV)*, number 2529 in Lecture Notes in Computer Science, pages 130–145. Springer, Berlin, Germany, Nov. 2002.

[20] S. Reiff-Marganiec and K. J. Turner. A policy architecture for enhancing and controlling features. In D. Amyot and L. Logrippo, editors, *Proc. 7th. Feature Interactions in Telecommunications and Software Systems*, pages 239–246. IOS Press, Amsterdam, Netherlands, June 2003.

[21] S. Reiff-Marganiec and K. J. Turner. The ACCENT policy server. Technical Report CSM-164, Department of Computing Science and Mathematics, University of Stirling, UK, Aug. 2004.

[22] S. Reiff-Marganiec and K. J. Turner. APPEL: The ACCENT project policy environment/language. Technical Report CSM-161, Department of Computing Science and Mathematics, University of Stirling, UK, Aug. 2004.

[23] S. Reiff-Marganiec and K. J. Turner. Feature interaction in policies. *Computer Networks*, 45(5):569–584, Aug. 2004.

[24] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnson, J. Peterson, R. Sparks, M. Handley, and E. Schooler, editors. *SIP: Session Initiation Protocol*. RFC 3261. The Internet Society, New York, USA, June 2002.

[25] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, editors. *RTP: A Transport Protocol for Real-Time Applications*. RFC 1889. The Internet Society, New York, USA, Jan. 1996.

[26] K. J. Turner. The ACCENT policy wizard. Technical Report CSM-166, Department of Computing Science and Mathematics, University of Stirling, UK, Aug. 2004.

[27] K. J. Turner and S. Reiff-Marganiec. Policy support for call control. *Computer Networks*, Sept. 2004. Under review.