Kenneth J. Turner, Stephan Reiff-Marganiec, Lynne Blair, Jianxiong Pang, Tom Gray, Peter Perry and Joe Ireland. Policy Support for Call Control, Computer Standards and Interfaces 28(6):635-649, June 2006

Policy Support for Call Control

Kenneth J. Turner [a], Stephan Reiff-Marganiec [b], Lynne Blair [a], Jianxiong Pang [a], Tom Gray [c], Peter Perry [c] and Joe Ireland [d]

[a] *Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, UK*

[b] *Computer Science, University of Leicester, Leicester LE1 7RH, UK*

[c] *Mitel Networks, 350 Legget Drive, Kanata, Ontario K2K 2W7, Canada*

[d] *MKC Networks, 555 Legget Drive, Kanata, Ontario K2K 2X3, Canada*

Abstract

The need for policies to control calls is justified by the changing face of communications. It is argued that call control requires distinctive capabilities in a policy system. A specialised policy language called APPEL (ACCENT Project Policy Environment/Language) has therefore been developed for this purpose. However the policy language is cleanly separated into a core plus specialisations for various application domains. The paper describes both the foundation and the call control ontologies. Sample policy examples are provided to illustrate use for call control. The paper also presents the policy system architecture in which the policy language is interpreted. The components of the policy system are described, particularly the policy server and the policy wizard.

Keywords: Call Control, Internet Telephony, Policy, Policy Conflict

1 Introduction

1.1 The Need for Policies

Communications 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 conventional or Internet telephony. Voice may be supplemented by video, data or other media. Calls may be supported by conventional telephones,

Email addresses: kjt@cs.stir.ac.uk (Kenneth J. Turner), srm13@le.ac.uk (Stephan Reiff-Marganiec), lb@comp.lancs.ac.uk (Lynne Blair), j.pang@sussex.ac.uk (Jianxiong Pang), digroup_codec@direcway.com (Tom Gray), peter_perry@mitel.com (Peter Perry), jireland@mkcnetworks.com (Joe Ireland).

cellphones, softphones, PDAs, voicemail, message transfer agents, and web browsers. As a consequence of this bewildering variety, there is an urgent need for flexible control of 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 suspend callers. 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 a single network operator. However, services are increasingly being deployed on 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.

Internet-based calling presents a striking difference from conventional telephony. The Internet philosophy is very much to have a simple and efficient core network, with complex facilities provided in the hosts and terminals. In contrast, conventional telephony emphasises the central role of the network in providing services to simple terminals. Thus in Internet telephony, the approach has been to support advanced call processing in the endpoints. SIP (Session Initiation Protocol [18]) is a good example. SIP services are typically deployed in enterprise proxy servers using SIP CGI (Common Gateway Interface [10]). SIP user agents (end-user interfaces) may allow users to define call preferences with CPL (Call Processing Language [9]).

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. The use of policies is attractive for call control. Policies tend to be high-level, flexible, goal-oriented and declarative. Policy support has arisen in areas such as distributed systems, access control and QoS. This paper reports a new kind of application for policies: call handling. 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 paper presents work by the ACCENT project (Advanced Call Control Enhancing Network Technologies). The aim of this project was to develop policy support for call control in an Internet setting. ACCENT was mainly focused on demonstrating policy support for SIP (Session Initiation Protocol [18]). However policy support for H.323 (a popular form of Internet telephony [8]) has also been undertaken in parallel work [6]. The present paper is fully complementary to [6], in that it explains the general foundation for policy support.

1.2 Related Work

CPL (Call Processing Language [9]) 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:

- CPL is limited in its network bindings (currently H.323 and SIP).
- CPL gives very limited control over calls, specifically just call setup. There is also a need

for mid-call control (e.g. when a new party is added to a call) and call tear-down control (i.e. when a call is disconnected).
- CPL supports only limited call control, e.g. through checks on the caller or the current time.
- CPL does not support a range of preferences (positive or negative, with different strengths).
- CPL has limited integration with presence and availability systems.
- CPL offers no mechanisms for detecting and resolving conflicts among call preferences.

Call centres and CTI (Computer Telephony Integration) support flexible call handling; see [4] for a survey of the approaches. Call centres rely on mechanisms such as CLI (Calling Line Identification) and ACD (Automatic Call Distribution) to route callers to appropriate agents. Call centres are designed for large businesses, unlike the work reported here which is intended for individual end users. Call centres essentially deal with routing within one organisation, whereas call policies handle calls on a global basis. Call centres also do not support the kinds of capabilities discussed in this paper. Policy-based support of calls is thus complementary to the techniques used in call centres.

Policies have been used in many kinds of management tasks. [11] defines policies as information that can be used to modify the behaviour of a system. This is a very general and open-ended definition. In the context of this paper, policies are interpreted as the goals for how calls should be handled. Policies lend themselves well to networked applications, where the very distribution demands careful management. Despite this, call handling systems have attracted little policy support. [1] uses fuzzy policies as a means of resolving feature interactions. From the most recent conference on *Feature Interactions in Telecommunications and Software Systems*, it is evident that many researchers see policies as important in future call handling.

Policy language developments in industry have largely focused on network management and QoS. For example, Cisco have developed policy support for control of security and QoS in routers. Lucent and Bell Labs have developed PDL (Policy Description Language) for network management. The IETF standard for COPS (Common Open Policy Service) is intended as a protocol for managing QoS. None of this work is of direct relevance to call control.

[13] discusses the kind of policies that are needed in call control. Initially, ACCENT evaluated some existing policy languages to determine their suitability for this application. For example, a detailed evaluation [13] was made of Ponder [3]. It was found that Ponder was only partly suitable for call control. Nonetheless, Ponder has been influential on ACCENT.

A policy language was therefore defined for ACCENT to overcome limitations of existing languages. A policy language should ideally have a form that is readily parsed by many tools. XML is widely employed for structured information, but is used by only a few policy languages.

The focus of ACCENT on call control is distinctive. It places different demands on a policy system, and of course it requires specialised support in a communications setting. The language developed by ACCENT for call control falls into the general category termed ECA (Event-Condition-Action). However 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. This is true of only some existing languages. Although a language for call control has been developed for ACCENT, the core of the language is separate and can be adapted for other uses. Even when used for call control, the ACCENT language has to be largely independent of the underlying communications system.

In systems management, a useful distinction can be often made between the subject of a policy (that performs an action) and the target of a policy (that is acted upon). A number of policy languages reflect this. In call control, the nature of subject and target becomes unclear. It can be argued that the subject is the caller, the call or the network, while the target is the callee, the call or the network. This is one reason why Ponder was found to be less appropriate for call control. In many application domains, the entities involved in policies are fairly static and predictable. This does not apply to call control, where any user (previously unknown) may call any other user. As a result, call control introduces a much more dynamic set of policies. In addition, policies may be introduced by the underlying networks as well as the call parties.

Most policy languages require specialised technical expertise, being designed for programmers or technicians. In contrast, policies for call control must be accessible to the ordinary telephone subscriber. This presents a major challenge, because the policy language and the supporting policy system must be usable by non-technical people. Communication is global, so policy support must also be truly international – specifically, multilingual.

Call control is more likely to lead to policy conflict because very many users with unpredictable policies may wish to communicate. Although conflict handling is mainly part of the policy infrastructure, the design of the policy language should assist conflict resolution. Furthermore, the guidance given for conflict handling needs to be in a form that ordinary end users can give.

Many policy languages support modal or deontic aspects such as obligation, permission (or authorisation) and interdiction (or refrain). Ponder has obligation, authorisation and refrain policies. Obligation and interdiction apply to the subject, while permission applies to the target. Since the notions of subject and target do not map so readily to call control, these modalities need some rethinking. Furthermore, obligations placed on end users have limited value since they cannot be enforced.

For the above reasons, it was concluded that no existing policy system would adequately serve for call control. The ACCENT project therefore developed its own policy language and policy support, inspired by the unique needs of call control. However the language has been cleanly separated into a core and its specialisation for some application domain (here, call control). This allows the policy system to be largely re-used in other contexts. In this respect, the ACCENT policy language resembles some others such as Ponder.

Distributed definition of policies can lead to incompatibilities among them. Policy conflict resembles the extensively studied feature interaction problem. It is argued in [14] that some techniques from feature interaction can be adapted for detection and resolution of policy conflicts. Nonetheless, conflict handling remains a challenging task. The approach taken by ACCENT is described in [2].

1.3 Structure of the Paper

Section 2 presents the concepts of the core policy language. These are generic, and are intended to be useful in any application domain. Section 3 then specialises the policy language for use in call control. The specific ontology required for this domain is summarised, and is illustrated with examples of call control policies. Section 4 explains the overall policy system architecture with specific reference to call control. The various components of the policy system are introduced, in particular the policy server and the policy wizard

2 Concepts of The Core Policy Language

The policy language developed by ACCENT is called APPEL (ACCENT Project Policy Environment/Language, a word play on the French for 'call'). The aim of this section is to explain the philosophy of the core language. [13] discusses the origins of the language. APPEL is fully defined in [17].

2.1 Approach

APPEL is intended as a general language for expressing policies in a variety of application domains. A clear separation is therefore made between the core language and its specialisation for concrete applications. This section describes the core language, while section 3 explains how it is specialised for call control.

Unlike many policy languages, APPEL is designed for end users rather than technicians or administrators. This has had a profound influence on the language. For example, the style of APPEL is closer to natural language than to programming. The benefit is that policies can more readily be formulated and understood by ordinary users. The disadvantage is that the language must then dress up subtle concepts that could be too complex for end users. As will be seen in section 4.6, the policy wizard has an key role in presenting APPEL in a comprehensible way.

APPEL is defined by an XML schema; policies are XML documents that conform to this schema. Policies are given meaning by being interpreted in a policy server. The policy system architecture is discussed in section 4.

2.2 Generic Policies

It can be convenient to define generic policies that are then instantiated as required. For example, a policy might forward incoming calls to some address when the callee is busy. Such a policy can have the forwarding address as a formal parameter. Formal parameters may be used in a policy wherever values are required. The actual values of formal parameters may be

defined separately as variables in a policy document. It is convenient, however, to instantiate a policy separately from its definition by using the policy wizard (see section 4.6).

It is also very useful to define template policies that require only a few parts to be completed. This makes it easy for novice users to adapt ready-made policies rather than having to define policies from scratch. Template policies are distributed with the policy wizard. The approach maintains the separation between the core language and specific application domains. It also facilitates the support of policies for vertical markets. For example in call control, the policies that are useful for a sales organisation differ from those that would benefit a medical clinic. A major problem for communications providers is that they have to offer a large package of features, even if only a small subset is employed in particular markets. The template approach could also encourage systems integrators to deliver communications systems with policies tailored for specific customers.

2.3 Domains

Policies have owners and apply to domains. These are often the same, i.e. when a person defines individual policies. However it is possible for someone (e.g. an administrator) to define policies that apply to others (e.g. in the same organisation). The owner is always a person, identified by an email-like address (e.g. alice@stir.ac.uk). The domain to which a policy applies may be an individual, a symbolic name for a group of individuals, or a list of both. Individuals may belong to several domains.

Groups are also identified by email-like addresses, e.g. @stir.ac.uk denotes anyone in the University of Stirling. When retrieving the policies that apply to someone, higher-level domain policies are also taken into account. For example alice@stir.ac.uk is subject to her own policies, as well as those of @cs.stir.ac.uk (Computing Science Stirling) and @stir.ac.uk (Stirling). Policies may exist at any level of this hierarchy, in principle including '@' meaning everyone.

2.4 Modality

APPEL adopts an everyday approach to defining policy modality in the form of a preference. The strength of feeling associated with a policy can be defined: **must**, **should** and **prefer**, plus the negative forms of these. Omitting the preference means that the user is neutral about how strongly the policy should apply. Preferences imply a relative ordering for conflicting policies. Some approaches require an explicit numerical weight to be used. However in a practical situation it is unclear how this weight can be determined. Instead APPEL relies on natural language terms to imply a relative ordering. Even though preferences then have a precise technical meaning, it is easier for end users to formulate such policies. In fact, policy modalities do not directly affect the execution of a policy. As discussed in section 4.4, they are used to guide the resolution of conflict among policies.

2.5 Rules

A policy document defines a number of policy rules. The execution of a rule depends on a number of factors: whether it is enabled at the current time, whether its trigger has occurred, whether its conditions are satisfied, and whether conflict resolution permits it to occur. To be enabled, a policy must be explicitly activated. Policies may also be associated with named profiles that the user may select (e.g. 'in the office', 'at home'). To be enabled, a policy must also belong to the current user profile (if defined). Finally, a policy may be enabled only within a certain time-frame.

Policy rules may be grouped. In particular, policies may be composed from pairs of rules using the following combinators:

Guarded choice: This is used to select rules based on generic information such as the current date or the type of event. The context severely limits the condition because no applicable trigger has yet been selected.

Unguarded choice: This is used when alternative rules might apply, and the user does not mind which is selected. If only one of the two policy rules is applicable, this will be chosen. Conflict resolution can influence the choice by determining that one rule should not apply. If both rules are applicable, the outcome is non-deterministic (system-defined).

Sequential composition: This is used when there are alternative rules that should be tried in a given order: the first applicable rule is executed.

Parallel composition: This is used when the order of execution is unimportant. The rules may be executed in parallel, sequentially, or in some system-defined order.

2.6 Rule Bodies

A rule body contains an optional trigger, an optional condition, and an action. The core language defines the structure but not the details of these: rather they are defined only in specific application domains. This allows the core language to be instantiated for different purposes.

Omitting a trigger means that a rule can be executed without an explicit event; it acts as a goal. To be exact, such a rule is implicitly triggered depending on its condition. The absence of a trigger severely restricts a following condition to general information such as the current time or the type of event. Omitting a condition means that a rule is always executed when its trigger occurs. If both the trigger and the condition are omitted, the action is immediately executed. If a rule is not applicable, it has no effect.

Triggers are caused by external events notified to the policy system. For example, in call control the triggers include the arrival of an incoming call or either party hanging up. A trigger may have parameters, such as the address of someone whose presence is to be checked. A trigger establishes information that is supplied by the external system. A call control system, for example, defines the caller and callee addresses when there is a call attempt. This information may be used by the conditions and actions of a rule.

Triggers may be combined using **and** and **or**, with the obvious meaning that both or either must occur. This affects the condition and action that follow a trigger, being governed by the union or intersection respectively of what the triggers imply.

Conditions may be combined with **and**, **or** and **not** with the expected meaning. A condition consists of a parameter established by a trigger, a comparison operator and a value. The operators are fixed in the language, but their interpretation is parameter-specific. For example, **lt** means 'less than' when used with numeric parameters and 'before' when used with time parameters. A condition value may be either a single literal or a list of literals. A list is used for membership or range checks, e.g. a date is one of the specified values or is within a range.

Actions have an effect external to the policy system. In call control, for example, the actions include forwarding or rejecting a call. An action may have parameters, such the address to which a call should be forwarded. Composite actions may be created using the following combinators:

and: Both actions are executed, but in a system-defined order: in sequence or in parallel. Conflict resolution may lead to a specific order being selected.

andthen: Both actions are executed in the order given. This is a stronger version of **and**, since the first action must precede the second in any execution.

or: One or other action is executed, the choice being made by the system. Conflict resolution may lead to a specific choice being made.

orelse: If there is a choice, the first action is taken. This expresses a user preference, but it is not guaranteed that this will be respected. Conflict resolution may require that only the second action be followed.

else: If there is an immediately prior condition, its value dictates selection of the first or second action. If there is no such condition (e.g. it was omitted), this combinator behaves like **or**.

3 Policies for Call Control

This section illustrates how APPEL is used in practice. The need for policies in call control is explained in [12]. The specialisation of APPEL for call control is fully defined in [17].

3.1 Policy Language Specialisation

The core language presented in section 2 defines a foundation ontology for policies. This can then be specialised by adding the information required in a concrete application domain. Explicitly, this means defining specific triggers, condition parameters and actions. As a major instance of APPEL, its specialisation for call control has been defined. This is intended to be a broad domain of application that includes:

- conventional telephony, whether using the PSTN (Public Switched Telephone Network), the AIN/IN (Advanced/Intelligent Network [7]) or a mobile telephone network
- Internet telephony, such as supported by H.323 [8] or SIP (Session Initiation Protocol [18])

Element	Call Control
Trigger	absent(*address*), available(*address*), bandwidth_request, †connect, †disconnect, event, †no_answer(*period*), present(*address*), †register, unavailable(*address*)
Condition	active_content, bandwidth, call_content, call_type, callee, caller, capability, capability_set, cost, date, destination_address, device, location, medium, network_type, priority, quality, role, signalling_address, source_address, time, topic, traffic_load
Action	add_caller(*method*), add_medium(*medium*), add_party(*address*), confirm_bandwidth, connect_to(*address*), fork_to(*address*), forward_to(*address*), log_event(*message*), note_availability(*topic*), note_presence(*location*), play_clip(*audio*), reject_call(*reason*), reject_bandwidth(*limit*), remove_medium(*medium*), remove_party(*address*), send_message(*address,message*)

Fig. 1. Triggers, Condition Parameters and Actions for Call Control

- non-voice calls such as used by pagers, email, transaction processing or web services.

The specialisation of APPEL for call control is detailed in [17]. It is not practicable to give a tutorial on the language here. Instead, the summary in figure 1 is provided as an overview; triggers marked † also exist in incoming and outgoing variants. The examples in section 3.2 give an insight into the approach.

3.2 Policy Examples

The following examples illustrate APPEL in a call control setting. Further sample policies can be found in [5,6,17]. A policy document defines policies and policy variables embedded in XML 'red tape' that is omitted here. For brevity, the obvious closing XML tags are also omitted. See section 4.6 for an example of how the policy wizard displays complex policies.

3.2.1 Forward Incoming Calls

Incoming calls for Alice *should* be forwarded to Bob during the dates 24th December 2004 to 5th January 2005 inclusive. A policy includes its owner, the domain it applies to, and its identifier. A policy is normally enabled, but can be deactivated. The date a policy is valid from or to can also be specified. When a policy is edited, the date and time of the change (in XML format) are stored along with it. Triggers and actions may have arguments, specified as XML attributes like *arg1*.

```
<policy owner="alice@stir.stir.ac.uk" applies_to="alice@stir.stir.ac.uk"
 id="Forward incoming calls" enabled="true"
 valid_from="2004-12-24T00:00:00" valid_to="2005-01-05T23:59:00"
 changed="2004-08-12T11:33:00">
  <preference>should
```

```
<policy_rule>
  <trigger>connect_incoming
  <action arg1="bob@stir.stir.ac.uk">forward_to(arg1)
```

3.2.2 Emergency Call Handling

Emergency calls must not be rejected. This policy applies to the cs.stir.ac.uk domain. The trigger *connect* refers to both incoming and outgoing calls. The call type may be given explicitly by the underlying communications system, or may be inferred from the use of a special address like 911 or 999. Because this is a general interdiction, the reason for rejecting a call (*arg1*) is irrelevant and is left empty. An interdiction can require a specific argument for an action, e.g. that emergency calls must not be rejected for reason 'busy'.

```
<policy owner="admin@cs.stir.ac.uk" applies_to="@cs.stir.ac.uk"
 id="Never reject emergency calls" enabled="true" changed="2004-08-02T11:46:00">
  <preference>must_not
  <policy_rule>
    <trigger>connect
    <condition>
      <parameter>call_type
      <operator>eq
      <value>emergency
    <action arg1="">reject_call(arg1)
```

3.2.3 Announcing Availability

Lecturers in Computing Science Stirling are now available for discussions about Java (*arg1*). This policy has no trigger or condition, so it is executed immediately on definition.

```
<policy id="Available for Java" owner="admin@cs.stir.ac.uk"
 applies_to="@lecturers.cs.stir.ac.uk" enabled="true" changed="2004-07-28T23:18:00">
  <policy_rule>
    <action arg1="Java">note_availability(arg1)
```

3.2.4 Using Capabilities

A policy can be governed by the capabilities of the caller. These might be provided explicitly by the caller, or might be extracted from a database. In the following policy, Alice accommodates deaf callers with textphones. A text operator is automatically conferenced in to transcribe Anne's speech into text form.

```
<policy owner="alice@stir.ac.uk" applies_to="alice@stir.ac.uk"
 id="Deaf caller" enabled="true" changed="2005-01-04T14:27:17">
  <policy_rule>
    <trigger>connect_incoming
    <condition>
      <parameter>capability
      <operator>eq
      <value>textphone
```

```
<actions>
  <action arg1="text_operator@stir.ac.uk">add_party(arg1)
```

3.2.5 Controlling Registration

Communications systems such as SIP and H.323 allow users to register their presence with a server. Policies can be defined to manage such registrations. As an example, a University administrator might permit registration only by staff (and not students). The policy parameter *staff* would be instantiated as a list of authorised users (or their domain); policy parameters are prefixed by ':'.

```
<policy owner="admin@stir.ac.uk" applies_to="@stir.ac.uk"
 id="Staff registration" enabled="true" changed="2004-12-17T16:13:57">
  <policy_rule>
    <trigger>register
    <condition>
      <parameter>signalling_address
      <operator>in
      <value>:staff
    <action arg1="only staff may register">reject_call(arg1)
```

3.2.6 Presence-Based Messaging

This example assumes a presence system, e.g. an active badge system that tracks where people are. Suppose that Alice works in Building 7. When the presence of Colin is reported, it is checked if he is in this building. If so, an email message is sent to him to propose dinner.

```
<policy owner="alice@stir.ac.uk" applies_to="alice@stir.ac.uk"
 id="Colin in Building 7" enabled="true" changed="2004-07-29T21:15:29">
  <policy_rule>
    <trigger>present(colin@acme.com)
    <condition>
      <parameter>location
      <operator>eq
      <value>Building 7
    <action arg1="mailto:colin@acme.com" arg2="Dinner at 8PM?">send_message(arg1,arg2)
```

4 Policy System Architecture

This section explains the overall policy system architecture and its components. The origins of the architecture are discussed in [14]. The ACCENT approach to policy conflict is discussed in [2,15]. The detailed implementation of the policy system is described in [16,19].

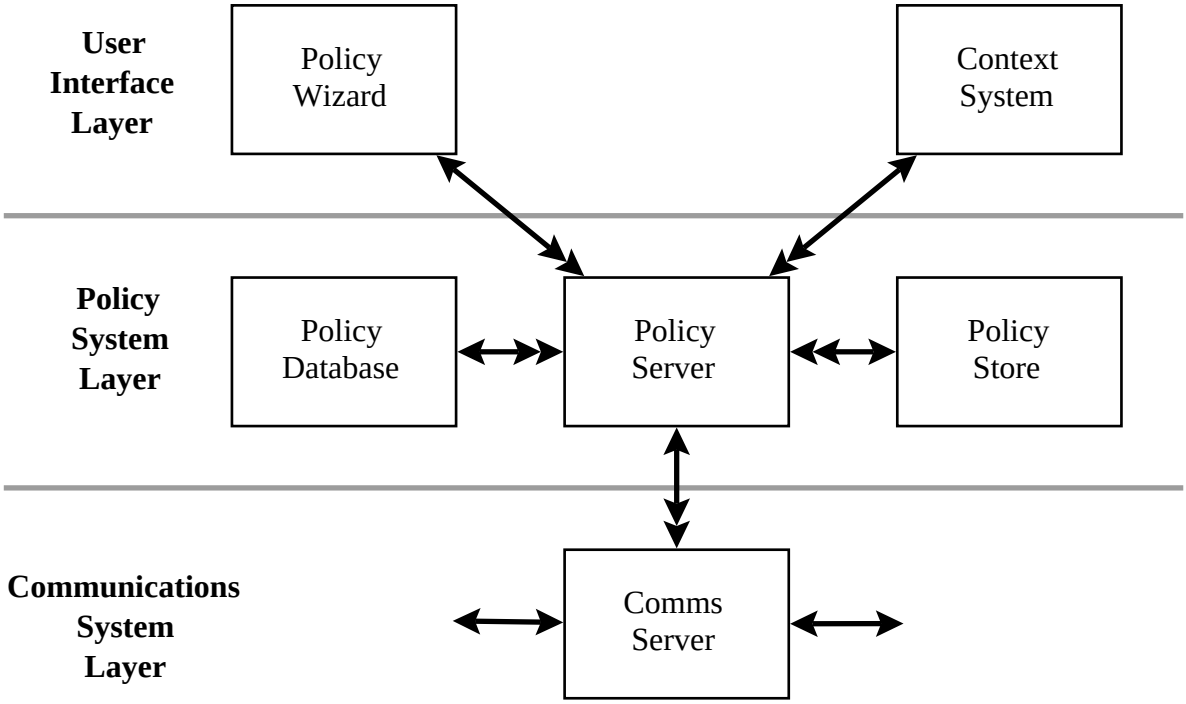

Fig. 2. Policy System Architecture

4.1 Policy System Environment

The policy language acquires meaning in the context of a domain-specific policy system. As an illustration, figure 2 shows the architecture adopted for call control. Arrows in the diagram are shown double-headed where many instances of a system may appear at this end. For example, a policy server may support many communications servers. In turn, one policy store may support many policy servers. All the arrows represent socket connections, so the system is truly distributed. The separate logical systems may, however, run on one physical system.

The policy system is conceptually divided into three layers. The user interface layer provides direct end user support. The policy system layer deals with policy handling. The communications systems layer provides support for whatever kinds of communications network are involved. These layers are deliberately separated so as to minimise their interdependence. For example, the policy system is largely independent of the underlying communications network. It has been used with both H.323 and SIP Internet telephony, and with four different types of communications server.

4.2 Communications Server

A communications server is presumed to be an existing part of the underlying communications network. For PSTN it would be a telephone exchange, for (A)IN an SSP (Service Switching Point), for H.323 a gatekeeper, and for SIP a proxy server. The policy system assumes that communications servers can provide information about calls. It is necessary to make a small

intrusion into each communications server in the form of a purpose-written server module. So far, four variants have been written: for an H.323 gatekeeper using GnuGK (*www.gnugk.org*), for any SIP proxy server that supports CGI, for the SIP Express Router (*www.iptel.org/ser*), and for the Mitel 7000 ICS (Integrated Communications Server, *http://www.mkcnetworks.com/products/7000ics.asp*). The server modules are relatively small (around 1000 lines of code). If the communications server is open-source, creating a module is reasonably straightforward. If the communications server is proprietary (like the Mitel 7000), then of course access to the server API is required.

A server module needs to be informed of significant call events such as user registration, call setup, mid-call events (such as adding a third party or new media) and call tear-down. The server module temporarily suspends call processing. It then connects to the policy server, sending trigger information such as the network type, the caller, and the callee. The specific information depends on the kind of network. Older networks such as the PSTN can provide only basic parameters. Newer approaches such as SIP can provide much more information, such as the topic of a call or how it was routed.

Trigger information is protocol-specific. The policy server therefore maintains a mapping from protocol terms to policy terms. For example an INVITE in SIP is mapped to *connect* in policy terms, while a BYE in SIP is mapped to *disconnect*. This allows policies and the policy system to operate in a protocol-independent manner. This is essential since different policies should not have to be written depending on how users are physically connected.

The policies that apply to a call dictate the actions to be performed, such as forwarding a call or logging it. A reverse mapping is performed from policy terms to protocol terms, and the actions are sent to the server module. This then instructs the communications server how to proceed with the call. The default action (if policies do not require anything specific) is to allow the call to continue as normal.

4.3 Policy Database and Policy Store

The policy database contains static information needed by the policy system. This includes the protocol to policy terminology mapping, and registered users of the policy system. A conventional relational database (MySQL, *www.mysql.com*) is used as the policy database.

The policy store contains relatively dynamic information needed by the policy system. This includes user policies, policy variables, and context information. A tuple space server (IBM TSpaces, *www.alphaworks.ibm.com/tech/tspaces*) is used as the policy store. Since policies are defined using XML, TSpaces is a good choice for storing policies as it has explicit support for storing and retrieving XML. Nonetheless, the policy server uses the policy store through a defined interface. TSpaces is just one implementation of this; a relational database or an XML database could be used as alternatives. Systems use the policy store via the policy server, so they are isolated from its exact implementation.

4.4 Policy Server

The policy server is the heart of the system. When it receives trigger information from a communications server, it retrieves policies that apply to the caller or the callee as appropriate. These policies are then filtered for applicability, eliminating those that do not apply.

Various agencies in a call may define policies that affect the call. For example if Alice makes a business call to Bob on his cellphone, the call is subject to their individual policies, those of their employers, and those of the service provider. In fact several service providers might be involved. Suppose that Alice is working off-site and staying at a hotel. When she makes a call to Bob, she is also subject to the hotel's policies.

The implementation of conflict detection and resolution is explained in [2]. The key point is that conflicts and their resolutions are *defined*, not built into the policy server. Conflicts are detected and handled by special resolution policies that resemble normal policies. For resolution, the triggers are actions that might conflict (e.g. adding and removing a call party). Condition parameters are those that might arise in a normal policy (e.g. the caller or the time). Resolution actions may either be absolute (e.g. add a specific party) or relative (e.g. prefer the caller's action). Policies that forward a call or fork it (i.e. try multiple destinations) cause extra complications. Conflict resolution must consider the sets of policies for each route so that the best resolution of conflicts can be achieved. This might result in forwarding or forking being denied.

Since the policy system has the ability to play media clips, it is possible to give the caller more information when a call cannot be put through. For example, the caller might be told: 'Alice is at a meeting, please try later'. Using Text-To-Speech, it is possible to play arbitrary announcements from a policy. However this risks compromising an individual's privacy, and must therefore be defined per user or per organisation.

4.5 Context System

Context is an increasingly important aspect of call control [20, Chapter 14]. This has a number of aspects including the following:

Presence: This means that the user is in some sense accessible. It might mean physically present (e.g. in the office), but these days is more likely to mean logically present (e.g. logged in). Presence information is used in buddy lists for ICQ ('I seek you'), and to alert cellphone users that a friend is nearby. Presence might be boolean, or might indicate a specific location. Presence information can be derived from a number of sources. For example, an active badge system can track where users are and report their location. A user's electronic diary can also provide information, e.g. that a user is currently on holiday or travelling.

Availability: This means that the user is available for communication. In conventional telephony, being off-hook means the user is unavailable. In more advanced forms of communication, it is possible to have a much more sophisticated approach. For example, suppose Ed is in his office and meeting a colleague. He considers himself unavailable for calls from

other colleagues. However, he wishes to be available for calls from important people such as managers or customers. Availability might be boolean, or might indicate topics for which the individual is available (e.g. discussions about project finance or sales predictions).

Role: The role of the call parties plays a part in determining call completion. The caller and callee addresses can be combined with an organisation chart to infer the role of the parties. Thus, principal@stir.ac.uk calling alice@stir.ac.uk implies a manager-subordinate role. In some communications networks such as H.323 and SIP, the subject of a call can be given. This can also provide a hint about the intended roles (e.g. a call about 'Your annual increment' is probably from a manager).

Capability: The capabilities of the call parties can also be important. For example, a French-speaking caller should be connected to a French speaker.

The context system is strictly outside the policy system. A full-blown context system would require techniques from artificial intelligence. Some aspects are also more sociological than computational. However the policy server supports an interface for a context system to provide supplementary information about calls, e.g. call party roles or capabilities. The policy server can act on presence and availability information from a context system.

The policy system records current user profiles, which effectively define the user situation or role. As a further demonstration of how context can link to policies, a system has been developed for presence and availability based on a user's diary. Appointments are extracted from a Microsoft Outlook calendar. Assuming reasonable use of Outlook, this allows the context system to infer presence and availability. Policies can be written to use this kind of information; see section 3.2.6 for an example.

4.6 Policy Wizard

The policy wizard is the primary interface between end users and the policy system. It allows non-technical users to define and edit policies. Presence, availability and profile can also be defined with the wizard. Considerable effort has gone into making the policy wizard easy to use. This is essential since the whole system is aimed at ordinary subscribers.

The policy wizard presents the policy system in a non-technical way. This is vital, since ordinary users must be able to take advantage of policies. It has been a challenge to define a policy system that is sufficiently comprehensive, and yet can be used easily for simple tasks. Extensive online help is provided, including the use of 'tool tips' when hovering over all aspects of a policy. Hints on forms are also provided as to the format of plausible values.

The policy wizard uses natural language. Although this is stylised natural language, it is easy to read; see figure 3 for an example. The focus of the policy system is not, however, linguistics so the approach is considered adequate. An important feature of the policy wizard is that it is multilingual. With communications being global, multilingual support is essential. Internally, the policy wizard maps between APPEL and the user's natural language.

The policy wizard is web-based. Apart from familiarity, this also means that policies can be

defined and modified while someone is off-site. For example, a user might have specified that calls be forwarded to a colleague during his absence. Anticipating a particular call, he can remotely modify his policy to forward this call to his current location. A web-based interface would be problematic for the partially sighted or for those on the move (with just a cellphone, for example). Investigations have therefore been carried out into a voice-based policy wizard using VoiceXML [21]. This is also multilingual, and can be tied into the policy system.

A large selection of template policies is provided, particularly as a convenience for the less expert user. Templates are simply selected from a list by name. Some templates define complete policies; others require the user to fill in specific values such as a forwarding address. Templates are in fact defined per locale. In principle templates are independent of the user's language and country. However the label of a template and its parameters are rendered in that language. It is also possible to vary templates by country to reflect local customs.

A user of the policy wizard has a defined skill level: novice, intermediate, expert or administrator. Less experienced users will be registered at a lower level. This automatically restricts the range of capabilities that the user sees. For example, an inexperienced user should not be exposed to guarded choice of policy rules or non-deterministic choice of policy actions. An administrator has full capabilities, including defining policies for other users. An administrator is also responsible for maintaining the details of registered users.

Among these features, the most challenging is making the policy wizard multilingual. Currently it supports English, French and German, including national variants such as United States English and Canadian French. Preliminary investigation of other languages suggests that the wizard can be adapted for many (but of course not all) languages. The main problem is a language whose sentence structure differs markedly English, since the wizard maps fragments of policies to fragments of sentences.

It is easy to incorporate a new language into the policy wizard; indeed it needs no programming. A single properties file is defined for mapping policy wizard output to that language. (A help page in the language is also required.) Achieving this simplicity has been at the expense of considerable complication in the coding. The problem is that natural language is required in many places: in web page text, in pop-up windows, in client-side or server-side code, and in form fields. The policy wizard carefully coordinates what happens at the client (form input, HTML, JavaScript) and what happens at the server (form processing, Java, JSP).

In fact, the policy system is intentionally not fully independent of the user's language. Some parameters are free-form text expressed in natural language. For example, a policy may be made dependent on a call topic such as 'manufacturing' in English or 'fabrication' in French. The policy system cannot address language translation issues. However this is unlikely to be a problem since most people communicating will use a common language. In a multilingual situation, policies can be defined to deal with multiple languages.

A further challenge for the policy wizard is allowing the user to edit the structure of complex policies. Virtually all elements of a policy displayed on the screen are hyperlinks. For example, clicking on a condition takes the user to a page where the condition can be edited. The XML schema defining APPEL has recursive definitions of many elements. The policy wizard must

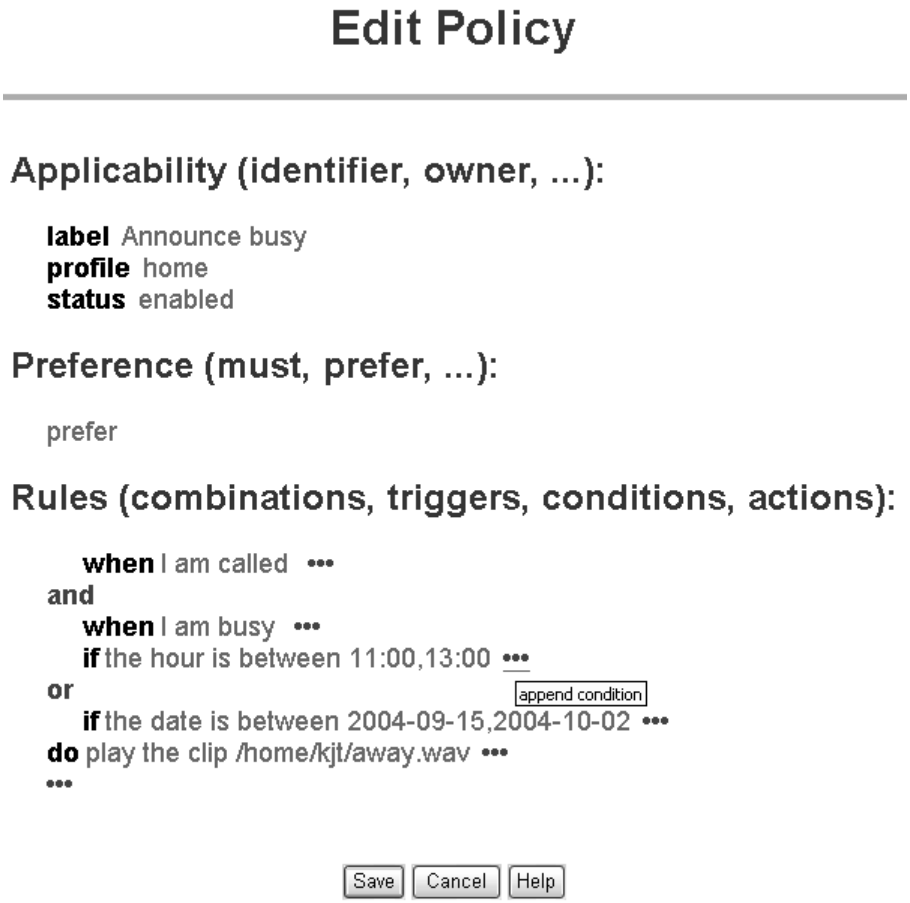

Fig. 3. Editing a Policy

therefore allow a tree structure to be edited, extended and contracted. Figure 3 shows a policy in the process of being edited. The ••• symbol appears wherever a policy element can be extended. In figure 3, the user has hovered over ••• after a condition. The 'append condition' tool tip indicates that a further condition can be added at this point. Clicking on ••• takes the user to a page where the combination of conditions can be selected. Choosing **and**, for example, combines the existing condition with a new blank condition. The new condition can then be edited. A combination can be contracted to one of its branches.

Once a policy has been finished, the policy wizard saves it to the policy server (which writes it to the policy store under the user's address and policy label). An existing policy can later be retrieved and edited further. The policy wizard, of course, creates policies that conform to the APPEL schema. More importantly, it also checks the static semantics of a policy. For example, conditions and actions must match triggers. For experts, the policy server also provides direct upload of policies. However they must then be created and validated in XML form.

5 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. The APPEL policy language has been presented in its core aspects and in its specialisation for call control. APPEL has met the challenges noted in section 1.2:

- it is focused on call control
- the core language can also be specialised for other domains, including various forms of communications systems
- it conforms to call control principles, avoiding concepts such as subject and target that cannot be readily related to this domain
- the approach allows many unknown users to communicate, taking into account policies from their organisations and their service providers as well
- the policy language and the policy system are accessible to ordinary users, allowing them to use their native language when defining policies
- a comprehensive strategy has been worked out for handling policy conflicts, making use of guidance that end users can readily give
- policy modalities can be defined in a straightforward way by end users
- the policy language uses XML as a widely accepted interchange format.

The ACCENT policy system architecture has been introduced. As the major components, the policy server and the policy wizard have been explained in some depth. They allow non-technical users to gain the benefits of policies. The policy system also links to other emerging systems providing contextual information.

The policy system is currently operational in a laboratory setting. So far it has been evaluated by the researchers and by others not involved in the development. Wider-scale industrial deployment is expected soon. In operation, the policy system is largely invisible to users and so does not place any technical demands on them. The policy wizard has been carefully designed for ordinary subscribers to benefit from the use of policies. In fact, it is easier to use the policy wizard than the special dialling codes normally required for functions like call forwarding or call blocking. It is planned to conduct a study of usability for end users, and an appraisal of performance under a variety of conditions.

Acknowledgements

The ACCENT project was funded by EPSRC (the UK Engineering and Physical Sciences Research Council, grant R31263) and by Mitel Networks (Canada). While at the University of Stirling, Stephan Reiff-Marganiec undertook the original design and implementation of the policy language and policy server. Grégory Estienne kindly improved the support of French in the policy wizard, while German was added by Mario Kolberg (University of Stirling).

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] L. Blair and K. J. Turner. Handling policy conflicts in call control. In S. Reiff-Marganiec and M. D. Ryan, editors, *Proc. 8th. Feature Interactions in Telecommunications and Software Systems*, Lecture Notes in Computer Science. Springer, Berlin, Germany, Mar. 2005. In press.

[3] 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.

[4] N. Gans, G. Koole, and A. Mandelbaum. Telephone call centers: Tutorial, review, and research prospects. *Manufacturing and Service Operations Management*, 5:79–141, Sept. 2002.

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

[6] T. Huang and K. J. Turner. Policy support for H.323 call handling. *Computer Standards and Interfaces*, Jan. 2005. In press.

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

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

[9] 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.

[10] J. Lennox, H. Schulzrinne, and J. Rosenberg, editors. *Common Gateway Interface for SIP*. RFC 3050. The Internet Society, New York, USA, Jan. 2001.

[11] E. C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans. on Software Engineering*, 25(6):852–869, Nov. 1999.

[12] S. Reiff-Marganiec. Policies: Giving user control over calls. In M. D. Ryan, J.-J. C. Meyer, and H.-D. Ehrich, editors, *Objects, Agents and Features*, number 2975 in Lecture Notes in Computer Science, pages 189–208. Springer, Berlin, Germany, May 2004.

[13] 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.

[14] 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.

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

[16] 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, May 2005.

[17] 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, May 2005.

[18] 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.

[19] K. J. Turner. The ACCENT policy wizard. Technical Report CSM-166, Department of Computing Science and Mathematics, University of Stirling, UK, May 2005.

[20] K. J. Turner, E. H. Magill, and D. J. Marples, editors. *Service Provision – Technologies for Next Generation Communications*. John Wiley and Sons, Chichester, UK, Mar. 2004.

[21] VoiceXML Forum. *Voice eXtensible Markup Language*. VoiceXML Version 2.0. VoiceXML Forum, Jan. 2003.