Stephan Reiff-Marganiec and Kenneth J. Turner. A Policy Architecture for Enhancing and Controlling Features. In Daniel Amyot and Luigi Logrippo, editors, Proc. Feature Interactions in Telecommunication Networks VII, 239-246, IOS Press, Amsterdam, June 2003.

A Policy Architecture for Enhancing and Controlling Features

Stephan Reiff-Marganiec and Kenneth J. Turner
Department of Computing Science and Mathematics
University of Stirling, FK9 4LA Stirling, Scotland, UK
{srm,kjt}@cs.stir.ac.uk

Abstract. Features provide extensions to a basic service, but in new systems users require much greater flexibility oriented towards their needs. Traditional features do not easily allow for this. We propose policies as the features of the future. Policies can be defined by the end-user, and allow for the use of rich context information when controlling calls. This paper introduces an architecture for policy definition and call control by policies. We discuss the operation of systems based on such an architecture. An important aspect of the architecture is integral feature interaction handling.

1 Motivation

Telecommunications has a central role in daily life, be it private or within the enterprise. We have left behind times when two users were simply connected for a verbal communication and now encounter a trend towards merging different communications technologies such as video conferencing, email, Voice over IP as well as new technologies like home automation and device control. This is combined with a move to greater mobility, e.g. wireless communications, mobile telephony and ad hoc networking. Services such as conference calling and voice mail have been added to help deal with situations beyond simple telephony.

Currently such services only support communication, that is they allow the user to simplify and more closely integrate telecommunications in their activities. In the future services will make use of the merged technologies on any device. We believe that services then will enable communications by allowing the user to achieve particular goals.

Increasing numbers of mobile users with multiple communication devices lead to a situation where users are always reachable. However, users might not always wish to be disturbed, or at least not for everyone or for any type of enquiry. Future services need to provide support for users to control their *availability*.

Availability will be highly dependent on the *context* of the user. Services must, amongst others, take into account where users are, what devices they currently use and who they might be with, as well as simple concepts such as time of day.

We conclude that the end-user must have a central place in communications systems. Services must be highly customizable by lay end-users, as only the individual is aware of his requirements. It might even be desirable for end-users to define their own services. However, any customization or service development must be simple and intuitive to suit lay users.

We discuss the advantages of policies over features, and define an architecture in which policies can be used to control calls. The operation of a system based on such an architecture is discussed. We will find that policies do not remove the feature interaction problem, but provide different angles on possible solutions. Interaction detection and resolution mechanisms will form an essential part of the proposed system. The system is user oriented and addresses the fact that future call control needs to enable individuals to achieve their goals.

2 Background

Policies and Services. Policies are defined as *information which can be used to modify the behaviour of a system* [6]. Considerable interest has been aroused by policies in the context of multimedia and distributed systems. Policies have also been applied to express the enterprise viewpoint [10] of ODP (Open Distributed Processing) and agent based systems [1].

In telecommunications systems, customized functionality for the user is traditionally achieved by providing services, i.e. capabilities offered on top of a basic service. Services are supplied by the network provider and thus do not offer completely customized functionality. Consider a call forwarding service: the user chooses whether the service is available or not, and which number the call gets forwarded to. There is no possibility for the user to forward only some calls or to treat certain calls differently, e.g. forward private calls to the mobile and others to a voicemail facility. It is exactly the flexibility, adaptability and end-user definition of policies that makes them an ideal candidate technology for services of the future.

Policies and Feature Interaction. One might hope that policies remove the feature interaction problem, simply by being higher-level. The policy community has recognised that there are issues, referred to as *policy conflict*, but has not considered any general solutions, assuming that this is not a crucial problem.

On the one hand, policies are defined by end-users, so policies will be larger in number and more diverse than features. Also, the lay nature of the user adds to the problem. On the other hand, policies can contain preferences. The context can also provide priorities usable to resolve conflicts in conjunction with richer protocols of new communications architectures.

3 The Policy Architecture

We propose a three layer architecture consisting of (1) the communications layer, (2) the policy layer and (3) the user interface layer. A three-tier architecture is used in completely different ways for other applications. However, a three-tier policy architecture emerges naturally.

When we consider existing call control architectures, similar architectures have been in use some time. For example, in the IN the three layers are given by the SSP (Service Switching Point), SCP (Service Control Point) and SCE (Service Creation Environment). Similarly in a SIP environment the layers are provided by the SIP proxies, CPL or CGI scripts and tools for creation of such scripts.

However our proposed architecture (Figure 1) differs in several key aspects, from similar existing architectures (e.g. the IN or SIP). Some of these aspects are:

- The Policy Servers can negotiate goals or solutions to detected problems.
- The User Interface Layer provides end-users with a mechanism to define functionality.
- The User Interface Layer and the Policy Servers make use of context information.

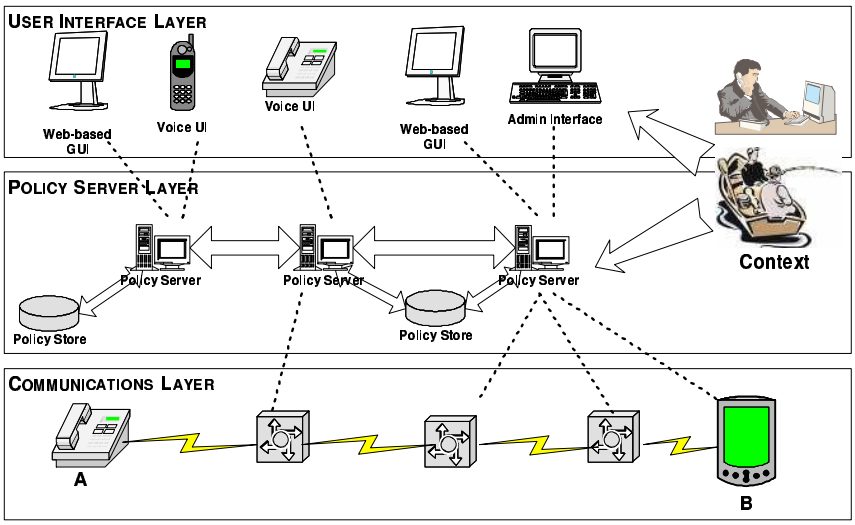

Figure 1: Overview of the Proposed Architecture

- The Policy Server layer is independent of the underlying call architecture.

The architecture makes the underlying communications network transparent to the higher layers. It enables end-user configurability and provides communication between policy servers which can be used for interaction handling. We will briefly discuss the details of each layer.

The Communications Layer. This represents the chosen call architecture. For this paper, we assume a general structure consisting of end-devices and a number of switching points. We impose two crucial requirements on the communications layer. (1) The policy servers must be provided with any message that arrives at a switching point; routing is suspended until the policy server has dealt with the message. (2) A mapping of low level messages of the communications layer into more abstract policy events must defined. Our investigations have shown that these are realistic assumption.

The Policy Server Layer. This contains a number of policy servers that interact with the underlying call architecture. It also contains a number of policy stores (database or tuple space servers) where policies are maintained by the policy servers as required. We assume that several policy servers might share a policy store, and also that each policy server might control more than one switching point or apply to more than one end device. The policy servers interact with the user interfaces in the policy creation process discussed in section 4. They also interact with the communications layer where policies are enforced; discussed in detail in section 5. The policy servers have access to up-to-date information about the user's context details which are used to influence call functionality.

The User Interface Layer. This allows users to create new policies and deploy these in the network. A number of interfaces can be expected here. We would assume the normal user to use a web-based interface for most functions. For mobile users, voice controlled interfaces are more appropriate. A voice interface is essential for disabled or partially sighted users. Both web and voice interfaces should guide the user in an intuitive way, preferably in natural language or in a graphical fashion. We also envision libraries of policies that users can simply adapt to their requirements and combine to obtain the functionality required, in a similar way that for example clip-art libraries are common today. System-oriented administrative interfaces exist

for system administrators to manage more complex functionality.

4 Defining Policies

Policies should provide the end-user with capabilities to get the most out of their communications systems. End-users usually use their communications devices in a social or commercial context that imposes further policies. For example an employee is often subject to company policies. Hence policies will be defined at different domain levels (users, groups, companies, social clubs, customer premises, etc.) by differently skilled users. Any policy definition process needs to take this into account.

A Policy Description Language. In previous work [9] we have introduced initial ideas for a policy description language (PDL) to express call control policies. Here we present detail relevant for this paper. We have analysed a set of more than 100 policies before defining this language. Further, any traditional feature can also be expressed. The policy definition language is defined as an XML schema and hence policies will be stored as XML.

Complex policies are combinations of policy rules. There are a number of ways in which simple rules can be combined: sequencing, parallel composition (simultaneous application of rules), guarded and unguarded choice (conditional choice of one rule or indifference as to which rule is applied). Each policy is uniquely identified by an id and can be activated or deactivated. Also, each policy states who it applies to; this can be the person who defined the policy or everyone within a certain domain. This mechanism allows policies to be defined at different levels, e.g. individual, group and enterprise.

Policy rules provide a means to express simple facts about what a user wishes to happen with her communications. Each policy rule is composed of a modalities block, an action block, a condition block and and an trigger event block. All parts but the actions part are optional.

We consider a number of modalities such as obligation, permission and interdiction. 'Never' and 'always' are further simple modalities. Preferences, e.g. wish or must, are highly relevant for call control policies and are considered as fuzzy modalities. Finally a class of temporal modalities, containing items like 'in the future', 'periodically' or 'now', is defined. Interactions can occur within each of these modality groups, but modalities can also be used by conflict resolution approaches to identify which policy should be given precedence.

The action block simply contains one or more instructions to be executed when the policy is applied. These instructions are typically actions as provided by the target system. In a call processing system, they can be actions such as "forward call", "originate call" or "contact". Note that at this level we are concerned with abstract actions which are mapped by the policy server to messages in the underlying communications layer.

A trigger event block describes when the policy should be applied: if it exists the policy should only be considered when the specified trigger occurs. The omission of a trigger block means that the rule is always to be considered, and we would refer to such a rule as goal.

Conditions restrict the applicability of a policy rule further. Typical conditions are equalities or inequalities on parameters associated with the call. There is a large set of these parameters, and others can be readily added. Examples are: caller (a user), call content (email, video, language), media (fixed, mobile, high speed), call type (emergency, long distance, intra-company), cost (of the call) or topic (project x, weekend plans). Other conditions are based on the context and attributes of users such as location (my office, at customer site), identity, role) (Mary, service representative) , and capabilities (Java expert, German speaker).

As it is impractical to require the user to provide the relevant information when establishing a call, most of the required information will be inferred from the context. For example, roles may be defined in a company organisation chart, the location can be established from the user's diary or mobile home location register. Or better, "mobile devices have the promise to provide context-sensing capabilities. They will know their location – often a critical indicator to the user's tasks." [4]. So for example, if you are in your boss's office, it is probably an important meeting and you do not wish to be disturbed.

User Interfaces. The presented language is quite expressive. However, we do not expect the end user to define policies directly in the policy description language. This would be unrealistic, as we expect lay users to be able to define their policies. We have developed a simple wizard that aids the user in creating and handling policies. The current interface is web-based, so users can access it from anywhere, with voice based interfaces being considered. A typical end-user can create new policies or edit, (de)activate and delete existing policies.

The whole process takes place in natural language and the gaps are filled by selecting values from the context (e.g. time from clocks and users or domains from hierarchies). Furthermore, the approach is language-independent so that in principle the user can work in his native language. Once a policy is finished, the user submits the policy to the policy server.

Upload of Policies. A policy server provides an interface via a TCP/IP socket to receive changes to policies (with appropriate authorisation). The user interface layer connects to this socket to submit the gathered information and to receive any feedback on success or failure through this connection. In the absence of the feature interaction problem, the policy store is updated to reflect the new policy. However, as conflicts are rather likely, a consistency check is performed on the set of policies applicable to the user before updating policies.

Interactions. An architecture that does not give rise for conflicts appears to be possible only at the cost of reducing the expressiveness of the policy language to trivial levels. We introduce a guided design process that automatically checks policies for the presence of conflict and presents any detected problems to the user, together with suggested resolutions.

When a policy is uploaded, it is checked against other policies from the same user, but also against policies that the user might be subject to (e.g. due to her role in the enterprise). Here we check for static interactions, i.e. those that are inherent to the policies. This suggests the use of offline detection methods and filtering techniques. Any inconsistencies detected need to be reported to the user. The resolution mechanism is either a redesign of the policy base or provisioning of information to guide online approaches.

Some methods considered most appropriate for the policy context are Anise [12] (predefined or user-defined operators allow to build progressively more complex features while avoiding certain forms of interaction) and Zave and Jackson's [14] Pipe and Filter approach. Also, Dahl and Najm [3] (occurrence of common gates in two processes) and Thomas [11] (guarded choices), where the occurrence of non-determinism highlights the potential for interactions are suitable. Note that these methods are applied at policy definition time, so execution times are less of an issue (as long as they stay within reason).

In fact we are not restricted to static interactions: we can also detect the potential for conflict. That is, we can filter cases where an interaction might occur depending on contextual data. A suitable approach might be derived from the work of Kolberg et al. [5].

Consistency could be checked at the user's device rather than the policy server. However, performing the check in the policy server has the major advantage that in addition to the user's policies, those from the same domain are accessible and can be considered. Furthermore, the

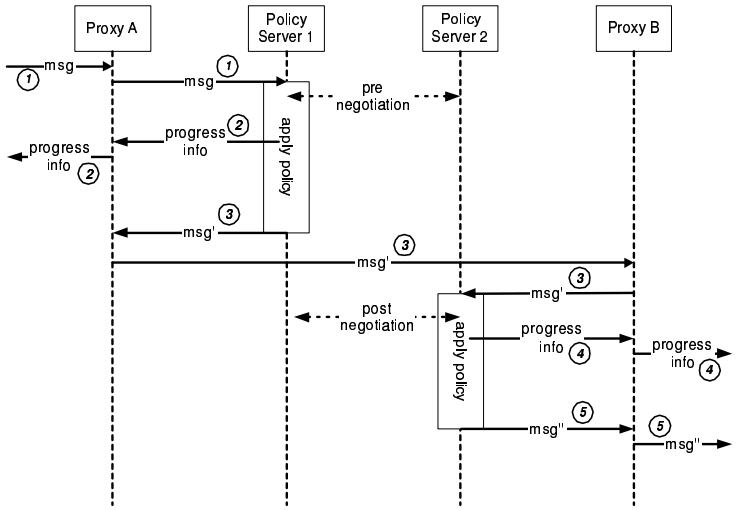

Figure 2: Policy Enforcement Process

user side of the implementation is kept light-weight, which is important for less powerful end-devices such as mobile phones or PDAs.

5 The Call Process

In the previous section we have discussed how users can define and upload their policies to the policy server. Now, we discuss how policies are enforced to achieve the goals they describe.

Applying Policies to a Call. When attempting to setup a call from A to B, A's end device will generate a message that is sent via a number of switching points to B's end device. Assuming that policy servers are associated with switching points, every switching point along the call path can sent the message to the policy server. Routing is suspended until the policy server allows continuation. In SIP, we can intercept, modify, fork and block messages by using SIP CGI.

The policy server processes any messages that it receives in a four stages: (1) all policies applicable to the source or target of the message are retrieved. (2) The policies are filtered, removing those where the trigger event does not match the received event and those where additional conditions are not satisfied by the current context. (3) The remaining policies are analysed for conflict and these are resolved. (4) The outgoing message(s) are produced. If there are no applicable policies, the outgoing message is simply the incoming message. If one or more policies apply, the required changes are made to the original message. Alternatively, new or additional messages can be created. The example in Figure 2 shows "progress info" messages that are generated by the policy server. Also, if a policy requires forking of a call (e.g. "always include my boss in calls to lawyers") the respective messages to setup the extra call leg need to be created.

Routing is resumed as normal, with the next switching point again forwarding the message to a policy server for the application of policies. Figure 2 shows an example of this process for a call between two parties where policies are enforced in one direction. Note that the figure shows two message exchanges between the policy servers, labelled pre- and post-negotiation.

More Interactions. While a call is actually taking place more interactions can arise, either forced by context information or between policies of the different parties involved. Any de-

tection mechanism incorporated in the enforcement part of the policy server needs to be able to detect and resolve such conflicts. The introduction of policy support must also not create unreasonable delays in call setups. However, if users are aware that complex steps are needed to resolve sophisticated policies, they should learn to live with the delays – and probably are happy to do so when the outcome is productive for them. It is for this reason that intermediate information messages are produced by the policy application process.

Online and hybrid feature interaction approaches are a possible solution. We believe that in a policy context the available information is sufficient to resolve conflicts. The underlying architecture provides protocols that are rich enough to facilitate exchange of a wide range of information. There are essentially two classes of run-time approaches. One is based around the idea of negotiating agents, the other around a (central) feature manager.

The feature manager in [7] detects interactions by recognising that different features are activated and wish to control the call. The resolution mechanism for this approach [8] is based on general rules describing desired and undesired behaviour. In negotiation approaches, features communicate with each other to achieve their respective goals [13]. Buhr et al. [2] use a blackboard technique for the negotiation, thus introducing a central entity.

Feature manager approaches lend themselves to the policy architecture, as their main requirement is that the feature manager is located in the call path. This is naturally the case with policy servers. However, feature manager approaches so far have suffered from a lack of information to resolve interactions (though some progress has been reported in [8]). Negotiation approaches can fill this gap. Their current handicap is the sophisticated exchange of information required; however this is addressed by allowing communication channels between policy servers. We foresee a combination of the two techniques to exploit their individual benefits.

Two forms of negotiation are practical in the policy architecture: pre- and post-negotiation. In the former case, the policy server contacts the policy server at the remote end and negotiates call setup details to explore a mutually acceptable call setup. The communications layer is then instructed to setup the call accordingly. In post-negotiation, the policies are applied while the call is being setup, thus potentially leading to unrecoverable problems.

6 Conclusion and Further Work

Evaluation. We have considered how policies can be used in the context of call control, especially how they can be seen as the next generation of features. The policy architecture allows calls to be controlled by policies. Each policy might make use of the context of a user. This allows context-oriented call routing, but goes far beyond routing by allowing for availability and presence of users to be expressed. Therefore, we can achieve truly non-intrusive communications that enable users to achieve their goals. Policies can be easily defined and changed by the end-user via a number of interfaces (e.g. Web or voice interfaces).

We have suggested some offline feature interaction techniques for policies to be checked for consistency when they are designed. They can be applied to new policies as well as to existing ones within the same domain. However, calls will eventually cross domains; then online techniques to detect and resolve any conflicts will be required.

Future Work. A prototype environment to create and enforce policies has been developed on top of a SIP architecture. Thus the proposed architecture has been implemented. The methods for detecting and resolving policy conflict identified in this paper need to be implemented in the prototype such that empirical data on their suitability can be gathered. These methods

require some more formal understanding of the policy language as well as the communications layers to be used (the mapping of low-level messages to policy events).

In the future we would like to test the upper layers on top of other communications layers. For example H.323 and PBX are planned. Further development of additional user interfaces should strengthen the prototype. Another research area is the automatic gathering of context details, which is interesting in itself but beyond the scope of our work.

Acknowledgements

This work has been supported by EPSRC (Engineering and Physical Sciences Research Council) under grant GR/R31263 and Mitel Networks. We thank all people who contributed to the discussion of policies in the context of call control. Particular thanks are due to Tom Gray, Evan Magill and Mario Kolberg.

References

[1] M. Barbuceanu, T. Gray, and S. Mankovski. How to make your agents fulfil their obligations. *Proceedings of the 3rd Int Conf on the Practical Applications of Agents and Multi-Agent Systems*, 1998.

[2] R. J. A. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. Feature-interaction visualization and resolution in an agent environment. In K. Kimbler and L. G. Bouma, editors, *Feature Interaction in Telecommunications Networks V*, pages 135–149. IOS Press (Amsterdam), 1998.

[3] O. C. Dahl and E. Najm. Specification and detection of IN service interference using LOTOS. *Proc. Formal Description Techniques VI*, pages 53–70, 1994.

[4] A. Fano and A. Gersham. The future of business services in the age of ubiquitous computing. *Communications of the ACM*, 45(12):83–87, 2002.

[5] M. Kolberg and E. H. Magill. A pragmatic approach to service interaction filtering between call control services. *Computer Networks*, 38(5):591–602, 2002.

[6] E. Lupu and M. Sloman. Conflicts in policy based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6), November/December 1999.

[7] D. Marples and E. H. Magill. The use of rollback to prevent incorrect operation of features in intelligent network based systems. In K. Kimbler and L. G. Bouma, editors, *Feature Interaction in Telecommunications Networks V*, pages 115–134. IOS Press (Amsterdam), 1998.

[8] S. Reiff-Marganiec. *Runtime Resolution of Feature Interactions in Evolving Telecommunications Systems*. PhD thesis, University of Glasgow, Department of Computer Science, Glasgow (UK), May 2002.

[9] S. Reiff-Marganiec and K. J. Turner. Use of logic to describe enhanced communications services. In D. Peled and M. Vardi, editors, *Formal Techniques for Networked and Distributed Systems (FORTE 2002)*, pages 130–145. Springer Verlag, November 2002.

[10] M. W. A. Steen and J. Derrick. Formalising ODP Enterprise Policies. In *3rd International Enterprise Distributed Object Computing Conference*. IEEE Publishing, September 1999.

[11] M. Thomas. Modelling and analysing user views of telecommunications services. In P. Dini, R. Boutaba, and L. Logrippo, editors, *Feature Interaction in Telecommunications Networks IV*, pages 168–182. IOS Press (Amsterdam), 1997.

[12] K. J. Turner. Realising architectural feature descriptions using LOTOS. *Networks and Distributed Systems*, 12(2):145–187, 2000.

[13] H. Velthuijsen. Distributed artificial intelligence for runtime feature interaction resolution. *Computer*, 26(8):48–55, 1993.

[14] P. Zave and M. Jackson. New feature interactions in mobile and multimedia telecommunication services. In M. Calder and E. Magill, editors, *Feature Interaction in Telecommunications and Software Systems VI*, pages 51–66. IOS Press (Amsterdam), 2000.