Stephan Reiff-Marganiec and Kenneth J. Turner,
Feature Interaction in Policies, Computer Networks,
45(5):569-584, Aug. 2004

Feature Interaction in Policies

Stephan Reiff-Marganiec * Kenneth J. Turner

Department of Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, United Kingdom

Abstract

Feature interaction is a problem mostly considered in the context of telephony features, but present in other domains. In this paper we consider policies (independent of the system that they control) as an application domain for feature interaction techniques. We present the feature interaction problem as it occurs in the policy context and show how it can be approached. We give a taxonomy for policy conflict, and introduce a generic architecture for handling policy conflict.

Key words: Feature Interaction, Policy, Policy Conflict

1 Introduction

Feature interaction has been identified as a problem in the telecommunications domain, where features are added as additional functionality to a basic system. This basic system used to be POTS, the plain old telephone service. However, it has since been recognised that the feature interaction problem occurs in many other application domains. These include multimedia, lift control systems, interactive voice response, internet telephony and home appliances. Problems that are very similar, but have not been considered much, are component-based systems especially web services and policy conflict. In this paper we concentrate on policy conflict.

The past few years have shown an increased interest in controlling systems by policies. An active research community is considering policies for access

* Since the initial submission of the paper, the corresponding author, Stephan Reiff-Marganiec, has moved to the University of Leicester. The new contact details are: Department of Computer Science, University of Leicester, Leicester LE1 7RH, UK.

Email addresses: srm13@le.ac.uk (Stephan Reiff-Marganiec), kjt@cs.stir.ac.uk (Kenneth J. Turner).

control, system management and QoS (Quality of Service) control. While this community has recognised that policy conflict is a problem it has been mostly put aside, often as an separate problem, to be addressed at a later stage. However, this problem will hinder the further uptake of policies. Also, when policies are moved outside their traditional domains the problem increases: for example if end-users employ policies to specify their needs.

These considerations make it timely to investigate the policy conflict problem. Due to its similarities to feature interaction, the feature interaction community can offer mature solutions. Many conflict issues are inherent to policies. We present a taxonomy to provide structure for a rather complex, multi-dimensional problem. As policies can be used to control systems, we also suggest a generic architecture that allows for this but can handle policy conflict.

Section 2 provides the setting for the work and introduces the fundamental concepts. Section 3 justifies policies as a new application domain. We introduce our taxonomy in section 4 and show how conflicts can manifest themselves in section 5. The latter illustrates the ideas with examples. Section 6 brings the more conceptual issues discussed in the taxonomy and examples into the context of a practical and open architecture. We conclude with a discussion of the paper and more general issues concerning policies as an emerging domain for feature interaction research.

2 Background

The work in this paper was influenced by and built upon work on feature interaction work on policies for system management. In general the idea of both features and policies is to adapt the behaviour of an existing system. Central are the concepts of feature, feature interaction, policy and policy conflict.

Features stem from the telecommunications industry, but similar concepts exist in other areas such as component-based systems. In general a feature is new functionality to enhance a base system. Features are often developed in isolation. Each individual feature's operation is tested with respect to the base system, and also with common known features.

Unfortunately, when two or more features are added to a base system, unexpected behaviour might occur. This is caused by the features influencing each other, and is referred to as feature interaction. Feature interaction is a technical issue with considerable significance for service developers and the public at large. As can be imagined, it is very costly to check for feature interactions and to implement solutions. It is common experience that introducing new features breaks others. For the ordinary telephone user, feature interactions may

result in unexpected behaviour or even failure of calls. This is unacceptable given the dependence of commerce and even safety on telephony.

Feature interaction has therefore been extensively studied and researched. A general discussion of the problem appears in [9,12,13]. The literature on feature interaction is large. A good source of information are the proceedings of the Feature Interaction Workshop, [6,10,14,18,22,37].

Policies are high-level statements to support personal, organisational or system goals. Policies have been defined as *information which can be used to modify the behaviour of a system* [25]. Policies have been particularly studied in applications such as distributed systems management, network management, Quality of Service, and access control. Many important papers are collected in the proceedings of the workshop on Policies for Distributed Systems and Networks [27,33]. Policies are not singular entities, they are generally arranged in groups or networks and collectively express overall goals.

Of particular relevance is the work on policy conflict, which can be regarded as similar to the feature interaction problem. In a distributed setting, policies may contradict since they may be set by different organisations or at different levels in the same organisation. Detecting and resolving such conflicts is vital.

Surprisingly, there does not appear to have been much work on policy conflicts. [19] recognises but does not address conflicts that arise in policy-driven adaptation mechanisms. [1] aims to define hierarchical policies such that, by definition, the subordinate policies cannot conflict. Conflicts are still possible if one policy in the hierarchy is changed. The use of meta-policies (policies about policies) is proposed as a solution, e.g. in [25], where meta-policy checks are applied when policies are specified and when they are executed. Similar ideas, where predefined rules and good understanding of the domain allow resolution of conflicts, are presented in [28]. In [5], it is anticipated that authorisation policies may lead to conflict. This is resolved by providing a function to compare policies and decide which should take precedence.

3 Policies as an Emerging Domain

Policy was a term much used at the 2003 Feature Interaction Workshop [2], showing that the feature interaction community has developed an interest in this topic. Their first mention in this context was in [1]. The views on policies and their use in a telecommunications setting vary widely.

Moving outside the telecommunications domain, Yee and Korba consider conflicts that arise during the use of privacy policies [39]. They introduce the

issues arising and suggests the use of negotiation to detect and resolve conflicts. However, they do not present any concrete techniques.

Other policy work is more concentrated on telecommunications. For example, [17] presents a feature interaction manager system where policies are used to steer the composition of services to avoid feature interactions.

The work by Gray *et al.* [21] describes an architecture to support features in a social context, following similar motivation to our own work. They foresee policy execution engines in their tripartite architecture, where policies replace features in all aspects where user intentions need to be expressed. In our work [32] we attempt to provide users with control over their communications through user-defined policies. The high level aim is to consider the purposes and not the mechanisms, as discussed in terms of features in [40]. In [32] we have discussed how a call control system can be enhanced by the use of policies, and have even suggested that policies might replace features in the future. We have considered policy conflict, but restricted our considerations to the application domain of call control.

In this paper we do not consider a particular system that can be enhanced by policies, but rather consider policies in their own right. Policies have proven useful and suitable in several domains; in all these domains the problem of policy conflict occurs. Our stance here is that the problem of policy conflict is inherent to policies and independent of the controlled system – although the latter might add conflicts of its own. The goal is to study policy conflict in general, leading to resolution strategies that can be particularised for each respective domain.

For the work in this paper we require a clear notion of what it means for policies to conflict. In an application context we say that two or more policies conflict when they are applicable at the same time and their actions conflict. This definition is only valid in a specific application domain as one must be aware of what conflicting actions are. For clarity, consider example 3.1, which shows conflicting actions in three distinct domains and it should be obvious that these actions are only conflicting because of the specific domain semantics.

Example 3.1 *The following are conflicting actions:*

Access Control: allowing access *and* denying access
Telephony: forwarding *and* blocking *a call*
POTS: forwarding to A *and* forwarding to B *with A and B being distinct.*

Several types of conflict exist in the policies themselves. In this paper we will concentrate on these.

4 A Taxonomy for Policy Conflict

Earlier we have discussed how policies are gaining popularity in many application areas. However, we have also identified that policy conflict is a problem that is inherent to policies and independent of the application domain. We consider policies to be a new application domain for feature interaction techniques.

In the early days of feature interaction research, researchers attempted to tackle a problem that was informally understood and manifold in its complexity. It is probably fair to say that the research into feature interaction detection and resolution techniques gained much structure after the publication of the feature interaction benchmark paper [11]. This provided a taxonomy for feature interaction and gave concrete examples. The latter was possible due to the restricted application domain, namely telecommunication systems. In this section we provide a taxonomy for policy conflict, in the hope of providing some structure for further research in policy conflict. Note that the taxonomy is an initial classification that we expect to be developed further.

4.1 Dimensions of Conflicts

We identify five principle dimensions of policy conflict: policy types, domain entities, roles, policy relations and modalities. Let us first discuss each of these dimensions before considering the conflicts that they can cause. Figure 1 represents an overview of the five dimensions. The Ponder policy framework [16] distinguishes authorisation and obligation policies by their point of enforcement: subject or target. This would form a suitable further dimension, but we believe that most conflicts in this category can be classified in either the domain entity or the modality dimensions.

Analysis of individual policies allows us to position them along each of the dimensions. In general the further a policy is from the origin, the more complex the conflict handling. We will now examine each of the dimensions.

4.2 Policy Types

We have identified a general structure of policies [32], which we have considered in two categories: event-condition-action (ECA) rules and goals. The former specify a trigger event which will lead to the action when the condition is fulfilled. Goals are somewhat more abstract, in that they do not specify a trigger event (we could see them as "CA rules"): they simply lead to the

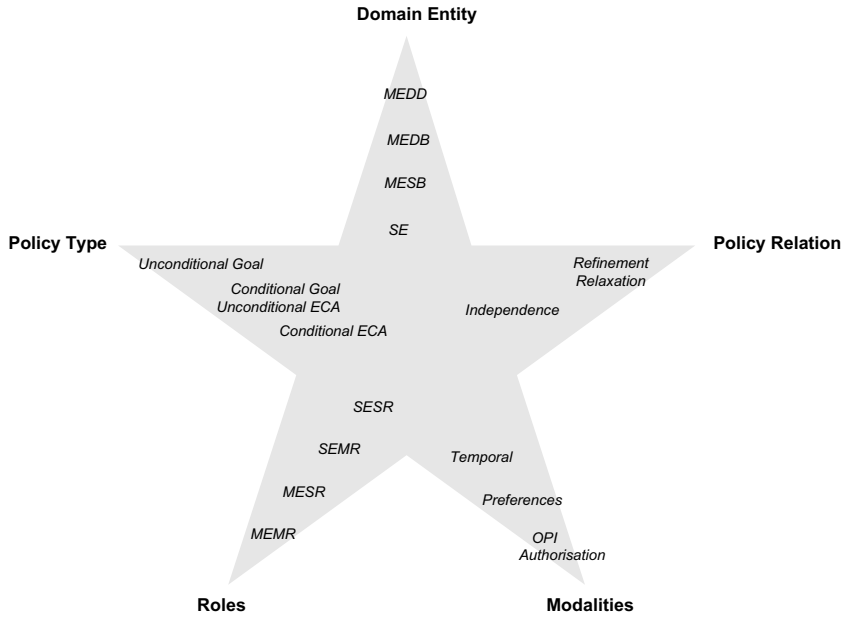

Fig. 1. Dimensions of Policy Conflict

stated action when the condition holds. However, we allow both types to omit the condition (i.e. the condition is trivially true).

For the purpose of the taxonomy we can distinguish four types of policies: unconditional goals (i.e. just actions), conditional goals, unconditional ECAs (i.e. an ECA with true as condition) and conditional ECAs. Clearly, if a policy only specifies an action, the notion of conflict is purely dependent on the application domain as the controlled system describes conflict between actions. However, if further information, such as conditions and triggers are available in the policy conflicts can be detected independent of the application domain (e.g. by detecting contradictory conditions). For example, a conditional goal with many conditions might provide much more information and hence be more suitable for domain-independent conflict detection than an ECA with a single, simple condition. A more concrete example is one where two rules have the same action, however the conditions of one prescribe that the action should never be executed during working hours, while the other states that the action should never be executed outside working hours – the conflict here is clear: the action can never be executed. This conflict can be detected independent of the application domain.

Here we would like to point out that none of the dimensions allows for a total ordering.

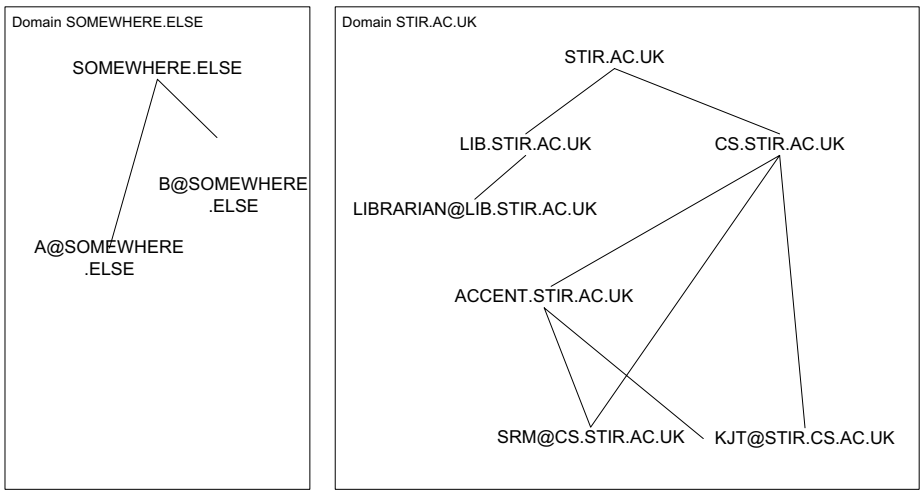

Fig. 2. A Typical Enterprise Domain Hierarchy

4.3 Domain Entities

Policies regulate the application domain by providing rules for the subjects that they talk about. Considering a business environment, one could see policies stipulated by the enterprise that apply to all employees, and also others that are more specific to certain work groups or to certain job profiles. Again, other policies might be specific to individuals in the organisation. This concept can be lifted from a business environment and be applied in a more general context. We often find that the subjects of the policies are related to each other in a hierarchical sense. However, entities might be part of the same hierarchy or of a different hierarchy; with the former their relation is obvious, with the latter we would consider the entities unrelated. An overlap of domain is not excluded. Note that we must assume that policies are maintained in a distributed fashion and that different domains or even entities far apart within a specific hierarchy are unwilling to exchange details of policies.

We have identified four relations among domain entities based on their place in the hierarchies: SE (the single entity), MESB (multiple entities in the same domain on the same branch), MEDB (multiple entities in the same domain on different branches) and MEDD (multiple entities in different domains). For examples of the groups, consult Fig. 2 where `LIB.STIR.AC.UK` and `LIBRARIAN@LIB.STIR.AC.UK` are MESB. Further, `SRM@CS.STIR.AC.UK` and `KJT@CS.STIR.AC.UK` are MEDB, and `SRM@CS.STIR.AC.UK` and `A@SOMEWHERE.ELSE` are MEDD. SE should be self explanatory, but might not be trivial if we allow for aliases.

In general we can assume that the more entities are involved and the greater their independence, the less knowledge will be available of policies that concern

them. Note that we assume here the most general case, one might consider architectures with a centralised policy manager that is aware of all rules and hence has all available knowledge. However, it is unlikely that such an entity will exist, partly for scalability reasons and partly for privacy reasons (entities might not be willing to share details of their policies).

4.4 Roles

When discussing domain entities, we said that we do not exclude an overlap of domains. In contrast, we would encourage this as typical users belong to many hierarchies having distinct roles in each. Considering a typical person, say John: he could be member of a sports club, have a role in his family, and have a role at work. He might also have a number of roles that are imposed by the relationship of his other roles: he will be the subordinate of some colleague, or he might be a business competitor of one of his sports club friends because they are in competing companies.

We can identify relations between entities and their roles that might lead to conflict: SESR (a single entity in a single role), SEMR (a single entity having multiple roles), MESR (multiple entities in the same role) and MEMR (multiple entities each in multiple roles). Examples are given by "Jill is the departmental secretary" (SESR), "Jill is the departmental secretary and also mother to her children" (SEMR). Any member of a support team has the same role: "support staff" (MESR). Finally John and Bill are golfing partners, but John is also director of his business and Bill is CEO of another business (MEMR).

4.5 Policy Relation

Considering two policies, we might be able to identify commonalities between them. For example one might be a more specific version of the other. That is policies might be derived from each other by specialising certain elements: for example a domain entity might be replaced with an entity from its sub-domain, or a time constraint might be replaced with a tighter or looser time constraint. However, policies could be completely independent of each other.

In general it is unlikely that the question whether policies are related or independent is decidable. However, within a specific application domain this might be decidable. Also, one could consider libraries of policy templates from which policies are derived – much as one would have inheritance in object oriented programming. As an example, Ponder provides a concept of inheritance.

4.6 Modalities

Modalities are very common in policies; examples are *obligation, permission and interdiction* in deontic logic based frameworks [4,3], *authorisation, obligation and delegation* in access control. Other deontic logic based policy frameworks have been proposed , for example in [24]. We have also considered temporal modalities (*now, every Christmas, weekends*) and preferences (*want, prefer, must*) in [31].

Again, we need to consider dependence on the application domain. Temporal modalities are concepts that are independent of the application domain as they refer to commonly understood concepts. Preferences might be somewhat more domain-dependent, especially if we consider the meaning of certain terms in subject-specific speech (e.g. in legal speech "shall" or "can" have a different meaning from their colloquial use). OPI and authorisation modalities are often quite dependent on the domain: violation of these modalities might be acceptable in some domains (albeit generally under penalties) but absolutely not in others. Consider a company policy that *staff is only permitted to hire small cars*, however a member of staff might decide that he urgently needs to attend a customer but due to non-availbility is unable to obtain a small hire car. In this case it is probably desirable that the policy is violated. On the other hand *only the holder is permitted to draw money from the bank account* is a rule that should not be violated under any circumstances.

5 Conflict Manifestation and Examples

Conflicts can manifest themselves within each of the dimensions discussed before, but a single conflict might involve more than one dimension. Depending on the exact manifestation we can recommend different categories of approaches to detect and resolve the conflicts – this is where the work conducted by the feature interaction community over the last decade will be valuable, but also where new work is required.

We will now consider how conflicts manifest themselves within each of the dimensions in turn.

5.1 Policy Types

Conflicts in this dimension are most likely to be dependent on the controlled system, as many conflicts here arise from conflicting actions. However, it de-

pends on the types whether the potential for conflict can be determined independently of the application domain.

Before considering policy types in more detail, it is worthwhile discussing actions. An action is the effect of a policy. In call control, a typical action could be the forwarding of a call; in access control, the denial of access or the request for a password. Actions can conflict, as is well known from feature interaction: for example in POTS simultaneous forwarding to two distinct telephones is not possible. However, this is tightly bound to the underlying system, as even in the telephony application domain the previous action might be meaningful in an Voice over IP context. The underlying system prescribes exactly which actions conflict. We will not pursue this further here due to its system dependence.

Two (or more) unconditional goals conflict if the actions they specify are inconsistent: this can only be determined if information about consistency of actions for the given application domain is available. Conditional goals are similar, in that they conflict if their actions conflict, however the conditions must also be overlapping – as illustrated by example 5.1. Thomas [34] has discussed methods to detect such cases in the context of feature interaction.

Example 5.1 *John is an administrator. The domain prescribes that allowing and denying access are incompatible. Then, the following two policies conflict:*

- *If the user is an administrator then allow access.*
- *If the user is John then deny access.*

ECA rules conflict if (1) their actions conflict and (2) they may be triggered at the same time and (3) their conditions overlap. Being triggered at the same time can have two interpretations: their trigger sets overlap (and the actual trigger is in the overlap) or the action of one is in the trigger set of the other. The former case has been called STI (Shared Trigger Interaction) and the latter SAI (Sequential Action Interaction). These cases, together with a hybrid method to handle them, have been discussed in [8]. Example 5.2 and example 5.3 show cases for STI and SAI respectively. The same holds for ECA rules with a trivial condition of true. Many interactions in this class can be detected by syntactic analysis since much information beyond the simple action is available.

Example 5.2 *Blocking and Forwarding are incompatible actions in the telecommunications domain. The following two policies lead to an STI as they are triggered by the same event but request probably incompatible actions:*

- *Incoming calls should be forwarded to John.*
- *Incoming calls should be blocked if they are not emergency calls.*

Example 5.3 *Blocking and Forwarding are incompatible actions in the telecommunications domain. The following two policies lead to an SAI as the forwarding requested by the first policy causes the second to be applicable:*

- *Incoming calls should be forwarded to John.*
- *Forwarded calls should be blocked if they are for John.*

We can also have interactions between goals and ECA rules. To detect and resolve these, we use methods that combine the issues discussed above for ECA and goals.

5.2 Domain Entities.

While the conflicts of policy types might be relatively simple, and somewhat similar to traditional feature interactions, domain entities introduce interesting new types of conflict and affect how conflicts in policy types can be detected and resolved.

All policies aid the users of a system to perform certain tasks, respecting their own wishes and the rules imposed on them by organisations. Users can serve as individuals or in certain roles, whereby a user can clearly assume more than one role at a time. Potential conflict exists between the policies applicable to a user, based on the diverse roles that the user assumes as well as on the roles of different users. We consider roles and users to be domain entities, embedded into a naturally occurring domain hierarchy of say an enterprise (see Fig. 2). Each entity in the hierarchy can have associated policies which might be applicable only to that entity, to all entities below it in the hierarchy, or to other sections of the hierarchy.

We can consider four fundamentally different cases, each of which has a significant impact on the the knowledge available for conflict detection and resolution.

SE (Single Entity): Multiple policies which are defined by a single entity might conflict. Such an example would be a user who already has a policy that *calls in the evening should be forwarded to voicemail* adds a new policy stating *calls after 16:00 should be forwarded to the home phone*. In the SE case it can be assumed that all the details of the policies are known, and that the entity has the possibility of changing all policies to achieve consistency. This case would clearly suggest the use of off-line detection methods and redesign as resolution strategy.

MESB (Multiple Entities same domain Same Branch): Here the conflicting policies are defined by entities that lie on the same branch in the domain hierarchy. A typical example would be a policy of an individual

and a policy imposed by the individual's role conflicting. Conflict detection methods can again assume full knowledge of the policy details and hence off-line methods are again suitable. With respect to the availability of details we should note that these might not be available to the individual user but are available to the system. For example, a company might have a policy to *disallow the use of company cars for non-management staff* with the "hidden" agenda of saving cost, the latter is stored in the policy but not told to the employees. Assuming that the entity farther from the root of the hierarchy is defining a conflicting policy, the resolution should be achieved by enforcing redesign of the policy. However if the entity closer to the root is defining a new policy that conflicts with policies of entities farther from the root, it is difficult to decide what a suitable resolution would be. In general, it appears that the resolution should be to disable all conflicting policies and enforce the new policy. An example would be a new enterprise policy on budget permissions *"only the CEO can delegate the right to sign purchase orders"*, which should invalidate all existing delegations given by (say) current budget holders.

MEDB (Multiple Entities same domain Different Branches): We are moving further away from the assumption that policy details are immediately known when policies are defined by entities that are only related at a higher level. While in the previous two cases a certain dependence between the entities could be assumed, and policy details could be assumed known, we no longer have this certainty when we consider entities on different branches. However, one could argue that we are still within the same domain, so access to policies might still exist. Thus, conflicts in this class can be detected and resolved by the same methods suggested for MESB or they might require the methods we suggest for MEDD.

MEDD (Multiple Entities Different Domains): The conflicting policies are defined by entities that are completely independent. A possible conflict in their policies will in general not matter, as they might never encounter each other. However, in the case that the two entities engage in contact which leads to the policies being activated, it is difficult to detect and resolve a conflict. This scenario does not generally occur in policies used for access control or system management, but is very likely in policies used for call control. The only possible solution is to detect conflicts at run-time when the actual contact is established. Resolution can only be achieved by automated run-time methods, as redesign is not an option here.

5.3 Roles.

Policies might be associated with individuals, but as they prescribe how these individuals act in a social or commercial context they are also related to the role of the individuals. Roles have been studied in social science, and we will

not go into any detailed discussion here. Role conflict is a well known area of study in social science.

[21] considers the importance of role when considering policies for call control. This work brought together social scientists with telecommunications engineers – showing that research into this area is not purely technical. We will here only consider a more technical stance: There are four possible role/domain entity relations with possible conflicts as follows:

SESR (Single Entity Single Role): A number of policies might be applicable to the same user in the same role. In this case no conflict should be caused by the role dimension. All conflicts will arise in the other dimensions.

SEMR (Single Entity Multiple Roles): However, a single user might be subject to policies of different roles. For example, a lecturer in a university typically has a teaching role, but also a research role and an administrative role. Policies might be in place to disallow teachers and researchers from seeing staff files of colleagues. On the other hand particular administration roles might have duties that can only be accomplished by seeing these files. A similar conflict occurs when we allow researchers on a certain project to speak to collaborators in another country but bar overseas calls for all research staff. A researcher on the project will be in both roles and hence subject to both policies. It cannot be assumed that all roles the entity might play are known beforehand; roles might dynamically change, and roles might stem from distinct domains and thus might not be immediately known. Clearly conflict detection and resolution require to be dynamic, however static techniques might be used to some extend.

MESR (Multiple Entities Single Role): Multiple entities might fill the same role. This is typical with administration or customer service staff. Any change to role policies will impact many domain entities and hence will not be simple; worse, the impact might not be known beforehand. Consider the second example given in SEMR. If the rule that research staff cannot initiate overseas calls is introduced as part of a money-saving scheme, it will certainly surprise researchers who were so far able to make such calls – and it might actually hinder project progress. Conflict detection and resolution might need to be dynamic, but depending on the domain one could assume certain precedences: for example, a role policy might always overrule a user policy. However, we then can find problems such as those discussed in the policy relation domain.

MEMR (Multiple Entities Multiple Roles): This category is clearly the most complex. Consider an example with John and Bill as golfing partners. John is product developer in company A, Bill is technical director in a competing company B. Company A has a rule that developers are not allowed to contact technical staff in competing companies. If John wants to make some arrangements with Bill for an upcoming golfing match, he might not be able to do so due to the company policy imposed on his and

Bill's respective company roles. Detection here can only be dynamic, and resolution needs to be handled dynamically too. However policies suggested by us in [31] and supported by the architecture below can take into account context, and this might be able to help resolve the problem. For John and Bill, if a clear indication is given that the contact is not work related it might be acceptable for it to take place.

5.4 Policy Relation.

Another challenging type of conflict is based on the relation between two policies. One policy might define a certain behaviour under certain circumstances. We can now define a second policy which adapts the behaviour or circumstances of the former.

Sometimes it might be highly desirable to allow refinement or even relaxation of policies, in other cases it might be highly undesirable. In general it is difficult to see a general solution to this type of conflict, so we see further work being required here. Let us conclude with another example to strengthen the point:

Example 5.4 *Consider four policies:*

(1) Managers can approve orders for their department
(2) Managers can approve urgent orders for any department
(3) Junior managers can approve orders under £500
(4) Senior managers can approve orders under £5000

Here (3) and (4) specialise (1), but they are actually vague about the department. (2) contradicts (1), but is often desirable. However, (2) also contradicts (3) and (4), and might be less desirable here – especially in the case of junior managers.

5.5 Modalities.

We can consider a number of modalities, such as obligation, permission and interdiction, or authorisation and obligation. On a different level, terms such as 'never' and 'always' indicate modalities. Preferences, e.g. wish or must, are highly relevant for call control policies. They can also be seen as modalities, albeit rather fuzzy ones. And finally a class of temporal modalities, containing items like 'in the future', 'periodically' or 'now', is also relevant.

There can be interactions within each of these modality groups, e.g. one might place an obligation to perform an action without the required permission ex-

isting (this has been considered in Lupu and Sloman [25]).

Lupu and Sloman have identified three types of conflict: O+ and O- (the obligation to perform and not to perform an action), A+ and A- (the authorisation and denial to perform an action) and O+ and A- (the obligation to perform an action that we are not authorised to perform). Similar conflicts have been described in papers on deontic logic [38].

Temporal modalities can lead to conflicts where inconsistent time intervals are specified such that the policy essentially never holds. Conflicts arising from temporal modalities should be easy to detect and resolve, probably due to the good common understanding of time-related concepts.

While preferences can lead to conflicts in themselves, it is more likely that they prove useful as a basis for conflict resolution in negotiation approaches where they can determine which policy should be given precedence by identifying how strongly users feel about their policy. Consider a user who at 8:00pm *urgently must talk to* a user who *prefers not to be called in evenings*. The system can use the preferences of "must" and "prefer" to resolve this conflict, in this case in favour of the caller.

6 An Architecture for Handling Policy Conflict

We have proposed a three-layer architecture for policies for call control in [32]; this has been implemented as part of the ACCENT [1] project. The same architecture is viable for the more general setting of policy-controlled systems. In particular, we have a policy definition layer, a policy server layer and the controlled system. We are not too concerned with the first and last here, but will briefly discuss their roles before concentrating on the policy server layer.

Policies are activated by events in the controlled system and the environment, and act on the controlled system (potentially changing the environment). It is here that conflicting policies might break the system or at least lead to annoyance of the users. Hence actions committed by the controlled system should be conflict-free. It is one purpose of the policy server layer to ensure this.

Policies need to be defined by users or system administrators and then be deployed in the system. The policy definition layer encapsulates user interfaces at different levels of complexity for different user groups. In this layer users formulate policies and submit them to the policy server layer. If users attempt to define inconsistent policies, the policy server layer will inform them about

[1] www.cs.stir.ac.uk/compass

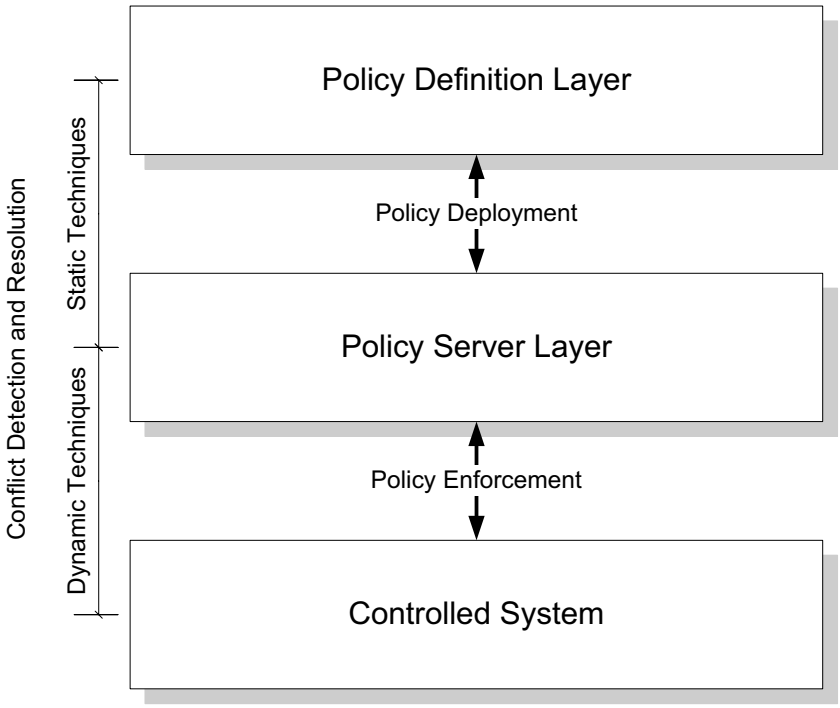

Fig. 3. Three-Layered Architecture

conflicts and enforce a redesign if desirable (i.e. by rejecting the submitted policy).

The foregoing should explain the central role of the policy server layer, but also give an insight into its complexity. The policy server layer consists of two main entities: policy stores and policy servers. Policy servers provide the intelligence, as well as the points of contact into this layer, while policy stores simply form repositories of policies. Policy stores should provide fast access to the stored policies. Tuple spaces [20] have been adopted for desirable technical reasons, but their details are not relevant for this paper. Note, that we assume a distributed architecture, where more than one policy server might use the same policy store. In addition each server might have several stores that it uses.

More interestingly the policy server provides two interfaces, one for policy deployment and one for policy enforcement. We will now consider each in turn before very briefly considering what feature interaction can contribute.

6.1 Policy Deployment

A (currently proprietary) protocol allows us to deploy policies by sending requests to a policy server on its deployment interface. The protocol provides

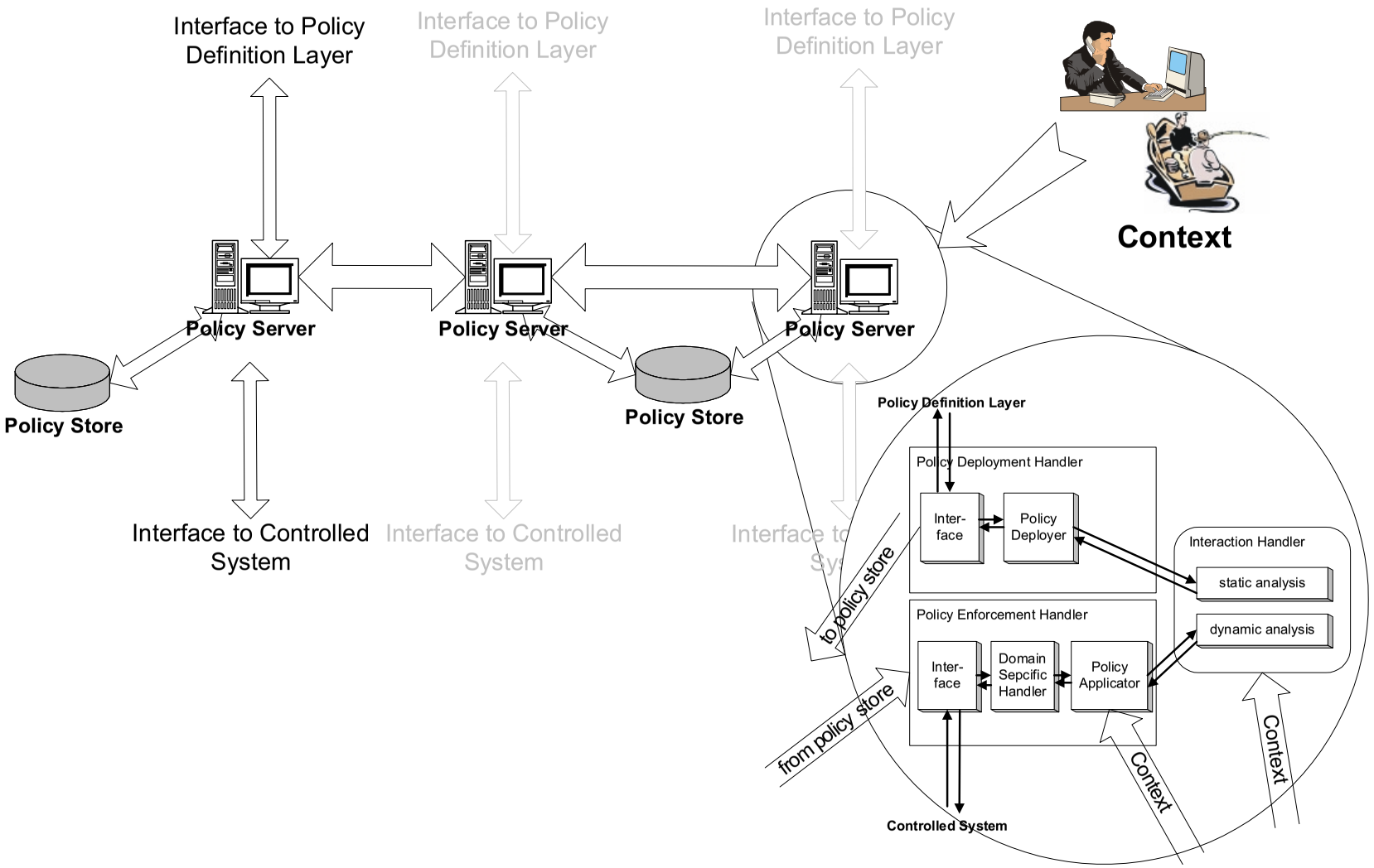

Fig. 4. The Policy Server Layer close up

six functions: upload, update, delete, enable, disable and query. These deploy a new policy, update or delete an existing policy, enable or disable policies (a disabled policy remains in the policy store but will not be applied), and query the policies that a user currently has deployed.

Each request consists of the given function, a user or policy identifier, and if required a policy description. Policy descriptions in APPEL (the Accent Project Policy Environment/Language) are encoded as XML, and hence it will be practical to move from the proprietary interface to a more general SOAP-based one – however, this will not impact on the functionality.

Received requests are passed to the policy deployer, which performs a simple algorithm which varies slightly depending on the function. The general operation is as discussed next. The policy store is queried for the existence of a policy with the provided id. This might raise an `ID` failure, which will lead to termination of the operation. If this succeeds, a static interaction handler will be created and the policy will be checked for conflicts. If a conflict occurs, a `CONFLICT` error will be reported. Assuming that no conflict occurs, the operation proceeds. If no system problem occurs, success will be reported, otherwise `GENERAL_FAILURE` is issued to the user interface. Each error message is accompanied by further details, in particular the `CONFLICT` error will be supported by a set of conflicting policies as well as suggestions for possible solutions.

Note that the interaction handler might make use of context information (such as domain hierarchies as appropriate). However, if conflict detection is dependent on such (semi-permanent) context information, one could only warn of possible conflict or not detect a conflict at all.

Also note that policy deployment process is completely independent of the application domain and the underlying system. However, to detect conflicts caused by actions the interaction handler requires a notion of what constitutes such interactions. Clearly, the policies themselves will be somewhat dependent on the controlled system as the actions they specify must be meaningful in the application domain.

6.2 Policy Enforcement

The process of policy enforcement might be more interesting. In order for this to work, information from the system must be passed to the policy server. An interface to the controlled system exists for this purpose, and the information passed depends on the controlled system. In call control, this will be the requests and responses sent through the network; in access control this might be the requests to access objects.

To keep the architecture open and as general as possible, the information from the network is provided again in a proprietary protocol (which again can be substituted by SOAP). This means that on the controlled system, an extra interface component needs to be deployed which extracts and packages the information into the required format. In general the format contains an identifier for the controlled system (which will be used to identify the domain-specific handler), the raw information and the structured information in key/value pairs. As extremes, we might not provide any structured information, letting the domain specific handler extract this from the raw information, or we might provide no raw data having extracted all required information at the controlled system side. Example 6.1 shows how the information passed might look for a SIP communications system.

Example 6.1

```
SIP\n
SERVER_NAME: d254196.cs.stir.ac.uk\n
MSG\n
INVITE sip:cs.stir.ac.uk SIP/2.0\n
Via: SIP/2.0/UDP 139.153.254.196:5062\n
CSeq: 3732 REGISTER\n
To: "Stephan" <sip:srm@cs.stir.ac.uk>\n
Expires: 900\n
From: "Stephan" <sip:srm@cs.stir.ac.uk>\n
Call-ID: 1815056773@139.153.254.196\n
Content-Length: 0\n
User-Agent: KPhone/2.11\n
Event: registration\n
Allow-Events: presence\n
Contact:"root"<sip:root\@139.153.254.196:5062;transport=udp>;
methods="INVITE, MESSAGE, INFO, SUBSCRIBE, OPTIONS, BYE, CANCEL, NOTIFY, ACK"\n
\n
```

The interface then identifies the appropriate domain-specific handler and passes the received information on. The domain-specific handler continues processing of raw data as required, and most importantly converts all information into the terms used by the policy language. The latter allows us to have a generic policy applicator that is independent of the controlled system. Finally the information is passed to the policy applicator, which will identify the relevant policies (dependent on the current context and received information).

A dynamic interaction handler is intended to determine any possibly conflicting policies, and also to resolve any conflict by either disabling policies or reordering them. The details of the policy conflict detection and resolution are dependent on the particular run-time resolution method chosen. For examples

of such methods please see section 6.3. Some methods might require negotiation between policy servers, and the presented architecture enables such contact. In particular we foresee that the originating policy server might contact the remote end policy server to "pre-negotiate" which is helpful to avoid actions that might lead to conflict later on or "post-negotiation" where the remote end on detecting a conflict must resolve this. The latter might be easier to implement, while the former might be more powerful. Details require further investigation. Once all conflicts are resolved, the policy applicator returns an ordered list of actions to be performed by the domain-specific handler.

The domain-specific handler must process the list of received actions (which is in policy terms) and convert them into the messages required by the system before being passed back to the controlled system and the action being enforced.

Clearly, policy enforcement is dependent on the controlled system, however, the architecture as presented neatly separates the generic from the system-dependent functionality justifying our claim as to its genericity.

6.3 Conflict Handling: A Glance at Traditional FI

Both the policy deployment and enforcement processes make use of an interaction handler to detect and resolve any policy conflicts. This is an area where we can refer to past work in the feature interaction community. There we find two main streams: off-line methods which detect conflict using some formal model of the system or using pragmatic techniques (resolution is by redesign); on-line methods which detect conflict at runtime and must also resolve conflicts at runtime. For a detailed review of the available techniques see [9]. Off-line methods are often static techniques whereas on-line methods are rather dynamic, taking into account the current context and state of the system.

Both categories can be useful here, as we have a requirement for static techniques for policy deployment and dynamic techniques for policy enforcement. We have made recommendations for both static and dynamic techniques in [32] to be used for the detection and resolution of policy conflict, which we will briefly repeat here.

Off-line methods considered most appropriate for the policy context are Anise [35] (pre-defined or user-defined operators allow to build progressively more complex features while avoiding certain forms of interaction) and Zave and Jackson's [41] Pipe and Filter approach. Also, Dahl and Najm's [15] (occurrence of common gates in two processes) and Thomas' [34] (guarded choices) approaches, where the occurrence of non-determinism highlights the potential

for interactions, are suitable. 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. [23].

There are also several on-line approaches that are suitable. The feature manager in [26] detects interactions by recognising that different features are activated and wish to control the call. The resolution mechanism for this approach [30] is based on general rules describing desired and undesired behaviour. In negotiation approaches, features communicate with each other to achieve their respective goals [36]. Buhr et al. [7] use a blackboard technique for the negotiation, thus introducing a central entity.

While these methods have been devised in the telecommunications domain, we believe that they are not bound to this domain and will be applicable in the more general context discussed in this paper. We also believe that other techniques might be adaptable or suitable for this new domain, and leave this as question for further investigation.

7 Policies as an Emerging Domain: Revisited

Policies have entered the area of feature interaction as is apparent from the recent feature interaction literature. There are several ways in which policies might be relevant for feature interaction. They could provide new handles on resolving feature interactions in existing systems by providing a higher level of abstraction. They could also replace features in call control. However, policies are not free of problems, and in particular the policy conflict problem closely resembles the feature interaction problem.

In this paper we have provided a taxonomy that structures the problem space of policy conflict, considering policies as the objects to be manipulated and not their specific application domain. Certain conflicts can occur in policies, others might be application-dependent. Both classes require solutions for detection and resolution of policy conflict, but we feel that the former is more important as it is inherent to policies. As such we believe that policies form an interesting new domain for feature interaction research.

We have shown that policies can be placed at different positions in each of the dimensions, and each of the dimensions provides different challenges. In general, the further a policy is from the origin of the domain, the less information is available about the conflicting policies and the more dependent they are on the current situation. This leads to a clear requirement for what has been considered on-line techniques in the feature interaction domain. On the other

hand as on-line techniques will always be under time constraints, one could say that leaving the solutions for problems until the last minute is bad design. One should attempt to solve conflicts occurring at design time if possible.

The forgoing clearly shows that both main fields of feature interaction research, on-line and off-line methods, should be pursued. Existing techniques need to be adapted and tried in the new context of policies. This should be relatively easy in the off-line area where many techniques have been developed. However, it will be more challenging in the on-line area where the number of existing techniques is scarce and their past development has often been hampered by poor signalling and lack of information. These problems are to some extent eased in systems controlled by policies.

7.1 Conclusions

We have attempted to classify the dimensions of policy conflict and discussed examples and possible solution strategies for each of the dimensions. In particular we have identified five dimensions that are inherent to policies, and have also seen that certain conflicts are introduced by the application domain. The latter were not considered in detail in this paper. The identified domains are policy type, domain entities, roles, policy relation and modalities – each contributing to the power of policies but introducing problems of their own.

We have also introduced a general architecture for controlling systems by policies. The architecture has been implemented and tried in the domain of call control as part of the ACCENT project. The ACCENT project is currently investigating different conflict detection and resolution techniques. However, the architecture is applicable in general as it does not make any assumption about the controlled system – beyond the fact that the system can be observed and influenced, which is fundamental for control by policies anyhow. The architecture considers both policy deployment (providing answers to the question of how policies get to their execution point) and policy enforcement (the actual influencing of the controlled system). The architecture is distributed, but can also handle the (simpler) case of centralised control.

A major advance over any existing architectures is the clear identification of 'hooks' for conflict detection and resolution techniques, both at policy deployment and policy execution time – a problem that has been mostly ignored to date.

As policies are considered an emerging domain, the work on policy conflict detection and resolution is in its infancy. Where work exists (such as [25]), it is tightly bound to a particular problem in a specific application domain rather

than being general. There is much potential for feature interaction research to shape and support the future of policy conflict detection and resolution in the emerging domain of policy control. In particular by considering the problem on a general policy level, rather than in an application-specific fashion, the feature interaction community can make a significant contribution extending and building on the taxonomy and architecture presented here.

Acknowledgements

This work has been supported by EPSRC (Engineering and Physical Sciences Research Council) under grant GR/R31263 and Mitel Networks Corporation. We thank all people who contributed to the discussion of policies in the context of call control which formed the basis for the more general considerations presented. Particular thanks are due to our colleagues at Mitel Networks Corporation, the University of Ottawa and here at Stirling University.

References