Department of Computing Science and Mathematics
University of Stirling

Handling Conflict-Prone Policies in Multiple Domains

Kenneth J. Turner

Technical Report CSM-199

ISSN 1460-9673

July 2014

Department of Computing Science and Mathematics
University of Stirling

Handling Conflict-Prone Policies in Multiple Domains

Kenneth J. Turner

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

Telephone +44 1786 467 421, Facsimile +44 1786 464 551
Email kjt@cs.stir.ac.uk

Technical Report CSM-199

ISSN 1460-9673

July 2014

Abstract: Features as self-contained units of functionality have been widely used in telephony and in many software systems. Feature interaction is a long-recognised and insidious problem whereby features can interfere with each other. Policies (user-defined rules) have been used as a flexible approach in system management, and have been extended to networked applications such as home automation and sensor networks. Policy conflict is the analogue of feature interaction. A new approach is described for identifying conflict-prone policies in multiple domains. This relies on domain knowledge that defines the abstract effects of policy actions. Conflict filtering is performed statically, but supports conflict detection and resolution dynamically. The technique has been implemented in the RECAP tool (Rigorously Evaluated Conflicts Among Policies). With limited user guidance, this tool automatically detects conflicting actions and automatically generates resolutions for these. The approach is generic, but is illustrated with policies for call control and home automation. The technique has improved the scalability of conflict handling, and has considerably reduced the manual effort previously required to deal with conflicts. The technique has also been evaluated in live practice.

Keywords: Call Control, Conflict Detection, Feature Interaction, Home Automation, Policy

Acknowledgements: The author is grateful to Gavin Campbell for collaboration on the early version of the approach that is extended in this report.

Contents

List of Tables

List of Figures

1 Introduction

This section gives the context and motivation for policies to control networked systems and to manage conflicts among these policies. The approach is compared to related work, including an early version of the technique in the report.

1.1 Call Control

Telephony has a long history. In the last twenty years or so digital telephony has become commonplace. This has encouraged the development of Internet telephony using approaches such as H.323 [18], SIP (Session Initiation Protocol [31]) and Skype. The digital approach has also supported the use of call control features as self-contained units of functionality.

Although features are well established in telecommunications, they are relatively limited from the user's point of view. Telephony features are mostly defined by the network operator. Users have little choice except to select the features they wish and to define a few feature parameters.

A key characteristic of telephony is that there is a basic system in the form of POTS (Plain Old Telephone Service). This can be extended by independently designed modules of functionality. Feature-based development was first pioneered in telephony. Telephone subscribers can choose separate capabilities in the form of features that augment the basic call. For example a subscriber might choose to forward incoming calls when busy, to queue incoming calls, or to block calls to certain numbers.

Of course, this approach is not unique to telephony. Many systems (including software systems) share this characteristic. For example, software packages often support plugins that add new functionality to the basic application. It is therefore not surprising that researchers have found feature interactions in new kinds of systems.

1.2 Home Automation

Home automation has been a goal for many years. For the most part, current commercial offerings might be better termed home control in that they support control of aspects such as lighting, heating, security, audio and video. These solutions tend to lack flexible management and automation of home functions – a goal of the work in this report. Home automation relies on devices in the home being networked, typically with a central hub that offers local or remote control.

Consumers have become increasingly knowledgeable about computer-based capabilities thanks to widespread use of personal computing, mobile phones, media players and the like. The time is therefore ripe to offer more sophisticated ways of managing the home. This can deal with aspects such as comfort, security, entertainment, and energy usage in the home. Home automation also underpins solutions in applications such as telecare (remote social care) and telehealth (remote health care).

1.3 Feature Interaction

Features are modular additions of functionality to a base system. Feature-based development was first pioneered in telephony, though the similar notion of a plug-in is common in software systems. Telephone subscribers can choose separate capabilities in the form of features that augment the basic call. For example, a subscriber might choose to forward incoming calls when busy.

Systems that offer multiple, independently defined features are prone to interactions. This is a well-known situation whereby the behaviour of one feature may affect another. Feature interaction is a long-recognised and insidious problem in systems that employ features. The issue is that independently designed or deployed features can interfere with each other.

The ideas behind feature interaction were first identified in telephony. As an example, a call forwarding feature (that forwards them) can interfere with a call waiting feature (that queues them). Similarly, call forwarding might interfere with outgoing call blocking because calls might be forwarded to a number that should be blocked. Detecting interactions is often problematic due to large numbers of features (hundreds in a typical PBX).

In general, feature interactions can be subtle and hard to anticipate. Normally feature interaction is seen as something to be avoided, but some interactions can be benign or even desirable. Indeed, there can be subjectivity in judging whether an interaction should be considered to exist and whether it should be avoided.

1.4 Policies

Many policy-based approaches have been developed to manage system behaviour and to give users more control. This added flexibility has the advantage that users can tailor systems more closely to their needs. Policies are user-defined rules that control a system dynamically through actions to be performed in specified circumstances. A policy typically defines event(s) that trigger it, condition(s) for the policy to apply, and resulting action(s) that are executed.

Features and policies are analogous, though their techniques and languages are very different. Feature interaction corresponds to policy conflict, i.e. the situation where different policies interfere with each other. The aim of this report has been to apply an idea from the world of features (identifying interaction-prone features) to the world of policies (for handling policy conflicts).

Historically, policy-based systems have been designed for applications such as access control, quality of service, security and system management. In all these applications, policies are typically created and maintained by administrators. They also require specialised technical knowledge and perhaps programming ability.

However, the policy approach used in this report is unusual in being designed for ordinary users. In telephony, policies offer much more flexibility than features [42]. Since policies operate at the edge of the network, users are not restricted to what the network operator provides. The approach is flexible and has been used in multiple domains. For example, it has been used to manage telephony, home care and sensor networks in wind farms. As important application areas, the report uses call control and home automation to illustrate the use of policies.

1.5 Motivation

Features and policies are analogous, though their techniques and languages are very different. Feature interaction corresponds to policy conflict, i.e. the situation where different policies interfere with each other. The work in this report has adapted an idea from the world of features (identifying interaction-prone features) to the world of policies (for handling policy conflicts).

The report describes work to deal with policy conflicts using domain knowledge captured in an ontology. Collecting this knowledge is a manual step, but conflict handling is then mostly automated. The idea is to automatically identify policy conflicts offline (statically, when policies are defined). This automatically creates conflict resolution policies that are then executed online (dynamically, when policies are executed).

The author and a colleague made an early study of policy conflict filtering for call control [9]. Although this showed promising results, the approach was significantly limited in a number of ways:

- conflict handling was restricted to call control, and could not readily be applied to other domains
- conflict detection was rather basic, and lacked the flexibility to deal with conflicts at multiple levels
- commonalities were not identified among conflicts, so the number of reported conflict cases was unnecessarily high
- conflict resolutions were crudely generated, resulting in an unmanageably large number of resolutions.

The new work has addressed all of these limitations, is more general and is more flexible. The author originally developed a technique for run-time handling of telephony policy conflicts [1]. This was later supported by filtering of conflict-prone telephony policies [9]. The supporting system was similarly designed for telephony but then extended for home automation [35]. Compared to this previous work, the present report focuses on automatically identifying conflicts among policies and generating resolutions for them. The use of automated conflict management in home automation is also new.

1.6 Related Work

1.6.1 Telephony

Feature interaction in telephony has a lengthy history. It was first discussed by [3]. Compatibility was then investigated among different telecommunications services [6]. A benchmark was developed for feature interaction techniques, particularly for use with Intelligent Network services [5]. In reviewing the state-of-the-art, [4] concluded that feature interaction techniques would be challenged by new developments such as Next Generation Networks, IP Networks, and Internet telephony

For interaction handling, one categorisation depends on the stage at which interactions are handled: avoidance (designing a system to avoid interactions), detection (discovering when an interaction occurs) and resolution (deciding how to deal with a detected interaction). Most techniques can be classified as offline/static (design-time analysis) or online/dynamic (execution-time analysis). Since exhaustive analysis may be impracticable or feature interactions may be subjective, a few techniques look for interaction-*prone* feature combinations rather than trying to definitively discover interactions.

For interaction resolution, a different range of techniques is applied. For example, resolution rules may be defined to handle run-time interactions. Typically this gives preference to one feature, but may enter into negotiation or may ask a user to resolve an interaction.

Many techniques have been developed to automate offline interaction detection. These techniques have focused heavily on formal methods such as process algebras, automata and (temporal) logic. This requires a precise knowledge of systems and their features. However, this is often unrealistic as the information is usually proprietary and unavailable. Formal techniques also require mathematical sophistication, static rather than dynamic analysis, and computationally intensive verification which tend to make the approach harder to use.

However, another offline technique (interaction filtering) offers a much more pragmatic solution. The notion of identifying interaction-prone features was initially presented in [20]. A filtering process is followed by detailed checking and refinement of conflicts.

A few tools support an automated approach to filtering feature interactions. One example is a prototype designed to detect interactions in a call environment [19]. This filters interactions in the Intelligent Network using simple descriptions of each service. [28] presents a filtering technique based on Use Case Maps and applies it to telephony features. [47] uses preconditions and postconditions to identify inconsistencies in features for LESS (Language for End System Services [46]).

1.6.2 Home Automation

A number of commercial solutions offer platforms for home automation such as the following. Although a degree of programmability is offered by several of these, this can be limited and require detailed technical knowledge.

Control4 (*www.control4.com*)**:** This is a leading approach that offers a framework for third-party devices to interoperate. Companies can embed the 'Control4 operating system' into their own devices, resulting in tightly integrated solutions.

Cortexa (*www.cortexa.com*)**:** This offers sophisticated home control. It is integrated with some of the most popular home automation technologies such as HAI and Insteon (*www.insteon.net*). Control can be exercised from a touch-screen in the home, a web interface or an iPhone.

Girder (*www.promixis.com*)**:** This supports a variety of home devices. Programmability is achieved by mapping input events to output events in a flexible way. This, however, is designed for those with specialised technical knowledge.

HAI (*www.homeauto.com*)**:** This deals with home control, safety, entertainment and energy usage. However, it is intended more for system installers than for programming by end users.

HomeSeer (*www.homeseer.com)***:** This is widely used to control a variety of home devices. Its advantages include remote control from devices such as PCs and mobile phones.

Many research projects have worked on smart homes, with an emphasis on either home automation or telecare/telehealth. Examples from a wide range include the Gator Tech smart house [16], the Gloucester smart house [30], House_n [17], MATCH (Mobilising Advanced Technologies for Care at Home [36]), Safe at Home [44], SAPHE (Smart and Aware Pervasive Healthcare Environment, *http://ubimon.doc.ic.ac.uk/saphe*), and SAPHIRE [15].

There has been work such as the following on feature interaction in home automation.

[25, 26] model appliances within a home network. Before a method can be invoked on an appliance, its pre- and post-conditions must be met. The approach is somewhat restrictive and assumes that precise models can be created for all the domestic appliances. A later development of the approach [27] allow environment impacts to be discovered, and explicitly recognises what features are designed to achieve.

[21, 43] adopt a high-level approach that aims to detect conflicts among services. The effects of each device on 'environment' variables are defined, and used to detect whether invoking a device action can lead to interference. The approach deals only with interaction detection and does not resolve interactions. It also does not take into account the subjectivity of conflicts. However, it provided some inspiration for the work in the current report.

[34] employs a tool-based approach to feature interaction in home-based systems. The event calculus is used to describe features formally. Logical reasoning is then applied to the combination of features, detecting interactions as constraint violations using a satisfaction solver. Resolution is aided by making explicit the assumptions of individual features.

[12] applies the idea of Software Product Lines to dealing with feature interactions in the home. Feature diagrams are combined with the Event Calculus. Interactions among features are found through static analysis with logical reasoning. The approach is illustrated with features for climate control, security and power control.

1.6.3 Offline Interaction Detection

Many techniques have been developed to automate offline interaction detection. Offline techniques have focused heavily on formal methods such as process algebras, automata and (temporal) logic. Formal techniques require a precise knowledge of systems and their features. This is often unrealistic as the information is usually proprietary and unavailable. Formal techniques also require mathematical sophistication and very lengthy run-times for analysis, both of which tend to make the approach harder to use.

However, another offline technique (interaction filtering) offers a much more pragmatic solution. The notion of identifying interaction-prone features was initially presented in [20]. A filtering process is followed by detailed checking and refinement of conflicts.

A few tools support an automated approach to filtering feature interactions. One example is a prototype designed to detect interactions in a call environment [19]. This filters interactions among Intelligent Network services, using simple descriptions of each service. Interactions are then detected for groups of services used in particular call scenarios. [28] presents a filtering technique based on Use Case Maps and applies it to telephony features. [47] uses preconditions and postconditions to identify inconsistencies in features for LESS (Language for End System Services [46]).

1.6.4 Policy Approaches

Many policy-based systems have been developed, usually for system management. As one of many examples, Ponder [13] is a popular approach. However, policy languages tend to require specialised expertise and thus to be less suitable for end users [42].

Policies have found applications in telecommunications, e.g. for controlling access to or managing network resources. As an example, COPS (Common Open Policy Service [14]) supports a client-server model for policy control over quality of service.

The work in this report is designed not to require detailed technical or programming knowledge. This report uses the policy language APPEL (Adaptable and Programmable Policy Environment and Language [41]). APPEL is interesting for analysis because it has been applied to telephony and has also been analysed by a number of authors.

Policy conflict is the analogue of feature interaction. [23] reported the first work on classifying policy conflicts, applying this to the Ponder policy language. A combination of obligation and authorisation aspects is used in both static and dynamic analysis. Meta-policies embody the constraints on policies that can be concurrently defined and executed. Conflict analysis has also been investigated in other policy approaches, e.g. [10] for automatic generation of conflict-free policies in IP Security.

As in feature interaction, formal techniques have been popular for analysing policy conflicts. For example, [11] is based on logic programming, [2] makes use of event calculus, and [33] employs model checking. However, as for feature interaction the use of formal techniques for policy conflicts lacks practicality.

Run-time handling of APPEL conflicts in call control is discussed in [1]. However, this still needs offline techniques to identify conflicts. [24] uses temporal logic to formalise the semantics of APPEL. This leads to a formal basis for automated detection of conflicts. In other work on APPEL, [22] presents a method for discovering call control conflicts based on the pre/post-conditions of actions. This allows semantically-based inferences to be drawn about the compatibility of actions. [9] reports early work on analysing APPEL conflicts in call control, using filtering for conflict-prone policies. This work showed promise but had significant limitations as noted in section 1.5.

1.6.5 Handling Feature Interactions

For interaction handling, one categorisation depends on the stage at which interactions are handled: avoidance (designing a system to avoid interactions), detection (discovering when an interaction occurs) and resolution (deciding how to deal with a detected interaction). Most techniques can be classified as offline/static (design-time analysis) or online/dynamic (execution-time analysis). Since exhaustive analysis may be impracticable or feature interactions may be subjective, some techniques look for interaction-*prone* feature combinations rather than trying to definitively discover interactions.

For interaction resolution, a different range of techniques is applied. For example, resolution rules may be defined to handle run-time interactions. Typically this gives preference to one feature, but may enter into negotiation or may ask a user to decide.

Many techniques have been developed to automate offline interaction detection. Offline techniques have focused heavily on formal methods such as process algebras, automata and (temporal) logic. This requires a precise knowledge of systems and their features. However, this is often unrealistic as the information is usually proprietary and unavailable. Formal techniques also require mathematical sophistication, static rather than dynamic analysis, and computationally intensive verification which tend to make the approach harder to use.

However, another offline technique (interaction filtering) offers a much more pragmatic solution. The notion of identifying interaction-prone features was initially presented in [20]. A filtering process is followed by detailed checking and refinement of conflicts.

A few tools support an automated approach to filtering feature interactions. One example is a prototype designed to detect interactions in a call environment [19]. This filters interactions in the Intelligent Network using simple descriptions of each service. [28] presents a filtering technique based on Use Case Maps and applies it to telephony features. [47] uses preconditions and postconditions to identify inconsistencies in features for LESS (Language for End System Services [46]).

1.6.6 Handling Policy Conflicts

Policy conflict is the analogue of feature interaction. [23] reported the first work on classifying policy conflicts, applying this to the Ponder policy language. The combination of obligation and authorisation aspects is used in both static and dynamic analysis. Meta-policies embody the constraints on policies that can be concurrently defined and executed. Conflict analysis has also been investigated in other policy approaches, e.g. [10] for automatic generation of conflict-free policies in IP Security.

As in feature interaction, formal techniques are commonly used for analysing policy conflicts. For example, [11] is based on logic programming, [2] makes use of event calculus, and [33] employs model checking. However, as for feature interaction the use of formal techniques for policy conflicts lacks practicality.

[32] is similar to the work in this report as it uses ECA rules for ambient intelligence in a home context. This work is complementary to that reported here as it aims to infer what home automation rules should be. The approach observes how people interact with their environment and can thus infer what their preferences are, such as for lighting levels. However, policy conflict is not addressed in this work.

This report uses the policy language APPEL (Adaptable and Programmable Policy Environment and Language [41]). APPEL is interesting for analysis because it has been used in various applications and is readily extended for new ones. APPEL has also been analysed by a number of authors.

Run-time handling of APPEL conflicts in call control is discussed in [1]. However, this still needs offline techniques to identify conflicts. [24] uses temporal logic to formalise the semantics of APPEL. This leads to a formal basis for automated detection of conflicts. In other work on APPEL, [22] presents a method for discovering call control conflicts based on the pre/post-conditions of actions. This allows semantically-based inferences to be drawn about the compatibility of actions. [9] reports early work on analysing APPEL conflicts in call control, using filtering for conflict-prone policies. This work showed promise but had significant limitations as noted in section 1.5.

1.6.7 Comparison with Related Work

This report differs in important respects from related work:

- Policies rather than features are used. Policies support higher-level statements of user intentions, and facilitate the resolution of conflicts.
- The approach is extensible for many domains including applications outside telephony (which tends to be *the* area for research into feature interaction).
- The approach deals with conflict detection *and* resolution within a single framework, unlike many approaches that deal with detection only.
- A formal specification of the system is not required. This is usually infeasible because the system is too complex, proprietary or open-ended. Formal analysis also needs specialised expertise and substantial computation.
- An intentionally less formal approach is followed. This aims to be simpler to define and to require only domain-specific knowledge. Domain experts, rather than formalists, define the information needed for conflict filtering.
- Conflicts are identified and resolved in a general way, resulting in a compact set of cases and resolutions.

1.7 Report Outline

Section 2 introduces the ACCENT policy system and its APPEL policy language. The use of dynamic conflict resolution is discussed. Section 3 presents the technique and tool used to handle conflict-prone policies. Sections 4, 5 and 6 explain how conflicts are handled in core policies, call control policies and home care policies respectively. Section 7 summarises and evaluates the work.

2 The ACCENT Policy System and APPEL Policy Language

This section introduces the ACCENT policy system and its APPEL policy language, with particular reference to policy conflicts and their resolution.

2.1 System Architecture

The ACCENT system (Advanced Component Control Enhancing Network Technologies, *www.cs.stir.ac.uk/accent*) was originally intended to allow users to tailor (Internet) telephony services to their own preferences [42]. However, ACCENT was designed to be extensible for new domains. For example, new extensions have been developed and evaluated for management of home automation [39] and of sensor networks in wind farms [8].

When used in call control, the ACCENT system manages a softswitch that supports H.323 (e.g. Gnu Gatekeeper, *www.gnugk.org*) or SIP (e.g. a Mitel softswitch, *www.mitel.com*, or the SIP Express Router, *www.iptel.org/ser*). When used in home automation, the ACCENT system manages a platform that supports a wide variety of domestic sensors and actuators [37].

The policy server interacts with the managed system, receiving events from it and responding with actions dictated by policies. Events trigger policies whose conditions hold. These are optimised to achieve the user goals and are then submitted to the conflict manager. This detects and resolves conflicts among policy actions, resulting in an optimal and compatible set of actions that is executed by the managed system.

2.2 Policy Language

APPEL (Adaptable and Programmable Policy Environment and Language, *www.cs.stir.ac.uk/appel*) is a comprehensive and flexible language, designed to express policies in multiple domains [41]. Key factors in the design of APPEL include ease of extension and orientation towards ordinary users. APPEL comprises a core language and its specialisations for different application domains. The original specialisations were for call control and conflict resolution, but new specialisations have been developed for home automation and wind farms.

APPEL defines the overall structure of policy documents that can contain goals, regular policies, resolution policies, prototype policies and policy variables. A policy consists of one or more rules in ECA form (Event-Condition-Action). Each rule has a combination of triggers (optional), conditions (optional), and actions (mandatory). The core language constructs are extended through specialisation for new applications. Only a small subset of APPEL is discussed in this report; see [41] for the full language definition.

A policy is eligible for execution if its triggers occur and its conditions apply. Additional restrictions may be imposed, such as the period during which the policy applies or the profile to which the policy belongs. As multiple policies can be triggered, conflicts may arise among their actions. The policies of just one user may conflict, perhaps due to contradictory goals like saving energy but staying comfortable. More typically, conflicts arise due to policies defined at different levels (e.g. by a resident, a carer and a doctor).

Policies can be neutral or can have a preference from *must* (strongest) to *should* (middling) to *prefer* (weakest); negative forms of these are used to express prohibition. In the event of policy conflict, the preference is one way of resolving this.

Internally, policies are represented in XML. APPEL is therefore defined by XML schemas. To make the language extensible, these are defined hierarchically. The schema for the core language is extended by schemas for regular policies and resolution policies. Each of these is then specialised further according to the domain.

2.3 Regular Policies

Obviously an end user is not expected to write or to understand XML. A variety of wizards therefore support user-friendly definition and editing of APPEL. For readability, examples of APPEL in this report use near-natural language in the style of the web-based wizard.

As an example for call control, the following regular policy deals with busy and unanswered calls, and also with call logging. If a busy user is called the call is forwarded to voicemail. Otherwise no answer within 5 seconds is dealt with. If the caller is not Acme or Tom then the call is fowarded to the user at home, else it is forwarded to the user's mobile. In parallel with these rules a further rule checks the type of call: business calls during office hours are logged in one way, while other calls are recorded in another way.

```
policy Busy or no answer
```

```
    when there is a call and I am busy
    do forward the call to voicemail
  otherwise
      when no answer in 5 seconds
      if the caller is not Acme and
         the caller is not tom@uni.edu
      do forward the call to home
    else
      do forward the call to my mobile
in parallel
  when there is a call
  if the call type is business and
    the time is in office hours
  do log an office hours call
else
  do log a general call
```

In domains such as home automation a very limited selection of action names is used. This is because actions are differentiated by multiple parameters rather than having many types of actions. Home automation mostly uses device input triggers (*device_in*) and device output actions (*device_out*). These carry the following parameters: message type (kind of trigger or action), entity name (class of device), entity instance (particular device), message qualifier (probability or timing) and parameter values.

Examples of these are as follows. The first is a trigger that reports a lounge temperature reading of 25.1°C with confidence 0.9. The second is an action that dims the lounge light to 30% in 10 minutes.

```
  device_in(reading,temperature,lounge,0.9,25.1)
  device_out(dim,light,lounge,10:00,30)
```

Obviously an end user is not expected to write or to understand XML. A variety of wizards therefore support user-friendly definition and editing of APPEL. For readability, examples of APPEL in this report use near-natural language in the style of the web-based wizard.

As an illustration, the following regular policy meets a common home care requirement. If older people need to go to the toilet at night, there is a risk that they will fall in the darkness. When an occupancy sensor reports that the user has got out of bed, the toilet light is switched on. Otherwise, when the bed is occupied again then the toilet light is switched off.

```
policy Night light
preference must
  when the bed becomes unoccupied
  if the time is 11PM to 7AM
  do turn on the toilet light
otherwise
  when the bed becomes occupied
  do turn off the toilet light
```

2.4 Conflict Detection and Resolution

Conflicts result from clashes between pairs of policy actions. As an example of conflict, one policy might wish to dim a light while another wishes to switch it off entirely. These contradictory actions must be identified and resolved, e.g. by choosing the more strongly preferred action. Conflict detection applies only to eligible policies. If a policy is not eligible, perhaps because it is triggered but its conditions are not satisfied, then conflict detection is not needed for its actions.

The ACCENT system allows for both static and dynamic conflict detection. Static detection is performed when a policy is defined and uploaded to the policy system, while dynamic detection occurs at run-time. Techniques and tools for APPEL were originally developed in telephony for static analysis [9] and for dynamic analysis [1].

This report discusses a more flexible and extensible approach to static conflict detection through identifying conflict-prone policies. Basically, the idea is to look for policy actions that have overlapping effects. This leads to the definition of special resolution policies that state which actions interfere, and also determine how the interference should be resolved. It is these resolution policies that are used dynamically.

Human guidance is often required to determine how best to handle conflicts. Only certain 'technical' conflicts (e.g. add + remove) can be detected fully automatically, but even then the resolution of a conflict may require judgment.

The effect of resolution is to process a set of proposed policy actions, selecting those that are compatible with the conflict handling rules. The nature of conflict ensures commutativity (if A conflicts with B then B conflicts with A) and associativity (if A conflicts with B and C, then A and B conflict with C). As a result, it is sufficient to check policy actions in ordered pairs for conflicts. In feature interaction it is possible, though rare, to find three-way (or even *n*-way) interactions that require more than two features to be considered in combination.

2.5 Action Conflicts

A policy action can have one or more effects that are positive (*effect*+), negative (*effect*-) or neutral (*effect*). As a call control example, adding video to a call increases bandwidth while removing it decreases bandwidth. However forwarding a call just affects routing without a particular positive or negative effect. As a home automation example, opening a window increases ventilation while closing it decreases ventilation. However the effect on indoor temperature is uncertain: how opening a window affects the indoor temperature depends on the outdoor temperature. Effects can be of two different kinds:

Environment: In call control, an environment effect changes the context of a call. For example adding a new user to a call reduces that user's availability for calls, while removing video from a call increases privacy. In home automation, an environment effect includes the usual environmental factors such as noise level and temperature. However it also includes concepts like security and ventilation, as well as device-specific factors like audio volume or the position when playing media.

Resource: In call control, a resource effect deals with consumption factors such as call bandwidth and cost. In home automation, a resource effect deals with with consumption factors such as electricity, gas or water that may be subject to a limit.

One reason for this distinction is that conflict detection depends on the kind of effect. For example, there is no conflict if two actions decrease an environment or resource effect. For an environment effect, if both actions increase it then this is not a conflict. However for a resource effect, if both actions increase it then a conflict may arise if there is a resource constraint.

For call control, two actions that increase privacy are compatible. However, two actions that significantly increase bandwidth may be incompatible if there is a limit on how much is available. If high-quality audio is in use, for example, then adding video might not be feasible.

For home automation, two actions that significantly increase power consumption may be incompatible if there is a limit on how much power may be consumed. For example, in some areas there is a limit on how much power a house can consume at any one time. If the air conditioning is running, for example, then operating a washing machine might be deferred.

A further consideration is the point at which action conflicts should be detected. The policy server first checks whether the optimised set of actions proposed by policies are compatible with each other. Checking simultaneous actions for interference is normal practice in call control. However, some applications such as home automation introduce a new issue: an action may conflict with a previously executed action rather than a simultaneous one. In effect, an action may conflict with the current state of the system.

The policy server keeps a (bounded) list of previously executed actions. Following a check for conflict among simultaneously proposed actions, the policy server extracts the most recent types of actions for the subscribers in the current call. For call control this would include conferencing and media actions, while for home automation it would include device actions. These previous actions are checked against the proposed actions using the same conflict detection technique.

As a call control example, suppose that high-quality audio has previously been selected but video is now required. This will be detected as a bandwidth conflict, perhaps resolving this by suppressing video. Sometimes it is the previous action that should be altered, e.g. downgrading audio quality so that video can be added.

As a home automation example, suppose that the heating has previously been turned on but a window is now to be opened. This will be detected as a temperature conflict so opening the window should be suppressed. Conversely, suppose that a window has previously been opened but the heating is now to be turned on. This will be detected as a conflict, but this time the new action should be executed along with closing the window.

2.6 Resolution Policies

Resolution policies are specified as an extension of the core APPEL language, and therefore use the same syntax as regular policies. Resolution is performed at just one level, i.e. resolutions are not allowed to create conflicts.

Resolution policies *define* conflicts among actions. Conflict handling is therefore intentionally not built into the policy server: it is defined externally and can therefore made be domain-dependent. The domain-specific actions of regular policies are the triggers of resolution policies. Conditions can also be imposed on resolution policies. Resolution policies dictate either generic or specific actions.

A generic action chooses one of the conflicting policies. This might select the one with the strongest preference, the one from the highest domain (e.g. organisational rather than departmental), the most recently defined one, etc. The default resolution action selects one of the conflicting actions: the stronger policy, else the newer policy, else the firmer (more confident) policy. If this still does not yield a unique result, one action is chosen (with a warning in the log).

Instead of a generic selection, a resolution can define specific actions from the set of regular policy actions. For example if the conflicting actions are to add or prevent use of video, the resolution might be to terminate the call altogether.

When a proposed action is matched to a resolution trigger, the action parameters are assigned to variables named in the resolution. The preferences of the associated policies are also implicitly assigned to special variables. The parameter and preference resolution variables are used in conditions to decide if a conflict would occur and how to resolve it.

As a call control example, the following resolution policy applies to actions that affect media in a call. Suppose one party wishes to add a medium to the call (e.g. video) while the other party wishes to suppress the same medium. If the preferences of both policies are in keeping (i.e. similar), the action of the more recent policy is chosen.

```
resolution Medium conflict
  when medium 1 is to be added and
         medium 2 is to be removed
  if media are the same and
     policy preferences are in keeping
  do choose the more recent policy
```

As a home automation example, the following resolution policy applies to power actions. Suppose the message types (actions) are different, the entity names and instances are the same, and the message qualifiers are the same. This means that different power actions are to be applied to the same device at the same time. There is then a conflict so the more strongly preferred policy is chosen.

```
resolution Power conflict
  when there are device outputs
  if message types are dim, off or on and
     message types are different and
     entities are the same and
     message qualifiers are the same
  do choose the stronger policy
```

Conflict handling within ACCENT is described in [38]. The main limitation of this previous work is that resolution policies have to be defined manually; this is tedious and error-prone. The work reported in [9] was a step towards identifying conflicts automatically and defining resolution policies for these. However, the limitations noted in section 1.5 meant that this solution saw little use until the improvements in this report.

2.7 Resolution Policy Structure

In general, resolution policies compare action parameters and policy preferences. Two preferences are out of keeping with each other (opposite) if one is positive and the other is negative, otherwise they are in keeping (similar).

Resolution policies often have checks on four combinations of parameter equality and preference similarity: equal-similar, equal-opposite, unequal-similar and unequal-opposite. This classification is not part of the policy language, but is a convenient way of analysing how a resolution is written.

In call control, a common pattern is equal-opposite. Suppose two call control policies try to fork a call (i.e. reach alternative destinations). If the destination parameters are the same and the preferences are out of keeping (equal-opposite), the resolution must decide whether to fork or not. If the destinations are the same and the preferences are in keeping (equal-similar), this is handled implicitly since one of the equivalent actions will be chosen by default. If the destinations are different and the preferences are in keeping (unequal-similar), the actions do not conflict with each other. If the destinations are different and the preferences are out of keeping (unequal-opposite), the action with negative preference will not be executed.

In home automation, suppose two actions have the same key device parameters (message type, entity name, entity instance and message qualifier). This means that both actions are trying to affect the same device in the same way at the same time. If the key parameters are the same and the preferences are in keeping (equal-similar), the resolution must decide which action to follow. If the key parameters are the same but the preferences are out of keeping (equal-opposite), this is handled implicitly since actions with negative preferences are not executed. If the key parameters are different (unequal-similar, unequal-opposite), this does not need a resolution since the actions do not conflict.

Defining resolution policies needs considerable thought and domain expertise. It is easy to get resolutions wrong or to miss cases that need resolution. It is therefore desirable to have automated tool support for definition of resolutions.

3 Offline Conflict Filtering

This section discusses how conflicts are filtered based on the abstract effects of policies as recorded in domain ontologies. An overview is given of the technique and tool for conflict detection and resolution generation.

3.1 Ontologies for Conflict Analysis

An ontology is a set of terms describing an area of knowledge, together with the logical relationships among these [29]. OWL (Web Ontology Language [45]) is a standard language for defining ontologies.

Ontology support is included in ACCENT through the POPPET tool (Policy Ontology Parser Program Extensible Translation [7]). OWL ontologies for ACCENT are defined at three levels:

GenPol (generic policies)**:** core language elements and the basic elements of policies.

WizPol (wizard policies)**:** information for policy wizards that is not part of the core language.

Domain: domain-specific ontologies that add application knowledge.

Since the ontologies and language schemas are defined in a hierarchical and domain-specific way, APPEL and its applications are fully extensible.

All information to handle conflicts is defined in the domain ontology, thus ensuring that the approach is general and extensible. This is achieved by defining certain action properties in the ontology. Concretely, this requires an ontology editor such as Protégé (*protege.stanford.edu*). Defining properties is done once by a domain expert as it requires specialised knowledge, though the task is straightforward.

Actions can be defined to have the following properties. The first of these determines how conflicts are detected, the second and third affect how resolutions are generated.

hasActionEffect: This defines one or more effects for some action (e.g. adding video to a call affects bandwidth and privacy, while turning on air conditioning affects humidity, power consumption and temperature). Effects can be defined at different levels.

hasPartialParameters: This applies to actions whose parameters are only partially checked for equality. The initial parameters distinguish actions and the final parameter is just a value. There is conflict between such actions if all parameters are the same and the preferences are opposite. There is also conflict if the parameters are the same except for the final one and the preferences are similar.

As a core policy example, consider the language action *start_timer(Identifier,Period)* that has a timer identifier and a timer period. Then 'must start timer *t* for period *p*' conflicts with 'must not start timer *t* for period *p*'. Also, 'must start timer *t* for period *p1*' conflicts with 'must start timer *t* for period *p2*' (periods *p1* and *p2* being different).

As a home automation example, consider the device output action with parameters as described in section 2.3. Then 'must do *device_out(m,e,i,q,p)*' conflicts with 'must not do *device_out(m,e,i,q,p)*'. Also, 'must do *device_out(m,e,i,q,p1)*' conflicts with 'must do *device_out(m,e,i,q,p2)*' (device parameter values *p1* and *p2* being different).

hasSimilarActions: This applies if duplicate types of action are allowed with different parameters. For call control, simultaneous actions to add different media to the call are not in conflict. Similarly for home automation, simultaneous outputs to different devices are not in conflict.

Conflict detection and resolution depend on the application domain, making use of the above action properties. Properties can be defined 'vertically' at different levels: actions of a given class (e.g. to manipulate call characteristics, or all home device actions); those with a given parameter type (e.g. some medium, or all home power actions); and those with an actual parameter (e.g. video selection for a call, or a home light dimming action). Effects can also be defined 'horizontally' for different combinations of parameters (e.g. for some timer name and period, or device outputs for a power action and a light). Compared to the early approach in [9], this flexibility makes conflict detection easier to define and also applicable to multiple domains.

3.2 Automated Identification of Conflicts

Conflicts arise between policy actions with certain parameters and abstract effects. The basic idea of conflict filtering is to compare the effects of actions. If these overlap then a conflict is likely (though not certain) and vice versa.

In call control, consider adding a new medium to a call. If this is done for video, it changes the video aspect (obviously) but also affects privacy (less obviously, since video is intrusive).

In home automation, activating or deactivating air conditioning will affect temperature, humidity and power consumption. Turning a washing machine on or off will affect power consumption. Opening or closing a window will affect security and indoor temperature. For these examples there is an overlap of effects (power) in turning on air conditioning and a washing machine; if power needs to be limited, this is a conflict. Similarly, turning on the air conditioning and opening a window leads to a conflict in temperature.

The approach described in this report is heuristic and is not guaranteed to give a correct result; the trade-off is between simplicity in definition and accuracy in analysis. However, as will be seen from later sections the approach gives good results for limited effort. Domain expertise is needed to associate actions with effects. Human judgement may be required to decide whether an effect is significant. For example, adding a third party to a call may be considered to affect privacy. However, this effect might be regarded as negligible and so not be recorded. Similarly, turning on a light will increase power consumption, temperature and security. However, these are likely to be small effects that can be ignored.

Action parameters may be enumerated types, e.g. a call can have media with possible values audio, video and whiteboard, while a home light can support the actions on, off and dim. Actions with actual parameters allow a deeper exploration of conflicts. Where an action has enumerated parameters, conflicts between instances of the same action are likely only if the parameters are the same. For call control, adding audio could conflict with a second such action since duplicate audio channels would not be desirable. However, if the second action wished to add video then this would not be an obvious conflict. For home automation, turning a light off twice would be pointless. For this reason, certain actions with different actual parameters can be defined as not conflicting.

Conflict analysis proceeds in two phases. A tool uses ontology information to automatically suggest what pairs of actions do or do not conflict. The tool user then reviews this information and adjusts the tool suggestions if required. As will be seen, automated identification of conflicts requires little manual editing. The list of conflicts and non-conflicts is then stored for future use, perhaps when the domain actions are updated.

3.3 Automated Generation of Resolutions

The ontologies also support automatic generation of resolution policies. The generated resolutions can be simplified because the policy server ignores duplicate actions in the same output response (e.g. two simultaneous attempts to remove the same party from a call). The default handling described in section 2.7 also allows uninteresting resolution cases to be omitted.

The early approach of [9] generated resolution policies in a rather crude way. A different resolution policy was generated for each identified conflict, resulting in an unmanageably large set of resolutions. The resolutions were even technically incorrect in that they did not conform to the language definition, though they worked with the policy server. A more sophisticated approach has now been implemented.

The new approach first generalises the conflict cases by looking for commonalities. One technique looks for ‘vertical’ similarities, i.e. conflicts where all values of an enumerated parameter type for one action are present. For call control, the action to add a third party may lead to conflict for all variations in the method (conference, hold, monitor, release and wait). For home care, a device action may lead to conflict for all variations in power setting (dim, off, on). These conflict cases can be collapsed into one for any action to add a party.

A second technique looks for ‘horizontal’ similarities, i.e. conflicts where each pair of actions uses the same value for an enumerated parameter. For call control, a pair of actions that add a medium may lead to conflict for all variations in the medium (audio, video, whiteboard). For home automation, a pair of actions for an audio-visual device may lead to conflict for all variations in the device type (DVB receiver, DVD player, radio, TV and VCR). In some applications these techniques reduce the number of resolution cases by 80%.

Another simplification is to group resolutions that have the same conflicting effects. Although this does not reduce the number of conflict cases, it substantially reduces the number of resolution policies (though each now has a number of cases). This makes it much easier to maintain resolutions. In some applications this technique reduces the number of resolution policies by 76%.

Within one resolution case, many variations are possible. Heuristics are used in order to generate resolutions that are close to human-created ones. There is also a dependency on domain knowledge embedded into the ontology.

As a call control example, the following resolution is part of what is automatically generated for audio conflicts:

- The first case applies when there are two requests to play an audio clip. The default resolution is applied if the clips are the same but the preferences are opposite (since the same clip cannot be both played and not played). The same applies if the clips are different but the preferences are similar (since different clips cannot be played simultaneously). This rule combines the equal-opposite and unequal-similar patterns.
- The second case applies if an audio clip is to be played but the audio medium is to be removed from the call (since the clip cannot be played if audio is removed). The default resolution is applied if the preferences are similar. This rule has an equal-similar pattern.

```
resolution Audio conflict
  when clip 1 is to be played and
         clip 2 is to be played
  if (clips are the same and
       preferences are out of keeping) or
     (clips are different and
       preferences are in keeping)
  do apply the default resolution
```

```
otherwise
  when clip is to be played and
        a medium is to be removed
  if the medium is audio and
    preferences are in keeping
  do apply the default resolution
```

As a home automation example, the following automatically generated resolution deals with security conflicts:

- The first case is for a close or open action on a blind, curtain, etc. coupled with an off or on action for a burglar alarm. These could conflict as they both affect security. If the preferences are in keeping then the default resolution is applied.
- The second case is for off or on actions with a burglar alarm. Suppose these actions are different but other parameters are the same. There is then conflict due to the opposing actions so the default resolution is applied.

```
resolution Security conflict
  when there are device outputs
  if message type 1 is close or open and
    entity 1 is blind, curtain, ... and
    message type 2 is off or on and
    entity 2 is burglar alarm and
    message qualifiers are the same and
    preferences are in keeping
  do apply the default resolution
otherwise
  when there are device outputs
  if message types are off or on and
    message types are different and
    entities are burglar alarm and
    message qualifiers are the same
  do apply the default resolution
```

Resolution generation proceeds in two stages. A tool automatically generates resolutions and uploads these to the policy server. Generated resolutions are uploaded in a disabled state, allowing the user to review and adjust them before they are deployed. The default resolution is generated automatically. This is usually appropriate, but a different resolution can be supplied by the user. As will be seen, automatically generated resolutions are mostly complete and need only limited modification. These can then be enabled for dynamic use by the policy server.

Besides generating resolution policies for upload, the tool also generates them in template form. This allows them to be incorporated in the policy wizard library.

3.4 Conflict Filtering Tool

The RECAP tool (Rigorously Evaluated Conflicts Among Policies) automates conflict detection and resolution generation as described in sections 3.2 and 3.3. The tool can be run as a stand-alone application. More conveniently it can also be run as part of ACCENT.

Using action effects, RECAP automatically constructs a table of ordered action pairs and highlights those that potentially conflict. A conflict can be unflagged if the domain expert decides that it is not real, and vice versa. Table columns can be sorted into ascending or descending order by clicking on their headers; this is convenient for checking groups of conflicts.

Figure 1 (at the end) illustrates what the tool looks like when analysing the call control domain. Conflict data can be loaded from an ontology or from a previous analysis. The table can show all possible cases or only conflicts. Conflict data can be saved to a file and uploaded to the policy server. The first highlighted line in figure 1 suggests that **add_caller**(conference) and **note_availability**(Topic) may conflict, the reason for this (a shared effect on availability), and when this conflict was analysed (automatically or manually). This conflict arises because adding someone to a call removes their availability, so it does not make sense to note their availability at the same time.

Action	Use	Parameters	Effects
log_event†	log message	Message	file
restart_timer†	restart running timer	Identifier	timer
send_message†	send user message	Recipient, Message	link
set_variable†§	set variable	Identifier, Expression	variable
start_timer†§	start new timer	Identifier, Period	timer
stop_timer†	stop running timer	Identifier	timer
unset_variable†	remove existing variable	Identifier	variable

Table 1: Core Action Effects († duplicates allowed with different parameters, § parameters partially matched)

Figure 2 (at the end) illustrates what the tool looks like when analysing the home automation domain. Conflict data can be loaded from an ontology or from a previous analysis. The table can show all possible cases or only conflicts. Conflict data can be saved to a file and uploaded to the policy server. The first highlighted line in figure 2 shows a pair of actions that respectively close or open a door or window. These actions conflict in security: closing improves it, but opening reduces it. They also conflict in ventilation: closing reduces it, but opening improves it.

The conflicts identified by RECAP may or may not be complete and correct. Subtle conflicts that are not automatically flagged may need to be indicated manually. As noted earlier, conflict handling usually requires human judgment and cannot be fully automated.

Resolution policies can be simple or complex, specific or generic, and depend on action parameters and policy details. Following human guidance, RECAP produces conflict resolution policies that are uploaded to the policy server (and also created in template form for the policy wizard). These policies define the conflicting triggers and parameter conditions, and also the resolution action. A policy wizard is then used to review and edit the resolutions.

4 Conflicts in Core Policies

This section illustrates how conflicts are handled in the core policy language.

4.1 Policy Actions

The core policy actions are domain-independent and performed internally by the policy server. As these actions are common to all domains, their conflicts are shared (by home automation here). Table 1 shows the core policy actions and their effects. Actions marked † may be duplicated with different parameters, while actions marked § have parameters checked only partially. (See *hasSimilarActions* and *hasPartialParameters* in section 3.1.) Parameter type names start with a capital letter.

4.2 Conflict Handling

The result of filtering core actions is shown in table 2. Of 28 distinct combinations of actions, 11 are automatically flagged as conflicts. The conflicts are numbered in the table with their corresponding effects shown below. As an example, action pair **start_timer** + **stop_timer** are in conflict. This exhibits a conflicting timer effect as indicated at their intersection (2).

From the 11 resolution cases, 4 resolution policies are automatically generated and uploaded to the policy server. Both conflict detection and conflict resolution aspects were checked by a domain expert for correctness and completeness. None of the automated generated resolutions had to be adjusted manually.

log_event (Message)	**restart_ timer** (Identifier)	**send_ message** (Recipient, Message)	**set_variable** (Identifier, Expression)	**start_ timer** (Identifier, Period)	**stop_ timer** (Identifier)	**unset_variable** (Identifier)	**Action1/Action2**
1							**log_event**(Message)
	2			2	2		**restart_ timer**(Identifier)
		3					**send_ message**(Recipient, Message)
			4			4	**set_variable**(Identifier, Expression)
				2	2		**start_ timer**(Identifier, Period)
					2		**stop_ timer**(Identifier)
						4	**unset_variable**(Identifier)

Conflicts: 1 file. 2 timer. 3 link. 4 variable.

Table 2: Automatically Identified Core Policy Conflicts

5 Conflicts in Call Control Policies

This section illustrates how conflicts are handled in call control policies.

5.1 Policy Actions

Table 3 shows the call control policy actions and their effects. Parameter type names start with a capital letter, while actual parameter values start with a small letter. Resource effects (only bandwidth in this example) start with a capital letter, while environment effects start with a small letter. Actions marked with † may be simultaneously executed as duplicates with different parameters but without causing conflict. (See the property *hasSimilarActions* in section 3.1.)

5.2 Conflict Handling

The result of filtering call control actions is shown in table 4. Of 120 distinct combinations of actions, 42 are automatically flagged as conflicts. The conflicts are numbered in the table with their corresponding effects shown below. Here, '&' is used for combined effects and '|' separates alternative effect combinations. As an example, the action pair **add_ caller** + **close** exhibits conflicting availability, party and privacy effects as indicated at their intersection (1). The action pair **add_ medium** + **close** has different conflicts at their intersection (5) depending on the actual medium.

From the 42 conflict cases, 11 resolution policies are automatically generated and uploaded to the policy server. Both conflict detection and conflict resolution aspects were checked by a domain expert for correctness and completeness. In the following case it was found necessary to refine the conflict detection. However, the conflict resolutions were found to be satisfactory.

The action pair **add_ medium**(video) + **add_ medium**(whiteboard) are reported as conflicting over bandwidth. These actions both add a significant amount of bandwidth and so might cause a limit to be exceeded. The generated resolution policy was therefore adjusted to detect a conflict if these actions would exceed the call bandwidth (as reported by the underlying communications service).

Action	Use	Parameters	Effects
add_caller	add caller to call with an H.323-like method	conference	availability-, party+, privacy-
		hold	
		monitor	
		release	
		wait	
add_medium†	add new medium	audio	audio+
		video	Bandwidth+, privacy-, video+
		whiteboard	Bandwidth+, whiteboard+
add_party†	add third party to call	Address	availability-, party+, privacy-
close	disconnect a call	–	audio-, availability+, Bandwidth-, call, party-, privacy+, route, video-, whiteboard-
confirm_bandwidth	confirm bandwidth	–	Bandwidth+
connect_to	connect to destination	Address	route
fork_to†	try parallel destinations	Address	route
forward_to	redirect call	Address	route
note_availability	note user availability	Topic	availability
note_presence	note user presence	Location	presence
play_clip	play audio clip	URI	audio
reject_bandwidth	reject bandwidth	-	Bandwidth
reject_call	reject call attempt	Reason	call
remove_medium†	remove medium	audio	audio-
		video	Bandwidth-, privacy+, video-
		whiteboard	Bandwidth-, whiteboard-
remove_party†	remove third party	Address	availability+, party-, privacy+

Table 3: Call Control Action Effects († duplicates allowed with different parameters)

6 Conflicts in Home Automation Policies

This section illustrates how conflicts are handled in home automation policies.

6.1 Policy Actions

Table 5 shows sample home automation policy actions and their effects. For illustrative purposes, this is only a subset of the full range that is supported. Environment effects start with a small letter, while resource effects start with a capital letter.

The device output action may be duplicated with different parameters, and is also subject to a partial parameter check (see section 2.6). For brevity only the message type (action) and entity name (device type) parameters are shown in the table; the entity instance, message qualifier and parameter values are omitted. Actual parameter values start with a small letter and are listed with '|' between alternatives. The actions from the core policy language discussed in section 4 also apply to this domain but are not repeated here.

6.2 Conflict Handling

The result of filtering home automation actions is shown in table 6. The table has been compressed by combining actions for same kind of entity; the dependency of conflicts on parameters is also not shown. Of 465 distinct combinations of actions, 39 are automatically flagged as conflicts. The conflicts are numbered in the table with their corresponding effects shown below. Here, '&' is used for combined effects. As an example, the action pair **device_out**(close, blind|...) + **device_out**(open, blind|...) exhibits light and security conflicts as indicated at their intersection (2).

add_caller (Method)	**add_medium**(Medium)	**add_party**(Address)	**close**	**confirm_bandwidth**	**connect_to**(Address)	**fork_to**(Address)	**forward_to**(Address)	**note_availability**(Topic)	**note_presence**(Location)	**play_clip**(URI)	**reject_bandwidth**	**reject_call**(Reason)	**remove_medium**(Medium)	**remove_party**(Address)	**Action1/Action2**
1			1					2					3	1	**add_caller**(Method)
	4		5	4									6	3	**add_medium**(Medium)
		1	1					2					3	1	**add_party**(Address)
			7	4	8	8	8	2		9	4	10			**close**
				4							4				**confirm_bandwidth**
					8	8	8								**connect_to**(Address)
						8	8								**fork_to**(Address)
							8								**forward_to**(Address)
								2						2	**note_availability**(Topic)
									11						**note_presence**(Location)
										9			9		**play_clip**(URI)
											4				**reject_bandwidth**
												10			**reject_call**(Reason)
															remove_medium(Medium)
														1	**remove_party**(Address)

Conflicts: 1 availability & party & privacy. 2 availability. 3 privacy. 4 Bandwidth.
5 audio | privacy & video | whiteboard.
6 audio | Bandwidth & privacy & video | Bandwidth & whiteboard.
7 call & route. 8 route. 9 audio. 10 call. 11 presence.

Table 4: Automatically Identified Call Control Action Conflicts

From the 39 conflict cases, 12 resolution policies are automatically generated and uploaded to the policy server. Both conflict detection and conflict resolution aspects were checked by a domain expert for correctness and completeness. In the cases below, it was found necessary to refine the conflict detection. However, the conflict resolutions were found to be satisfactory.

- The action pair **device_out**(close, blind|...) + **device_out**(dim|on, lamp|light) are reported as having a conflict over light. During the day it would not make sense to close the blinds (reducing illumination) and also to turn on a light (increasing illumination). However, at night this would be sensible. For a conflict to exist the light level must therefore be above some threshold. The generated resolution was therefore adjusted to detect a conflict only if the light level exceeds 20% of full daylight.

- Actions **device_out**(on, air_conditioning), **device_out**(on, central_heating|...) and **device_ out**(on, drier|...) are reported as having a pairwise conflict over power. These appliances consume a lot of power and so might cause a limit to be exceeded. The resolution was therefore adjusted to detect a conflict only if the total power exceeds 10kW.

7 Conclusion

7.1 Summary

It has been seen that feature interaction arose in telephony but is a problem in many application domains. Policy-based management is widely used, and exhibits the analogous issue of policy conflict. The report

Action	Use	Parameters		Effects
		Message Type	Entity Name	
device_out§	device output	channel_down| channel_set| channel_up	dvb|dvd|radio| tv|vcr	channel
		close	blind|curtain| shutter	light-, security+
		open		light+, security-
		close	door|window	security+, ventilation-
		open		security-, temperature, ventilation+
		dim	lamp|light	light
		off		light-
		on		light+
		off	air_conditioning	humidity+, Power-, temperature+
		on		humidity-, Power+, temperature-
		off	burglar_alarm	security-
		on		security+
		off	central_heating| heater	Power-, temperature-
		on		Power+, temperature+
		off	drier|washer	Power-
		on		Power+
		off	extractor|fan	ventilation-
		on		ventilation+
		fast_forward| forward|play|record	cd|dvd|vcr	position+
		fast_reverse|reverse		position-
		pause|stop		position
		track_next| track_previous| track_set		track
		volume_down| volume_mute| volume_up	cd|dvb|dvd|hifi| radio|tv|vcr	volume

Table 5: Home Automation Action Effects (§ parameters partially matched)

has covered work on adapting the idea of interaction-prone feature filtering to policy conflict in multiple domains.

The approach makes use of the ACCENT policy system and its APPEL policy language. The RECAP tool and the technique it embodies have been discussed for automated handling of conflict-prone policies. Action effects defined in ontologies allow conflicting action pairs to be discovered as potential conflicts. Resolution policies are automatically generated from this analysis. These are then used to detect and resolve conflicts at run time.

The approach has been validated in several different policy applications: core policies, call control policies, and home automation policies. It has been seen that a high level of automation is possible in handling conflicts, though some limited manual intervention is required. Since detecting and resolving policy conflicts is tedious and error-prone, the approach offers significant benefits.

7.2 Practical Evaluation

For call control, the approach has been evaluated in practice using a Mitel softswitch. The original, manually defined resolution policies used for telephony have been substantially extended by the new, automatically generated resolutions.

For home automation, the automatically generated resolution policies have been evaluated in practice.

device_out(channel_down|..., dvb|...)	**device_out**(close|open, blind|...)	**device_out**(close|open, door|...)	**device_out**(dim|off|on, lamp|...)	**device_out**(off|on, air_conditioning)	**device_out**(off|on, burglar_alarm)	**device_out**(off|on, central_heating|...)	**device_out**(off|on, drier|...)	**device_out**(off|on, extractor|...)	**device_out**(fast_forward|..., cd|...)	**device_out**(volume_down|..., cd|...)	**Action1/Action2**
1											**device_out**(channel_down|..., dvb|...)
	2	3	4		3						**device_out**(close|open, blind|...)
		5		6	3	6		7			**device_out**(close|open, door|...)
			4								**device_out**(dim|off|on, lamp|...)
				8		10	9				**device_out**(off|on, air_conditioning)
					3						**device_out**(off|on, burglar_alarm)
						10	9				**device_out**(off|on, central_heating|...)
							9				**device_out**(off|on, drier|...)
								7			**device_out**(off|on, extractor|...)
									11		**device_out**(fast_forward|..., cd|...)
										12	**device_out**(volume_down|..., cd|...)

Conflicts: 1 channel. 2 light & security. 3 security. 4 light. 5 security & temperature & ventilation. 6 temperature. 7 ventilation. 8 humidity & Power & temperature. 9 Power. 10 Power & temperature. 11 position & track. 12 volume.

Table 6: Automatically Identified Home Automation Action Conflicts

The home system uses ACCENT in an environment that links networked sensors, actuators and services to support care delivery within the home [39, 40]. Policies to manage home care are defined by users and their carers. Inevitably, policy conflicts can arise – particularly if policies are defined by different people (e.g. the resident and a carer).

Operational experience has been positive, though it raises the question of what a failure in conflict handling actually means. In general, the system might incorrectly identify policies as conflicting (a false positive) or might fail to spot a conflict (a false negative).

For a false positive, the system will forbid certain actions that might have been harmless. This is visible to the user through the system not reacting as expected (e.g. refusing to place a call, or not turning on the heating). Such a situation could have safety implications, for example if an alert message is suppressed due to incorrect identification of a conflict.

For a false negative, the system will allow certain actions that should not get through. This is visible to the user through the system performing unexpected actions (e.g. allowing high-quality audio as well as video and thus causing both to be degraded, or switching a light on then immediately turning it off again). However, the system is designed to be tolerant of such situations and will not fail (even if its behaviour might seem odd to the user).

An objective assessment has been performed through an analysis of the policy server log. These record the operation of the system, and in particular its handling of policy conflicts. The only small anomaly that was noticed was a very occasional situation where multiple resolutions were possible. This arises if the resolution policies are too broad such that more than one can be applied. In such a case, the system chooses just one resolution. This led to a couple of resolution cases being tightened up.

For home automation, a subjective assessment has been performed through gathering information from a trial home. No problems related to conflict handling were reported by the users (though there was a wider discussion of the approach to home automation).

7.3 Assessment

Although telephony has attracted a lot of interest from the feature interaction community, only a few studies have considered policy conflicts in this domain. The handling of policy conflicts in home automation is novel, though there has been work on feature interaction in home automation.

The approach has been assessed in three policy applications: core policies, call control and home automation. Across all three domains, of 613 distinct action combinations the automated analysis identifies 92 potential conflicts. Of these, 5 conflict detections were manually adjusted to take account of factors such as bandwidth limit.

This means that the heuristic approach correctly deals with 94% of conflicts. Given the difficulty of handling conflicts through purely manual analysis, this is a significant benefit. In fact, conflict handling can never be fully automated. However, the important point is that the heuristic approach gives good results with little effort. Once the domain expert has labelled actions with their effects, the analysis is fully automated and needs little manual intervention.

Overall, the approach has substantially extended the set of resolution policies that had been previously created by laborious manual means. This has improved the scalability of conflict handling, and has significantly reduced the effort and complexity of dealing with conflicts. Associating actions with their effects is simple compared to formal methods, but yields good results.

It is convenient that the conflict analysis results are stored for future usage. Although the main use is for filtering conflicts in the initial stages of specifying policies in a new domain, the results are useful for later revisions of the language to refine conflicts and resolutions.

Compared to the original conflict filtering solution in [9], all the limitations identified in section 1.5 have been overcome. The approach is extensible for multiple domains and is not just for use in call control (which many feature interaction techniques are limited to). Conflicts can now be handled at multiple levels: actions, parameter types, actual parameters, and multiple parameters. Commonalities are recognised among conflicts, reducing the number of detection cases by 80% in some applications. Resolution policies are now grouped by effect, reducing the number of resolution policies by 76% in some applications.

The report has focused on conflict handling for APPEL. Although the techniques are general, the tool implementation is restricted to working with APPEL and ACCENT. However, it would be feasible to use the same approach for other policy languages and systems that follow a similar ECA approach.

The approach has few competitors in practice, since alternative techniques tend to be hard to use (e.g. formal methods) or to be for a single domain (e.g. for telephony). The approach to conflict filtering is much easier than one that requires a formal model. Formal methods require a precise knowledge of systems and their features which is usually unavailable or requires computationally expensive analysis. The heuristic approach yields good results for limited effort and is easily extended for new applications.

References

[1] L. Blair and K. J. Turner. Handling policy conflicts in call control. In S. Reiff-Marganiec and M. D. Ryan, editors, *Proc. 8th Int. Conf. on Feature Interactions in Telecommunications and Software Systems*, pages 39–57. IOS Press, Amsterdam, Netherlands, June 2005.

[2] J. M. Blum. Handling emergent conflicts in adaptable rule-based sensor networks. Technical Report CSM-196, Computing Science and Mathematics, University of Stirling, UK, Oct. 2012.

[3] T. F. Bowen, F. S. Dworack, C. H. Chow, N. Griffeth, G. E. Herman, and Y.-J. Lin. The feature interaction problem in telecommunications systems. In *Proc. 7th Int. Conf. on Software Engineering for Telecommunication Switching Systems*, pages 59–62. Institution of Electrical and Electronic Engineers Press, New York, USA, July 1989.

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

[5] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen. A feature-interaction benchmark for IN and beyond. *IEEE Communications Magazine*, 31(8):18–23, Aug. 1993.

[6] E. J. Cameron and Y.-J. Lin. Real-time transition model for analyzing behavioral compatibility of telecommunications services. *ACM Software Engineering Notes*, 16(5):101–111, Dec. 1991.

[7] G. A. Campbell. Ontology for call control. Technical Report CSM-170, Computing Science and Mathematics, University of Stirling, UK, June 2006.

[8] G. A. Campbell and K. J. Turner. Goals and policies for sensor network management. In M. Benveniste, B. Braem, C. Dini, G. Fortino, R. Karnapke, J. L. Mauri, and M. S. H. Monsi, editors, *Proc. 2nd Int. Conf. on Sensor Technologies and Applications*, pages 354–359. Institution of Electrical and Electronic Engineers Press, New York, USA, Aug. 2008.

[9] G. A. Campbell and K. J. Turner. Policy conflict filtering for call control. In L. du Bousquet and J.-L. Richier, editors, *Proc. 9th Int. Conf. on Feature Interactions in Software and Communications Systems*, pages 83–98. IOS Press, Amsterdam, Netherlands, May 2008.

[10] C.-L. Chang, Y.-P. Chiu, and C.-L. Lei. Automatic generation of conflict-free IPSec policies. In F. Wang, editor, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE XVIII)*, number 3731 in Lecture Notes in Computer Science, pages 233–246. Springer, Berlin, Germany, Oct. 2005.

[11] J. Chomicki, J. Lobo, and S. Naqvi. A logical programming approach to conflict resolution in policy management. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. Principles of Knowledge Representation and Reasoning*, pages 121–132. Morgan Kaufmann, 2000.

[12] A. Classen. Problem-oriented modelling and verification of software product lines. Master's thesis, Institut dInformatique, Facultés Universitaires Notre-Dame de la Paix, Namur, France, May 2007.

[13] N. Damianou, N. Dulay, E. C. Lupu, and M. Sloman. Ponder: A language specifying security and management policies for distributed systems. Technical Report 2000/1, Imperial College, London, UK, 2000.

[14] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, editors. *The COPS (Common Open Policy Service) Protocol*. RFC 4261. The Internet Society, New York, USA, Jan. 2000.

[15] A. Hein, O. Nee, D. Willemsen, T. Scheffold, A. Dogac, and G. B. Laleci. SAPHIRE – Intelligent healthcare monitoring based on semantic interoperability platform – The homecare scenario. In A. Meier, editor, *Proc. 1st European Conference on eHealth*, pages 15–21. Gesellschaft für Informatik, Bonn, Germany, Oct. 2006.

[16] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah, and E. Jansen. Gator Tech smart house: A programmable pervasive space. *IEEE Computer*, 38(3):50–60, Mar. 2005.

[17] S. S. Intille. The goal: Smart people, not smart homes. In C. Nugent and J. C. Augusto, editors, *Proc. 4th Int. Conf. on Smart Homes and Health Telematics*, pages 3–6. IOS Press, Amsterdam, Netherlands, June 2006.

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

[19] D. O. Keck. A tool for the identification of interaction-prone call scenarios. In K. Kimbler and W. Bouma, editors, *Proc. 5th Int. Conf. on Feature Interactions in Telecommunications and Software Systems*, pages 276–290. IOS Press, Amsterdam, Netherlands, Sept. 1998.

[20] K. Kimbler. Addressing the interaction problem at the enterprise level. In P. Dini, R. Boutaba, and L. M. S. Logrippo, editors, *Proc. 4th Int. Workshop on Feature Interactions in Telecommunication Networks*, pages 13–22. IOS Press, Amsterdam, Netherlands, June 1997.

[21] M. Kolberg, E. H. Magill, and M. E. Wilson. Compatibility issues between services supporting networked appliances. *IEEE Communications Magazine*, 41:136–147, Nov. 2003.

[22] A. F. Layouni, L. Logrippo, and K. J. Turner. Conflict detection in call control using first-order logic model checking. In L. du Bousquet and J.-L. Richier, editors, *Proc. 9th Int. Conf. on Feature Interactions in Software and Communications Systems*, pages 66–82. IOS Press, Amsterdam, Netherlands, May 2008.

[23] E. C. Lupu and M. Sloman. Conflict analysis for management policies. In *Proc. 5th. International Symposium on Integrated Network Management*, pages 430–443. Chapman-Hall, London, UK, 1997.

[24] C. Montangero, S. Reiff-Marganiec, and L. Semini. Logic based detection of conflicts in APPEL policies. In A. Movaghar and J. Rutten, editors, *Proc. Int. Symposium on Fundamentals of Software Engineering*, volume 4767 of *Lecture Notes in Computer Science*, pages 257–271. Springer, Berlin, Germany, Oct. 2007.

[25] M. Nakamura, H. Igaki, and K. Matsumoto. Feature interactions in integrated services of networked home appliances. In S. Reiff-Marganiec and M. D. Ryan, editors, *Proc. 8th Int. Conf. on Feature Interactions in Telecommunications and Software Systems*, pages 236–251. IOS Press, Amsterdam, Netherlands, June 2005.

[26] M. Nakamura, H. Igaki, Y. Yoshimura, and K. Ikegami. Considering online feature interaction detection and resolution for integrated services in home network system. In M. Nakamura and S. Reiff-Marganiec, editors, *Proc. 10th Int. Conf. on Feature Interactions in Software and Communications Systems*, pages 191–206. IOS Press, Amsterdam, Netherlands, June 2009.

[27] M. Nakamura, K. Ikegami, and S. Matsumoto. Considering impacts and requirements for better understanding of environment interactions in home network services. *Computer Networks*, 57(12):2442–2453, Aug. 2013.

[28] M. Nakamura, T. Kikuno, J. Hassine, and L. M. S. Logrippo. Feature interaction filtering with Use Case Maps at requirements stage. In M. H. Calder and E. H. Magill, editors, *Proc. 6th Int. Conf. on Feature Interactions in Telecommunications and Software Systems*, pages 163–178. IOS Press, Amsterdam, Netherlands, May 2000.

[29] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory, Stanford, USA, Mar. 2001.

[30] R. Orpwood. The Gloucester smart house for people with dementia. In *Proc. Workshop on Technology for Aging, Disability and Independence*, Swindon, UK, June 2003. Engineering and Physical Sciences Research Council.

[31] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnson, J. Peterson, R. Sparks, M. Handley, and E. Schooler, editors. *SIP: Session Initiation Protocol*. RFC 2543 *bis* 09. The Internet Society, New York, USA, Feb. 2002.

[32] L. S. Shafti, P. A. Haya, M. García-Herranz, and E. Perez. Inferring ECA-based rules for ambient intelligence using evolutionary feature interaction. *Ambient Intelligence and Smart Environments*, 5(6):563–587, Dec. 2013.

[33] M. H. ter Breek, S. Gnesi, C. Montangero, and L. Semini. Detecting policy conflicts by model checking UML state machines. In M. Nakamura and S. Reiff-Marganiec, editors, *Proc. 10th Int. Conf. on Feature Interactions in Software and Communications Systems*, pages 59–74. IOS Press, Amsterdam, Netherlands, June 2009.

[34] T. T. Tun, R. Laney, Y. Yu, and B. Nuseibeh. Specifying software features for composition: A tool-supported approach. *Computer Networks*, 57(12):2545–2556, Aug. 2013.

[35] K. J. Turner. Flexible management of smart homes. *Ambient Intelligence and Smart Environments*, 3(2):83–110, May 2011.

[36] K. J. Turner, editor. *Advances in Home Care Technologies: Results of The* MATCH *Project*. IOS Press, Amsterdam, Netherlands, Oct. 2012.

[37] K. J. Turner. The ACCENT policy system for home care. Technical Report CSM-188, Computing Science and Mathematics, University of Stirling, UK, Apr. 2014.

[38] K. J. Turner and L. Blair. Policies and conflicts in call control. *Computer Networks*, 51(2):496–514, Feb. 2007.

[39] K. J. Turner, G. A. Campbell, and F. Wang. Goals and policies for home care. In K. J. Turner, editor, *Advances in Home Care Technologies: Results of The* MATCH *Project*, pages 3–49. IOS Press, Amsterdam, Netherlands, Oct. 2012.

[40] K. J. Turner and C. Maternaghan. Home care systems. In K. J. Turner, editor, *Advances in Home Care Technologies: Results of The* MATCH *Project*, pages 11–29. IOS Press, Amsterdam, Netherlands, Oct. 2012.

[41] K. J. Turner, S. Reiff-Marganiec, L. Blair, G. A. Campbell, and F. Wang. APPEL: Adaptable and Programmable Policy Environment and Language. Technical Report CSM-161, Computing Science and Mathematics, University of Stirling, UK, Apr. 2014.

[42] K. J. Turner, S. Reiff-Marganiec, L. Blair, J. Pang, T. Gray, P. Perry, and J. Ireland. Policy support for call control. *Computer Standards and Interfaces*, 28(6):635–649, June 2006.

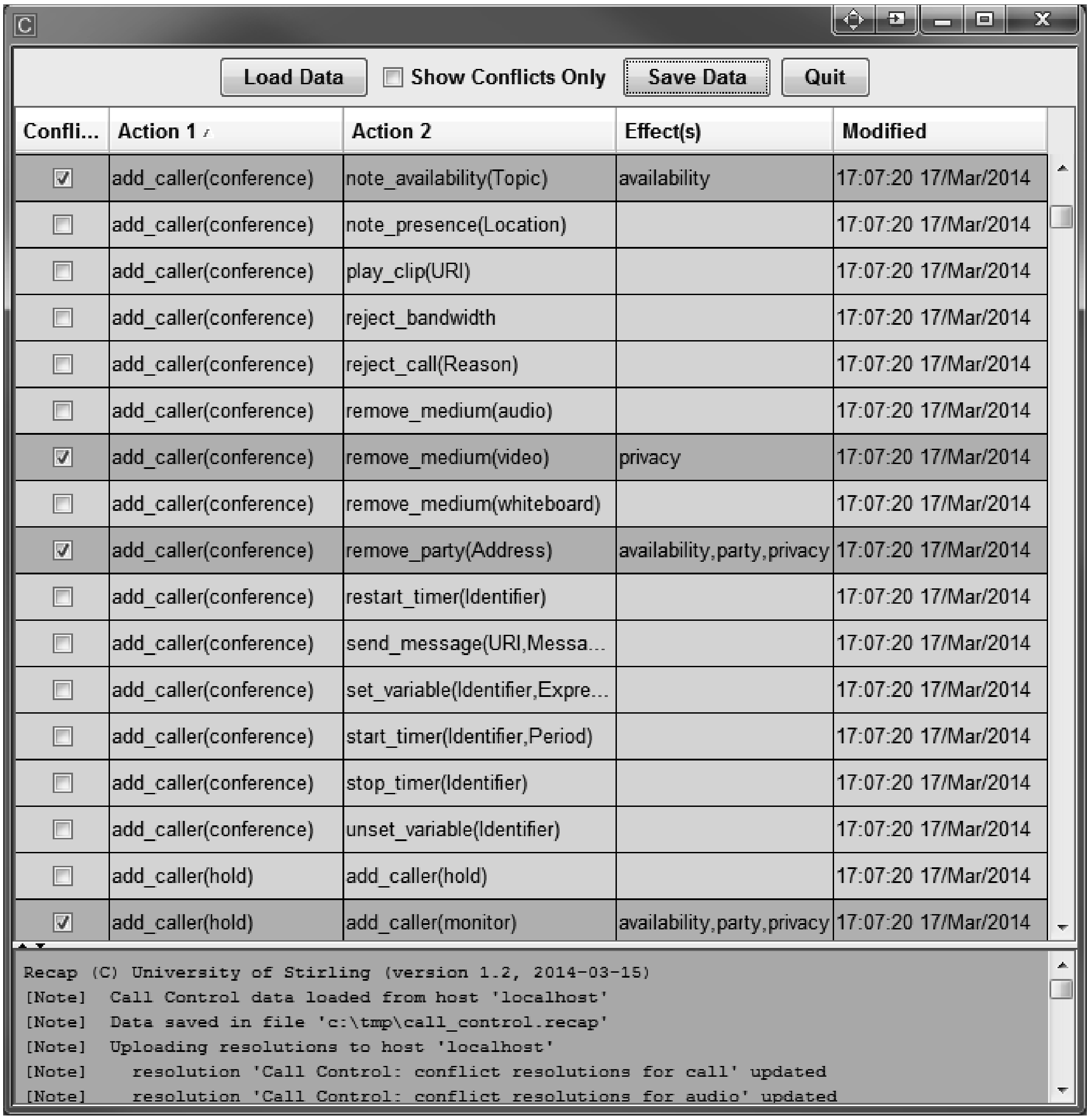
[43] M. Wilson, M. Kolberg, and E. H. Magill. Considering side effects in service interactions in home automation – An online approach. In L. du Bousquet and J.-L. Richier, editors, *Proc. 9th Int. Conf. on Feature Interactions in Software and Communications Systems*, pages 172–187. IOS Press, Amsterdam, Netherlands, May 2008.

[44] J. Woolham, B. Frisby, S. Quinn, A. Moore, and W. Smart. *The Safe at Home Project*. Hawker Publications, London, 2002.

[45] World Wide Web Consortium. *Web Ontology Language (OWL) – Reference*. Version 1.0. World Wide Web Consortium, Geneva, Switzerland, Feb. 2004.

[46] X. Wu and H. Schulzrinne, editors. *LESS: Language for End System Services in Internet Telephony*. Internet Draft. The Internet Society, New York, USA, Aug. 2005.

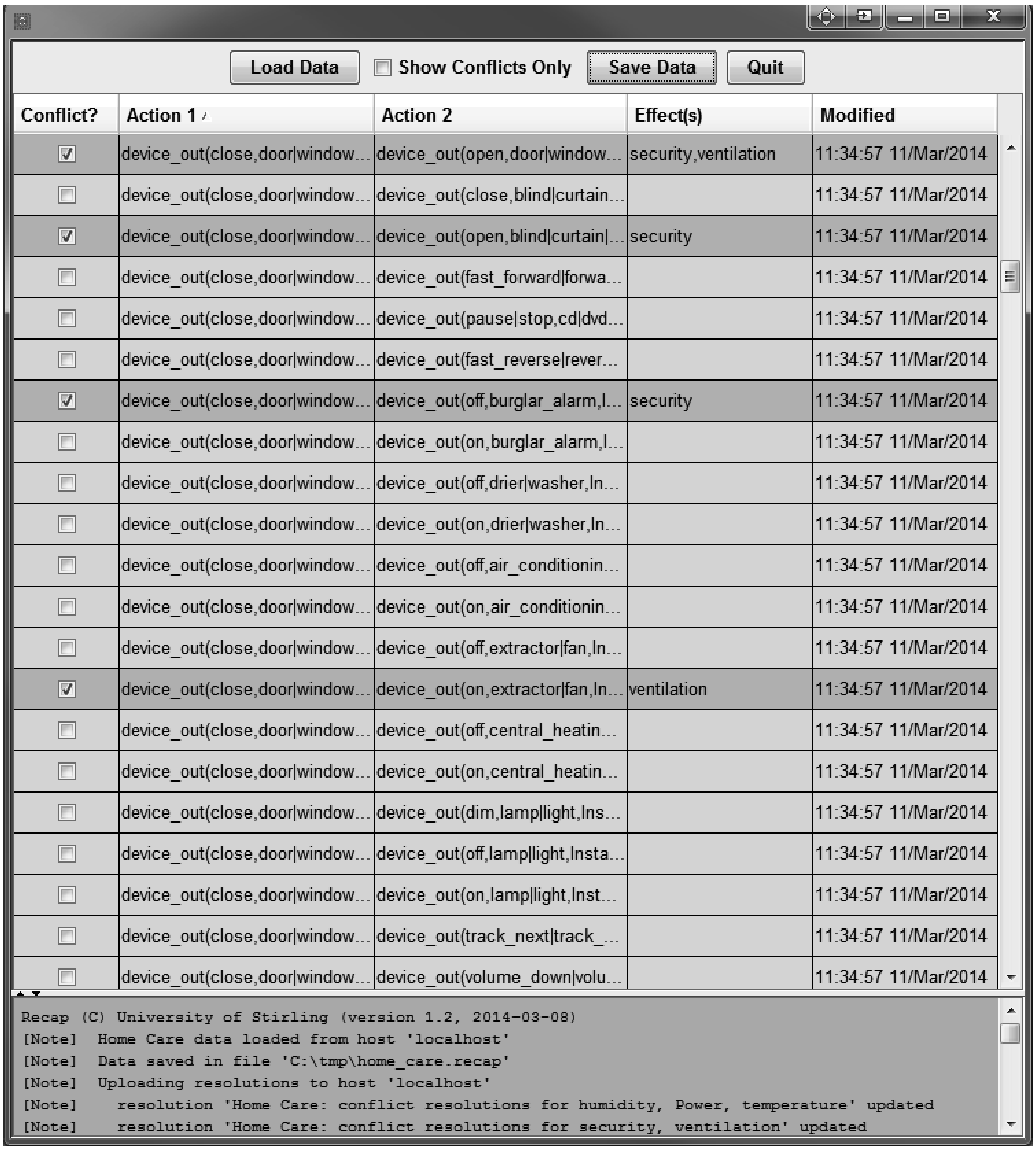
[47] X. Wu and H. Schulzrinne. Handling feature interactions in the language for end system services. *Computer Networks*, 51(2):515–535, Jan. 2007.



Confli...	Action 1	Action 2	Effect(s)	Modified
✓	add_caller(conference)	note_availability(Topic)	availability	17:07:20 17/Mar/2014
	add_caller(conference)	note_presence(Location)		17:07:20 17/Mar/2014
	add_caller(conference)	play_clip(URI)		17:07:20 17/Mar/2014
	add_caller(conference)	reject_bandwidth		17:07:20 17/Mar/2014
	add_caller(conference)	reject_call(Reason)		17:07:20 17/Mar/2014
	add_caller(conference)	remove_medium(audio)		17:07:20 17/Mar/2014
✓	add_caller(conference)	remove_medium(video)	privacy	17:07:20 17/Mar/2014
	add_caller(conference)	remove_medium(whiteboard)		17:07:20 17/Mar/2014
✓	add_caller(conference)	remove_party(Address)	availability,party,privacy	17:07:20 17/Mar/2014
	add_caller(conference)	restart_timer(Identifier)		17:07:20 17/Mar/2014
	add_caller(conference)	send_message(URI,Messa...		17:07:20 17/Mar/2014
	add_caller(conference)	set_variable(Identifier,Expre...		17:07:20 17/Mar/2014
	add_caller(conference)	start_timer(Identifier,Period)		17:07:20 17/Mar/2014
	add_caller(conference)	stop_timer(Identifier)		17:07:20 17/Mar/2014
	add_caller(conference)	unset_variable(Identifier)		17:07:20 17/Mar/2014
	add_caller(hold)	add_caller(hold)		17:07:20 17/Mar/2014
✓	add_caller(hold)	add_caller(monitor)	availability,party,privacy	17:07:20 17/Mar/2014

```
Recap (C) University of Stirling (version 1.2, 2014-03-15)
[Note]  Call Control data loaded from host 'localhost'
[Note]  Data saved in file 'c:\tmp\call_control.recap'
[Note]  Uploading resolutions to host 'localhost'
[Note]    resolution 'Call Control: conflict resolutions for call' updated
[Note]    resolution 'Call Control: conflict resolutions for audio' updated
```

Figure 1: Screen-Shot of Conflict Filtering in Call Control



Conflict?	Action 1	Action 2	Effect(s)	Modified
☑	device_out(close,door|window...	device_out(open,door|window...	security,ventilation	11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(close,blind|curtain...		11:34:57 11/Mar/2014
☑	device_out(close,door|window...	device_out(open,blind|curtain|...	security	11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(fast_forward|forwa...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(pause|stop,cd|dvd...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(fast_reverse|rever...		11:34:57 11/Mar/2014
☑	device_out(close,door|window...	device_out(off,burglar_alarm,l...	security	11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(on,burglar_alarm,l...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(off,drier|washer,In...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(on,drier|washer,In...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(off,air_conditionin...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(on,air_conditionin...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(off,extractor|fan,In...		11:34:57 11/Mar/2014
☑	device_out(close,door|window...	device_out(on,extractor|fan,In...	ventilation	11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(off,central_heatin...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(on,central_heatin...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(dim,lamp|light,Ins...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(off,lamp|light,Insta...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(on,lamp|light,Inst...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(track_next|track_...		11:34:57 11/Mar/2014
☐	device_out(close,door|window...	device_out(volume_down|volu...		11:34:57 11/Mar/2014

```
Recap (C) University of Stirling (version 1.2, 2014-03-08)
[Note]  Home Care data loaded from host 'localhost'
[Note]  Data saved in file 'C:\tmp\home_care.recap'
[Note]  Uploading resolutions to host 'localhost'
[Note]    resolution 'Home Care: conflict resolutions for humidity, Power, temperature' updated
[Note]    resolution 'Home Care: conflict resolutions for security, ventilation' updated
```

Figure 2: Screen-Shot of Conflict Filtering in Home Automation