Department of Computing Science and Mathematics

University of Stirling

Ontologies for Resolution Policy Definition and Policy Conflict Detection

Gavin A. Campbell

Technical Report CSM-172

ISSN 1460-9673

February 2007

Department of Computing Science and Mathematics
University of Stirling

Ontologies for Resolution Policy Definition and Policy Conflict Detection

Gavin A. Campbell

Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland
Telephone +44-1786-467421, Facsimile +44-1786-464-551

Email **gca@cs.stir.ac.uk**

Technical Report CSM-172

ISSN 1460-9673

February 2007

Abstract

In previous work, the author devised a collection of ontologies to model the generic structure and characteristics of the APPEL policy description language [18] utilised within the ACCENT policy system [1]. Two ontologies, namely `genpol.owl` and `wizpol.owl`, were defined using OWL (Web Ontology Language [10]) to describe the generic aspects of the policy language and aspects of how the policy wizard (user interface) uses the language. These generic ontologies are explained in CSM-169 [3], while an ontology modelling a domain-specific implementation of the policy language for call control is described in CSM-170 [4]. This document describes how these ontologies have been extended to define the structure of resolution policies, in addition to standard domain policies. A resolution policy has a similar structural composition to a standard policy, but places restrictions on the characteristics of its components. While a standard policy is used to define how events within the domain are handled, a resolution policy is defined purposely to resolve run-time conflicts between standard domain policies. Conflict arises between a pair of standard domain policies whose actions clash if executed simultaneously. A resolution policy specifies the action to be taken when such conflict occurs.

This report distinguishes between a standard policy and a resolution policy in APPEL, outlining the ontology description for each and highlighting subtle differences in their form. In particular, it demonstrates extensions to `genpol` and `wizpol` to specify generic resolution policies. Based on the generic extensions, `callcontrol.owl` was also expanded to include generic call control resolution actions. The call control ontology is described here as a concrete example of a domain-specific resolution policy language definition for the purposes of managing call conflicts. In addition, the report describes generic and domain-specific ontology extensions to aid in policy conflict detection using a filtering technique.

Keywords: ACCENT, Call Control, Conflict Filtering, Conflict Detection, Conflict Resolution, Ontology, OWL, Policy.

Table of Contents

Table of Figures

Conventions

1. Ontology conventions

The ontology documents described in this report use a specific naming convention with respect to class and property objects. The format adopted reflects a widely acknowledged general convention for OWL ontology design.

Ontology class naming convention
Ontology class names begin with a capital letter and do not contain spaces. Multiple words in a class name string start with a capital letter, conforming to what is known as 'CamelBack' notation.

For example: `PolicyVariableAttribute`

Ontology property naming convention
Ontology properties follow a similar convention to class names but start with a lower case letter. Property names begin with the word 'has' for clearer meaning in their application.

For example: `hasPolicyRule`

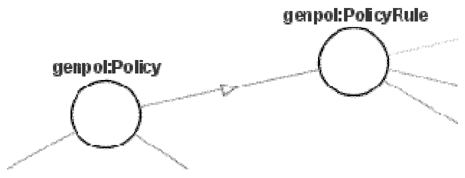
2. Diagram conventions

Diagrams depicted in this report were generated using the Jambalaya plug-in tool [7] and the OWLViz graphical plug-in tool [13] for Protégé-OWL 3.2. A key to the graphical notation used in each tool is outlined below.

Jambalaya
An ontology class is depicted by a single circle with the class name positioned directly above. By default, where applicable, Jambalaya displays the namespace prefix of a class (e.g. `genpol` or `wizpol`) in addition to its name, separated by a ':' symbol.

A property restriction is displayed as a straight line with a hollow triangle positioned at the mid-point. The 'point' of the triangle faces the target class, thus indicating the direction of the relationship. In the example below, the class Policy 'has' some relation with the class *PolicyRule*. Policy is the source class and *PolicyRule* is the target class of the illustrated restriction. Although not shown, a plausible restriction would be '*hasPolicyRule'*.

For example:

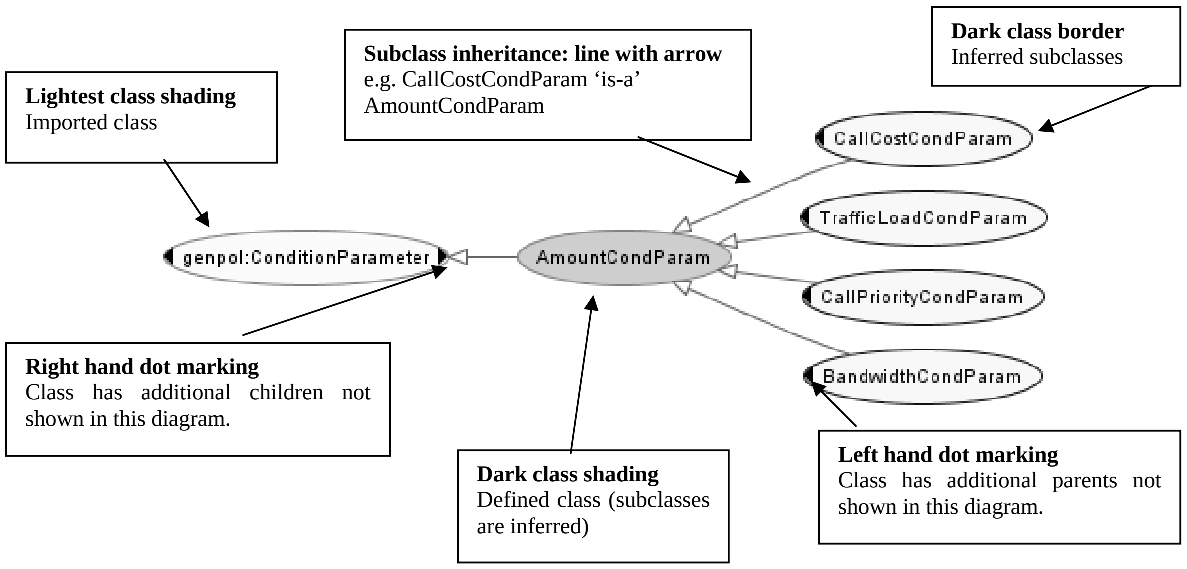


Sub-class (inheritance) is shown by a solid straight line without a triangle.

OWLViz
Each ontology class is represented by an oval shape with any subclass relationship shown by a curved line with a hollow triangular arrow head located at the superclass. OWLViz is used to illustrate ontology class inheritance only. Notation is not dissimilar from that of UML (Unified Modelling Language), signifying class inheritance or an 'is-a' relationship. Class names are displayed inside the oval class body, and imported class names are preceded by their namespace prefix (e.g. `genpol`, `wizpol`) separated by a ':' symbol.

Shading indicates whether a class is imported, defined or undefined. Imported classes have the lightest shading. Defined classes[1] are the most darkly shaded. Undefined classes have a darker shading than that of imported classes but not as dark as a defined class. Classes outlined with a darker border represent inferred subclasses.

OWLViz has the ability to restrict levels of class hierarchy displayed in a single diagram for ease of clarity. A class with additional parents (direct or inferred superclasses) not currently displayed is marked with a small black dot to the left hand side – indicating the class has further parent classes, but they are hidden in the current diagram. Similarly, a class may have additional children (direct or inferred subclasses) which may be omitted from a diagram. This is indicated by a black dot to the right-hand side of the class.


Subclass inheritance: line with arrow
e.g. CallCostCondParam 'is-a' AmountCondParam
Dark class border
Inferred subclasses
Lightest class shading
Imported class
CallCostCondParam
TrafficLoadCondParam
genpol:ConditionParameter
AmountCondParam
CallPriorityCondParam
BandwidthCondParam
Right hand dot marking
Class has additional children not shown in this diagram.
Left hand dot marking
Class has additional parents not shown in this diagram.
Dark class shading
Defined class (subclasses are inferred)

3. Report conventions

Throughout this report, ontology class, property and OWL file names are formatted using a `Courier` font. OWL ontology documents named `genpol.owl` and `wizpol.owl` are referred to as `genpol` and `wizpol` respectively.

For example the name of an ontology class is *LogEventAction*, an ontology property is *hasPolicyRule*, and similarly an OWL file name is recognised as `genpol`.

[1] A defined OWL ontology class is a class which has at least one property restriction deemed to be both necessary and sufficient. For further information refer to [6]. Typically, defined ontology classes are those whose subclasses are intended to be purely inferred.

1 Overview

An ontology describes a particular area of knowledge, including the key terms, their semantic interconnections and certain rules of inference. Using ontology to describe a domain allows a common understanding of the structure of information to be shared between software applications or agents. A further benefit is the ability to separate domain-specific knowledge from common operational knowledge in a system.

These advantages of ontology use have been employed in a move to generalise the ACCENT policy-based management system [1]. Previous implementation of this system saw both core policy language information and details specific to the original application domain (call control) embedded within the system interface. The lack of domain-independence imposed by such hard-coding, rendered the policy wizard incapable of easy adaptation to a new domain.

In previous work, the generic constructs of APPEL were defined in an ontology known as `genpol` [5]. A second ontology, named `wizpol`, directly extends `genpol` to specify common features employed to manipulate the language within the ACCENT system 'policy wizard' user interface. These ontologies form the general base on top of which any domain ontology may provide language details specific to the field it describes. This includes a definition of the precise triggering events, conditions and actions which may be used in policy execution within that domain. The ontology for call control (`callcontrol.owl`) is a concrete example of how an ontology may extend `genpol` and `wizpol` to provide a domain-specific implementation of the APPEL policy language. The OWL ontology documents `genpol.owl`, `wizpol.owl` and `callcontrol.owl` may be accessed at [5], [21] and [9] respectively.

In additional work, a system called POPPET (Policy Ontology Parser Program Extensible Translation) was developed to parse, interpret and incorporate ontology information within the ACCENT policy wizard. The system is described further in CSM-168 [2].

In the following introductory sub-sections, Section 1.1 provides an introduction to APPEL – the core policy description language used by the ACCENT system. Section 1.2 outlines the motivation for using ontology to define standard and resolution policy information, while Section 1.3 provides an overview of the language and tools chosen for ontology development. Section 1.4 describes the hierarchical stack of ontologies used to construct domain-specific standard and resolution policies.

1.1 APPEL Policy Description Language

A comprehensive policy description language called APPEL (the ACCENT Project Policy Environment/Language [18]) was designed to facilitate the creation of policies. APPEL comprises a core language schema which can be extended to support policy management for any given domain. APPEL was previously described using XML-based grammar – its syntax defined by means of XML Schema. Policies themselves are stored within the ACCENT system as XML documents.

The ACCENT system supports rule-based policies in event-condition-action (ECA) form. In relation to the concept of ECA, a policy rule broadly consists of three main components:

- A **trigger** set (events which potentially cause a policy to be executed)
- A **condition** set (contextual variables used to determine whether the triggers justify policy execution)
- An **action** set (output or resulting actions taken by the system upon policy execution).

The APPEL language describes the make-up of a policy. This includes the definition of a Policy Document which may be either a standard domain policy or a resolution policy. Both types of policy are outlined in this report and the differences between them explained.

1.2 Motivation for Ontology Usage

The motivation behind using ontologies stems from the need to generalise the ACCENT policy wizard so it may facilitate user-friendly policy creation for any customised domain. As the APPEL language contains a core component structure which may be reused across any domain-specific policy language, generic aspects of the language defined in the `genpol` ontology can be extended to suit the area in question. The use of ontology brings many benefits including the ability to define complex knowledge structures, reason with these using existing inference tools, and import and extend ontology structures. These features are key to achieving an extensible language framework, and are not possible using XML Schema alone.

1.3 OWL/Protégé Overview

A variety of specialised languages exist to define ontologies. OWL (The Web Ontology Language [10]) was the language chosen for ontology development. The language is XML-based and was officially standardised by the World Wide Web Consortium (W3C) in February 2004. OWL was chosen primarily due to its recent standardisation, the benefits this brings in terms of available software tool support, and compatibility with existing and future industrial and academic projects. In addition, OWL provides a larger function range than any other ontology language to date.

Ontology documents expressed in OWL are intended for use in applications where ontological content must be processed rather than simply extracted and presented to the human eye. OWL was designed to combine and extend the customisable tagging of XML with the flexible data representation ability of RDF (the Resource Description Framework [16]) with a view to formally describing the semantics of terminology in a domain.

The OWL language is broken down into three sub-languages that provide mounting strengths of expressiveness to meet the needs of different users and implementers. For a complete formal definition of the differences between OWL dialects, refer to [12]. In descending order, the dialects are:

- **OWL Full:** The complete OWL language, OWL Full provides maximum expressiveness in an ontology. It permits all the syntactic freedom of RDF but gives no computational guarantee that statements will be logically inferable using existing Description Logic reasoners.

- **OWL DL (Description Logic):** Designed to provide complete computational compatibility with Description Logic reasoners, OWL DL contains the full range of OWL language constructs, but places certain restrictions on how they are used. The result is an extremely expressive sub-language that can be used in conjunction with existing reasoning systems.

- **OWL Lite:** The weakest dialect, providing only a subset of OWL language constructs, OWL Lite was designed for users requiring simple constraints and a class hierarchy. Additionally, tool support for OWL Lite ontologies is easier to implement, and the documents themselves are more compact. As OWL Lite is a condensed subset of OWL DL, it also offers compatibility with existing reasoning tools.

To be compatible with existing formal reasoning tools, the ontologies outlined in this report were designed to conform to the OWL DL sub-language. An ontology can be validated to ensure its structure is compliant with the desired OWL sub-language. There are multiple online sources which provide a free validation service, including the WonderWeb OWL Ontology Validator [21]. To check the ontology described in this report, point the validator to the ontology URL as specified in [9].

Using OWL, an ontology is created by defining various classes, properties and individuals. A class represents a particular term or concept in the domain, while a property is a named relationship between two classes. An individual is an instance or 'member' of a class, usually representing real data content within an ontology. Properties are applied to classes in the form of 'restrictions'. A property restriction describes an 'anonymous' class, that is, a class of all individuals that satisfy the restriction. In OWL, each property restriction places a constraint on the class in terms of either a value (class or data type), or cardinality (number of values the property may be related to). The language also supports inheritance within class and property structures. A property restriction placed upon a class is automatically inherited by any of its subclasses. The Web Ontology Language Reference document [11] provides a complete description of all language constructs.

OWL ontology documents are often very large and complex to edit manually – especially when using OWL DL or OWL Full sub-languages as these utilise a broad range of constructs. Protégé [14] is a widely used tool throughout industry and academia for the creation of ontologies. Under continual, active development, it provides an effective user interface framework through which to define and edit ontology documents, and supports automated reasoning capability via any external Description Logic compatible reasoning engine. An extendable framework, Protégé supports the creation of OWL ontologies via a dedicated plug-in. Additional plug-in modules provide further specialised functions, such as graphical visualisation of ontology structure and class hierarchy diagram generation. Both of these were utilised for the figures within this report. The Protégé framework and all OWL modules are available to freely download.

Inference support during ontology development was achieved using the RacerPro reasoning engine [15]. Diagrams were generated with the aid of the OWLViz [13] and Jambalaya [7] plug-in tools for Protégé.

1.4 The OWL Ontology Stack

The aim of ontology development was to provide a solid knowledge base describing the generic aspects of APPEL, which could be extended to create a larger ontology specific to a particular domain application. Using OWL, two generic ontologies were created. A previous report CSM-169 [3] describes the contents for standard policies. This report describes their extensions for resolution policies.

At the base level, the `genpol` (Generic Policy Language) ontology describes the core constructs of the APPEL policy description language. This includes definition of key policy-related concepts such as Policy Document, Policy Variable, Policy Rule, Trigger, Condition and Action. Relationships between these concepts describe named associations, inheritance properties and cardinality restrictions. This ontology specifies a skeleton structure of ontology classes and properties, which can be imported and extended.

Rather than work directly with XML, the ACCENT system includes a policy wizard that provides a graphical user interface through which users can create and edit policies. Thus, the wizard contains explicit knowledge of both the generic aspects of the APPEL language and its domain specialisations. The policy wizard incorporates a number of features that control and manipulate domain data prior to its display. Such features are not part of the policy language itself, but are common and useful in any domain-specific ontology that is geared towards use with the policy system. Examples include categorisation of triggers, conditions, actions and operators, and the inclusion of 'user-level' grouping categories to restrict the range of language functionality depending on a user's skill or authorisation level. This additional, wizard-related knowledge is defined in a second base ontology known as `wizpol` (the Wizard Policy Language ontology). `Wizpol` directly extends the `genpol` ontology, thus specialising the APPEL language for use with the policy wizard.

OWL supports the sharing and reuse of ontologies by means of ontology importation. Using this mechanism, all definitions of classes, properties and individuals within an imported ontology, may be used by the importing ontology. `Wizpol` imports the `genpol` ontology and extends it to provide additional user interface features not directly related to the APPEL policy language. In turn, `wizpol` is then extended to specialise the language for a particular application. Extending ontologies in this way has resulted in an ontology 'stack' or layered model, through which to build a domain-specific policy language ontology. This stack is shown in Figure 1.

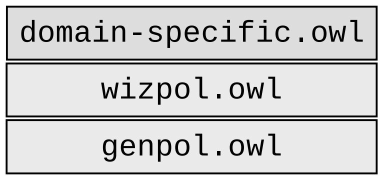


Figure 1. OWL Ontology Stack

In Figure 1, `domain-specific.owl` represents an ontology used to describe a particular domain, such as callcontrol.owl described in this report. These ontologies define only the structure of policy-related knowledge and not actual policy data. For this reason, none of the developed ontologies contain individuals or 'instances' of ontology classes. All constraints have been applied strictly to 'anonymous' classes. That is, relationships between classes are described in purely abstract terms.

The ontologies of `genpol` and `wizpol` are intended to be entirely reusable. Due to the recursive nature of the OWL import mechanism, a domain-specific ontology is required to import only `wizpol` – importation of `genpol` is inherently automatic. A domain-specific ontology – call control in this report – extends the class hierarchy of the imported `wizpol` ontology structure to define additional sub-classes and properties together with applicable constraints. This includes definition of the trigger events, condition parameters and actions particular to standard and resolution policies.

Although the developed ontology stack structure is specially geared towards tailoring the language for use within the policy wizard, a domain-specific ontology may include additional non-policy related knowledge. The presence of the `genpol` and `wizpol` ontology structure ensures compatibility with the ACCENT system, but the same ontology may contain additional domain knowledge and an unlimited number of imported ontologies for use by other applications or agents.

2 Generic Policy Structure

The generic aspects of the APPEL policy description language [18] are defined within the ontology known as `genpol`. This section describes the structure of a policy document in APPEL which is used to define both standard and resolution policies.

2.1 Generic Policies in APPEL

At the highest level of policy language abstraction, the outline of a policy described within `genpol` is depicted in Figure 2. Each labelled rectangle represents an ontology class, while linking arrows are ontology properties which indicate relationships between two classes. In brief, A *PolicyDocument* is related to *Policy* and *PolicyVariable* instances. A *Policy* has both *PolicyAttribute* and *PolicyRule* instances. Each *PolicyRule* has an associated *TriggerEvent*, *Condition* and *Action*. Both *TriggerEvent* and *Action* may have *Argument* instances, while a *Condition* is related to a *ConditionParameter*, a *ConditionOperator* and a *ConditionValue*.

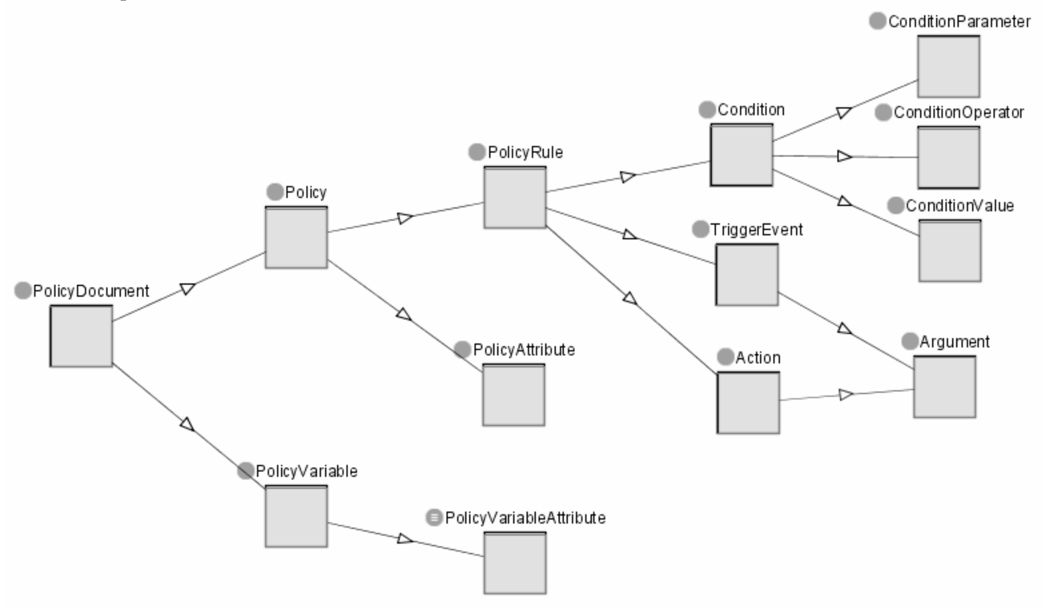


Figure 2. High-level Generic Policy Structure

2.2 Types of Policy

A *Policy* can be either a *StandardPolicy* or a *ResolutionPolicy*, as shown in Figure 3. Each oval represents a class. The connecting lines depict class inheritance. Standard policies and resolution policies share a similar structure in the policy language components they are constructed with.

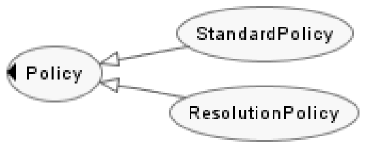


Figure 3. Types of Policy

A distinction is made between the types of triggers, conditions and actions used to define the characteristics of standard and resolution policies. These are shown in Figure 4, Figure 5 and Figure 6 respectively. Where a class appears with darker shading, this indicates it has a specific set of values described within the ontology. Arrows to the left or right of a class indicate presence of further superclasses or subclasses respectively, which are not shown.

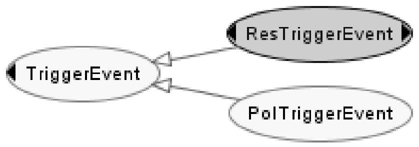


Figure 4. Types of TriggerEvent

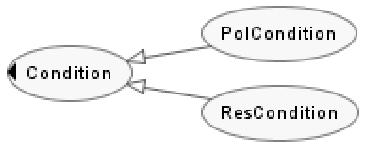


Figure 5. Types of Condition

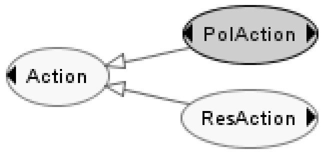


Figure 6. Types of Action

Section 3 describes the components of a standard policy, while section 4 describes the components for a resolution policy. Section 5 outlines how a resolution policy may be customised for a particular domain. Section 6 describes and explains how additional ontology information is used to aid in conflict detection, using the call control domain as an example.

3 Standard Policies

A *StandardPolicy* is a policy which is used to express how particular situations in a domain may be handled. Standard policies offer control of domain events. `Genpol` describes the make-up of a *StandardPolicy* as shown in Figure 7.

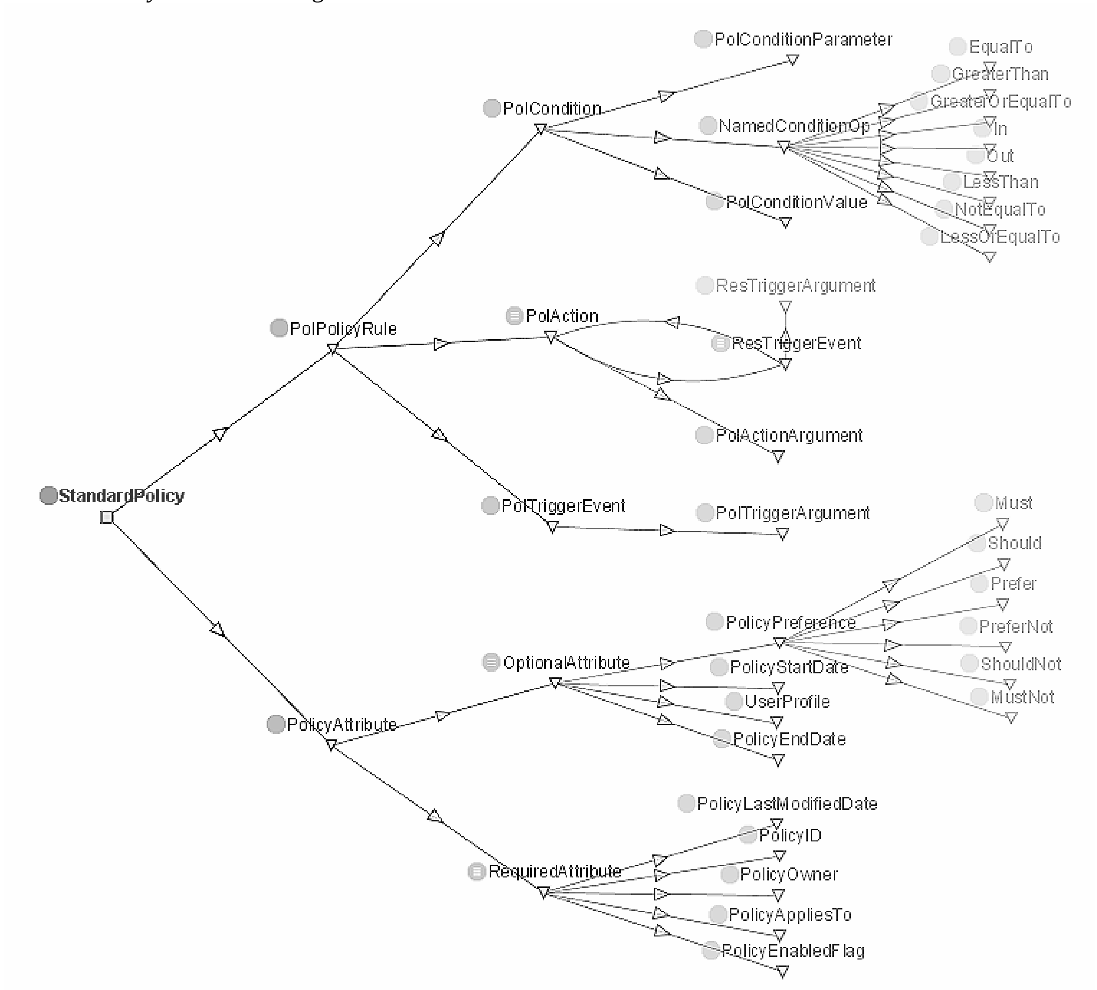


Figure 7. Standard Policy Class Hierarchy

3.1 Standard Policy Rules

A *PolicyRule* within a *StandardPolicy* is defined to have zero or more associations with *PolTriggerEvent*, zero or more associations with *PolCondition*, and at least one association with *PolAction*.

3.2 Standard Policy Triggers

A *PolTriggerEvent* is defined as a subclass of *TriggerEvent* as shown in Figure 8.

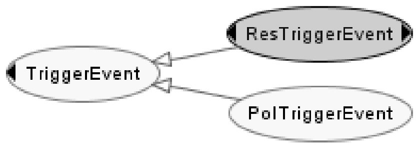


Figure 8. Standard Policy TriggerEvent

3.3 Standard Policy Conditions

A *PolCondition* is defined to have a single association with each of *PolConditionParameter*, *ConditionOperator* and *PolConditionValue*. The parameter and value of a *StandardPolicy* is customisable to the domain whereas the operator must be selected from a named set of operators defined within the ontology. Defined operators, such as *EqualTo*, *NotEqualTo, GreaterThan*, and *LessThan* for example, are shown in Figure 7.

3.4 Standard Policy Actions

A *PolAction* is defined as a subclass of *Action* as shown in Figure 9. A standard policy action can also be a trigger within a resolution policy. Therefore, *PolAction* is also a subclass of *ResTriggerEvent*. Conversely, *ResTriggerEvent* is a subclass of *PolAction*.

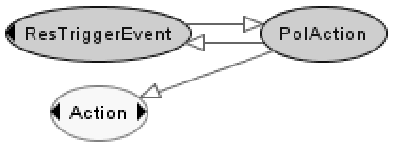


Figure 9. Standard Policy Action

4 Resolution Policies

A *ResolutionPolicy* is a policy which is used to express how conflicts between actions of standard policies may be resolved. Resolution policies follow the same structural format as standard policies, with differences occurring in the types of arguments and parameters of triggers, conditions and actions. `Genpol` defines the make-up of a *ResolutionPolicy* as shown in Figure 10.

The following subsections describe the format of triggers, conditions, actions and variables for resolution policies.

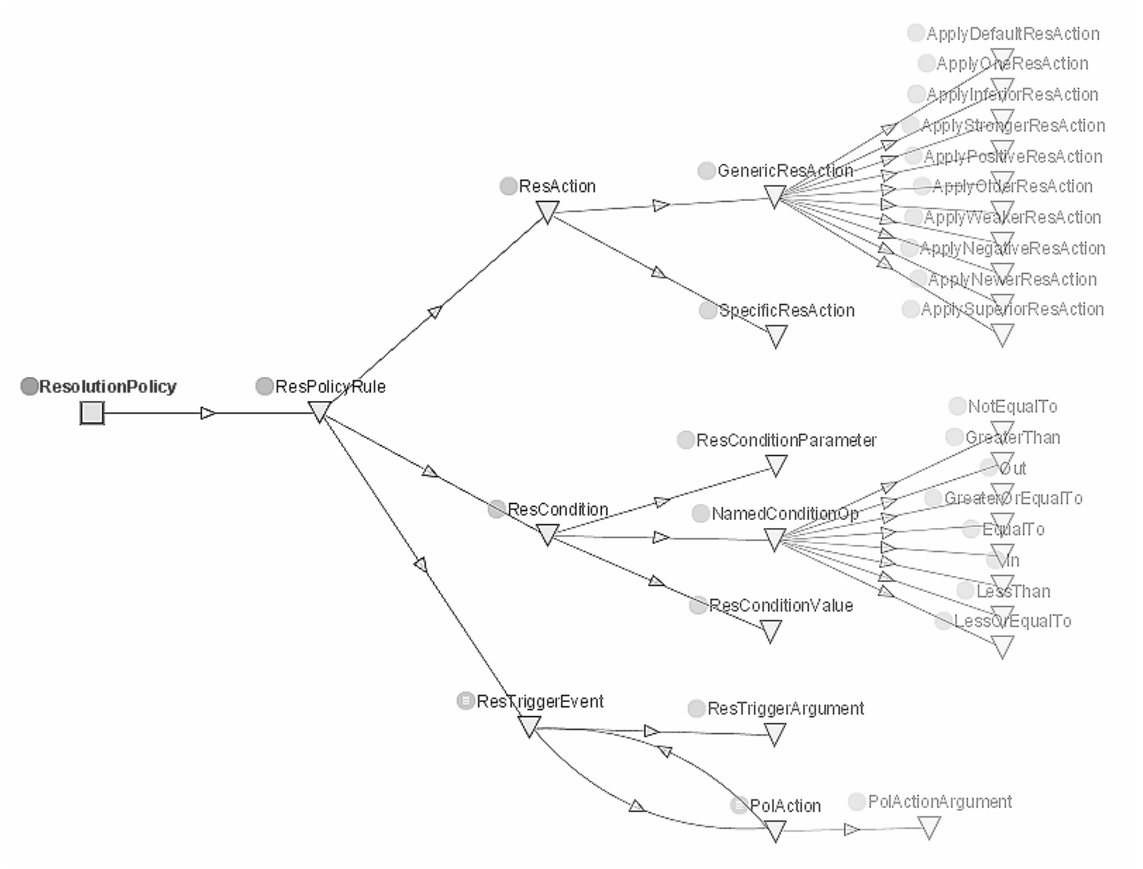


Figure 10. Resolution Policy Class Hierarchy

4.1 Resolution Policy Rules

A *PolicyRule* within a *ResolutionPolicy* is defined to have a minimum of two associations with *PolTriggerEvent*, zero or more associations with *PolCondition* and at least one association with *PolAction*. The most notable difference between a *ResPolicyRule* and a *PolPolicyRule* is that a resolution policy must define at least two triggers – relating to the pair of conflicting policy actions. A *PolPolicyRule* may contain zero or more triggers.

4.2 Resolution Policy Triggers

As resolution policies exist to define how conflicts between standard policy actions are handled, the triggering events for a *ResolutionPolicy* are in fact a combination of domain-specific standard policy actions. This is shown in Figure 11, where a *ResTriggerEvent* is defined to be a *PolAction* (standard policy action). Conversely, a *PolAction* is also a *ResTriggerEvent*. In general, a resolution policy will have just two triggers, relating to the pair of conflicting actions to be resolved.

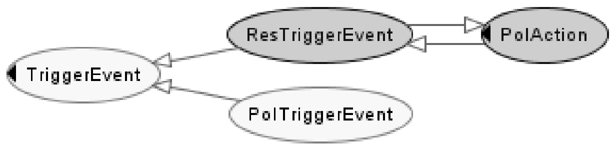


Figure 11. Resolution Policy Trigger Class Hierarchy

4.3 Resolution Policy Conditions

Resolution policy condition rules follow the same format as a standard policy condition. Each *ResCondition* has three component parts of a parameter, an operator and a value. The ontology definition is shown in Figure 12.

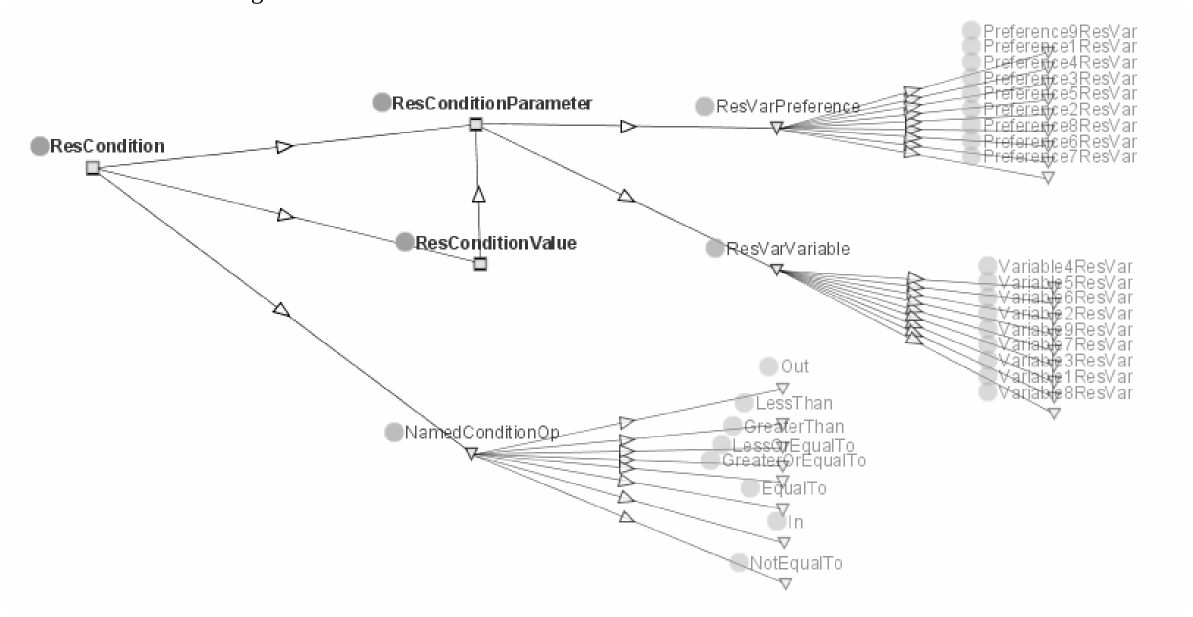


Figure 12. Resolution Policy Condition Class Hierarchy

With reference to Figure 12, a *ResConditionParameter* may be either a *ResVarPreference* (variable bound to the preference value of one of the conflicting standard policies in question) or a *ResVarVariable* (variable bound to the actual arguments of a trigger). The range of condition operators which may be used is common for all types of policies – both standard and resolution. In contrast to a standard policy condition, a *ResConditionValue* of a *ResCondition* may be any value used as a *ResConditionParameter*, so parameters (standard policy preferences or trigger arguments) may be compared. Additionally, a *ResConditionValue* can be a literal value.

4.4 Resolution Policy Actions

In principle, the action of a *ResolutionPolicy* could simply be to execute one of the two conflicting standard policy actions. However, the action could also be any available standard policy action defined for the domain in question. Therefore, actions of a *ResolutionPolicy* may be either generic or specific, as shown in Figure 13.

A *GenericResAction* is a general action which is used to select one of the conflicting policy actions for execution. These actions are defined within `genpol` and can be used in resolution policies for any domain. They consider standard policies in a generic sense, comparing attributes of the policies such as policy preference level (e.g. *ApplyNegativeResAction*, *ApplyStrongerResAction*), definition date (e.g. *ApplyOlderResAction*, *ApplyNewerResAction*) and domain (e.g. *ApplyInferiorResAction*, *ApplySuperiorResAction*). In addition, where applying a single generic action does not result in elimination of one of the conflicting actions, the *ApplyDefaultResAction* and *ApplyOneResAction* options can be used to narrow it down to a single selection.

A *SpecificResAction* is a standard policy action defined in a domain-specific implementation of the policy language. Examples are shown in section 5 for the domain of call control.

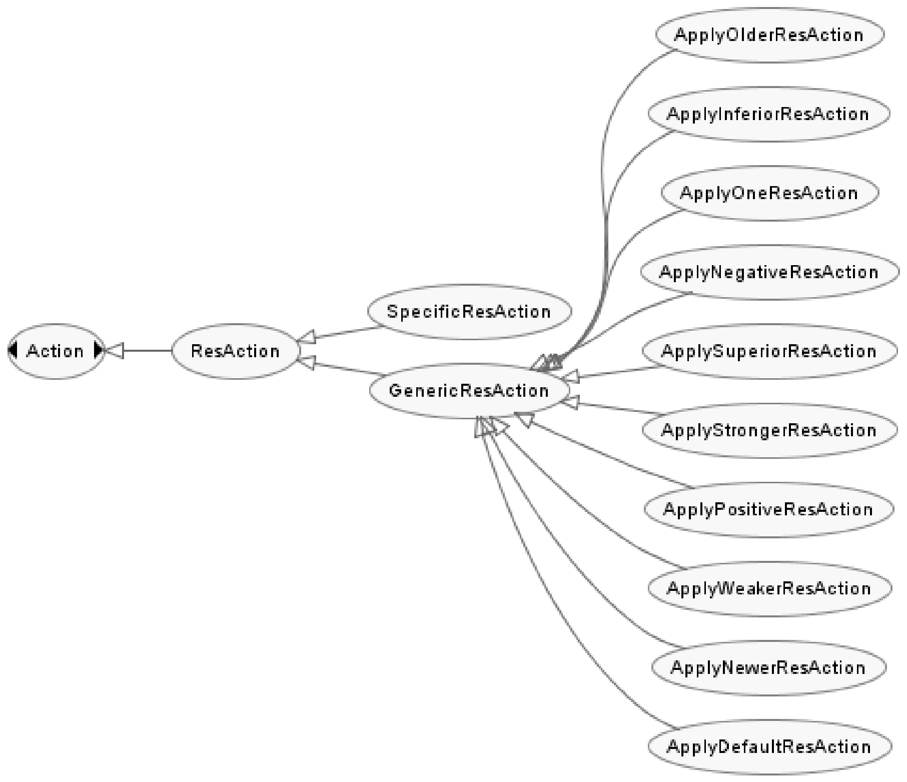


Figure 13. Resolution Policy Action Class Hierarchy

4.5 Resolution Policy Variables

Resolution variables are specifically bound to data values relating to preferences or arguments within conflicting standard policies. The `genpol` ontology class definition is shown in Figure 14. A *ResVariable* may be either a *ResVarVariable* or a *ResVarPreference*.

A *ResVarVariable* is a variable bound to an actual argument of a trigger. A *ResVarPreference* is a variable bound to a preference value of one of the conflicting policies. Although typical resolution policies will require two variables to represent policy trigger arguments and two variables to represent policy preferences, the policy language makes provision for an arbitrary number of variables. In this case, the ontology defines nine variables to represent *ResVarVariable* and nine variables to represent *ResVarPreference*.

4.6 Resolution Policy Attributes

A *ResolutionPolicy* has a range of *RequiredAttribute* instances as shown in Figure 15. Unlike a *StandardPolicy*, resolution policies do not include *OptionalAttribute* instances as depicted in Figure 7 for standard domain policies.

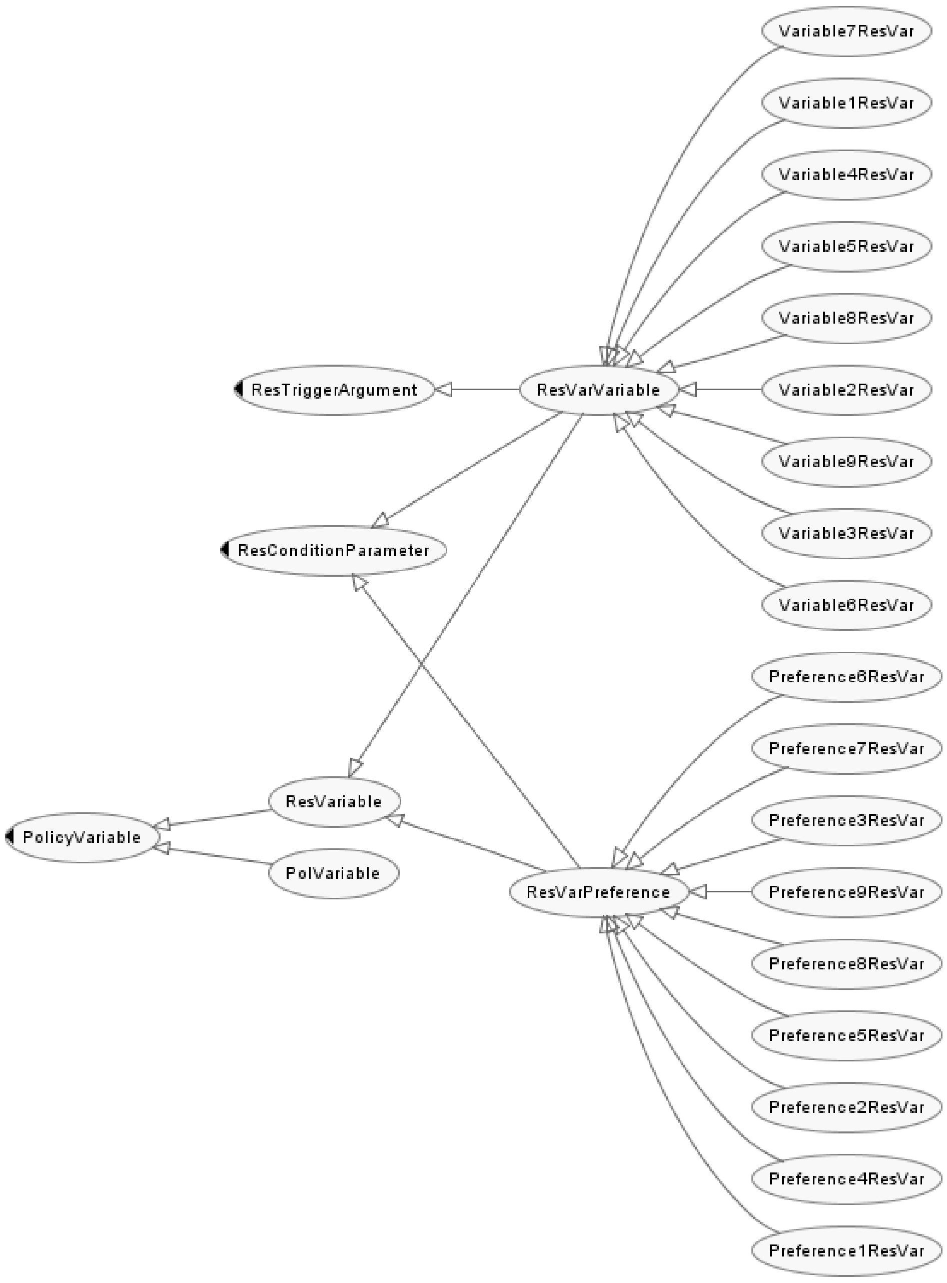


Figure 14. Resolution Policy Variables

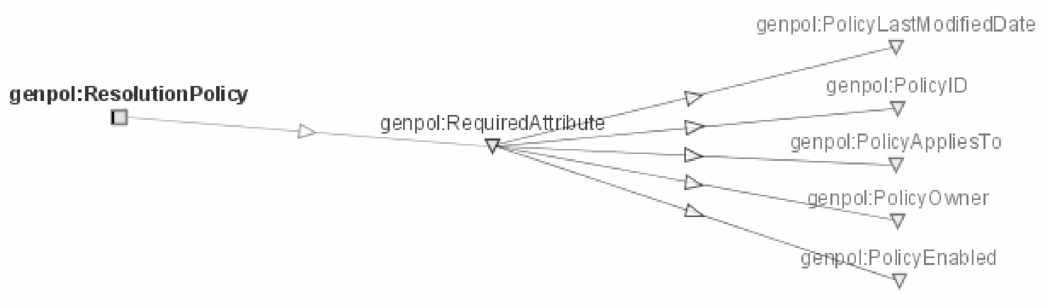


Figure 15. Resolution Policy Attributes

5 Domain-Specific Resolution Policies

In section 4, the basic form and structure of a resolution policy was described. Such generic language details are contained within the genpol ontology. In this section, domain-specific resolution policies are described using the example domain of call control. These details are defined within the ontology named callcontrol.owl, which imports and extends the ontologies of wizpol and genpol as previously outlined in **Error! Reference source not found.**. Specifically, the domain ontology extends the generic resolution policy description within genpol to define *SpecificResAction* subclasses particular to call control.

While genpol defines a number of generic resolution actions, these are applicable to any domain policy. Within the call control ontology, *SpecificResAction* is extended to define resolution actions particular to call control. As shown in Figure 16, the actions defined are *ApplyCalleeResAction* and *ApplyCallerResAction*. These actions can be used in resolution policies to handle call control policy conflicts. Their effect is to execute the action associated with the callee or caller respectively when a conflict occurs. For example, if two policies conflict when the caller wishes to refrain from using video during a call and the callee wishes to include video, one resolution may be to apply the caller's policy as they are paying for the call.

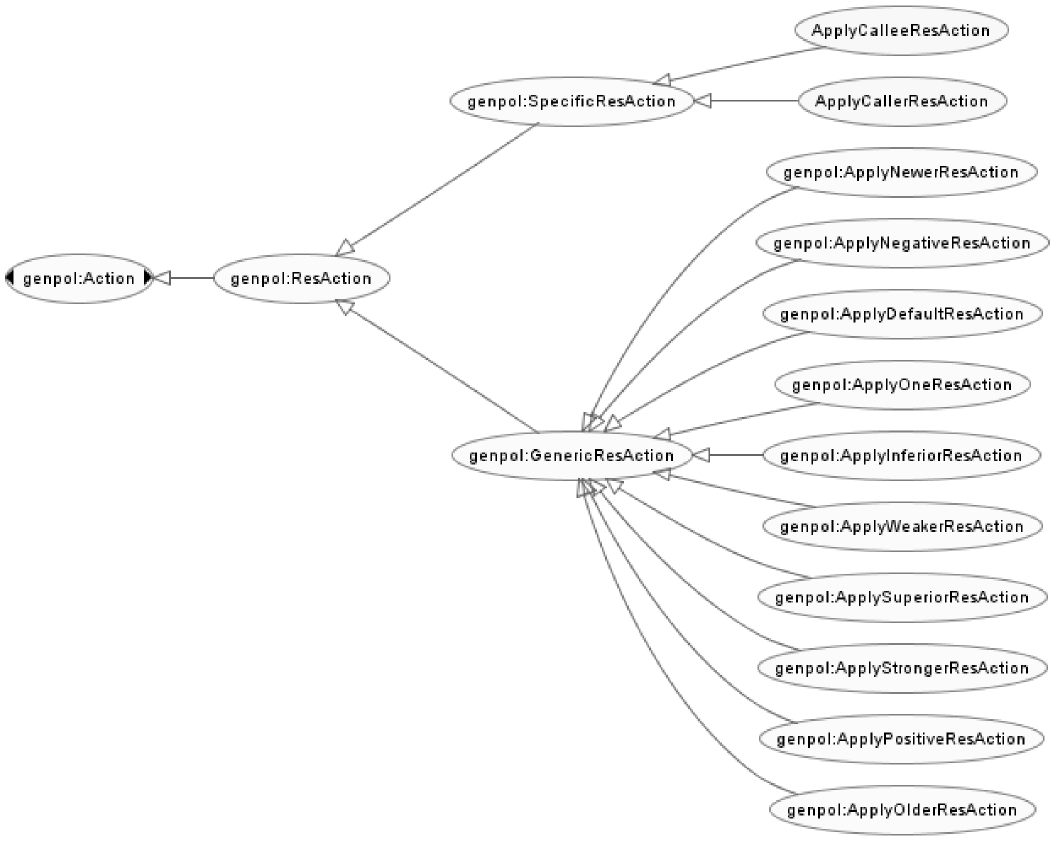

ApplyCalleeResAction
genpol:SpecificResAction
ApplyCallerResAction
genpol:ApplyNewerResAction
genpol:ApplyNegativeResAction
genpol:Action
genpol:ResAction
genpol:ApplyDefaultResAction
genpol:ApplyOneResAction
genpol:GenericResAction
genpol:ApplyInferiorResAction
genpol:ApplyWeakerResAction
genpol:ApplySuperiorResAction
genpol:ApplyStrongerResAction
genpol:ApplyPositiveResAction
genpol:ApplyOlderResAction

Figure 16. Specific Resolution Actions

In addition, any domain-specific action defined within callcontrol.owl can be used as a resolution policy action. These actions are listed in the following section which describes how a domain-specific ontology may be utilised in the process of policy conflict detection.

6 Policy Conflict Detection Support

During policy execution, the outcome of two policies may conflict. For example, one policy may have an action to add a medium to the call, whereas another may have an action to remove a medium from the call. If both policies become eligible for execution at the same time, the policy system must detect they conflict and resolve the situation – typically by choosing one of the two policies to execute. For the purposes of conflict filtering, it is necessary to consider every possible paired combination of actions statically to identify those with similar effects on the environment. The subject of policy conflict filtering and techniques to aid in detection and resolution of conflicts is discussed in [20].

6.1 Action Effect Ontology Support

For the purposes of conflict filtering, the ontology stack includes an additional class and object property to define action effect categories and associate them with particular actions. As this information is not part of the policy language but is common to any domain-specific policy language specialisation, it is omitted from `genpol` and defined in `wizpol`.

The new class was named *ActionEffect* and the property named *hasActionEffect*. Domain-specific ontology extensions for the policy language can define particular effect categories as subclasses of *wizpol:ActionEffect*. Domain actions are associated with these categories using the *wizpol:hasActionEffect* property.

6.2 Detecting Conflicts Between Actions

The call control ontology defines a set of standard policy actions as shown in Figure 17. Each action is deemed to have a particular effect on the execution environment. The effect categories of these actions are shown in Figure 18.

Conflicts can occur between actions themselves or between the parameters of actions. While two actions may appear to be unrelated, both may utilise a parameter which could potentially conflict. Therefore, conflicts must be considered between all action→action pairs and all action(argX)→action(argX) pairs for actions with enumerated parameter values.

Action parameters are defined as *genpol:ActionArgument* subclasses in the call control ontology. Call control action arguments are shown in Figure 19. For example, *AddMediumAction* is associated with *MediumActionArg*. Argument types with an explicit set of values have these defined as subclasses. For example, *MediumActionArg* is the parameter type, while possible parameter values are either *AudioMedium*, *VideoMedium* or *WhiteboardMedium*.

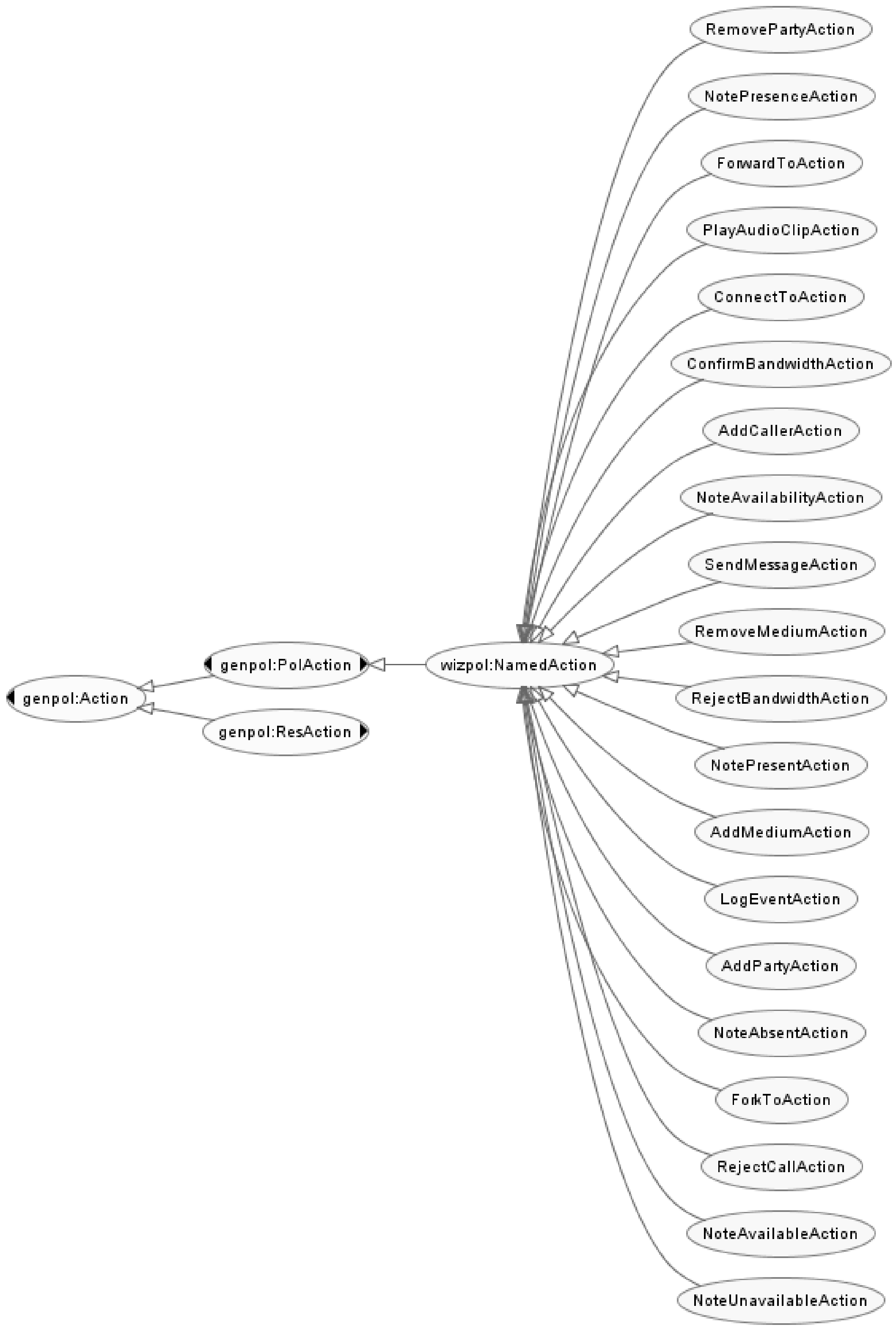


Figure 17. Call Control Actions

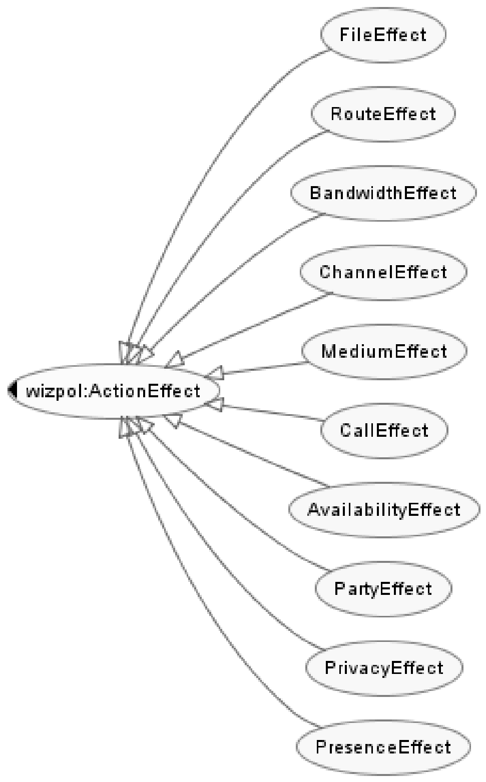


Figure 18. Call Control Action Effect Categories

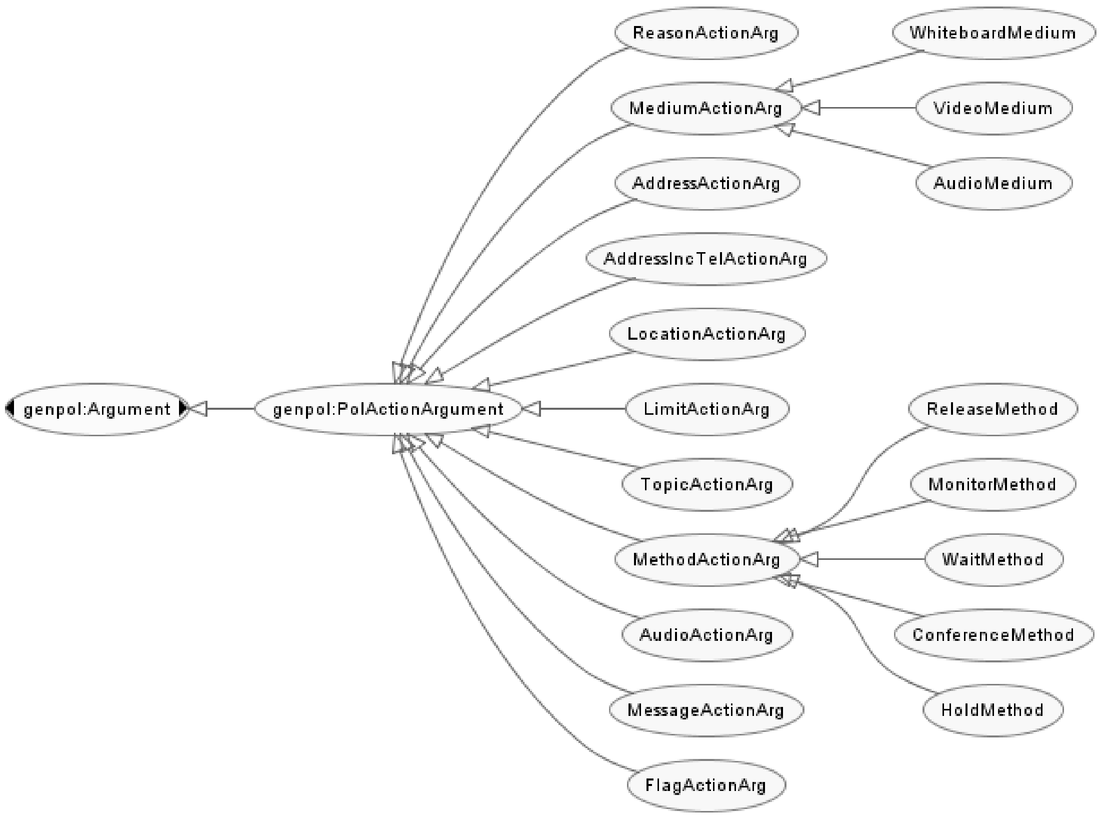


Figure 19. Call Control Action Arguments

To associate actions with effect categories, the property *wizpol:hasActionEffect* is used. An example is shown in Figure 20. The action *AddMediumAction* is shown linked along this property with *MediumEffect* and *PrivacyEffect*. This is because the action of adding medium to a call may affect both ‘medium’ (as audio or video may be added to the call), and also ‘privacy’ (as the introduction of, say, video may conflict with the callee who does not want to have their image broadcast).

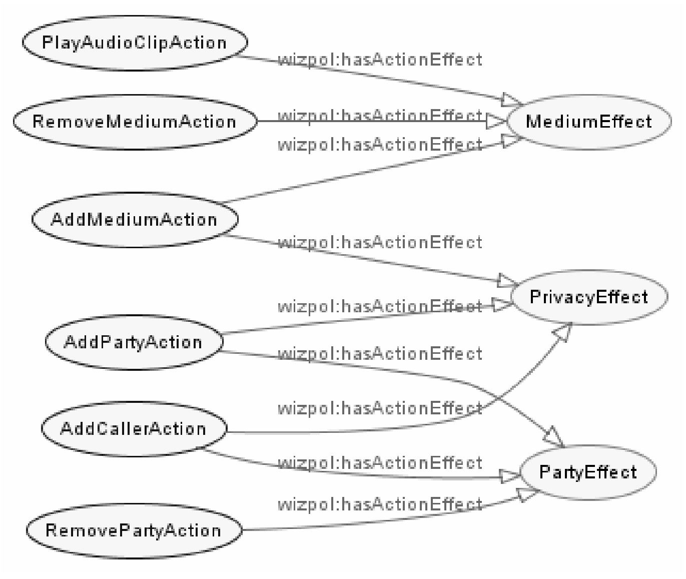


Figure 20. Example Actions and Effect Categories

Together with *AddMediumAction*, Figure 20 shows the actions *PlayAudioClipAction* and *RemoveMediumAction* are also defined to affect medium. From this it can be inferred that any pairing of these three actions could potentially result in conflict.

7 Conclusion and Future Work

This document outlined how a series of ontology documents was extended to describe the generic structure and characteristics of resolution policies in the APPEL policy description language. A resolution policy shares similar structural components to a standard domain policy in APPEL, but differs in the form of the components. Standard policies define how certain events in a domain should be handled, whereas a resolution policy defines the action that should be taken when the actions from a pair of standard policies conflict if they become eligible for execution as a result of the same triggers.

The extensions made to the generic language ontology `genpol` describe the components used to form resolution policies and also a set of generic resolution policy actions. Resolution actions specific to a particular domain can be defined in a domain ontology. This report used an ontology for call control as a working example.

A technique to aid in policy conflict filtering was outlined in section 6. The procedure is not related to the APPEL language so its implementation required n extension to `wizpol` and the call control ontology only. The technique aids in the early stages of conflict detection between policies in APPEL using the concept of 'action effect categories' which standard domain policy actions can be linked with. Actions with similar effect categories may be potentially conflicting. The ontologies contain classes and property descriptions to specify the links. Using this technique, external tools can make use of effect information and automatically generate lists of conflicting actions. This allows creation of skeleton resolution policies to handle such conflicts. A stand alone application to perform such semi-automated conflict filtering and resolution policy generation is under development.

References

[1] ACCENT Policy-based system Project home page: http://www.cs.stir.ac.uk/accent.

[2] Campbell, G.A. An Overview of Ontology Application for Policy-based Management using POPPET. Technical Report CSM-168. June 2006.

[3] Campbell, G.A. Ontology Stack for A Policy Wizard, Technical Report CSM-169, June 2006.

[4] Campbell, G.A. Ontology for Call Control, Technical Report CSM-170. June 2006.

[5] Generic Policy Language Ontology document (genpol.owl). Located online at URL: http://www.cs.stir.ac.uk/schemas/genpol.owl, February 2007.

[6] Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C. A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. The University of Manchester, August 2004.

[7] Jambalaya visual plug-in tool for Protégé. Home page: http://www.thechiselgroup.org/~chisel/projects/jambalaya/jambalaya.html, February 2007.

[8] Knublaugh, H. User-defined Datatypes in Protégé-OWL. Located online: http://protege.stanford.edu/plugins/owl/xsp.html, Last updated: August 2005. Last accessed: February 2007.

[9] Ontology of (Internet) Call Control (callcontrol.owl). Located online at URL: http://www.cs.stir.ac.uk/schemas/callcontrol.owl, February 2007.

[10] OWL: The Web Ontology Language. http://www.w3.org/2004/OWL/, February 2007.

[11] OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, February 2007.

[12] OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, February 2007.

[13] OWLViz graphical plug-in tool. Home page: http://www.co-ode.org/downloads/owlviz/, February 2007.

[14] Protégé home page: http://protege.stanford.edu/, February 2007.

[15] Racer Systems GmbH & Co. KG. Home Page and download links: http://www.racer-systems.com/index.phtml, February 2007.

[16] RDF: The Resource Description Framework. http://www.w3.org/RDF/, February 2007.

[17] RDF Schema (RDFS): http://www.w3.org/TR/rdf-schema/, February 2007.

[18] Reiff-Marganiec, S., Turner, K.J. APPEL: The ACCENT Project Policy Environment/Language. Technical Report CSM-161, June 2005.

[19] Turner, K.J, The ACCENT Policy Wizard. Technical Report CSM-166, May 2005.

[20] Turner, K.J and Blair, L. Policies and Conflicts in Call Control, Computer Networks, 51(2):496–514, February 2007.

[21] Wizard Policy Language Ontology document (wizpol.owl). Located online at URL: http://www.cs.stir.ac.uk/schemas/wizpol.owl, February 2007.

[22] WonderWeb OWL Ontology Validator. University of Manchester, 2003. Service located online at URL: http://phoebus.cs.man.ac.uk:9999/OWL/Validator, February 2007.