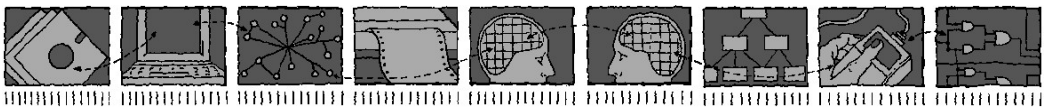
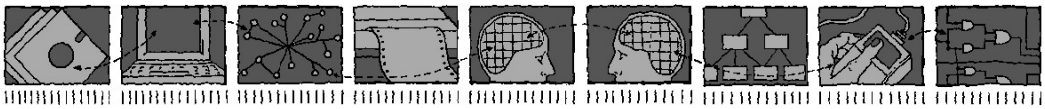


Department of Computing Science and Mathematics
University of Stirling



Ontology Stack for A Policy Wizard

Gavin A. Campbell

Technical Report CSM-169

ISSN 1460-9673

June 2006

Department of Computing Science and Mathematics
University of Stirling

Ontology Stack for A Policy Wizard

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-169

ISSN 1460-9673

June 2006

Abstract

An ontology provides a common vocabulary through which to share information in a particular area of knowledge, including the key terms, their semantic interconnections and certain rules of inference. The ACCENT policy-based management system uses a policy description language called APPEL and supports policy document formation through the use of a comprehensive user interface wizard. Through the use of OWL (the Web Ontology Language), the core aspects of APPEL have been captured and defined in an ontology. Assigned the acronym `genpol`, this ontology describes the policy language independent of any user interface or domain-specific policy information. A further ontology has been developed to define common interface features implemented by the policy wizard [17]. This ontology, referred to as `wizpol`, directly extends `genpol`. `It` provides additional information to the language itself, whilst retaining freedom from any domain-specific policy details. Combined, both `genpol` and `wizpol` act as a base for defining further domain-specific ontologies which may describe policy options tailored for a particular application.

This report presents a technical overview of both the generic policy language ontology (`genpol`) and the wizard policy ontology (`wizpol`), expressed in the form of graphical depictions of OWL classes and properties.

Keywords: ACCENT, Policy, Ontology, OWL.

Table of Contents

Abstract ..... **i**
Table of Contents ..... **ii**
Table of Figures ..... **iii**
Conventions ..... **iv**

1 Overview **1**
1.1 APPEL Policy Description Language 1
1.2 Motivation for Ontology Usage 1
1.3 OWL/Protégé Overview 2
1.4 The OWL Ontology Stack 3

2 The Generic Policy Language Ontology: `genpol.owl` **5**
2.1 Overview of genpol 5
2.2 Policy Document 5
2.3 Policy 6
2.4 Policy Rule, Trigger, Condition and Action. 6
2.5 Arguments 7
2.6 Policy Attribute 7
2.7 Policy Variable Attribute 9
2.8 Operators 9
2.8.1 Condition Operators 9
2.8.2 Combination Operators 10

3 Wizard policy language ontology: `wizpol.owl` **12**
3.1 Trigger, Condition and Action Class Wizard Extension 12
3.2 Class Categorisation 13
3.2.1 User-Level Categorisation 14
3.2.2 Internal Use Categorisation 14
3.2.3 Trigger, Condition Parameter and Action Categorisation 15
3.3 Operator Extension (User-Level Provision) 15
3.3.1 Admin Level Operators 15
3.3.2 Expert Level Operators 17
3.3.3 Intermediate Level Operators 19
3.3.4 Novice Level Operators 21
3.4 Status Variables 23
3.5 Data Typing 24
3.6 Unit Typing 24

4 Conclusion **25**
4.1 Evaluation of OWL/Protégé 25
4.2 Future Application 25

Appendix A: `Genpol` Properties **27**
Appendix B: `Wizpol` Properties **29**
References **30**

Table of Figures

Figure 1.1 OWL Ontology Stack 3

Figure 2.1 Top-Level `genpol` Structural Overview 5
Figure 2.2 Policy Document Property Structure 5
Figure 2.3 Policy Structure 6
Figure 2.4 Policy Rule 6
Figure 2.5 TriggerEvent Relationship with TriggerArgument 6
Figure 2.6 Condition Relationship with Related Classes 7
Figure 2.7 Action Relationship with ActionArgument 7
Figure 2.8 Argument Hierarchy 7
Figure 2.9 Policy Attribute Top-Level Hierarchy 7
Figure 2.10 Policy Attribute Required Attributes 8
Figure 2.11 Policy Attribute Optional Attributes 8
Figure 2.12 Policy Attribute: Policy Preference Optional Attribute 8
Figure 2.13 Possible Policy Variable Attributes 9
Figure 2.14 Condition Operator Hierarchy 9
Figure 2.15 Action Combination Operator Hierarchy 10
Figure 2.16 Trigger Combination Operator Hierarchy 10
Figure 2.17 Condition Combination Operator Hierarchy 10
Figure 2.18 Policy Rule Combination Operator Hierarchy 11

Figure 3.1 `Wizpol` Subclass Extension to `genpol:TriggerEvent` 12
Figure 3.2 `Wizpol` SubclassExtension to `genpol:ConditionParameter` 13
Figure 3.3 `Wizpol` Subclass Extension to `genpol:Action` 13
Figure 3.4 Class Categorisation Top-Level `wizpol` Hierarchy 14
Figure 3.5 Defined User-Level Categories 14
Figure 3.6 Operator Extension for User-Level Association 15
Figure 3.7 Admin Level Condition Operators 16
Figure 3.8 Admin Level Condition Combination Operators 16
Figure 3.9 Admin Level `PolicyRule` Combination Operators 16
Figure 3.10 Admin Level Trigger Combination Operators 17
Figure 3.11 Admin Level Action Combination Operators 17
Figure 3.12 Expert Level Condition Operators 18
Figure 3.13 Expert Level Trigger Combination Operators 18
Figure 3.14 Expert Level Action Combination Operators 18
Figure 3.15 Expert Level `PolicyRule` Combination Operators 19
Figure 3.16 Expert Level Condition Combination Operators 19
Figure 3.17 Intermediate Level Condition Operators 20
Figure 3.18 Intermediate Level Condition Combination Operators 20
Figure 3.19 Intermediate Level Trigger Combination Operators 21
Figure 3.20 Intermediate Level `PolicyRule` Combination Operators 21
Figure 3.21 Novice Level Condition Operators 22
Figure 3.22 Novice Level Trigger Combination Operators 22
Figure 3.23 Novice Level Condition Combination Operators 23
Figure 3.24 Novice Level `PolicyRule` Combination Operators 23
Figure 3.25 Status Variable Class Structure 23
Figure 3.26 Top-Level DataType Hierarchical Structure 24
Figure 3.27 String and Boolean DataType Options 24
Figure 3.28 UnitType Top-Level Definition 24

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`

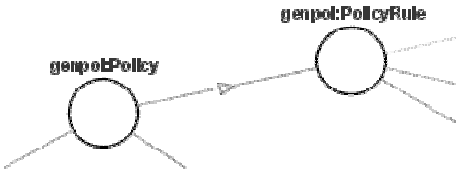
2. Diagram conventions

Diagrams depicted in this report were generated using the Jambalaya plug-in tool [6] and the OWLViz graphical plug-in tool [11] for Protégé-OWL Beta 2.2 (Build 288). 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:

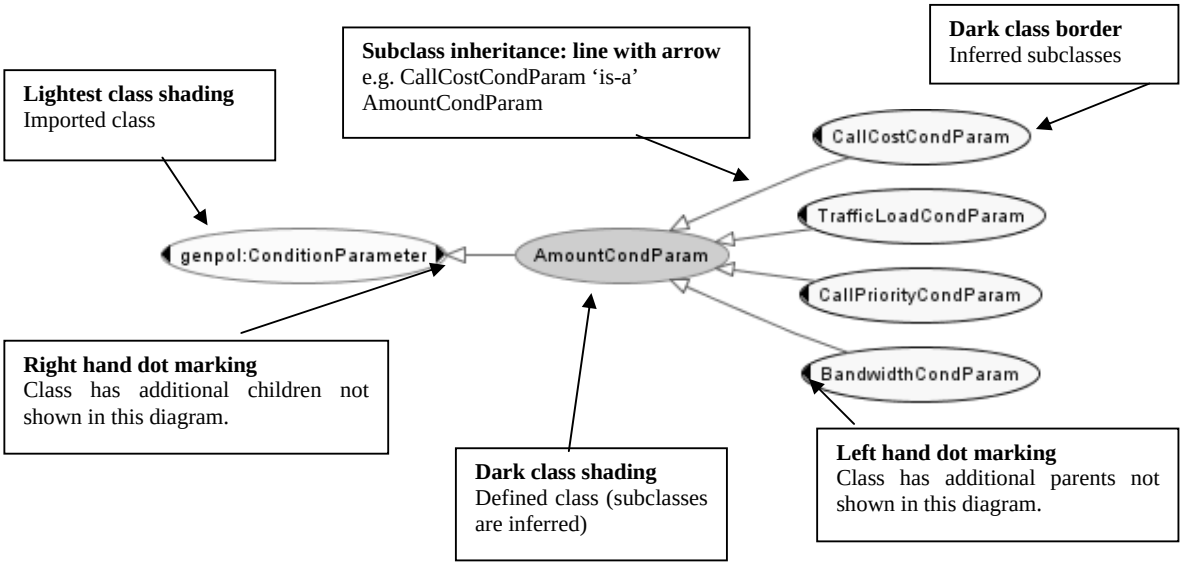


Sub-class (inheritance) is shown by a solid straight line without a triangle.

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.



3. Report conventions

Throughout this report, ontology class, property and OWL file names are formatted using `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 [5]. Typically, defined ontology classes are those whose subclasses are intended to be purely inferred.

1 Overview

Ontologies can be used to describe a particular area of knowledge, including the key terms, their semantic interconnections and certain rules of inference. Once defined, an ontology has several major benefits when utilised by software applications or agents in a variety of contexts. Using an ontology, a common understanding of the structure of information within a domain may be shared between applications. Another major 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.

Section 1.1 provides an introduction to APPEL – the policy description language used by the ACCENT system and modelled in the ontologies described in this report. Section 1.2 outlines the motivation for using ontology, while Section 1.3 provides an overview of the language and tools chosen for ontology development. Section 1.4 describes the various ontologies which were created and explained in this report.

1.1 APPEL Policy Description Language

A comprehensive policy description language called APPEL (the ACCENT Project Policy Environment/Language [16]) 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. As a brief overview, this includes the definition of a Policy Document which may contain zero or more Policy definitions which, in turn, may contain zero or more Policy Rule definitions. A Policy Rule may contain zero or more Triggers, Conditions and at least one Action. Further to these main components, the language outlines various policy attributes and definitions of variables, together with a range of operators and rules governing how they may be applied to combine various statement blocks. This report describes how these core aspects of APPEL were encapsulated in an ontology.

1.2 Motivation for Ontology Usage

The use of ontology helps generalise the ACCENT policy wizard so it may facilitate user-friendly policy creation for any customised domain. As the APPEL language contains a core 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.

In a wider context, once a domain-specific policy language ontology is produced it may be integrated with the policy wizard. To achieve this, a special integration system known as POPPET (Policy Ontology Parser Program Extensible Translation) was developed to access, parse and process ontology data and offer an interface through which the policy wizard can query such data. Subsequent to the creation of a suitable ontology to model the policy language, the original policy wizard was re-engineered to remove hard-coded domain details and instead integrate it with the POPPET system. A

technical description of the POPPET system and how it is used to integrate OWL ontologies with ACCENT is presented in the technical report 'An Overview of Ontology Application for Policy-based Management using POPPET' [2].

1.3 OWL/Protégé Overview

A variety of specialised languages exist to define ontologies. OWL (The Web Ontology Language [8]) 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 [14]) 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 [10]. 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 [19]. To check the ontologies described in this report, point the validator to the relevant ontology URL as specified in [4] or [18].

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 [9] 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 [12] 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 [13]. Diagrams were generated with the aid of the OWLViz [11] and Jambalaya [6] 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 particular domain application. Two ontologies were developed using OWL. The first defines the core constructs of APPEL, and the second extends this to specify common features employed to manipulate this for user-interface display.

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 within a domain-specific ontology.

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. The `genpol` ontology document may be accessed at [4] and the `wizpol` ontology document accessed at [18].

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. Extending ontologies in this way has resulted in an ontology ‘stack’ or layered model, on top of which any domain-specific ontology may be built and easily integrated with the ACCENT policy system, as shown in Figure 1.1.

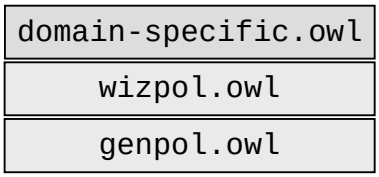


Figure 1.1 OWL Ontology Stack

These ontologies define only the structure of policy-related knowledge and not actual policy data. For this reason, `genpol` and `wizpol` contain no 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. Once included, an ontology may extend the class hierarchy of the imported ontology structure to define additional sub-classes and properties together with applicable constraints. In particular, this includes the definition of specific trigger events, condition parameters and actions associated with the domain in question. The implemented domain-specific ontology for call control is described within the technical report 'Ontology for Call Control' [3].

This report describes the structure of the base ontologies `genpol` and `wizpol`. Through a series of graphical representations, ontology class and property hierarchies are explained together with a view of how `genpol` was extended to create `wizpol`.

2 The Generic Policy Language Ontology: genpol.owl

The generic policy language ontology, referred to hereafter by the acronym genpol, describes the key components of the APPEL policy description language [16] implemented in the ACCENT policy system [1]. Contained within this ontology is a definition of key language terms and how they relate to one another. This includes the concept of a "policy document" and its various constituent parts – including policy rules, trigger events, conditions, actions and additional attributes, variables and operators. Relationships between such concepts are defined by way of a specified property or traditional inheritance.

2.1 Overview of genpol

The top level structural overview of genpol in terms of defined named property restrictions is shown in Figure 2.1. Classes shown are related via named specific properties and not by any notion of inheritance or hierarchical structure.

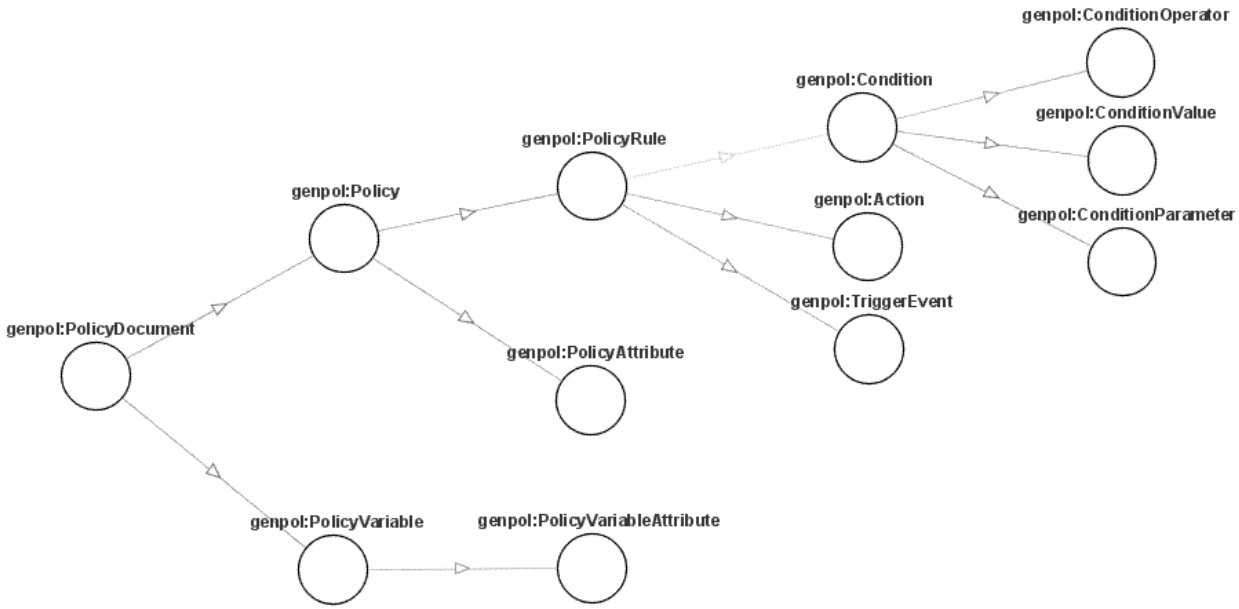


Figure 2.1 Top-Level genpol Structural Overview

2.2 Policy Document

A PolicyDocument is the highest conceptual level component of the APPEL policy description language. It is defined to have zero or more Policy instances and may have zero or more associations with the PolicyVariable component as shown in Figure 2.2.

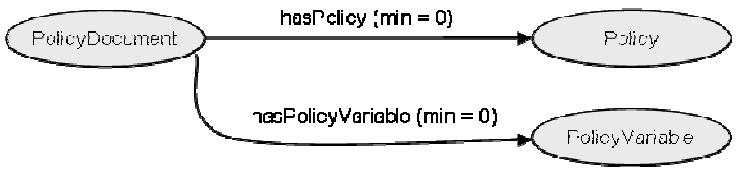


Figure 2.2 Policy Document Property Structure

2.3 Policy

A Policy is defined to have at least one `PolicyRule` and must have `RequiredAttribute` instances. It may also have any number of `OptionalAttribute` instances. Figure 2.3 demonstrates these properties in `genpol`.

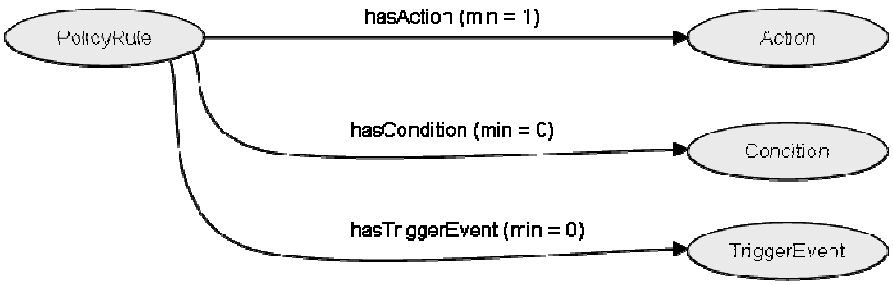


Figure 2.3 Policy Structure

2.4 Policy Rule, Trigger, Condition and Action.

A `PolicyRule` may have zero or more `TriggerEvent` or `Condition` associations, but must have at least one `Action` as shown in Figure 2.4.

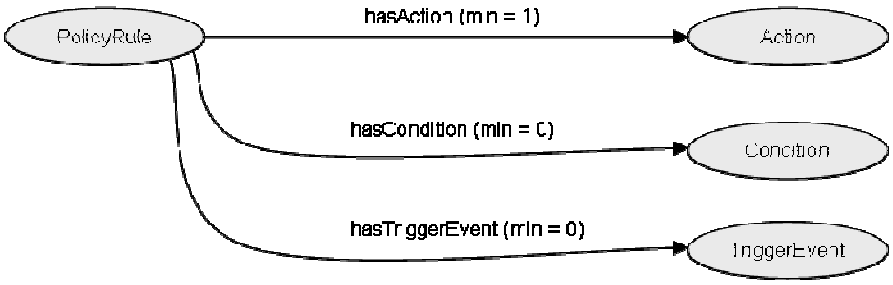


Figure 2.4 Policy Rule

A `TriggerEvent` may be linked with a `TriggerArgument` along the `hasTriggerArgument` property restriction as shown in Figure 2.5. For the purposes of the ontology no cardinality restriction has been placed on this relationship. There may be triggers which do not have any arguments.

Figure 2.5 TriggerEvent Relationship with TriggerArgument

A Condition must be associated with a single `ConditionParameter`, `ConditionOperator` and `ConditionValue as` depicted in Figure 2.6. This is defined using the properties `hasConditionParameter`, `hasConditionOperator` and `hasConditionValue` combined with a set of associated cardinality restrictions.

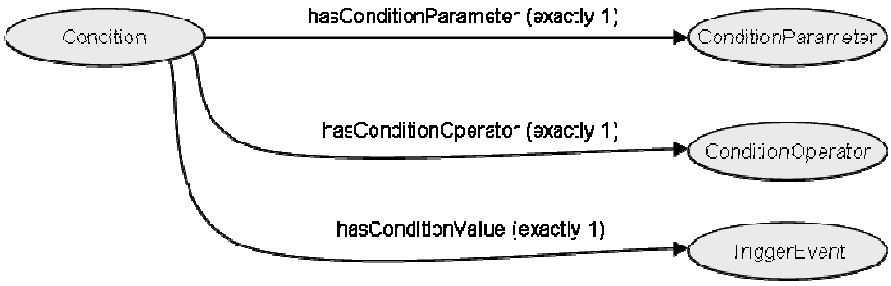


Figure 2.6 Condition Relationship with Related Classes

An `Action` may be linked with an `ActionArgument` along the `hasActionArgument` property restriction as shown in Figure 2.7. No cardinality restriction has been placed on this relationship within the ontology as there may be actions which do not have any arguments.

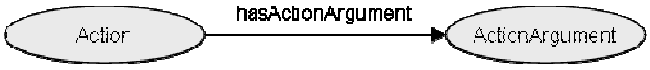


Figure 2.7 Action Relationship with ActionArgument

2.5 Arguments

An `Argument` may either be a `TriggerArgument` or an `ActionArgument` as shown in Figure 2.8. An `Argument` represents particular parameter values associated with a trigger or action in a policy. `Genpol` defines this top level class structure which a domain-specific ontology may extend to define its own named arguments.

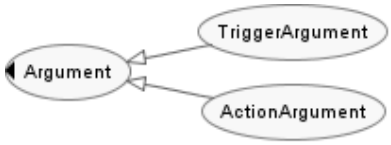


Figure 2.8 Argument Hierarchy

2.6 Policy Attribute

A `Policy` has a number of attributes. A `PolicyAttribute` may be either required or optional as shown in Figure 2.9. APPEL defines a number of required attributes as shown in Figure 2.10, all of which exist within every `Policy`. Optional attributes can be seen in Figure 2.11. These attributes exist for each policy but may not have values. In particular, Figure 2.12 lists the restricted set of preference choices permitted for the `PolicyPreference` attribute.

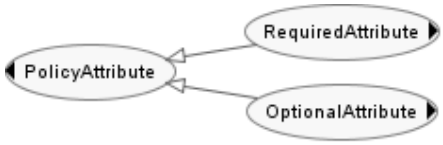


Figure 2.9 Policy Attribute Top-Level Hierarchy

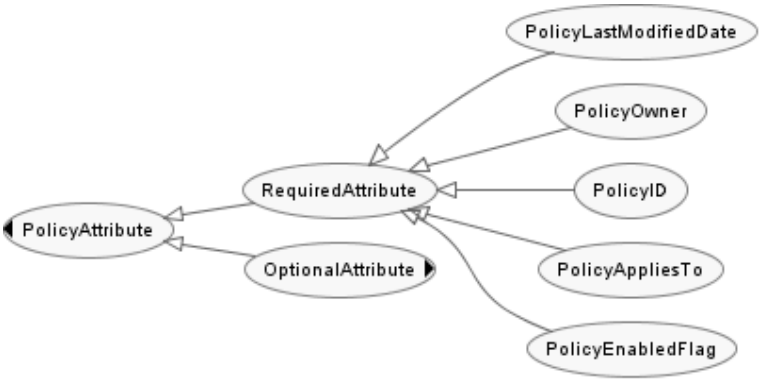


Figure 2.10 Policy Attribute Required Attributes

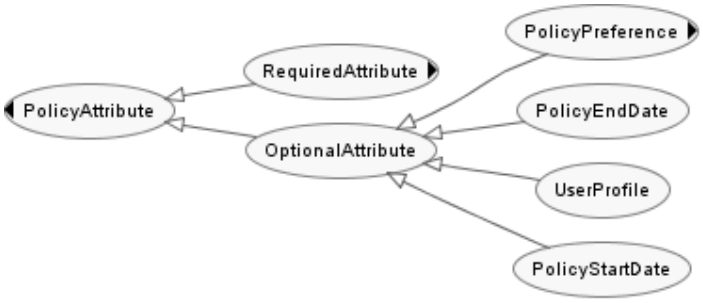


Figure 2.11 Policy Attribute Optional Attributes

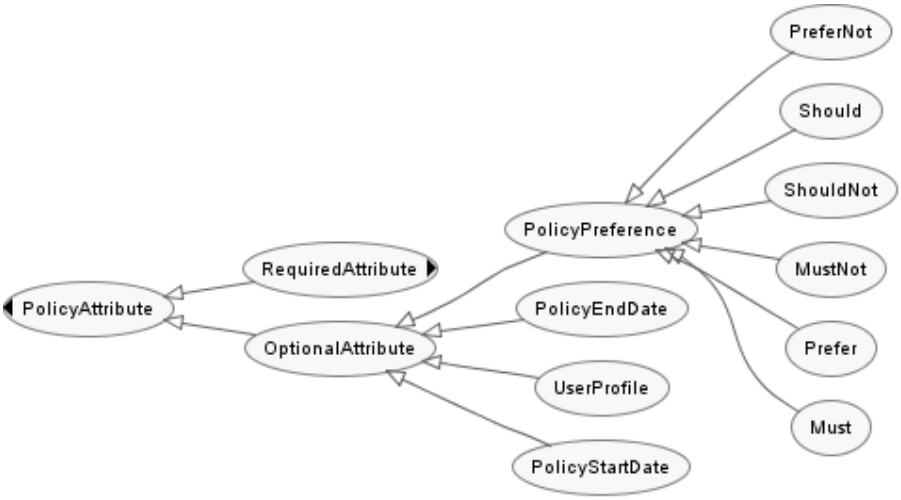


Figure 2.12 Policy Attribute: Policy Preference Optional Attribute

2.7 Policy Variable Attribute

A `PolicyVariable` may have a number of attributes associated with it. The list of specific `PolicyVariableAttribute` types is detailed in Figure 2.13.

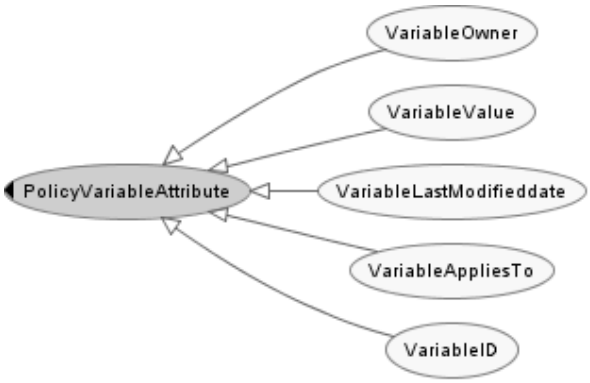


Figure 2.13 Possible Policy Variable Attributes

2.8 Operators

There are two types of operators in a policy: a `ConditionOperator` used within a `Condition` and a `CombinationOperator` used to integrate two policy rules.

2.8.1 Condition Operators

Named operators applicable within a `Condition` component of a `PolicyRule` are outlined in Figure 2.14.

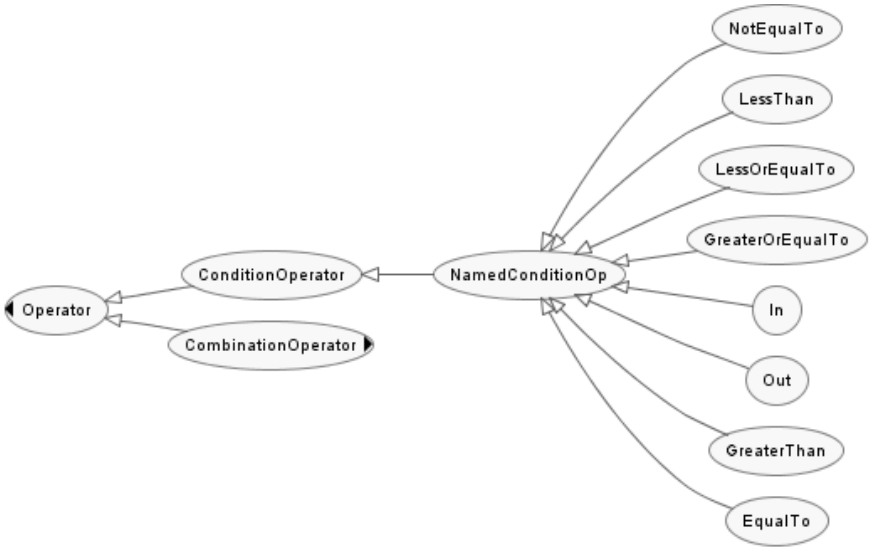


Figure 2.14 Condition Operator Hierarchy

2.8.2 Combination Operators

Different combination operators apply to actions, conditions and trigger nodes as well as to policy rules themselves. Each is outlined in Figure 2.15, Figure 2.16, Figure 2.7 and Figure 2.18.

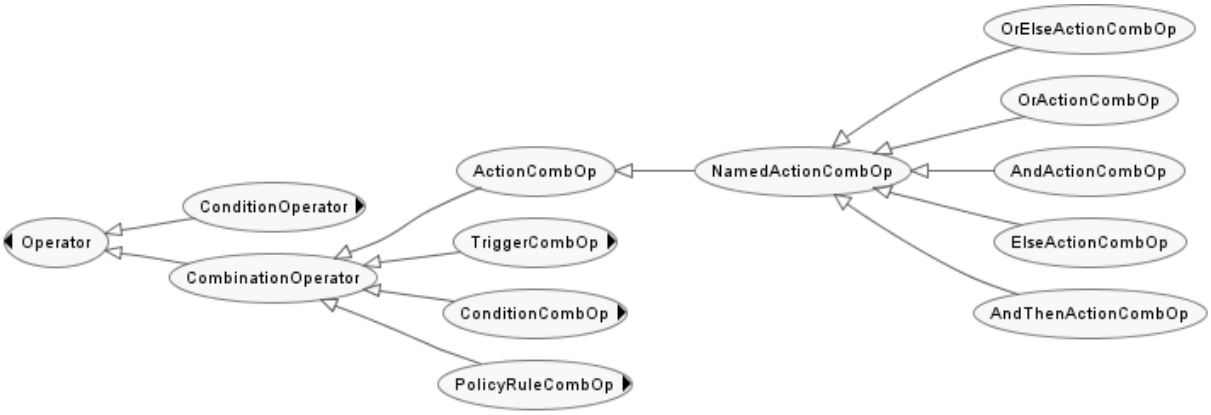


Figure 2.15 Action Combination Operator Hierarchy

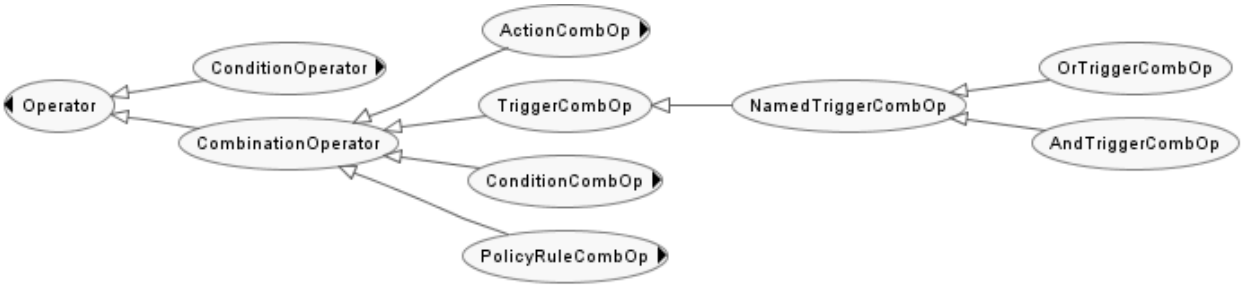


Figure 2.16 Trigger Combination Operator Hierarchy

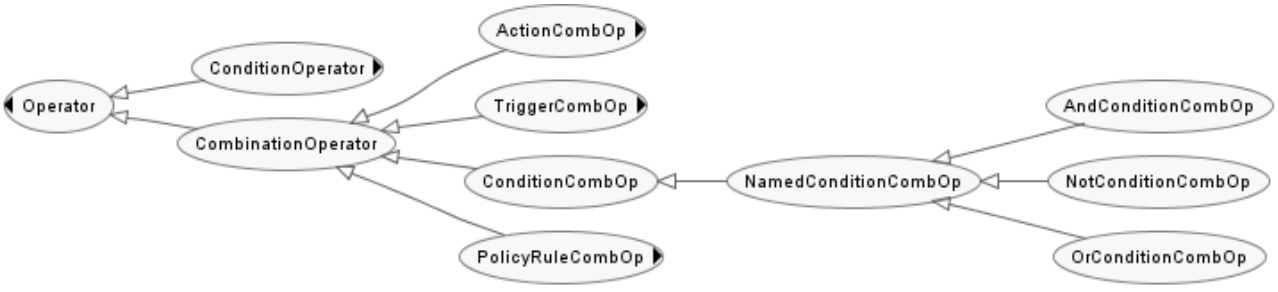


Figure 2.17 Condition Combination Operator Hierarchy

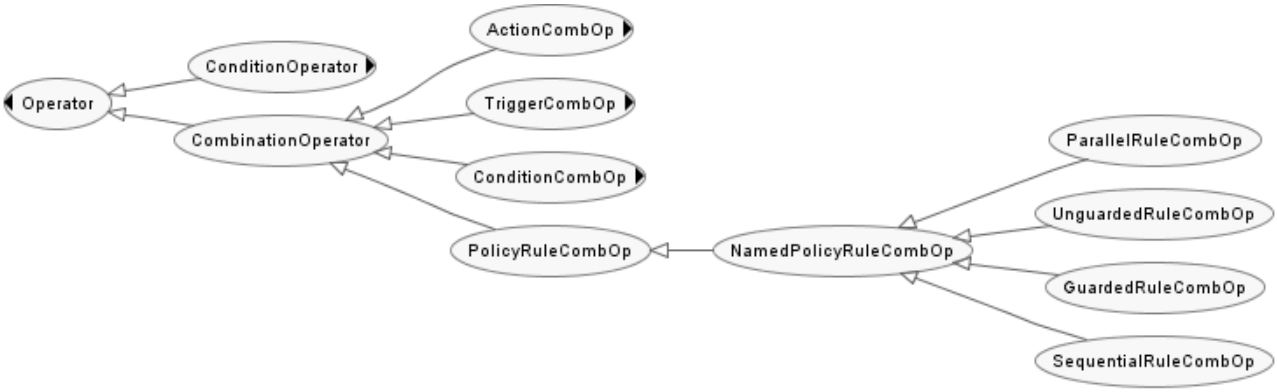


Figure 2.18 Policy Rule Combination Operator Hierarchy

3 Wizard policy language ontology: `wizpol.owl`

The ACCENT Policy Wizard [17] provides a user-friendly means of creating and editing policies. Many features of this interface are core to handling and displaying policy information. While distinct from the policy description language aspects, they are required for any domain-specific implementation of the policy system. The wizard policy language ontology (referred to by the acronym `wizpol`) was developed as a means of extending the description of the policy description language (`genpol`) to define common information structures specific to the policy system user interface.

In particular, `wizpol` expands the class hierarchy of `genpol` classes `TriggerEvent`, `ConditionParameter` and `Action`. It also provides a range of wizard-specific properties (restrictions) including user-levels, categorisation and internalisation, which are used to categorise triggers, conditions and actions in a domain. The ontology also defines additional class structures used to specify wizard-related information in the form of data typing and unit typing. A detailed explanation of `wizpol` ontology structure is explained in the following subsections.

3.1 Trigger, Condition and Action Class Wizard Extension

The `TriggerEvent`, `ConditionParameter` and `Action` structure of `genpol` has been extended to include a 'Named' class which represents the top level through which domain-specific triggers, conditions or actions may be defined as subclasses. For example, actual domain-specific trigger classes are defined as subclasses of `NamedTriggerEvent`.

In addition, `wizpol` defines five extra subclasses in the class hierarchies of `genpol:TriggerEvent`, `genpol:ConditionParameter` and `genpol:Action`. Four of the additional classes represent user level categorisations ("admin", "expert", "intermediate", and "novice") and one signifies "internal use". The subclasses of each are inferred by placing `hasUserLevel` or `hasInternalUse` property restrictions on domain-specific triggers, conditions and actions.

The extended class hierarchies of `TriggerEvent`, `ConditionParameter` and `Action` are shown in Figure 3.1, Figure 3.2 and Figure 3.3 respectively. In addition, as a `TriggerEvent` is not a compulsory element within a Policy (a policy may contain zero or more triggers), `wizpol` defines a class `EmptyTriggerEvent` to represent such a scenario, shown in Figure 3.1.

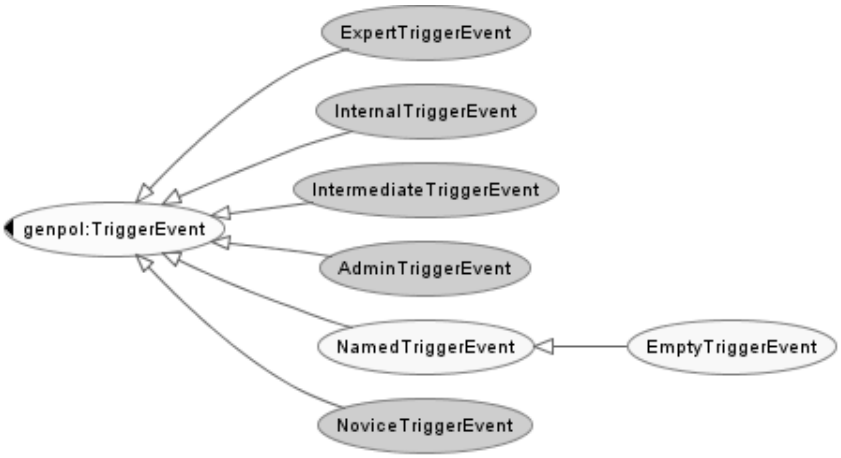


Figure 3.1 `Wizpol` Subclass Extension to `genpol:TriggerEvent`

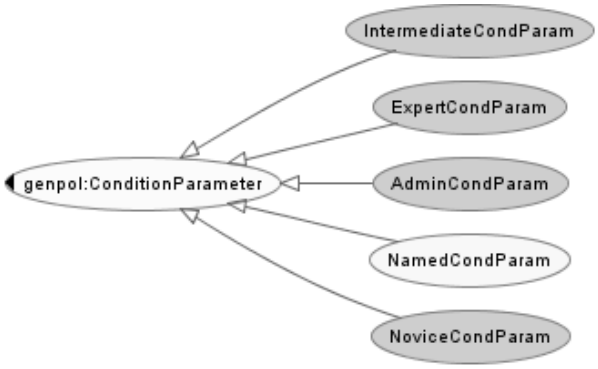


Figure 3.2 `Wizpol` SubclassExtension to `genpol:ConditionParameter`

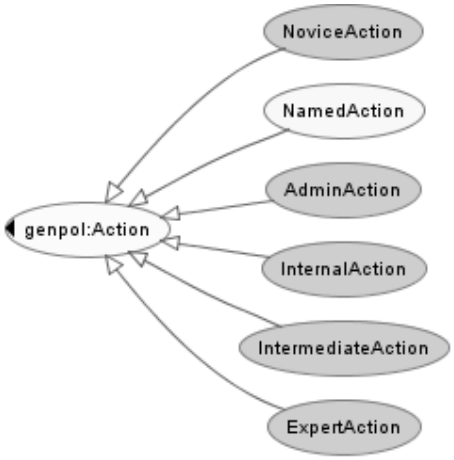


Figure 3.3 `Wizpol` Subclass Extension to `genpol:Action`

3.2 Class Categorisation

A crucial feature of the policy wizard is its ability to categorise or group related triggers, conditions, actions and operators in a domain for processing and display purposes. Groupings, such as "user-level" grouping or implying some action or trigger "has internal use" within the policy system, all require some form of class categorisation.

There are three categorisation types defined in `wizpol`, as shown in Figure 3.4. The top class in this structure is `ClassCategorisation`. Defined subclasses of this class represent the `UserLevelValue`, `InternalUse`, and three top-level categories through which domain-specific trigger, condition parameter and action categories can be specified. These are explained in turn within the following subsections.

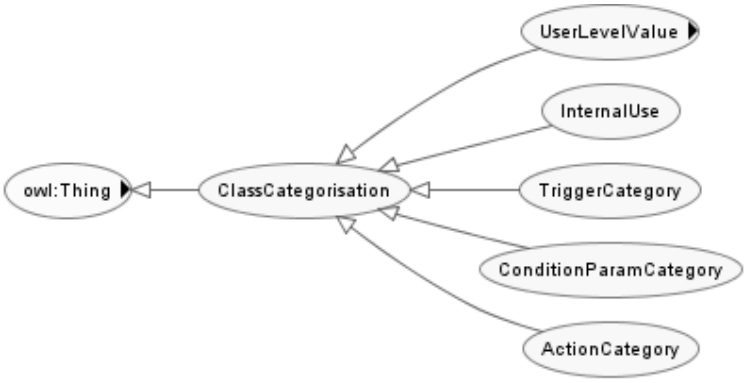


Figure 3.4 Class Categorisation Top-Level `wizpol` Hierarchy

3.2.1 User-Level Categorisation

The policy system interface supports a four-level classification of its users, offering varying degrees of functionality depending on the expertise of a user. In particular, each user level corresponds to a specific subset of triggers, condition parameters and actions permitted for display and selection. The top level is 'Admin' which permits the full range of options, while the remaining levels of "Expert", "Intermediate" and "Novice" may either retain or reduce this range respectively. For example, in the call control domain [3], an Admin and Expert user have equivalent option ranges, with an Intermediate user utilising a subset and a Novice user condensing this set further still. Additional details regarding user-level categorisation can be found in the ACCENT wizard technical report [17].

Each user level is defined as a subclass of `UserLevelValue` as shown in Figure 3.5. Specific trigger, action, condition and operator subclasses may be associated with one or more user levels using the property restriction `hasUserLevel`.

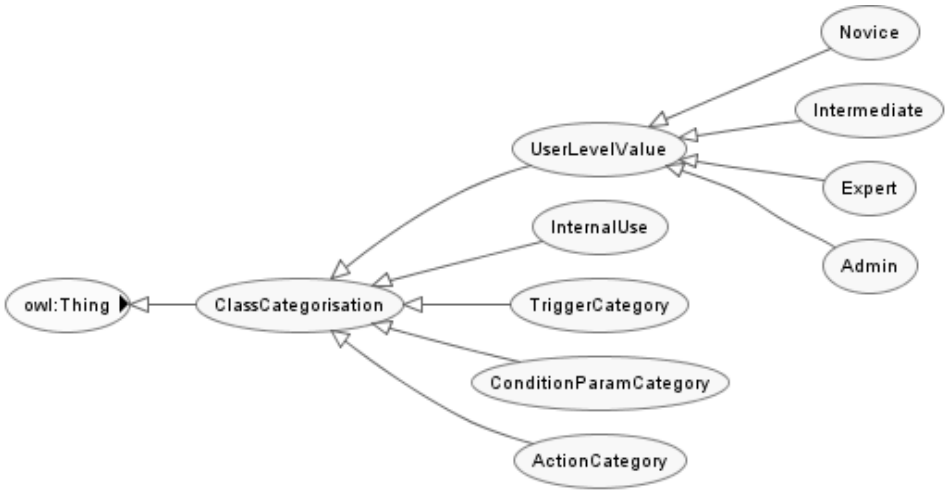


Figure 3.5 Defined User-Level Categories

3.2.2 Internal Use Categorisation

Within a domain, a trigger, condition or action may be defined which accesses or modifies a variable stored locally in the policy system. Such instances can be classified as having "Internal usage". `Wizpol` provides an `InternalUse` class and the property `hasInternalUse`. These can be used together to restrict classes deemed as internal.

3.2.3 Trigger, Condition Parameter and Action Categorisation

Rather than displaying each trigger, condition and action option set as large, continuous lists, the wizard assembles related classes into categories, which are presented to the user as shorter sub-lists. This categorisation is useful not only for display purposes, but also for grouping options with similar properties, such as by number of parameter arguments or by related parameter data types. The `wizpol` ontology defines top-level categories of `ActionCategory`, `TriggerCategory` and `ConditionParamCategory` as shown in the diagram of Figure 3.5. In a domain-specific ontology, named categories are defined as subclasses of these.

3.3 Operator Extension (User-Level Provision)

The policy system predefines associations between user levels and both condition parameter and policy rule combination operators. The `wizpol` ontology uses the `UserLevel` categorisation to place restrictions on the condition and combination operators defined in `genpol` as a means of associating each operator with permitted user levels. The top-level operator extension is outlined in Figure 3.6. Specific operator subsets associated with each user level category are described in the following subsections.

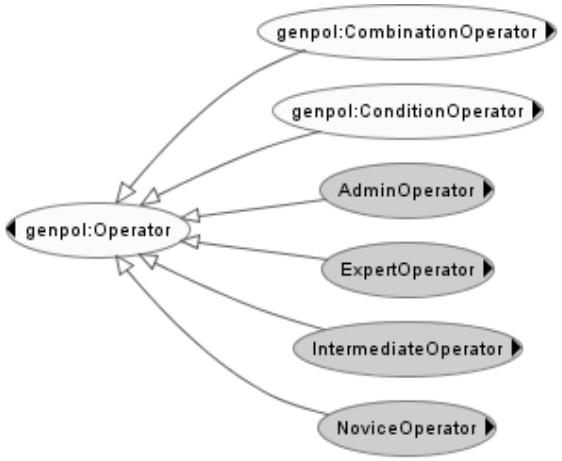


Figure 3.6 Operator Extension for User-Level Association

3.3.1 Admin Level Operators

Admin level operators are shown in Figure 3.7, Figure 3.8, Figure 3.9, Figure 3.10 and Figure 3.11.

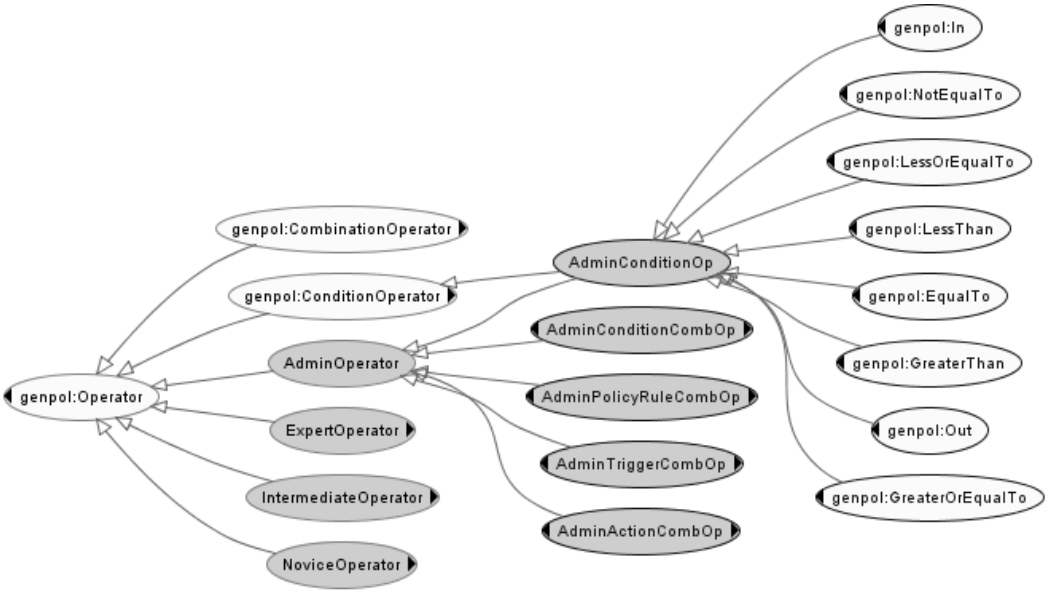


Figure 3.7 Admin Level Condition Operators

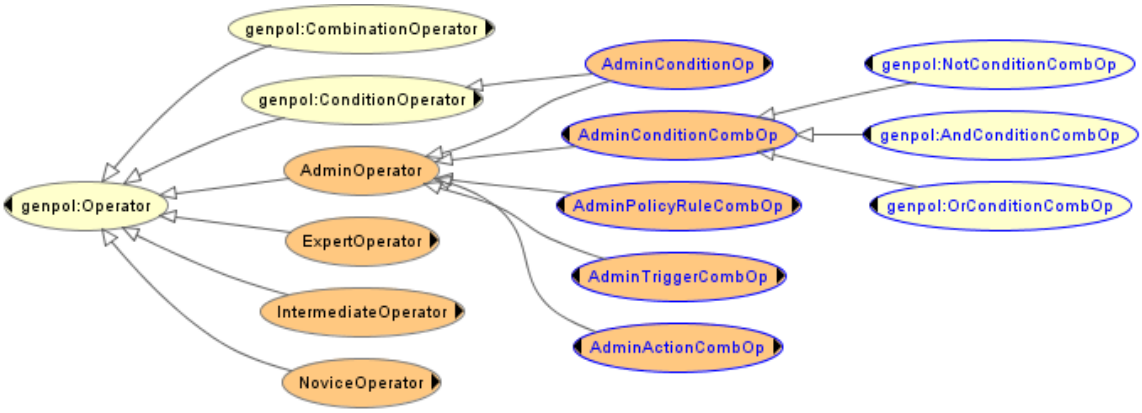


Figure 3.8 Admin Level Condition Combination Operators

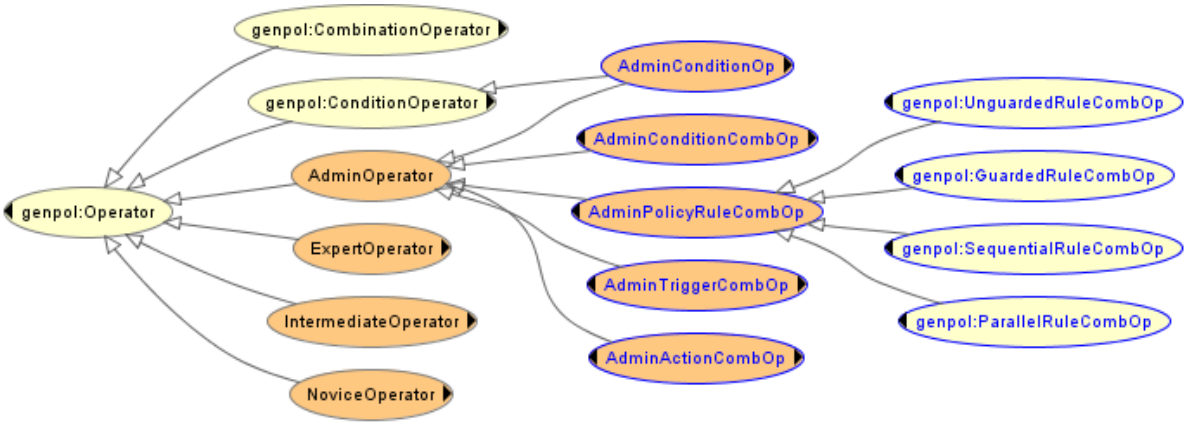


Figure 3.9 Admin Level `PolicyRule` Combination Operators

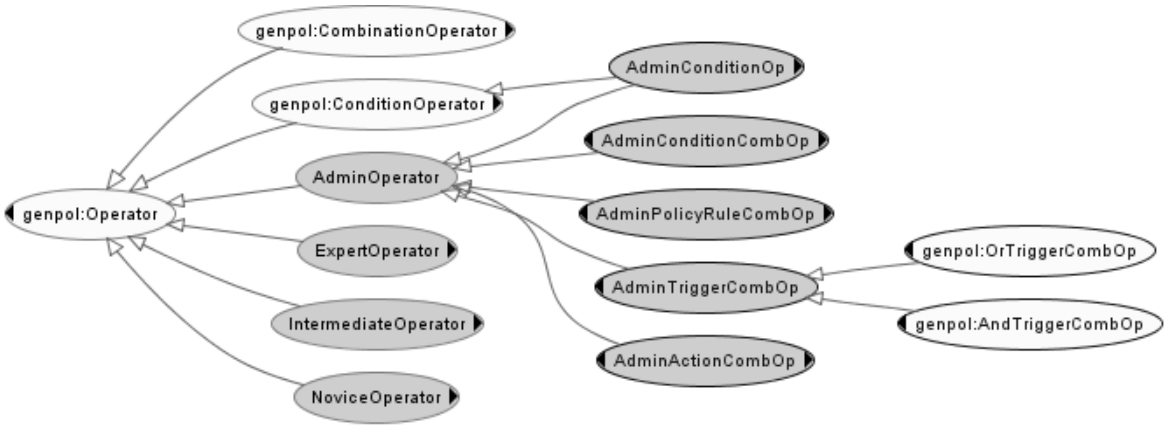


Figure 3.10 Admin Level Trigger Combination Operators

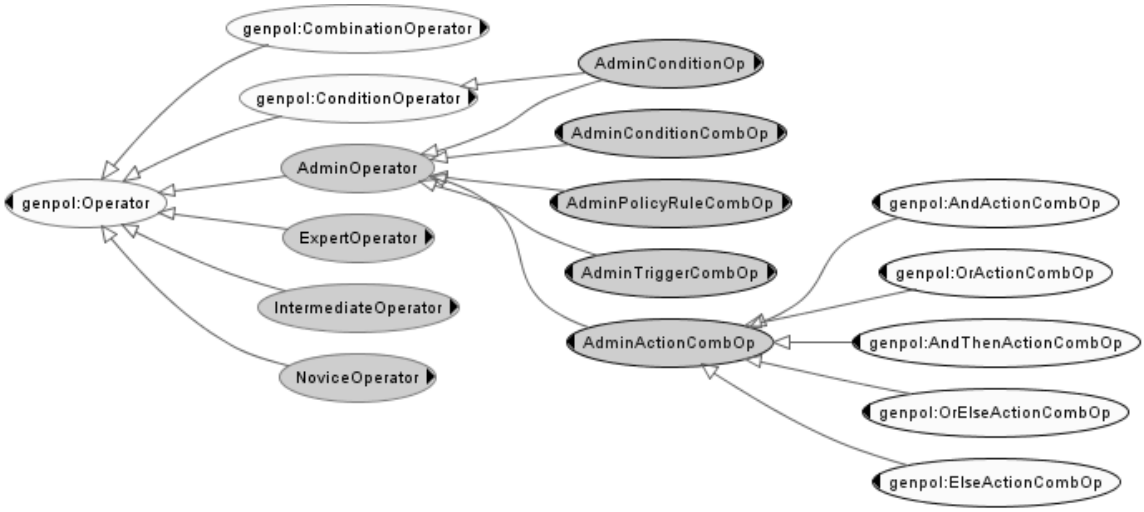


Figure 3.11 Admin Level Action Combination Operators

3.3.2 Expert Level Operators

Expert level operators are listed in Figure 3.12, Figure 3.13, Figure 3.14, Figure 3.15 and Figure 3.16.

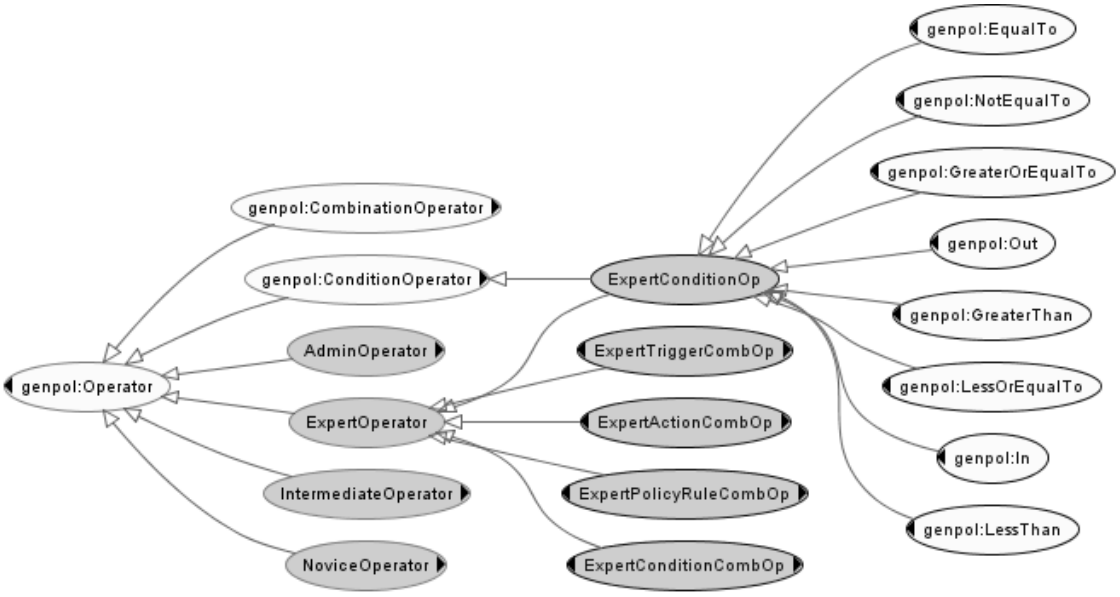


Figure 3.12 Expert Level Condition Operators

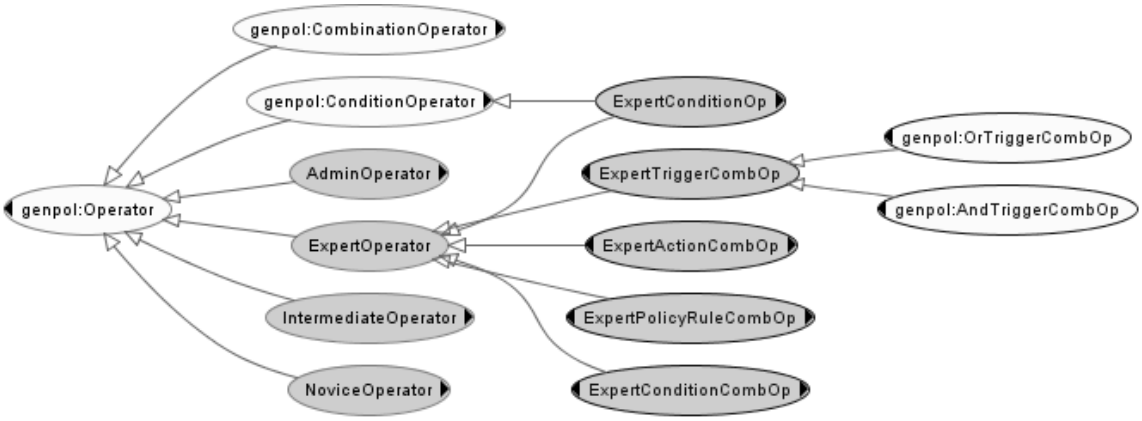


Figure 3.13 Expert Level Trigger Combination Operators

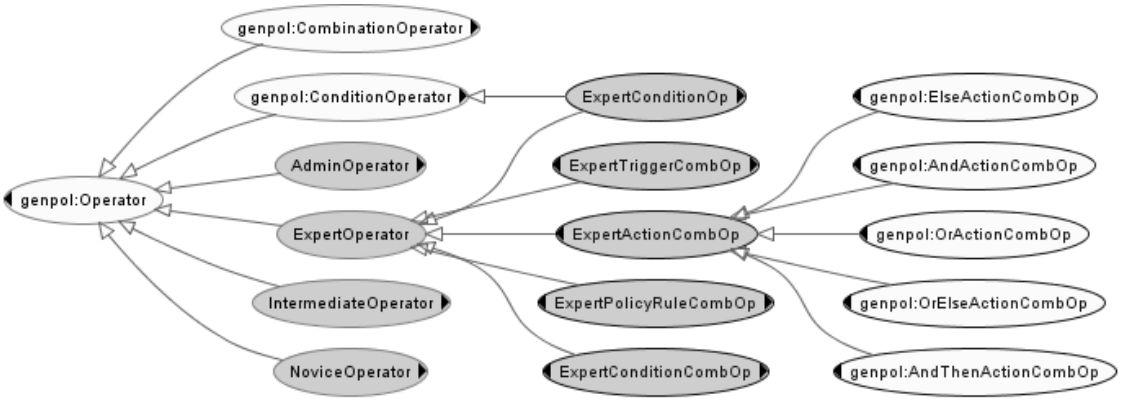


Figure 3.14 Expert Level Action Combination Operators

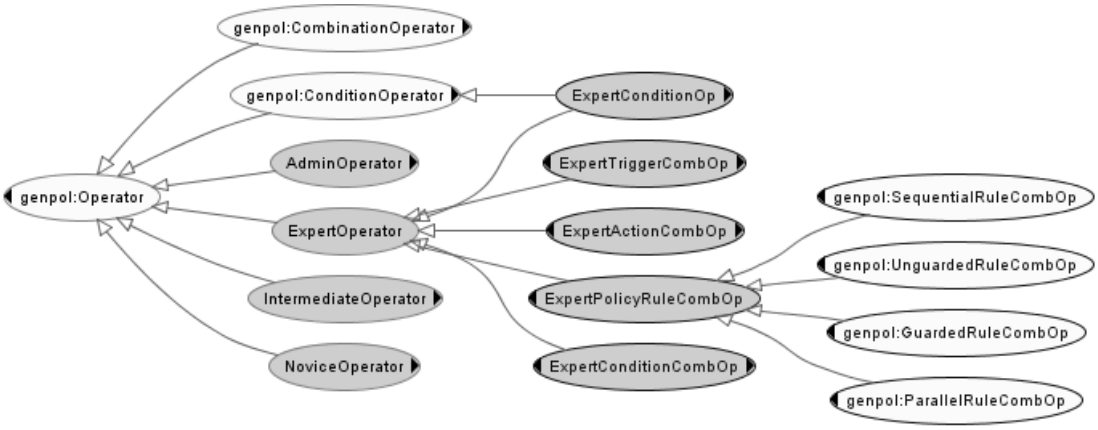


Figure 3.15 Expert Level `PolicyRule` Combination Operators

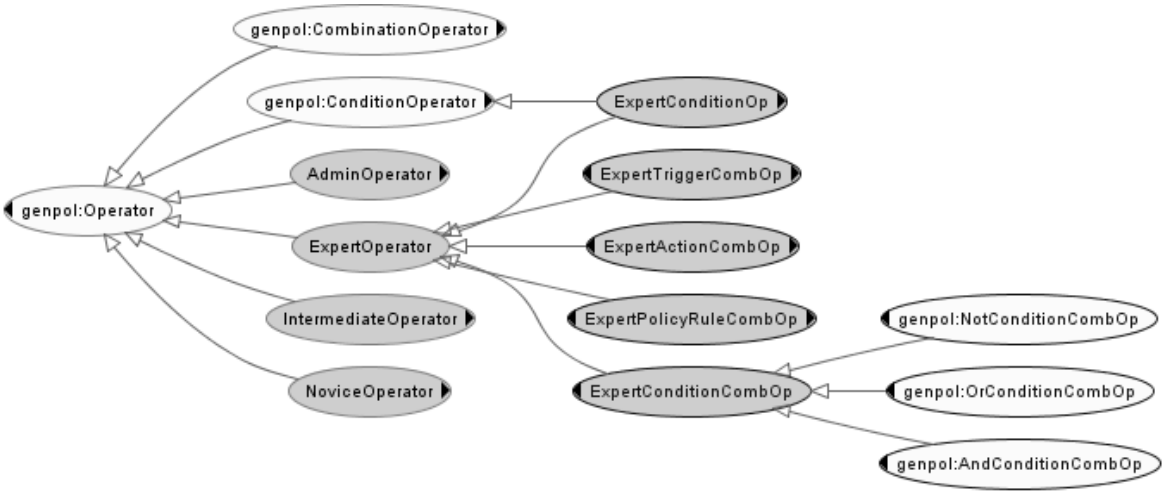


Figure 3.16 Expert Level Condition Combination Operators

3.3.3 Intermediate Level Operators

Intermediate level operators are defined in Figure 3.17, Figure 3.18, Figure 3.19 and Figure 3.20. Note that at intermediate level there are no defined Action combinator operators. Intermediate (and in turn Novice) users are not permitted to define more than one `Action` within a `PolicyRule`.

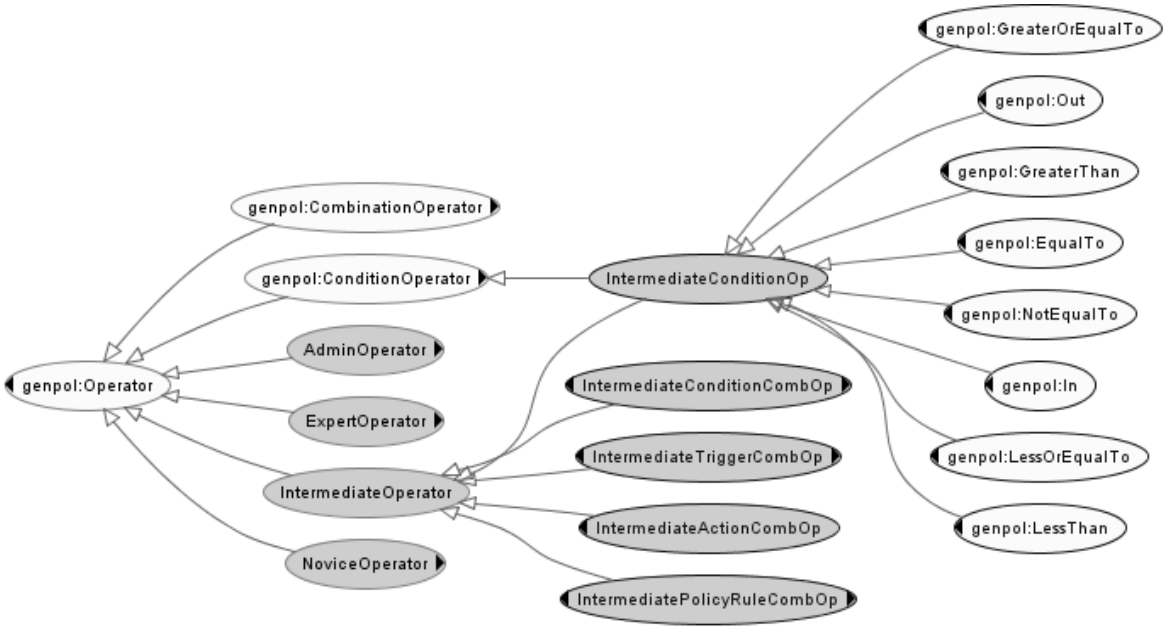


Figure 3.17 Intermediate Level Condition Operators

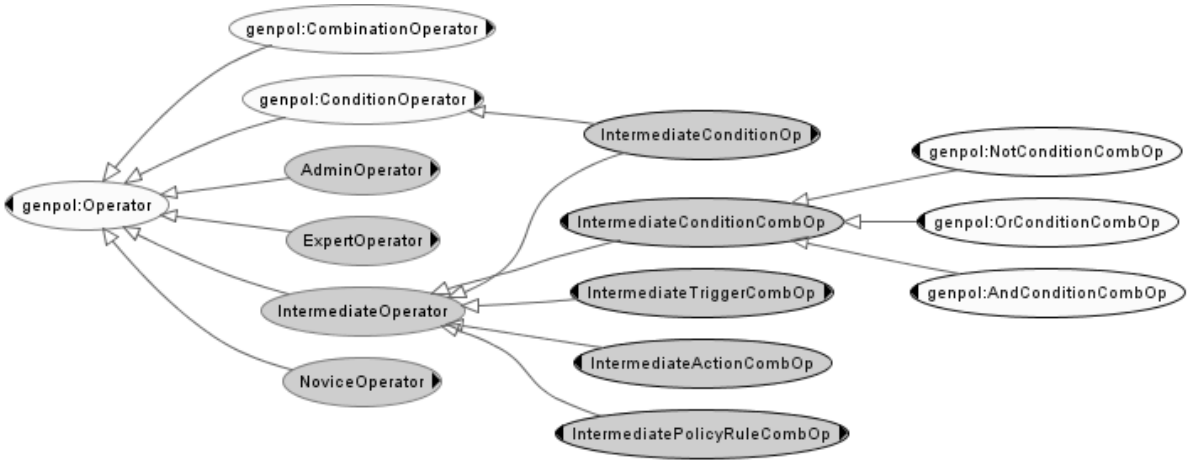


Figure 3.18 Intermediate Level Condition Combination Operators

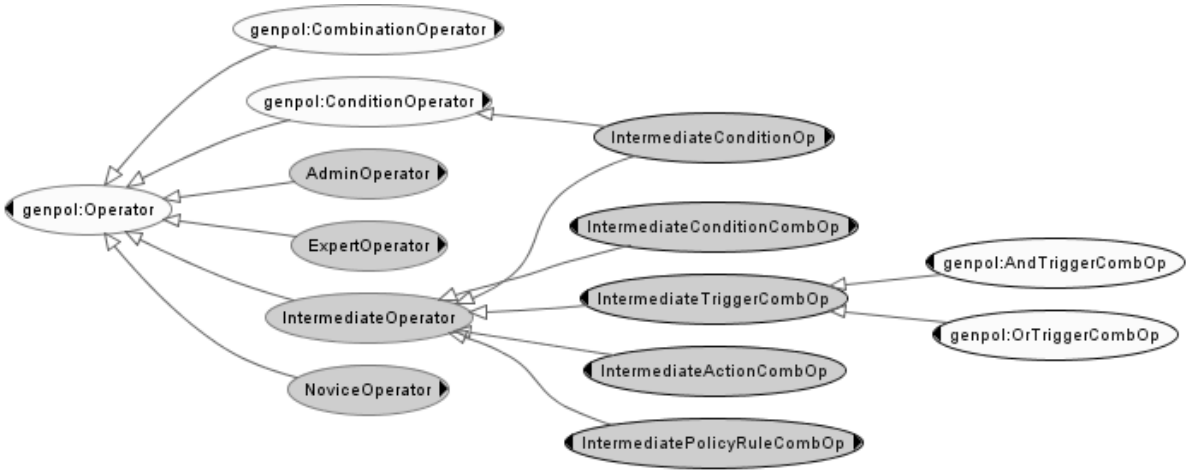


Figure 3.19 Intermediate Level Trigger Combination Operators

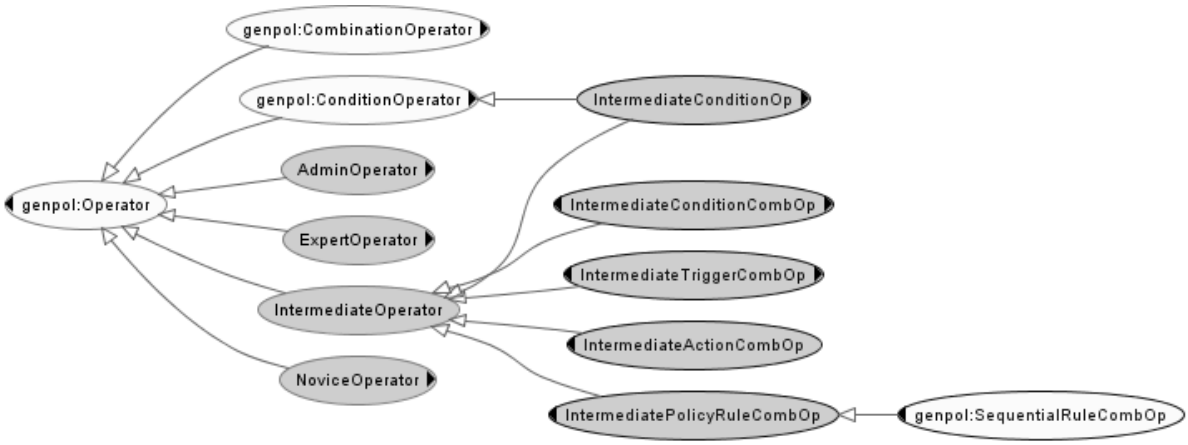


Figure 3.20 Intermediate Level `PolicyRule` Combination Operators

3.3.4 Novice Level Operators

Novice level operators are shown in Figure 3.21, Figure 3.22, Figure 3.23 and Figure 3.24. Note that at novice level there are no defined action combinator operators. Novice users are not permitted to define more than one `Action` within a `PolicyRule`.

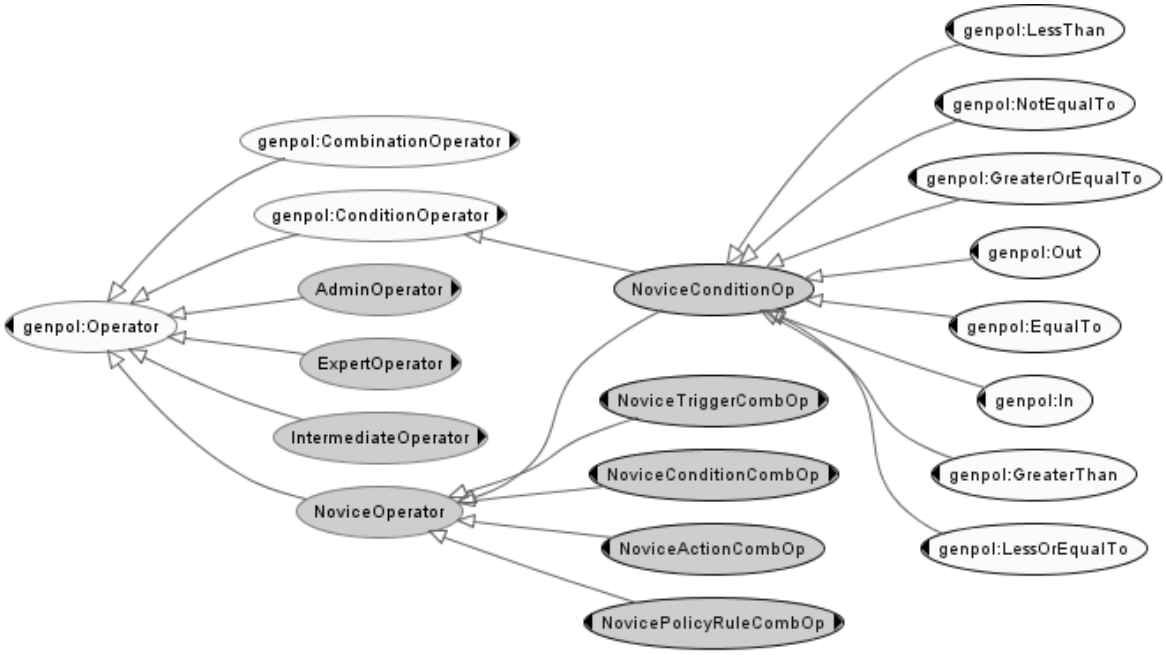


Figure 3.21 Novice Level Condition Operators

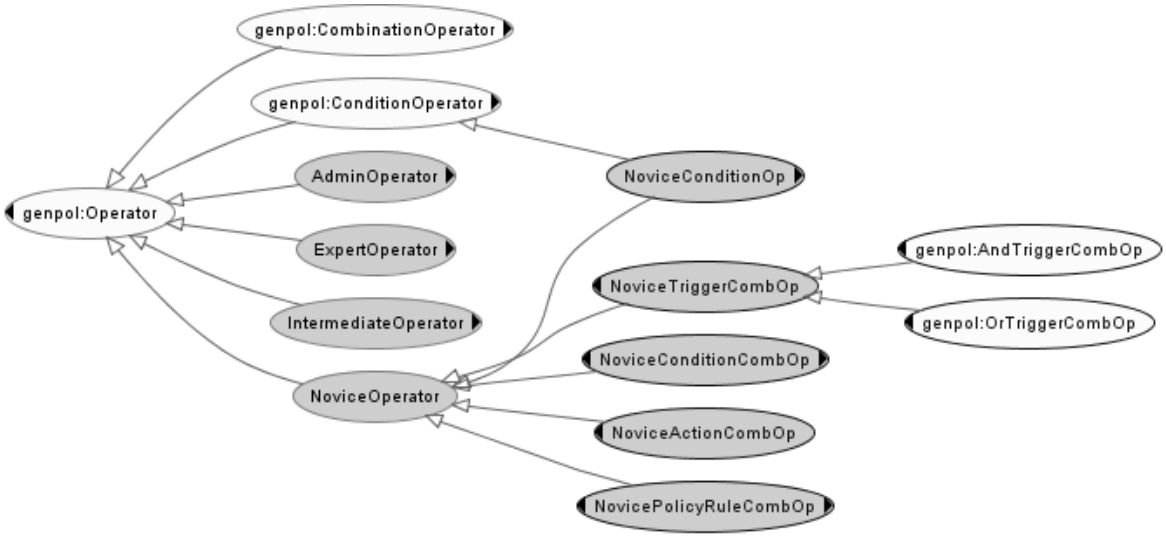


Figure 3.22 Novice Level Trigger Combination Operators

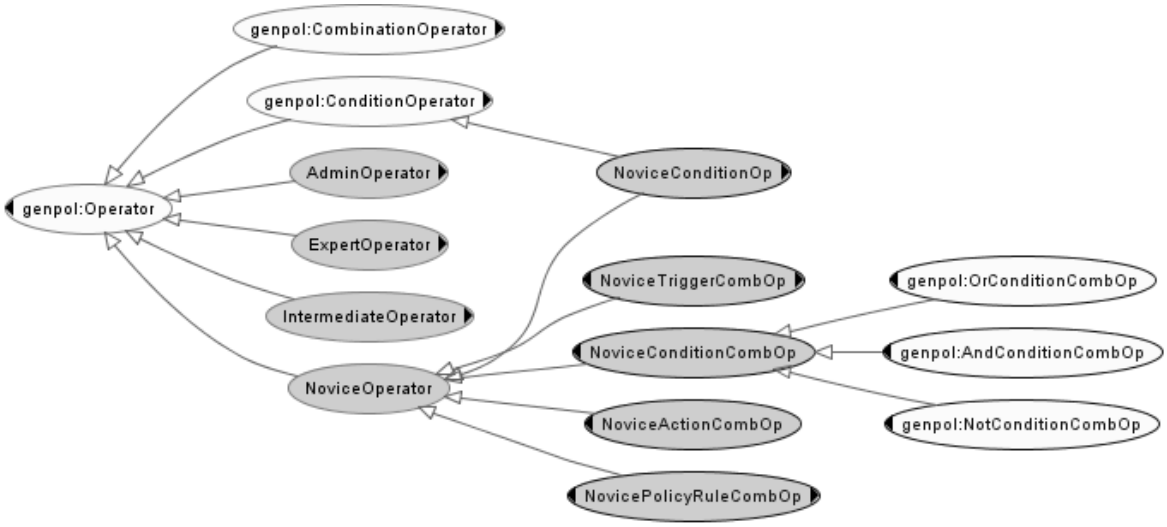


Figure 3.23 Novice Level Condition Combination Operators

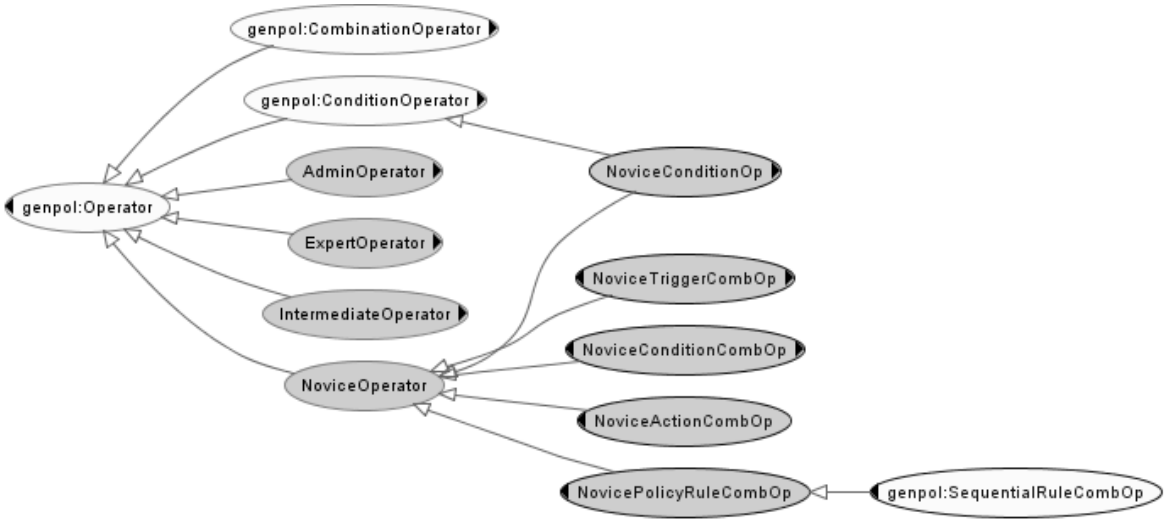


Figure 3.24 Novice Level `PolicyRule` Combination Operators

3.4 Status Variables

In addition to general policy variables a user may define for sole personal use, the policy system holds a more concrete set of variables that describe state information applicable to all system users. These variables are interpreted by the policy wizard in a special way and are represented in `wizpol` under the class `StatusVariable` as shown in Figure 3.25. Specific status variables may be defined as subclasses of this structure in a domain-specific ontology. Note there is a compulsory status variable representing the profile of a user.

Figure 3.25 Status Variable Class Structure

3.5 Data Typing

Action and trigger arguments (parameters) associated with certain `Action` and `TriggerEvent` subclasses may have a specific data type eligible for definition within the ontology. Unfortunately, the current OWL specification is limited in its provision for built in data-type restrictions. Although OWL supports the definition of a data-type property (for example, a property `hasBandwidth`) there is no facility to place specific restrictions on its values (such as restricting `hasBandwidth` to a particular numeric range). Therefore, in the absence of a general framework for customised data types, the ontology simply defines a structure of classes to represent types of data. This solution works as a method of describing extra data type knowledge within the ontology but is extremely general and gives no real semantic control. Should the OWL specification be updated to support customised datatyping, this method would be re-implemented.

The `wizpol` ontology defines the top-level hierarchical structure for the `DataType` option shown in Figure 3.26, listing two initial types of Boolean and String.

Figure 3.26 Top-Level DataType Hierarchical Structure

The `BooleanType` class contains both true and false values. The `StringType` class is defined to contain a default general string option used in the wizard, as shown below in Figure 3.27. Domain-specific ontologies are encouraged to extend this list and to define other data types if necessary, including their own `StringType` options.

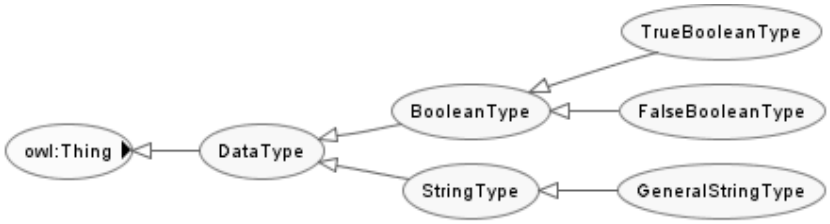


Figure 3.27 String and Boolean DataType Options

3.6 Unit Typing

For display purposes, the policy wizard may output unit annotations to values entered as action or trigger arguments or condition values. For example, in the call control domain, the trigger 'call not answered after' requires an input argument representing a time in some unit of measurement. This can be defined as a subclass of `UnitType` (i.e. `SecondsUnitType`) and linked to the corresponding trigger class via the property restriction `hasUnitType`. The top-level hierarchical structure in `wizpol` is shown below in Figure 3.28. Again, domain-specific ontologies are encouraged to extend this list and define relevant unit type options.

Figure 3.28 UnitType Top-Level Definition

4 Conclusion

This report used a series of graphical representations to describe the ontologies of `genpol` and `wizpol`, created to define the generic constructs of the APPEL policy description language and features common to the ACCENT policy wizard. Together, these ontologies form the basis of a reusable policy language ontology stack, which may be extended to define a domain-specific policy language. `Genpol` is the base level ontology which encapsulates the core, generic constructs of APPEL, including the syntax used to describe a policy document, policy rules and additional policy attributes, variables and operators. The `genpol` ontology is imported and extended within `wizpol`, to define how constructs may be categorised and processed for display by the ACCENT policy wizard.

Using the concept of ontology, these documents go beyond a simple syntactic definition of the policy language, as is presented through XML Schema, to express knowledge of the semantics surrounding the constructs used. In an ontology, the structure of the policy language is also defined in such a way that it may be reasoned about and extended through importation within additional ontologies. Both ontologies were defined using OWL and developed under the Protégé ontology environment. The following subsections evaluate this choice of ontology language and support tools, describing how the developed policy language ontology framework may be applied.

4.1 Evaluation of OWL/Protégé

OWL was used as it sports a broader set of functions than any existing ontology language. Compatibility with existing reasoners, such as RacerPro and Pellet, is offered through the OWL DL sub-language. This is useful for the current ontology set and also for future extensions to these ontologies. Alternative languages are either too formally expressive for the current ontologies, or lack the portability and support provided by an XML-based syntax like OWL. Additionally, as OWL is a recent standard, there is greater scope for standardised extension to its functionality.

The most noticeable flaw of the current OWL specification stems from a lack of support for customised data typing, which prevents ontologies from placing restrictions on data-type values. While OWL supports cardinality restrictions to specify the number of values associated with a property, it does not give the ability to state further restrictions upon data type values, such as a specific numeric range of Integer values or the minimum and maximum lengths of a String. There is a plan to extend OWL to integrate and reuse the mechanisms of the XML Schema specification, which allows detailed definition of user-defined data-types [7]. However, as XML Schema does not derive from an RDF-based format, there are issues regarding its syntactical compatibility with OWL. With these issues under debate, it is hoped a solution may be implemented in the near future which will allow OWL to support data-type restrictions in a standard way.

OWL ontologies are intended for use by software applications. Due to the large number of additional statements required in an ontology for compliance with OWL DL, the documents themselves become extremely large and complex to work with directly. Therefore, adequate tool support is essential. Without the use of Protégé, understanding and applying the range of OWL language constructs would have been a much slower, less efficient and highly error prone process. In addition, the graphical plug-in tools obtainable for the Protégé framework provided a useful means of analysing and presenting an ontology – especially in the latter stages of development when documents became much more complex.

The only notable drawback of using Protégé is the changeable state of software releases. As the tool is under constant development and the OWL language is still relatively young, the interface contains a number of bugs and inconsistencies. Also, as the framework was originally designed for general ontology support, the interface contains several functions not applicable to OWL. For these reasons, the tool is undergoing frequent revisions to improve its functionality and reliability. Currently, there is no other freely available tool which provides the same level of support for OWL.

4.2 Future Application

The key structure of the APPEL policy language is contained within the `genpol` ontology. In policy language terms, this ontology describes the core structure through which any domain-specific policy language must be based. Although `wizpol` extends this structure to provide additional constraints useful when interpreting the language within the policy wizard, it is not a compulsory extension to the

language itself. Therefore, genpol may be extended in two different ways depending on the application for which it is intended.

Specifically, the policy language defined in genpol was intended for specialisation and reuse within the ACCENT policy system. To achieve this, the language could be specialised by extended the structure of wizpol. This would tailor the language for use in a particular domain and allow for successful integration with the current ACCENT policy wizard.

However, as the core policy language details and wizard extensions have been defined within separate ontologies, the language could potentially be specialised through direct extension of genpol alone. This would be useful if the language was intended for use in another application or with a different user interface. For example, if the language was to be applied within another policy system, genpol could be extended directly. Also, should the policy wizard be altered in any way, wizpol could be adjusted accordingly or a new ontology created that imported genpol to describe the new interface.

Certainly, there is sufficient scope for the reuse of both genpol and wizpol, either as extensions of one another or independently. In a move to apply genpol and wizpol to the ACCENT policy system, the developed ontology stack framework was taken and extended to produce a domain-specific language ontology for (Internet) call control. The specialised policy ontology language is described in the technical report ‘Ontology for Call Control’ [3].

Appendix A: `Genpol` Properties

The table below lists the properties defined under the `genpol.owl` ontology document and a brief description of their usage within the ontology. Inverse properties simply reverse a restriction application – that is, they are placed upon the domain class instead of the range class.

Property Name	Description of usage
`hasAction`	A `PolicyRule` has at least one `Action`
`hasActionArgument`	An `Action` may have an `ActionArgument`
`hasCondition`	A `PolicyRule` has a `Condition`
`hasConditionOperator`	A `Condition` has a `ConditionOperator`
`hasConditionParameter`	A `Condition` has a `ConditionParameter`
`hasConditionValue`	A `Condition` has a `ConditionValue`
`hasPermissibleAction`	A `TriggerEvent` has some permissible `Action`(s)
`hasPermissibleParameter`	A `TriggerEvent` has some permissible `ConditionParameter`(s)
`hasPolicy`	A `PolicyDocument` has at least one `Policy`
`hasPolicyAttribute`	A `Policy` has some `PolicyAttribute`
`hasPolicyRule`	A `Policy` has at least one `PolicyRule`
`hasPolicyVariable`	A `Policy` has some `PolicyVariable`
`hasPolicyVariableAttribute`	A `PolicyVariable` may have some `PolicyVariableAttribute`(s)
`hasTriggerArgument`	A `TriggerEvent` may have a `TriggerArgument`
`hasTriggerEvent`	A `PolicyRule` has zero or more `TriggerEvent`(s)
`isActionOf`	Inverse property of `hasAction`
`isActionArgumentOf`	Inverse property of `hasActionArgument`
`isConditionOf`	Inverse property of `hasCondition`
`isConditionOperatorOf`	Inverse property of `hasConditionOperator`
`isConditionParameterOf`	Inverse property of `hasConditionParameter`
`isConditionValueOf`	Inverse property of `hasConditionValue`
`isPermissibleActionOf`	Inverse property of `hasPermissibleAction`
`isPermissibleParameterOf`	Inverse property of `hasPermissibleParameter`
`isPolicyAttributeOf`	Inverse property of `hasPolicyAttribute`

isPolicyOf	Inverse property of hasPolicy
isPolicyRuleOf	Inverse property of hasPolicyRule
isPolicyVariableAttributeOf	Inverse property of hasPolicyVariableAttribute
isPolicyVariableOf	Inverse property of hasPolicyVariable
isTriggerArgumentOf	Inverse property of hasTriggerArgument
isTriggerEventOf	Inverse property of hasTriggerEvent

Appendix B: `Wizpol` Properties

The table below lists the properties defined under the `wizpol.owl` ontology document and a brief description of their usage within the ontology. Inverse properties simply reverse a restriction application – that is, they are placed upon the domain class instead of the range class.

Property Name	Description of usage
`hasAbilityToQuery`	Can be applied to any class in a domain-specific ontology to indicate a form of relationship with an internally classed variable. In the current implementation, there is no inverse equivalent
`hasCategory`	Used to categorise triggers, condition parameters, actions and operators
`hasDataType`	Used to assign a particular defined subclass of `DataType` to a `TriggerArgument` or `ActionArgument`
`hasInternalUse`	Used to categorise a domain-specific trigger, condition parameter or action as Internal in its use in a domain
`hasUnitType`	Used to associate particular units for display alongside a `TriggerArgument` or `ActionArgument` or `ConditionParameter`
`hasUserLevel`	Used to categorise triggers, conditions, actions and operators in groups according to user level applicability
`isCategoryOf`	Inverse property of `hasCategory`
`isDataTypeOf`	Inverse property of `hasDataType`
`isInternalUseOf`	Inverse property of `hasInternalUse`
`isUnitTypeOf`	Inverse property of `hasUnitType`
`isUserLevelOf`	Inverse property of `hasUserLevel`
`matchValue`	This is an annotation property which has special function and is a form of meta-data. In OWL, this type of property acts as a class attribute rather than a restriction. It is applied in a similar way to the `rdfs:label, rdfs:comment or owl:versionInfo` predefined annotations defined for each class. The `matchValue` is used to define an alternative action or trigger class in a policy depending on the input value of an argument for a trigger or action. It contains a literal string value that links it with another ontology class. The string is interpreted and processed by an application (POPPET) reading the ontology.

References

[1] ACCENT Policy-based system Project home page: http://www.cs.stir.ac.uk/accent, May 2006.

[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 for Call Control*. Technical Report CSM-170. June 2006.

[4] Generic Policy Language Ontology document (genpol.owl). Located online at URL: http://www.cs.stir.ac.uk/schemas/genpol.owl, May 2006.

[5] 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.

[6] Jambalaya graphical plug-in tool for the Protégé ontology development environment. Home page: http://www.thechiselgroup.org/~chisel/projects/jambalaya/jambalaya.html, May 2006.

[7] Knublaugh, H. User-defined Datatypes in Protégé-OWL. Located online: http://protege.stanford.edu/plugins/owl/xsp.html, Last updated: August 2005. Last accessed: June 2006.

[8] OWL: The Web Ontology Language. http://www.w3.org/2004/OWL/, May 2006

[9] OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, June 2006.

[10] OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, June 2006.

[11] OWLViz graphical plug-in tool. Home page: http://www.co-ode.org/downloads/owlviz/, May 2006.

[12] Protégé home page: http://protege.stanford.edu/, May 2006.

[13] Racer Systems GmbH & Co. KG. Home Page and download links: http://www.racer-systems.com/index.phtml. May 2006.

[14] RDF: The Resource Description Framework. http://www.w3.org/RDF/, May 2006.

[15] RDF Schema (RDFS): http://www.w3.org/TR/rdf-schema/, June 2006.

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

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

[18] Wizard Policy Language Ontology document (wizpol.owl). Located online at URL: http://www.cs.stir.ac.uk/schemas/wizpol.owl, May 2006.

[19] WonderWeb OWL Ontology Validator. University of Manchester, 2003. Service located online at URL: http://phoebus.cs.man.ac.uk:9999/OWL/Validator, June 2006.