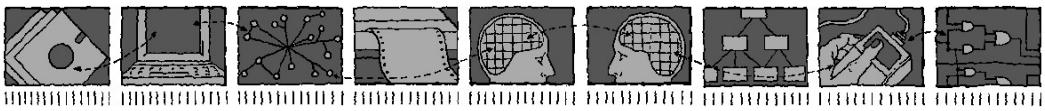
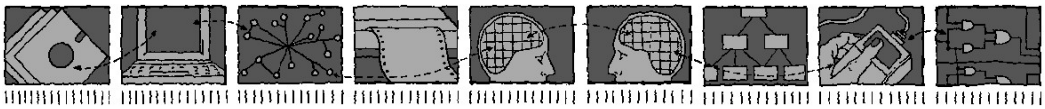


Department of Computing Science and Mathematics

University of Stirling



Ontology for Call Control

Gavin A. Campbell

Technical Report CSM-170

ISSN 1460-9673

June 2006

Department of Computing Science and Mathematics
University of Stirling

Ontology for Call Control

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-170

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. Using OWL (The Web Ontology Language), an ontology has been developed describing the domain of (Internet) call control. In particular, the ontology focuses on the use of call control in conjunction with its application within the ACCENT policy-based management system. The structure of the ontology builds heavily on previously developed ontologies `genpol` [4] and `wizpol` [20]. These describe generic aspects of the system, including the core policy description language on which it is based. This report presents a technical overview of the ontology for (Internet) call control, illustrated by way of graphical depictions of OWL class and property implementation.

Keywords: ACCENT, Call Control, Ontology, OWL, Policy, Internet Telephony.

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`

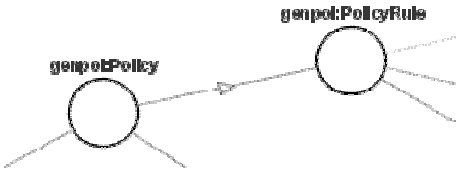
2. Diagram conventions

Diagrams depicted in this report were generated using the Jambalaya plug-in tool [6] and the OWLViz graphical plug-in tool [13] 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:

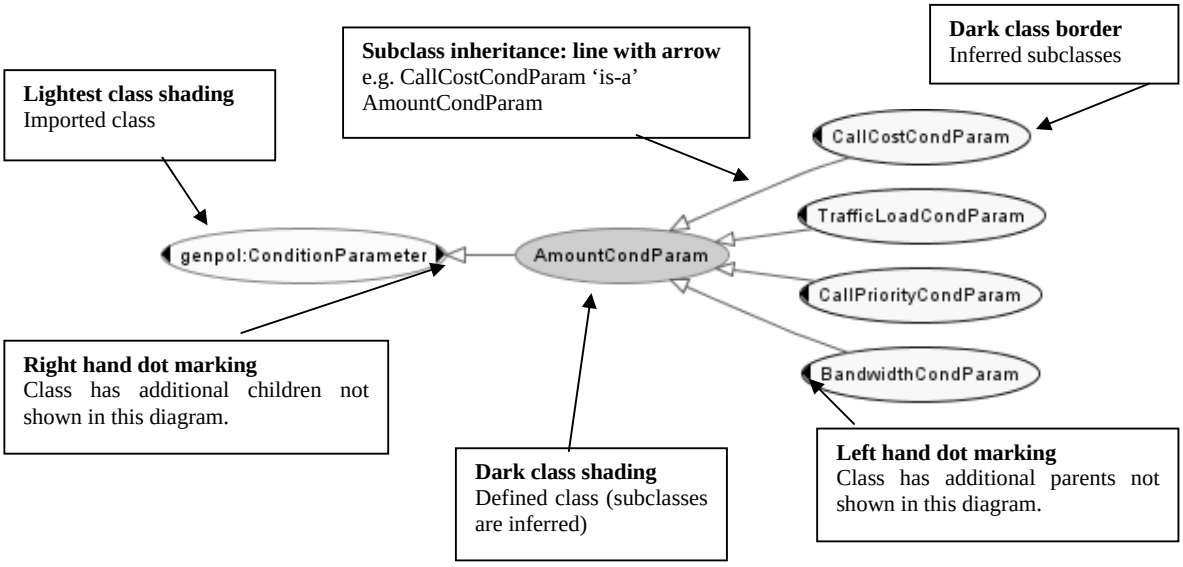


Sub-class (inheritance) is shown by a solid straight line without a triangle.

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 the `Courier New` 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

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.

The ontology of (Internet) call control defines the language through which policies may be defined and applied within the ACCENT policy-based management system [1]. The language itself has been specialised for the call control domain through extensions to core constructs of the APPEL policy description language [18]. This includes a definition of the triggering events, conditions and actions which may be used in policy execution. The ontology also describes higher-level call control concepts not explicitly intended for use by the policy system.

In previous work, the generic constructs of APPEL were defined in an ontology known as `genpol` [4]. A second ontology, `wizpol` [20], directly extends `genpol`, to specify common features employed to manipulate the language within the ACCENT system 'policy wizard' user interface.

As this report outlines extensions to the class, property and conceptual structure defined in the `genpol` and `wizpol` ontologies, it is recommended that the reader is familiar with the content of these documents as explained in the technical report 'Ontology Stack for a Policy Wizard' [3]. The call control ontology itself may be accessed at [8].

In the following introductory sub-sections, Section 1.1 provides an introduction to APPEL – the core policy description language used by the ACCENT system and specialised in the ontology 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 hierarchical stack of ontologies used to construct the call control policy language ontology.

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. 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. These core aspects of APPEL were encapsulated in an ontology named, which can be extended to tailor the language to a specific domain. This report describes how the core language of APPEL was extended for the domain of call control.

1.2 Motivation for Ontology Usage

The motivation behind creating this ontology 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.

In a wider context, the designed ontology for the call control domain is fully eligible for integration with the ACCENT policy wizard. This is achieved using a special integration system known as POPPET (Policy Ontology Parser Program Extensible Translation), which was developed to access, parse and process ontology data, and offer an interface for the policy wizard to query it. The original policy wizard, holding hard-coded call control details, was re-engineered to remove these references and integrate ontology knowledge via 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 [9]) 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 [11]. 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 [8].

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 [10] 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 [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 a particular domain application. Using OWL, two generic ontologies were created which have been extended to form the ontology described in this report.

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.

The `genpol` ontology document may be accessed at [4] and the `wizpol` ontology document accessed at [20]. An in-depth review of both these ontologies is given in the technical report ‘Ontology Stack for a Policy Wizard’ [3]. For clarity, it is recommended the reader is familiar with the content of this document prior to studying its specialisation for call control within this report.

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. The stack is shown in Figure 1.1.

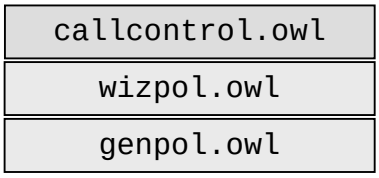


Figure 1.1 OWL Ontology Stack

These ontologies define only the structure of policy-related knowledge and not actual policy data. For this reason, 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. The call control ontology 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 the call control domain.

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.

This report describes an ontology of call control based on the ontology stack of Figure 1.1. The ontology describes how the core policy language defined and extended in `genpol` and `wizpol`, was specialised for this domain.

2 TriggerEvent Classes for Call Control

The top-level `genpol:TriggerEvent` hierarchy is shown in Figure 2.1. Specific triggers for the call control domain are defined as subclasses of `wizpol:NamedTriggerEvent`, as shown in Figure 2.2.

Domain-specific direct subclass extensions to `genpol` are category classes used to infer groups of related triggers. For call control, the defined trigger categories are “Availability” and “Call”. Inferred subclasses of these categories are shown in Figure 2.3 and Figure 2.4 respectively. A trigger may belong to exactly one category – no trigger may be associated with multiple categories.

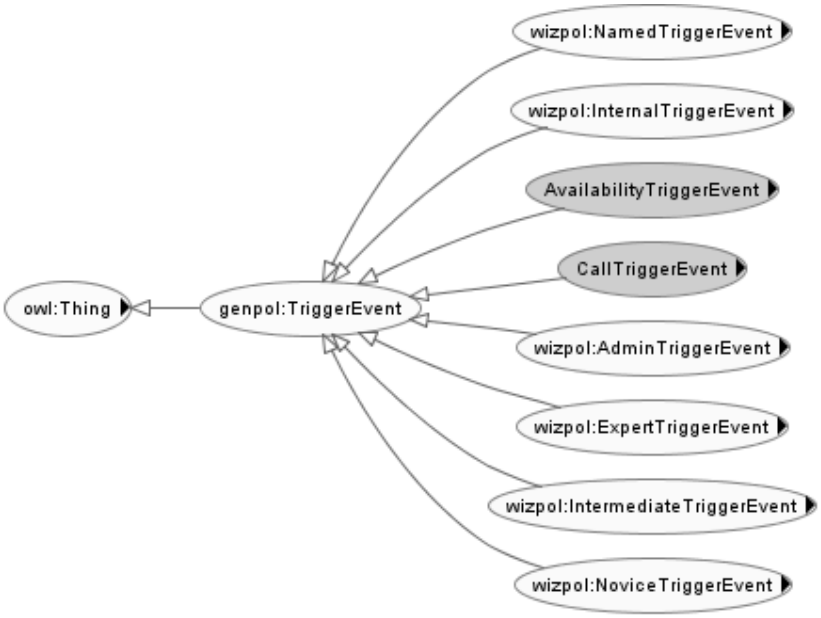


Figure 2.1 Top-Level Trigger Hierarchy

Inferred subclasses of “admin”, “expert”, “intermediate” and “novice” level named triggers are outlined in Figure 2.5, Figure 2.6 and Figure 2.7 respectively. Note that admin and expert level trigger sets are identical and contain the full set of triggers in the domain. This is the chosen set-up for this domain, but may of course vary for other domain-specific policy ontologies.

Inferred internal triggers are shown in Figure 2.8. This class contains a subset of triggers defined to be of internal use within the system.

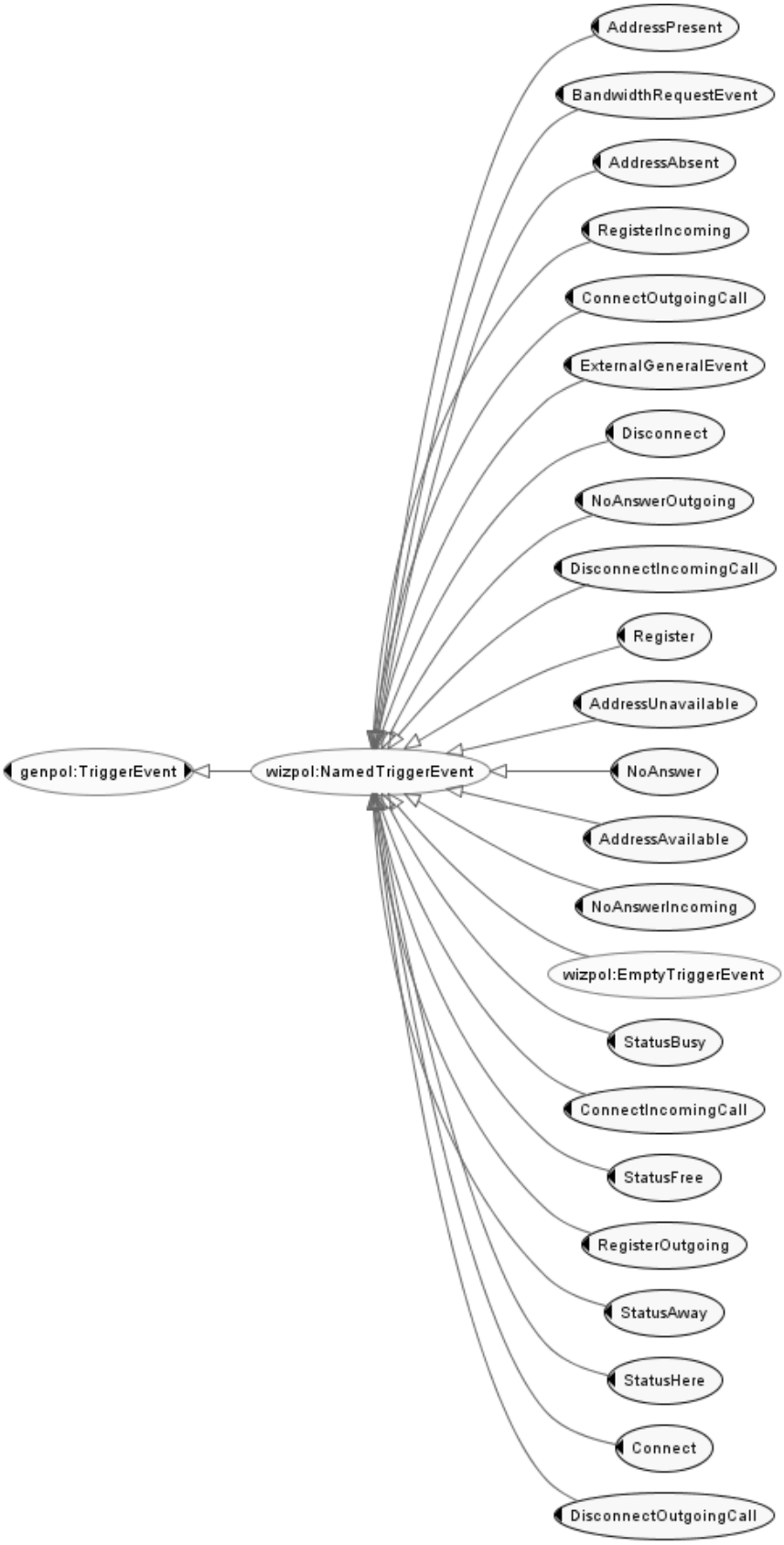


Figure 2.2 Named Triggers for The Call Control Domain

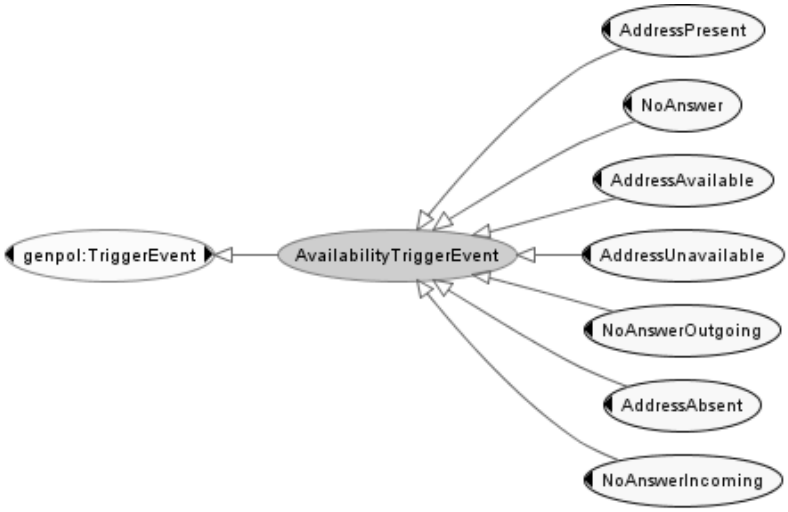


Figure 2.3 Inferred Triggers for The Availability Category

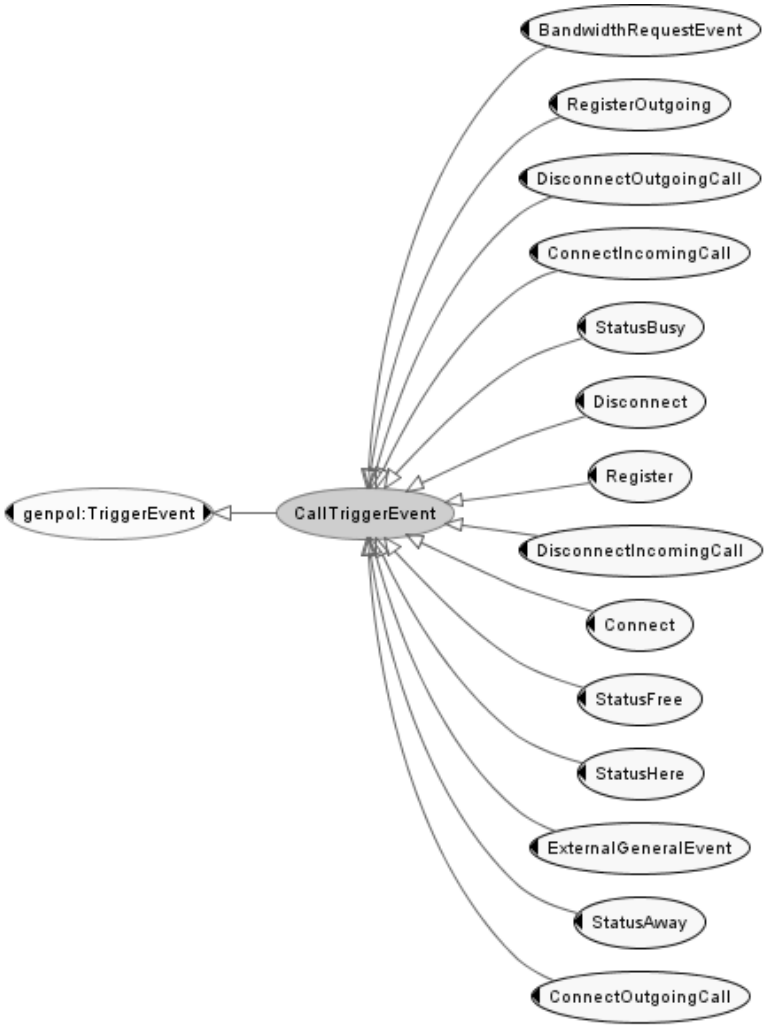


Figure 2.4 Inferred Triggers for The Call Category

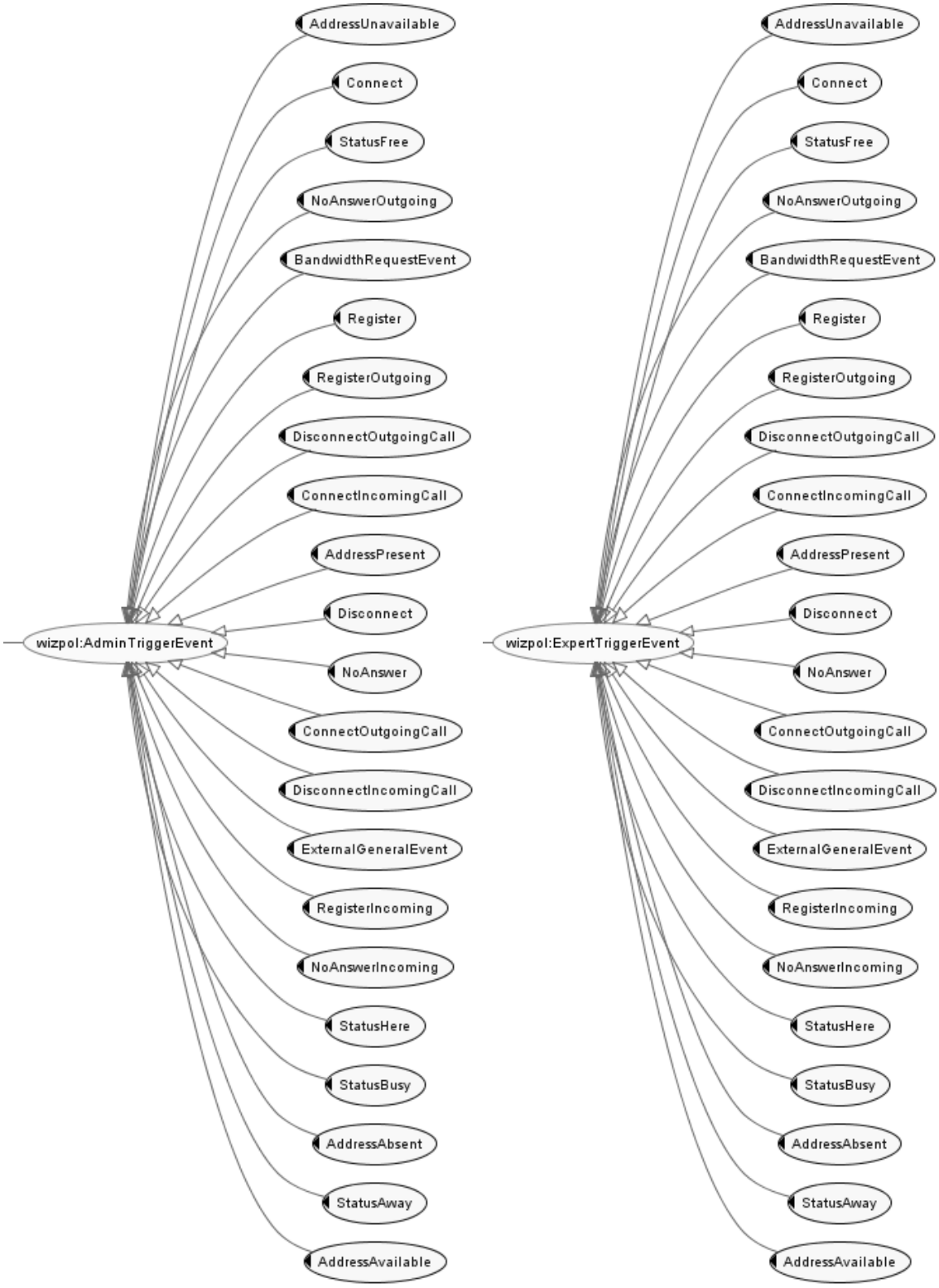


Figure 2.5 Inferred Admin and Expert Level Triggers

Note: Both sets are identical and contain all named triggers. Admin and Expert level users within the call control domain are granted permission to utilise all defined triggers.

Figure 2.6 Inferred Intermediate Level Triggers

Figure 2.7 Inferred Novice Level Triggers

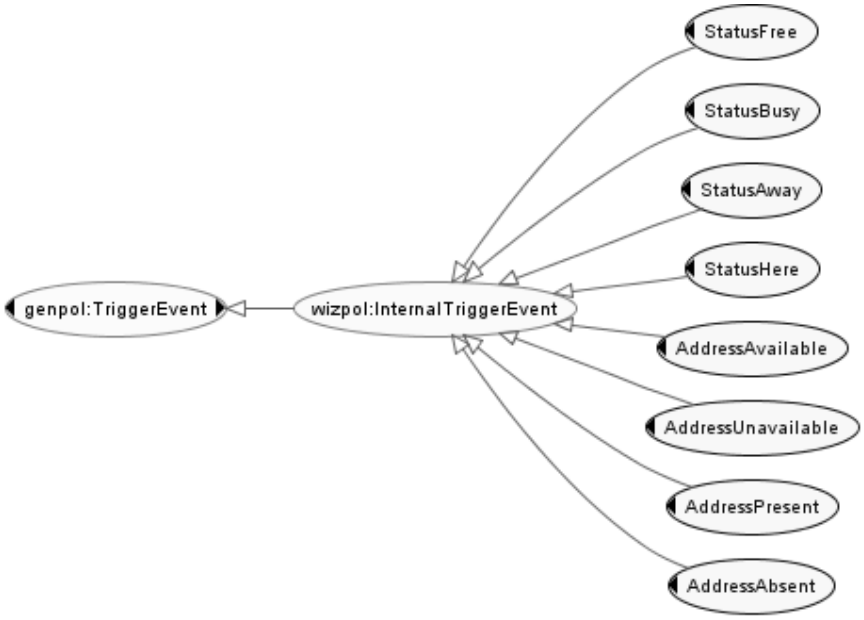


Figure 2.8 Inferred Internal Triggers

3 Condition Parameters

The call control ontology directly extends the top level genpol:ConditionParameter class structure, defining subclasses representing the condition categories “Address”, “Amount”, “Description” and “Epoch”. This is shown in Figure 3.1.

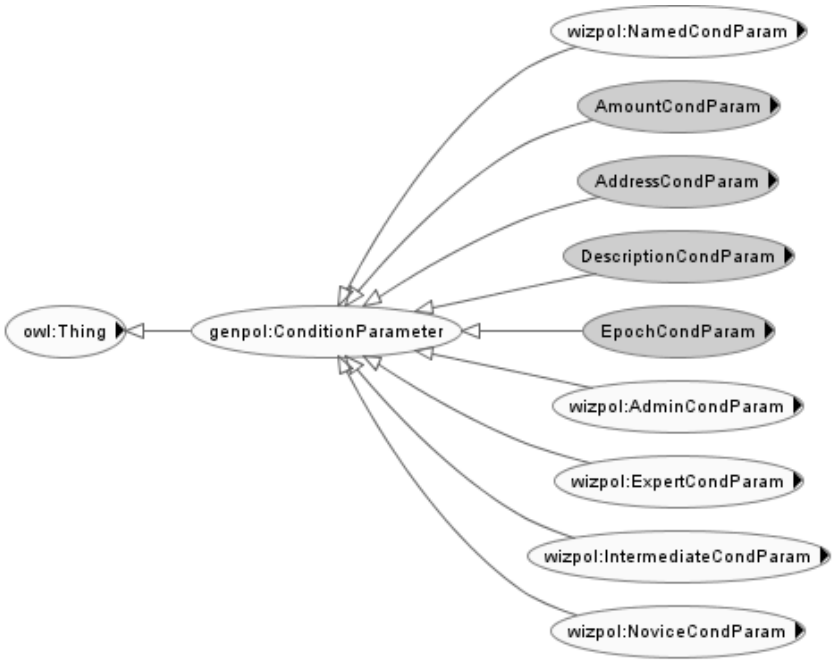


Figure 3.1 Top-Level Condition Parameter Class Structure

Each category has an inferred set of subclasses which are subsets of condition parameters defined under `wizpol:NamedConditionParameter`. Each named condition parameter is related to exactly one category – no condition may be associated with multiple categories.

The full set of defined call control condition parameters is shown in Figure 3.2. Inferred sets of condition parameters for the “Address”, “Amount”, “Description” and “Epoch” categories are shown in Figure 3.3, Figure 3.4, Figure 3.5 and Figure 3.6 respectively.

Inferred condition parameters for each user level within the policy system (“admin”, “expert”, “intermediate” and “novice” levels) are shown in Figure 3.7, Figure 3.8 and Figure 3.9.

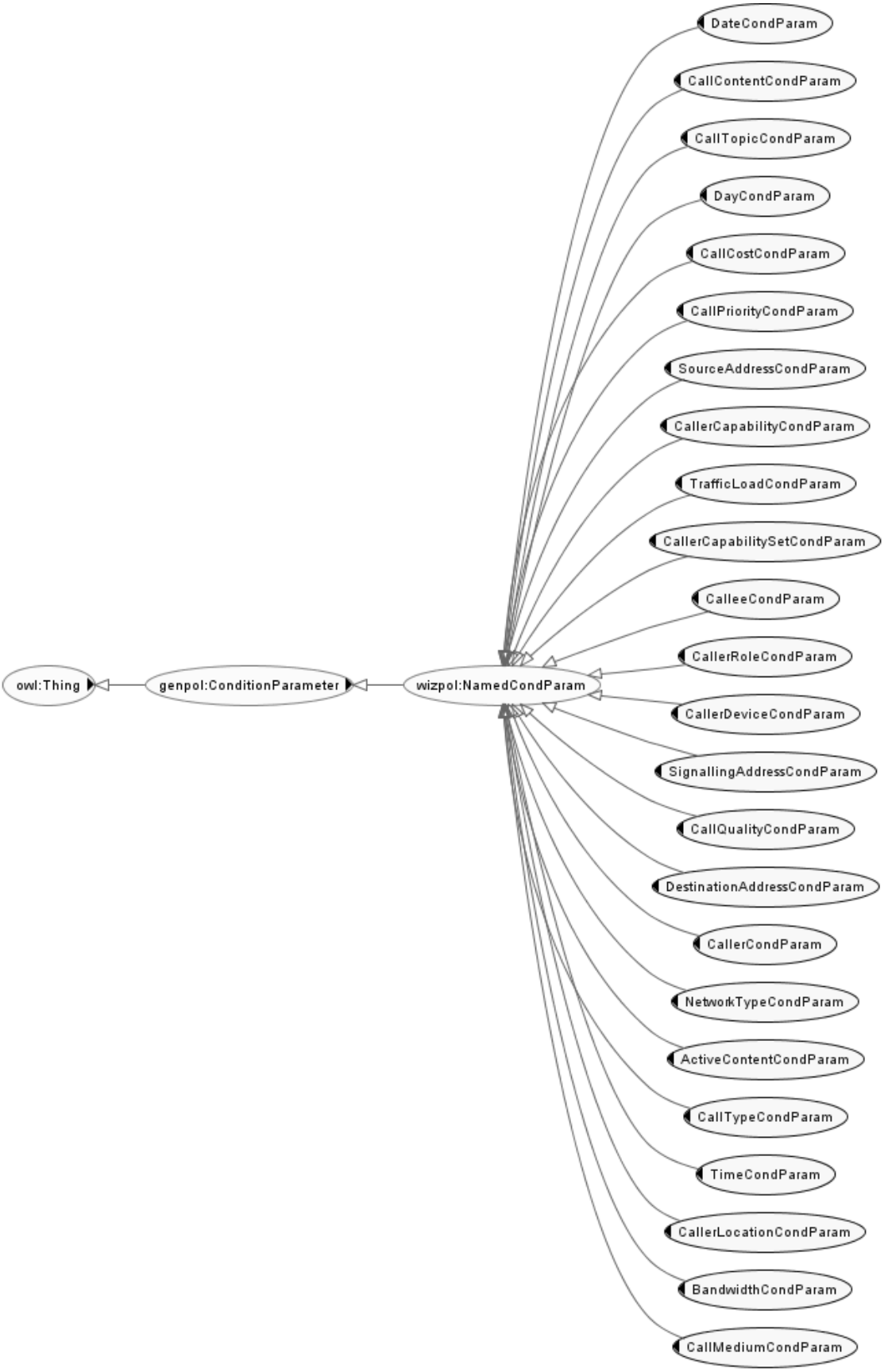


Figure 3.2 Named Condition Parameters for The Call Control Domain

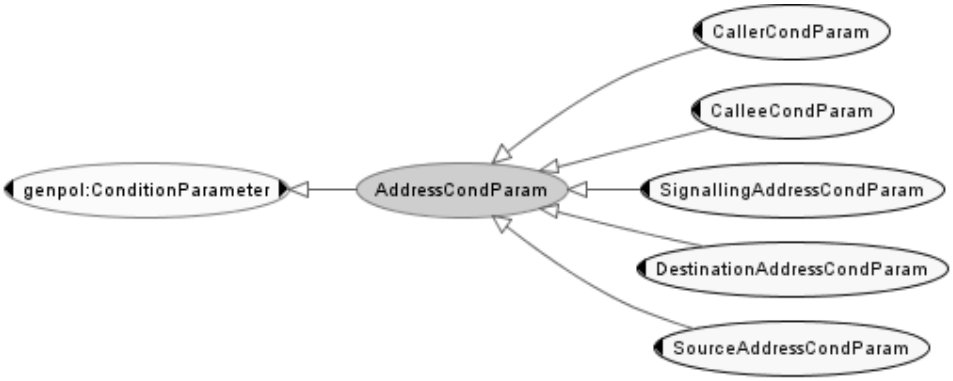


Figure 3.3 Inferred Address Category Condition Parameters

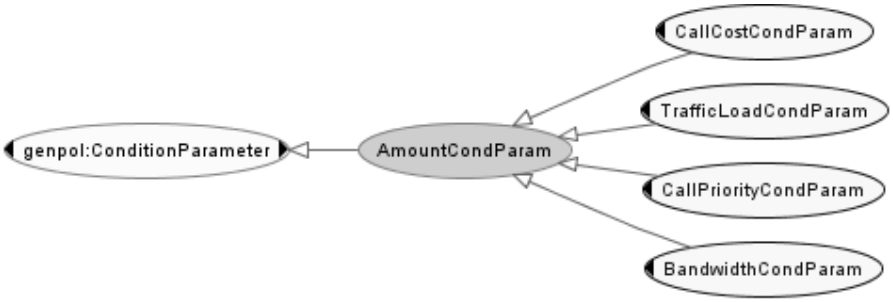


Figure 3.4 Inferred Amount Category Condition Parameters

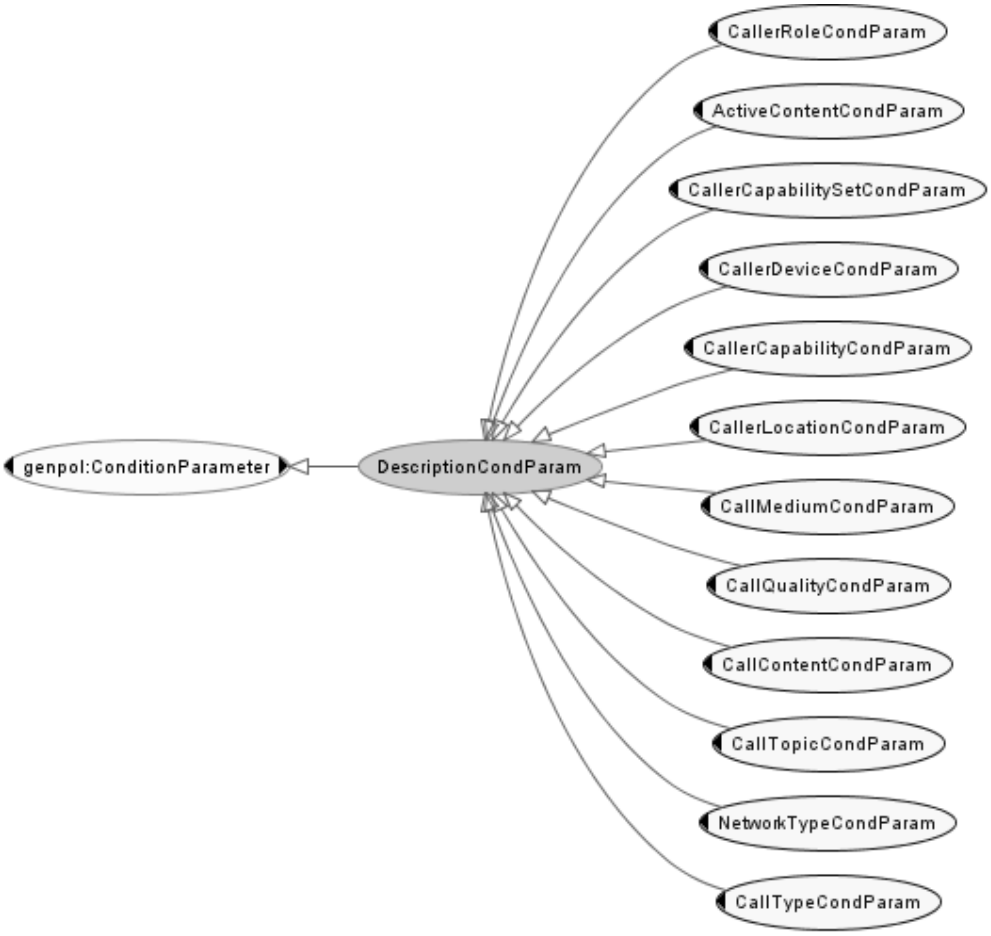


Figure 3.5 Inferred Description Category Condition Parameters

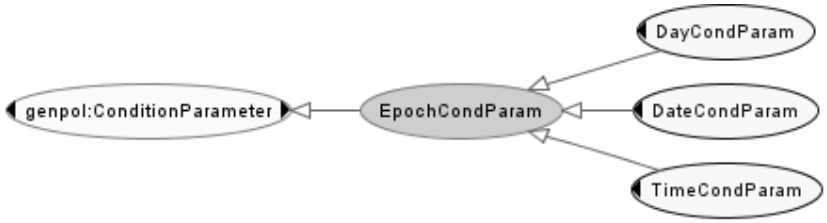


Figure 3.6 Inferred Epoch Category Condition Parameters

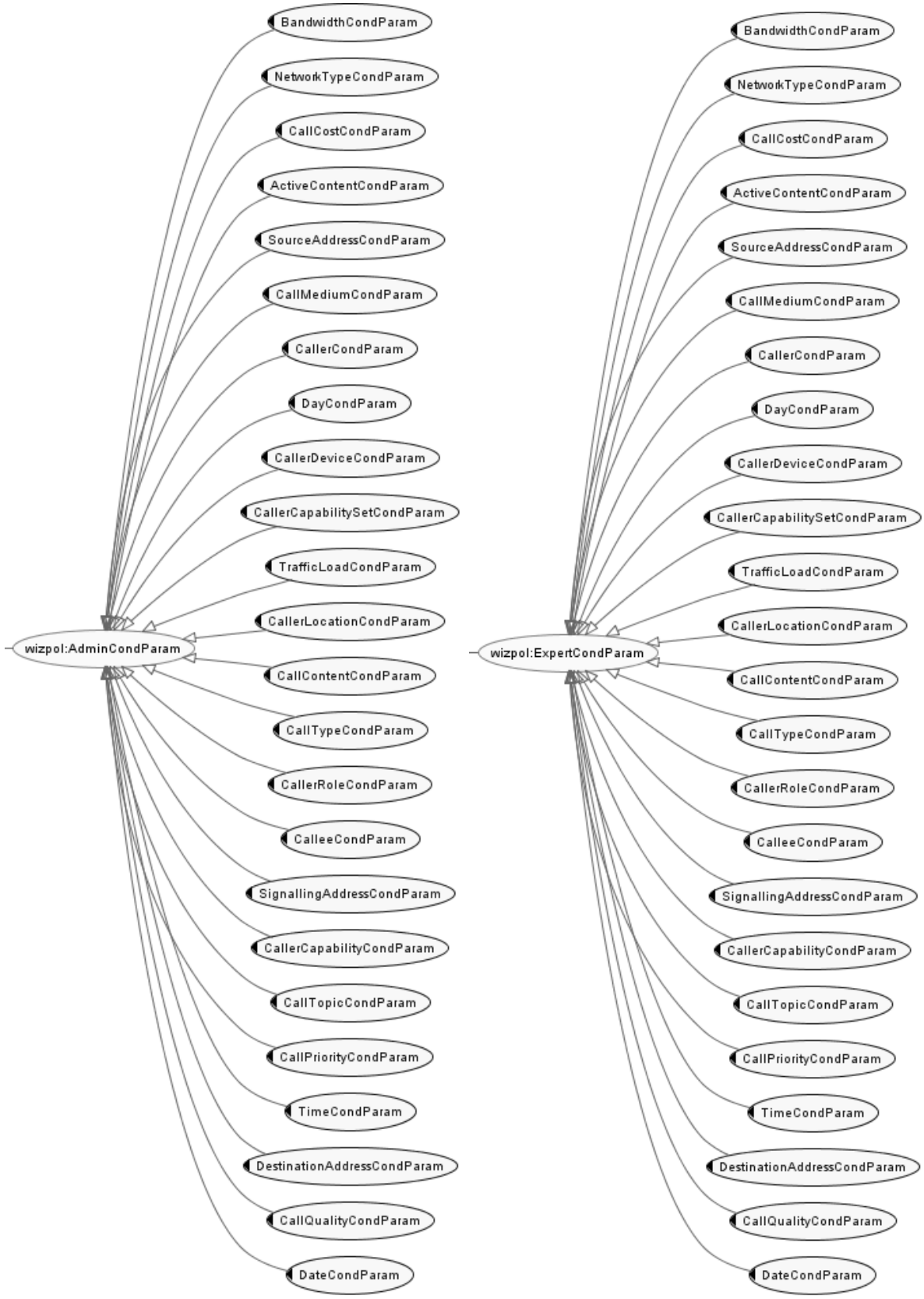


Figure 3.7 Inferred Admin and Expert Level Condition Parameters

Note: Both sets are identical and contain every named condition parameter. Admin and Expert level users within the call control domain are granted permission to utilise all defined condition parameters.

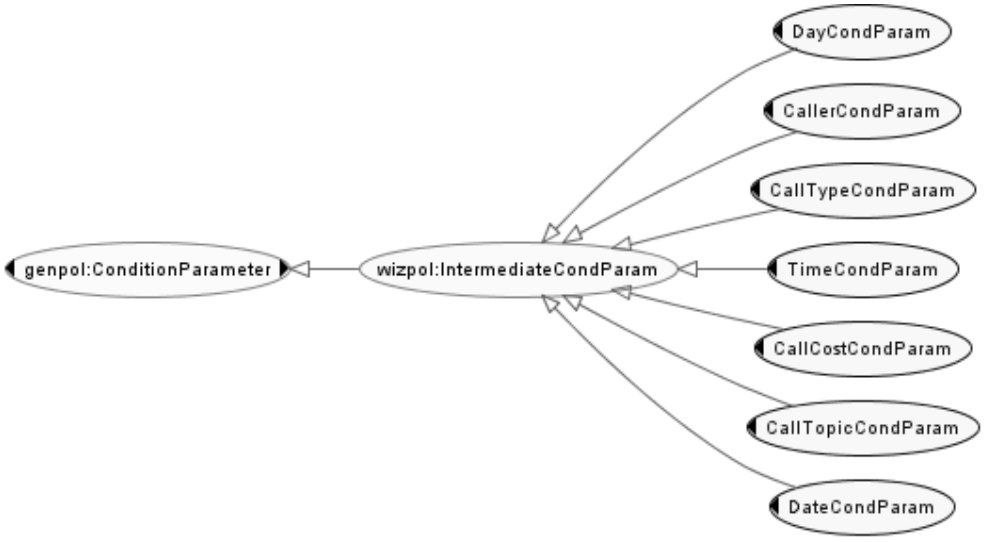


Figure 3.8 Inferred Intermediate Level Condition Parameters

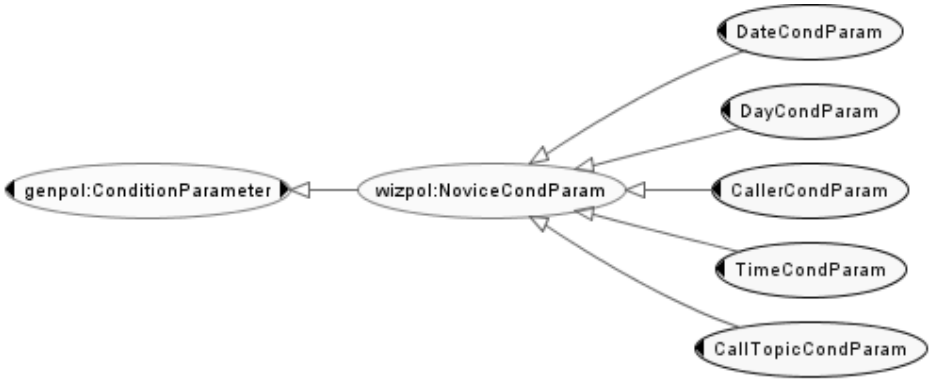


Figure 3.9 Inferred Novice Level Condition Parameters

4 Actions

The top level `genpol:Action` hierarchy is shown in Figure 4.1. Specific actions for the call control domain are defined as subclasses of `wizpol:NamedAction`, as shown in Figure 4.2.

From Figure 4.1, direct subclass extensions are category classes used to group related actions. For call control, the defined action categories are “Call”, “Send” and “Update”. Inferred subclasses of these categories are shown in Figure 4.3, Figure 4.4 and Figure 4.5 respectively. An action is defined to be a member of one category only – no action may be associated with multiple categories. Note that the “Call” trigger category (`CallTriggerEvent` class) is different to the “Call” action category (`CallAction` class).

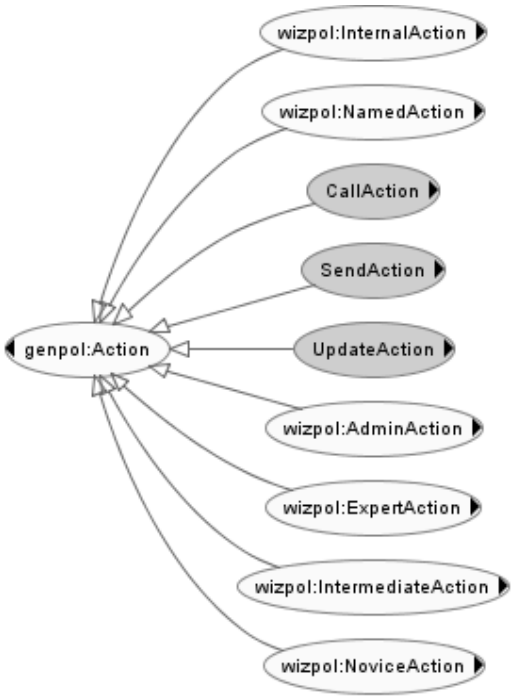


Figure 4.1 Top-Level Action Hierarchy

Inferred subclasses of “admin”, “expert”, “intermediate” and “novice” level named actions are outlined in Figure 4.6, Figure 4.7 and Figure 4.8 respectively. Note that “admin” and “expert” level trigger sets are identical and contain the full set of named actions. This is the chosen set-up for this domain, but may vary in other domain-specific policy ontologies.

Inferred internal actions are shown in Figure 4.9. This class contains a subset of actions defined to be of internal use to the policy system.

Figure 4.2 Named Actions for The Call Control Domain

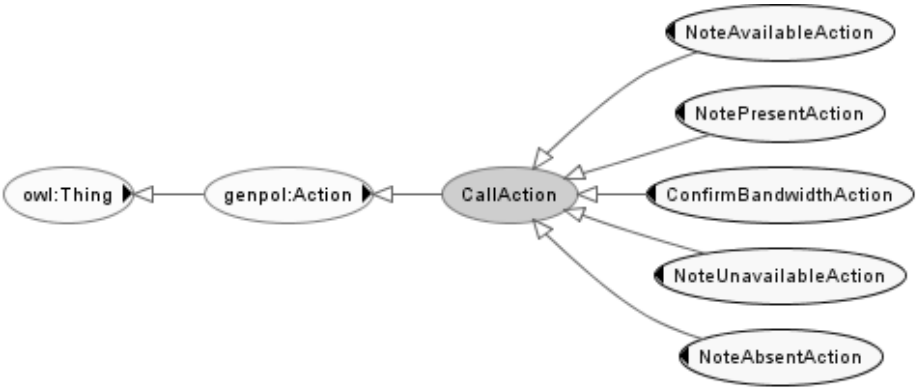


Figure 4.3 Inferred Actions for The Call Category

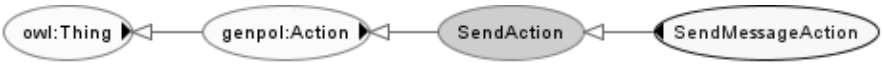


Figure 4.4 Inferred Actions for The Send Category

Figure 4.5 Inferred Actions for The Update Category

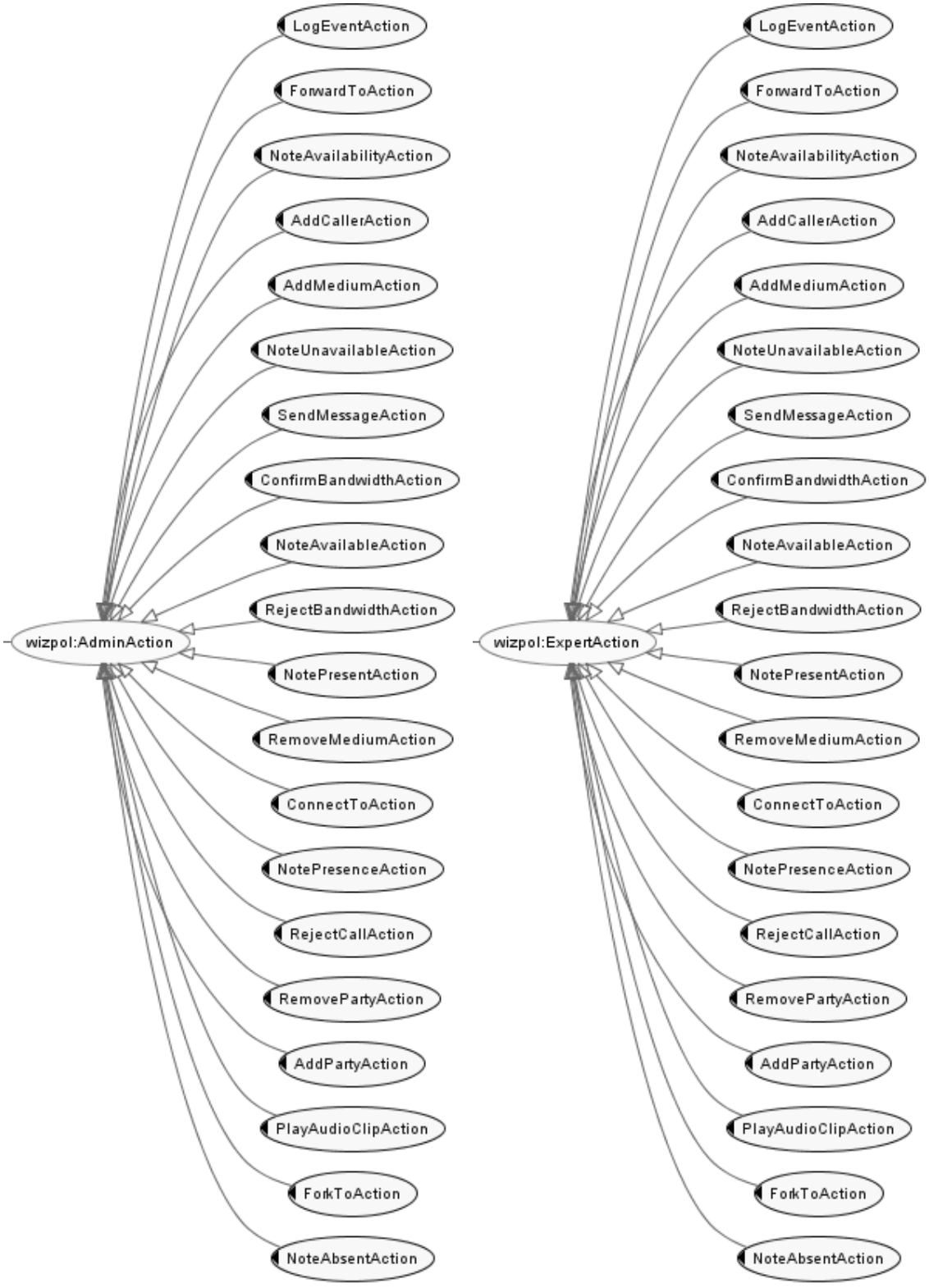


Figure 4.6 Inferred Admin and Expert Level Actions

Note: Both sets are identical and contain every named action. Admin and Expert level users within the call control domain are granted permission to utilise all defined actions.

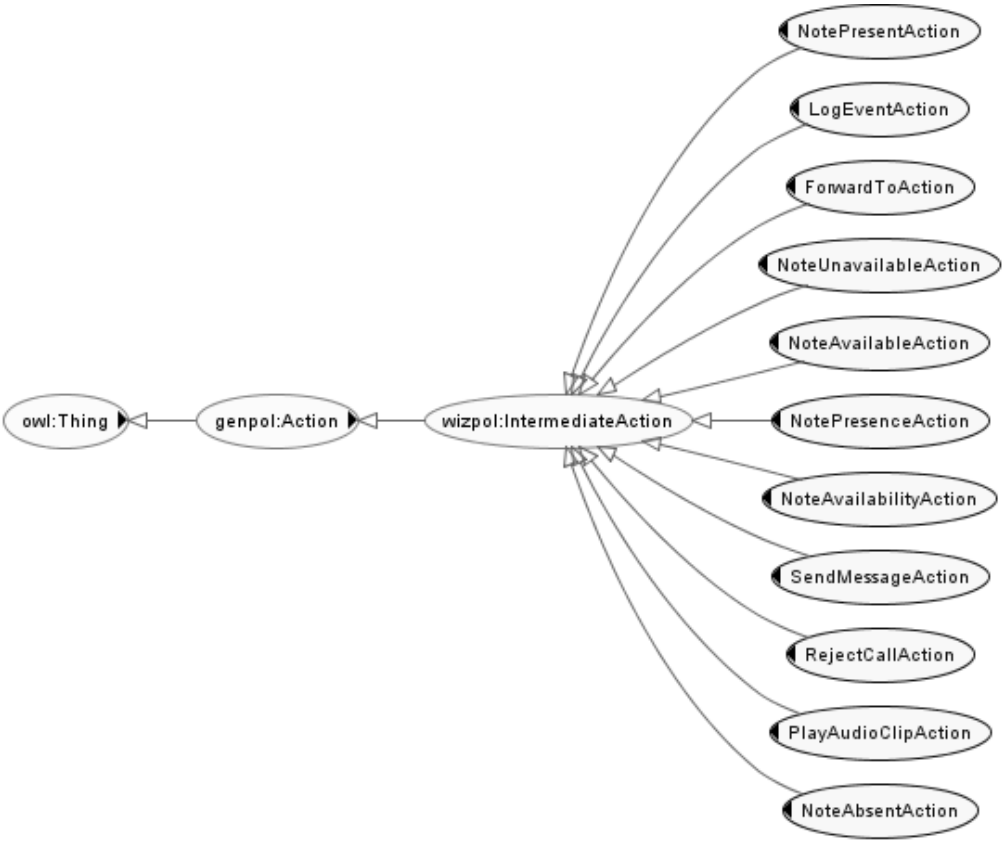


Figure 4.7 Inferred Intermediate Level Actions

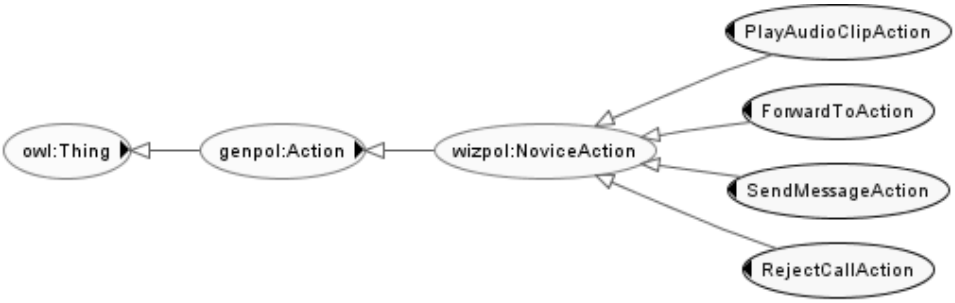


Figure 4.8 Inferred Novice Level Actions

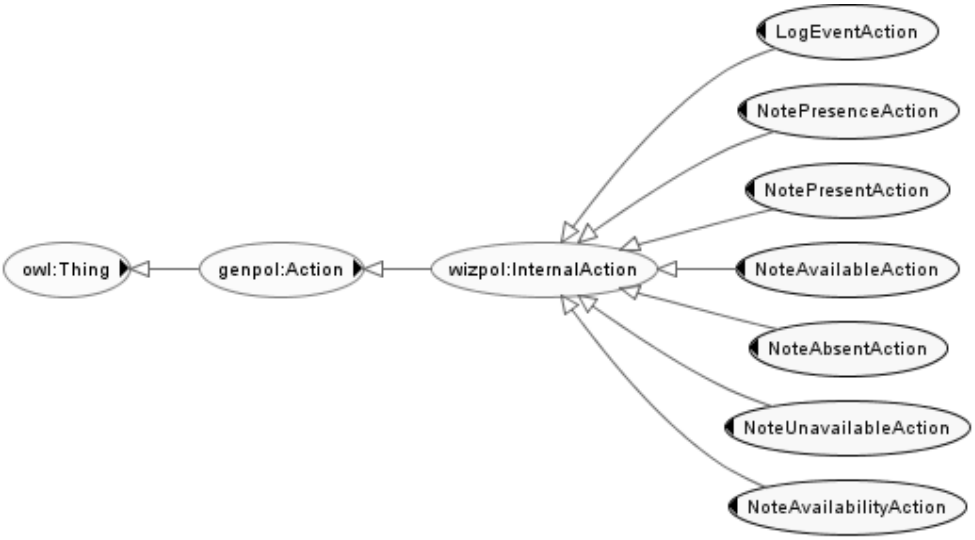


Figure 4.9 Inferred Internal Actions

5 Class Categorisation

The call control ontology extends `wizpol:ClassCategorisation` to define categories of triggers, conditions and actions specific to the call control domain, as shown in Figure 5.1. Categories defined under this structure are used to achieve category inference throughout the ontology.

Figure 5.1 Trigger, Condition Parameter and Action Categories

6 Arguments

Domain-specific arguments associated with triggers and actions in a policy are defined as subclasses under the `genpol:Argument` class hierarchy. Arguments describe the type of information which is supplied as a parameter with an action or trigger in a policy.

Arguments which accompany call control actions extend the `genpol:ActionArgument` class as shown in Figure 6.1. Similarly, trigger arguments extend the `genpol:TriggerArgument` class as shown in Figure 6.2. Each argument may be associated with more than one action or trigger respectively.

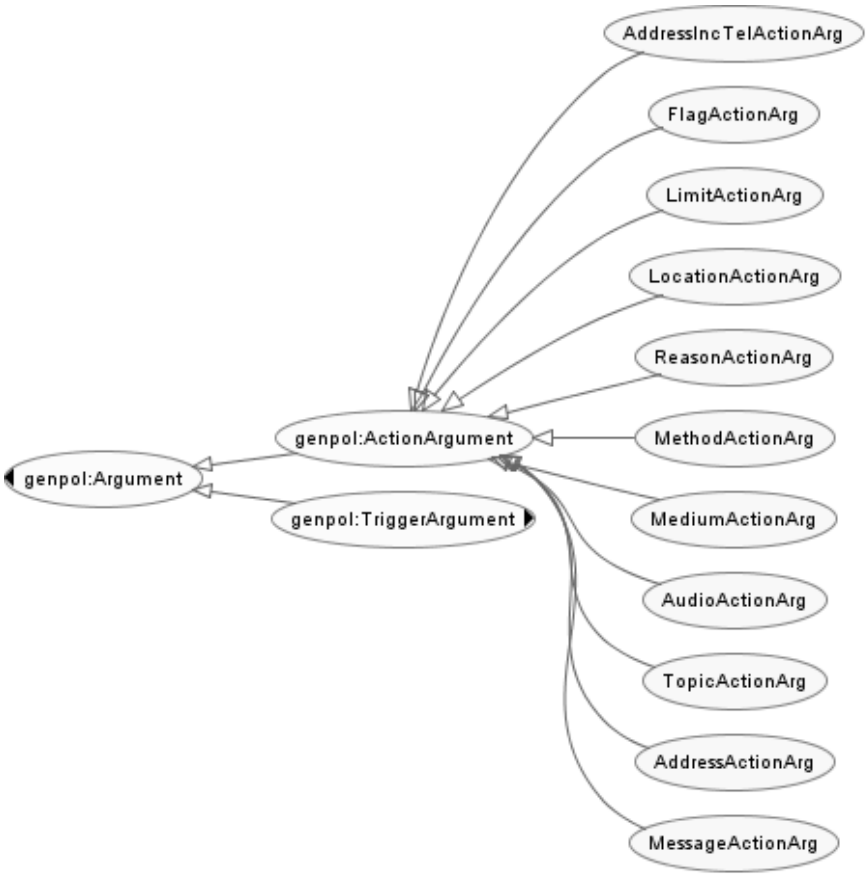


Figure 6.1 Action Arguments for The Call Control Domain

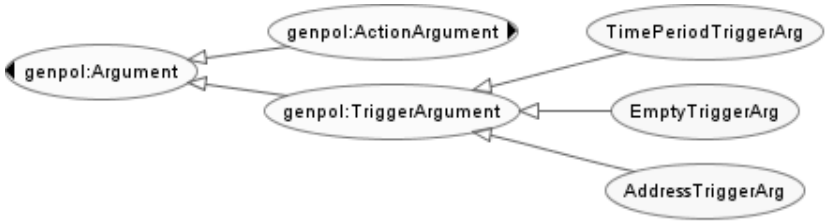


Figure 6.2 Trigger Arguments for The Call Control Domain

7 Condition Operators

The call control domain extends regular condition operators defined in `genpol` for the purposes of natural language translation by the policy wizard. These new operators are clones of original `genpol` operators (in their defined restrictive properties and the way in which they are interpreted by the policy system) but have a different `label` value attribute which translates to an alternative display string depending on the category they are used in.

For example, the `LessOrEqualToEpoch` class represents an alternative translation of the `LessThanOrEqualTo` operator when used in the context of the Epoch (date, day, time) category. Specifically, this translates to “is on or before” instead of the regular phrase “is less than or equal to”. Extensions are defined for the “description” and “epoch” condition categories as outlined in Figure 7.1.

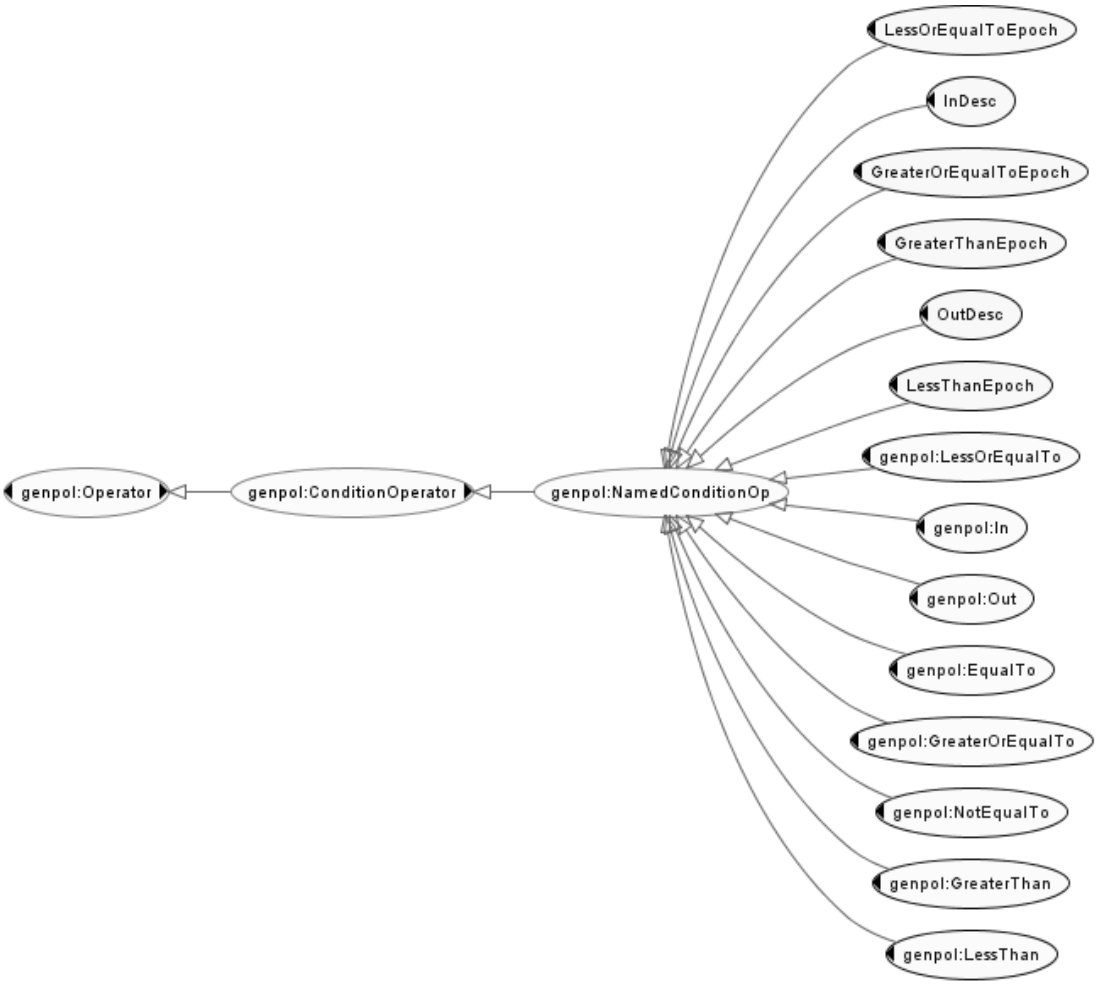


Figure 7.1 Extended Condition Operators

Additionally, subclasses of `genpol:ConditionOperator` are defined which hold inferred sets of subclasses of condition operators associated with each condition category. These inferred subsets for the “Address”, “Amount”, “Description” and “Epoch” condition categories are shown in Figure 7.2, Figure 7.3, Figure 7.4 and Figure 7.5.

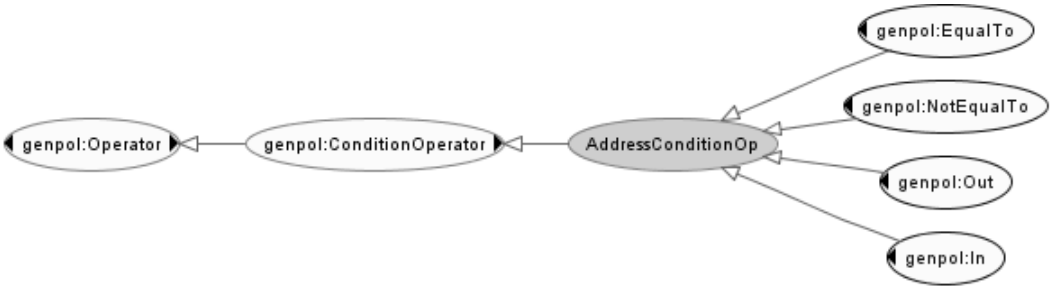


Figure 7.2 Inferred Address Category Condition Operators

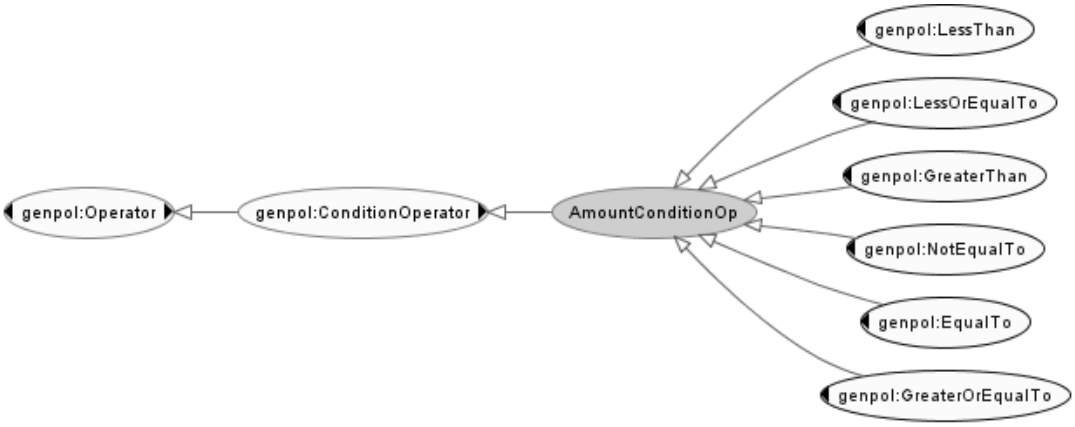


Figure 7.3 Inferred Amount Category Condition Operators

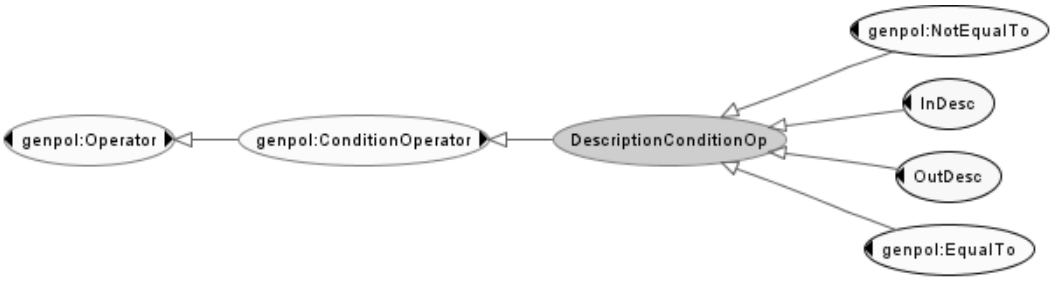


Figure 7.4 Inferred Description Category Condition Operators

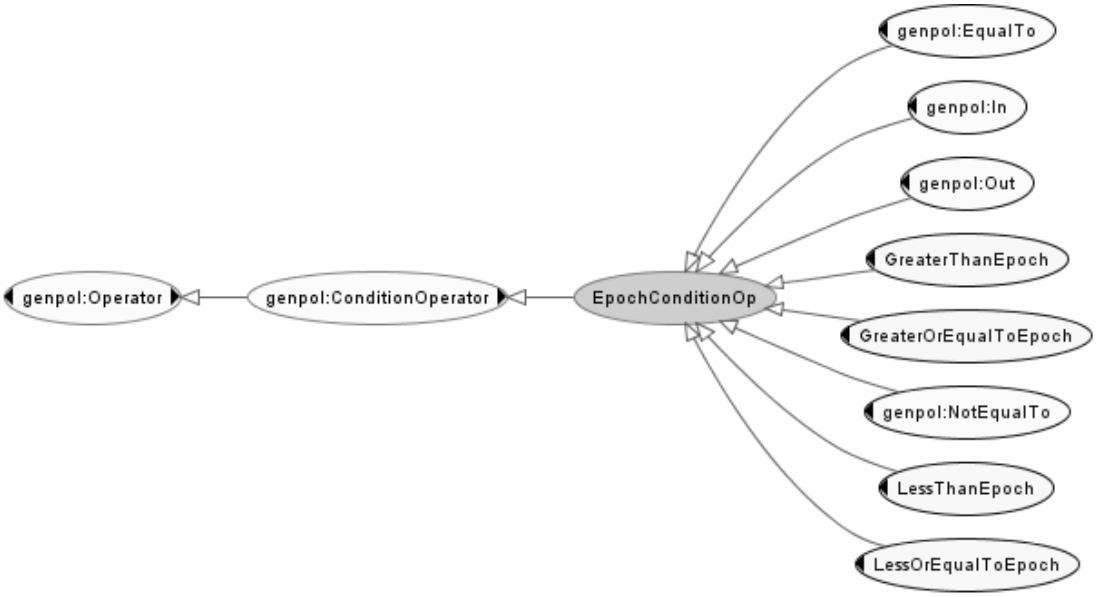


Figure 7.5 Inferred Epoch Category Condition Operators

8 Status Variables

A status variable is treated differently from an ordinary policy variable in the ACCENT system. General policy variables can be defined by individual users (or an administrator) and may represent information for use solely within policies they define. Status variables are domain-specific and apply to all users of the system. Within the call control domain, there are two defined status variables in addition to the 'profile' of a user. These are the 'presence' and the 'availability' of a user, which are defined as subclass extensions of `wizpol:StatusVariable` as shown in Figure 8.1.

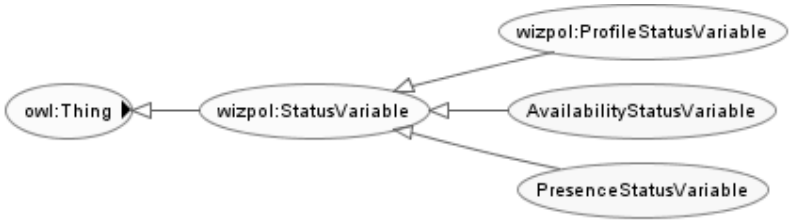


Figure 8.1 Status Variables for Call Control

The 'availability' status variable can be set to indicate whether a user is generally available (to receive calls) or available for calls relating to a specified topic. The 'presence' status variable may be set to indicate whether a user is generally present or present within a specific location only.

Status variables are internal to the policy system. Certain triggers and actions are defined based on their value at a given time. While their values may be modified by the user, they cannot be removed and no additional status variables may be defined.

9 Unit Types

Within the call control domain, there are three unit types typically linked with action, trigger and condition values as shown in Figure 9.1. The “Kbps”, “general” and “seconds” unit types are used to associate appropriate textual labels with displayed argument values within in the policy wizard. For example, the named trigger `BandwidthConditionParam` has a restriction along the `hasUnitType` property linking it with the `KbpsUnitType` class.

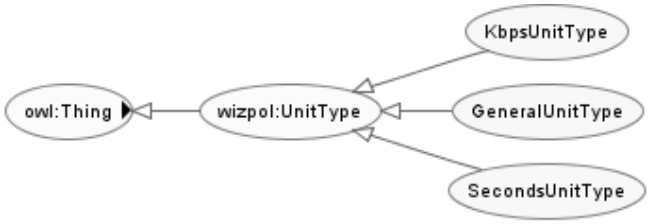


Figure 9.1 Call Control Unit Types

10 Data Types

`Wizpol` defines the `DataType` class structure outlined below in Figure 10.1. The call control ontology defines another direct subclass representing integer data types: `IntType`.

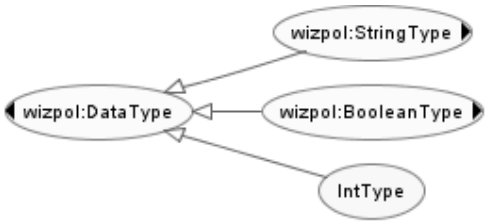


Figure 10.1 `DataType` Top-Level Class Extension

The call control domain also extends the variety of string data types as shown in Figure 10.2. Classes `CallMediumType` and `CallMethodType` define enumerated sets of condition values for the condition parameter `CallMedium` and `CallMethod` respectively.

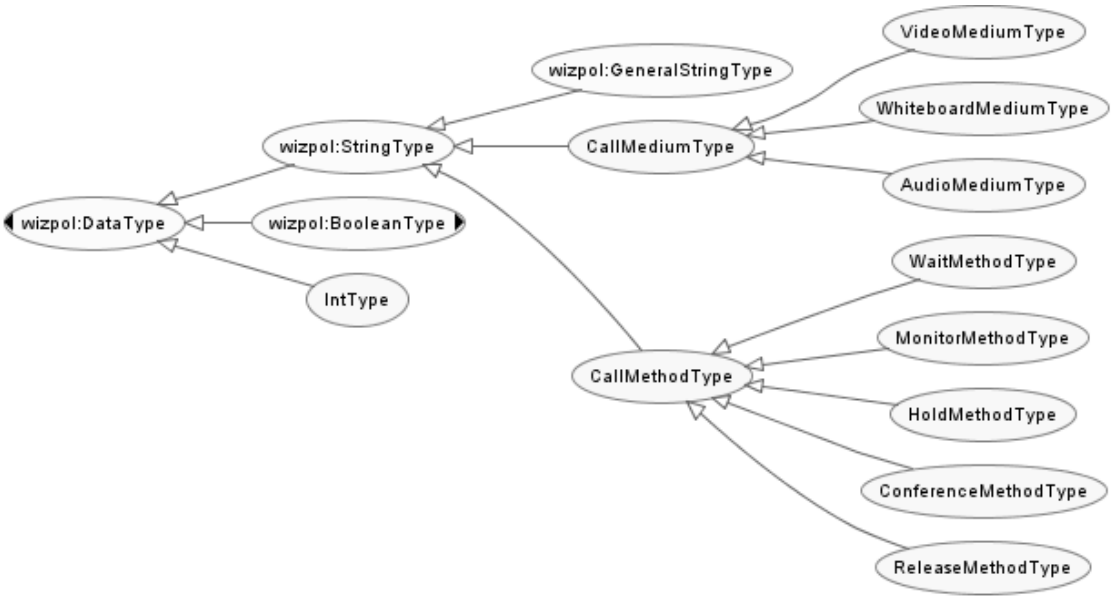


Figure 10.2 String `DataType` Extension for Call Control

11 Call Control Domain-specific Information

While the call control ontology primarily defines domain-specific class and property extensions to the base ontologies of `genpol` and `wizpol`, additional knowledge is included not directly intended for policy system usage. This knowledge relates to a higher level conceptual view of the form and operation of (Internet) call control, as shown in Figure 11.1.

For further details, including comment descriptions of each class, please refer to the generated ontology documentation OWLDoc [12].

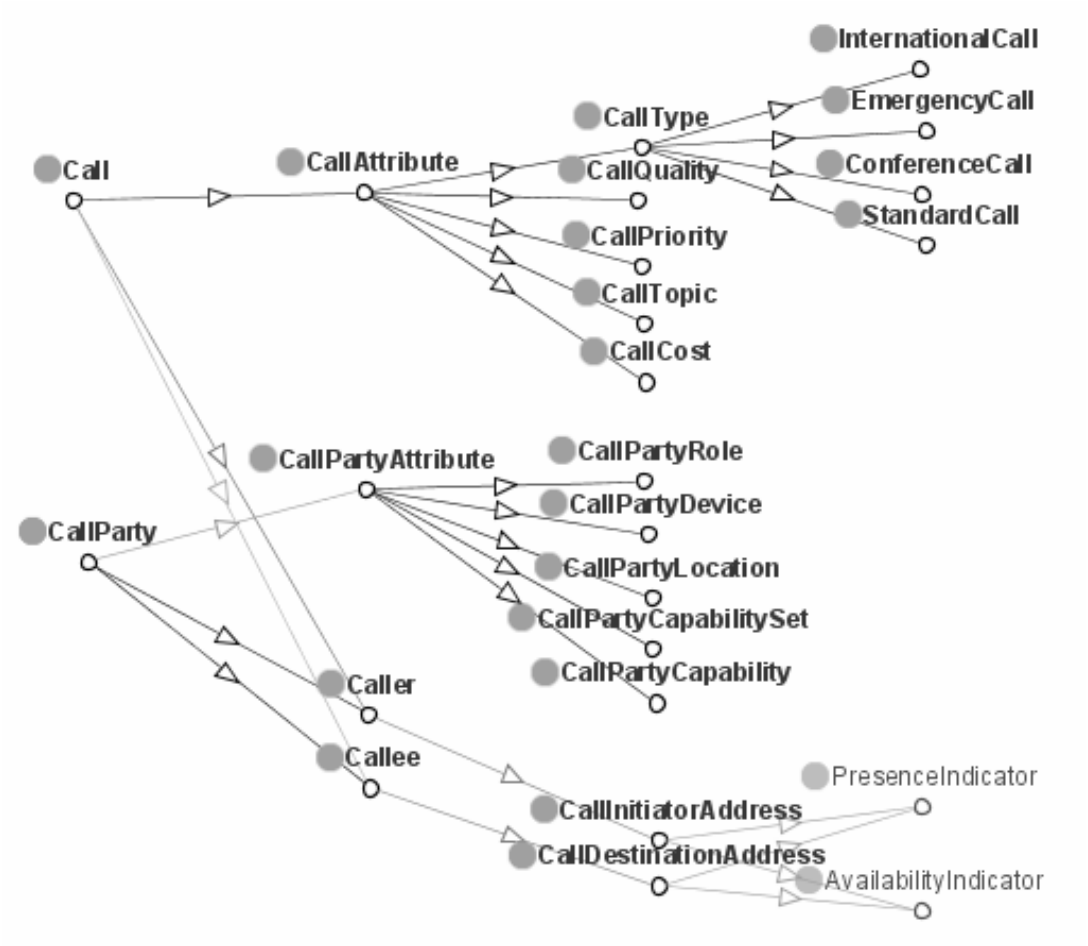


Figure 11.1 Additional Call Control Domain Knowledge
Diagram generated using Jambalaya [6]

12 Conclusion

This report used a series of graphical representations to describe an ontology of (Internet) call control. The ontology defines the complete language used to specify policies within the ACCENT policy management system [1]. Based on generic constructs of the APPEL policy description language, the ontology extends a previously developed ontology stack, to specialise the language for use in the domain of call control. In particular, this includes an extension to the ontologies of `genpol` and `wizpol`. `Genpol` is the base ontology which encapsulates the core, generic constructs of APPEL, including the elements 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. The ontology of call control extended the framework further, to define domain-specific details of the policy language including triggering events, condition parameters, actions, arguments, variables and data types. This report provided an overview of these specialised constructs. Additionally, the ontology described more general, non-policy related knowledge of the domain.

Defining an ontology to describe the call control domain goes beyond a simple syntactic definition of the policy language, as is presented through XML Schema, to express greater knowledge of the semantics surrounding the constructs used. Using ontology, the structure of the policy language is also defined in such a way that it may be reasoned over and extended through importation within additional ontologies. The ontology of call control was defined using OWL and developed under the Protégé ontology environment. The following subsections evaluate this choice of ontology language and support tools and describe how the developed domain-specific ontology may be applied.

12.1 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.

12.2 Future Application

The ontology for call control is a domain-specific specialisation of the core APPEL policy description language. In its entirety, the ontology comprises three separate ontology structures which extend each other in a hierarchical fashion to specify language related information. These ontologies are `genpol` (generic policy language constructs), `wizpol` (policy wizard specific customisation) and the call control ontology itself. As the ontology has been built using this structure, it is automatically usable within the ACCENT policy system. As described in Section 1.2, this can be achieved using POPPET [2].

Additionally, the ontology describes general knowledge of the call control domain, not directly related to the policy language, as described in Chapter 11. This knowledge could theoretically be used within the existing ACCENT policy system by components other than the policy wizard, such as the policy server or context system. Also, the ontology can be directly extended following future revisions of the policy language, or should the policy wizard be adapted to include more general domain knowledge.

The call control ontology may also be utilised by other applications not connected to the ACCENT system or policy processing in the large. As an OWL ontology is stored globally at a specific URL, another source may access it, processing its contents within a larger knowledge base or application system. Such sources might include an OWL ontology that imports and extends the call control ontology for use in another context, or by another parsing software agent or application.

In conclusion, there are several ways the developed ontology for call control may be utilised or extended, both within its intended field of policy-based management of calls and in other more general contexts.

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] Generic Policy Language Ontology document (`genpol.owl`). Located online at URL: http://www.cs.stir.ac.uk/schemas/genpol.owl, June 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 visual plug-in tool for Protégé. Home page: http://www.thechiselgroup.org/~chisel/projects/jambalaya/jambalaya.html, June 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] Ontology of (Internet) Call Control (`callcontrol.owl`). Located online at URL: http://www.cs.stir.ac.uk/schemas/callcontrol.owl, June 2006.

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

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

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

[12] OWLDoc for call control (including `genpol` and `wizpol` class and property descriptions): http://www.cs.stir.ac.uk/~gca/owl/cc/doc/index.html. June 2006.

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

[14] Protégé home page: http://protege.stanford.edu/, June 2006.

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

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

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

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

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