Gemma A. Campbell and Kenneth J. Turner. Ontologies to support Call Control Policies.
In N. Meghanathan, D. Collange and Y. Takasaki, editors, Proc.
3rd. Advanced International Conference on Telecommunications (AICT'07),
pages 5.1-5.6, IEEE Computer Society, New York.

Ontologies to support Call Control Policies

Gemma A. Campbell and Kenneth J. Turner
Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, UK
Email gca | kjt @cs.stir.ac.uk

Abstract—**The topic of policy-based management is introduced. Its specific application by the ACCENT project to call control is then discussed. The APPEL policy language supports regular policies as well as resolution policies that deal with conflict handling. The core APPEL language can be specialised, e.g. for call control. Ontologies are introduced as a means of capturing domain-specific knowledge – here, about calls. It is seen how this has allowed the ACCENT policy system to be generalised for use in a variety of domains. This is supported by a stack of interrelated ontologies: for generic policy aspects, for a policy definition wizard, and for call control. The approach has been integrated with the ACCENT system, allowing its extension for policy-based management in new domains.**

Index Terms—**Call Control, Internet Telephony, Ontology, OWL, Policy.**

I. INTRODUCTION

This paper explores the use of new techniques in advanced telecommunications. Policies are used to personalise control of (Internet) telephony, while ontologies are used to define a solid foundation for the application domain.

Traditional telephony services, such as call diversion, are centralised and limited in their effectiveness. Their invocation cannot take account of individual preference or the dynamic context of the call. Policies have emerged as a promising method of promoting and managing decentralised services in networks to give end-users more control. Using policies, a user may customise a service and define high-level goals for actions a system should take depending on the circumstances in which an event occurs. A policy defines how to modify the behaviour of a system, depending on whether defined conditions (e.g. time or user context) are detected.

This paper reports a specialisation of the policy-based management system developed by the ACCENT project (Advanced Call Control Enhancing Network Technologies [1]). Although ACCENT focused on Internet call processing, it developed a general approach for policy-based management of any kind of service. The ACCENT system supports creation, editing, deployment and execution of policies expressed in a policy description language called APPEL (ACCENT Project Policy Environment/Language [8]). The paper focuses on how APPEL was modelled using a framework of ontologies which separately encapsulate generic aspects of the policy language and specialised aspects dealing with call control.

Using ontologies to describe the policy language and its specialisation for call control goes beyond simple syntax, as it allows a deeper knowledge of the application domain to be expressed. The motivation for defining the APPEL language in this way was to enable greater flexibility in support of the core language structure and those of its specialisations.

Section II provides background on policy-based systems and languages, together with an overview of the ACCENT policy system. An introduction to ontologies and the Owl ontology language is also given. Section III describes the ontology framework developed for describing call control. Support for policy conflict handling is discussed in section IV, where the approach is extended for resolution policies. Section V evaluates the approach and highlights future work.

II. CONTEXT AND BACKGROUND

A. Policy Languages

Policy-based management techniques have historically been employed for purposes such as access control, quality of service, and security. However, policy-based systems have found much wider application. The work by ACCENT on management of (Internet) call control is a novel application of policies. A policy is defined by users in some high-level language that specifies the syntax and semantics of the policy constructs. Many policy languages have been developed. However, this paper focuses on the ACCENT approach because of the distinct advantages it offers, including its design for users not programmers, extensibility of the core language, and proven suitability for the unique requirements of call control. The place of the ACCENT work in the general context of policy systems is discussed in [11].

B. The ACCENT Policy System

The ACCENT policy-based management system [11] allows users to specify high-level policies for how they wish calls to be handled. The major components of the ACCENT system have the three-layer structure as shown in Figure 1.

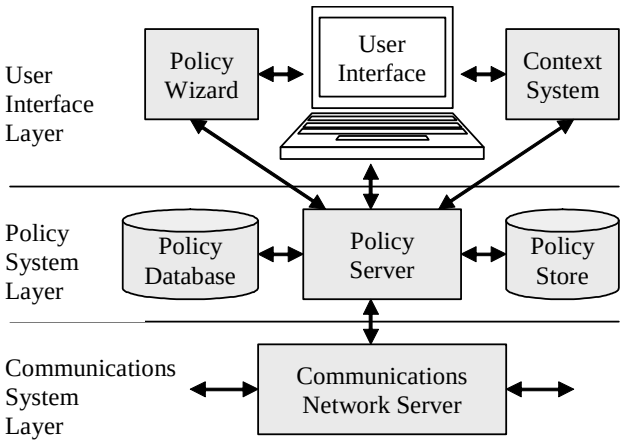

Figure 1. ACCENT Policy System Architecture

At the lowest level is the Communications System Layer that connects the system to its external environment. Policy enforcement is handled by the Policy System Layer that incorporates the Policy Server and Policy Stores. At the top level is the User Interface Layer, where users create policies and contextual information is obtained. Users define and edit policies via the Policy Wizard [10]. This supports a familiar web-based interface, which allows policies to be managed irrespective of the user's location. For a detailed explanation of the ACCENT system architecture refer to [9].

The system supports rule-based policies in event-condition-action (ECA) form. A policy rule broadly consists of three main elements:

- a combination of triggers: events that potentially cause a policy to be executed
- a combination of conditions: predicates over context variables that determine whether a policy may execute
- a combination of actions: outputs dictated by a policy.

A policy is eligible for execution if its triggers occur simultaneously and its conditions apply. Additional conditions may be imposed, such as the period during which the policy applies, or the profile to which the policy belongs. When the policy system is informed of an event, the applicable policies are retrieved, and applied if eligible. Multiple policies can be triggered, which may lead to conflict if their actions clash. The policy server automatically detects and resolves such conflicts.

A comprehensive policy description language called APPEL [8] was designed to facilitate the creation of policies within the ACCENT system. APPEL comprises a core language schema and its specialisations for different application domains. For example, there are specialisations for call control and for conflict resolution. APPEL defines the overall structure of a policy document, including regular policies, resolution policies, and policy variables. A policy consists of one or more policy rules. Each of these contains an optional trigger, an optional condition, and a compulsory action. APPEL specifies how compound triggers, conditions and actions can be defined. Other core facilities of the language include a range of operators for conditions.

To give a feel for the approach, the following are *simple* examples of the kinds of policies that can be expressed. APPEL is capable of describing much more complex or subtle policies.

- Calls to department staff must never be diverted to Mary.
- Ken is available for calls about policy languages.
- When Evan arrives, alert Ken by email to call him.
- Calls for Gemma should be sent to voicemail if she is busy. However, calls from Bob must continue to ring.
- Calls from French speakers should be answered by Solange or Michel.
- International calls must not be forwarded.

C. Handling Policy Conflicts

Policy conflict resembles the well-known feature interaction problem in traditional telephony. Conflicts in a policy-based environment are caused by the simultaneous execution of policies with contradictory actions. The ACCENT approach is described in [12]. Run-time conflict detection and resolution is carried out during policy execution. Conflict handling is defined by resolution policies that are distinct from regular policies. This gives considerable flexibility in that conflict handling is not hard-coded into the policy system – it is defined externally, and can be domain-specific.

Resolution policies express when and how the system should respond to conflicts. Their effect is to filter a set of proposed policy actions, selecting those that are compatible and in accordance with the stated conflict handling rules. As an example, the caller may wish to use video while the callee does not. Their respective policies propose 'add video' and 'avoid video' actions that are obviously contradictory. This will be determined as a conflict and resolved, e.g. the caller (as the bill payer) may be given priority.

Resolution policies are specified as an extension of the core APPEL language, and therefore use the same syntax as policies themselves. However, resolution policies use a different vocabulary because they govern different things. When (domain-specific) actions are proposed by regular policies, these become the triggers of resolution policies. Resolution policies can dictate generic outcomes (selecting among the proposed actions) or specific outcomes (dictating domain-specific actions, e.g. for call control).

D. Ontologies

An ontology is the set of terms used to describe and represent an area of knowledge, together with the logical relationships among these. It provides a common vocabulary to share information in a domain, including the key terms, their semantic interconnections, and some rules of inference. Ontologies confer the ability to share a common understanding of how information is structured in a particular domain. Ontologies also enable separation of domain knowledge from common operational knowledge in a system. A more in-depth review of ontologies can be found in [5].

A variety of specialised languages are used to define ontologies. OWL (Web Ontology Language [7]) is an XML-based language that was standardised by the World Wide Web Consortium in 2004. Due to its standards status, OWL gains through widely available software support, as well as compatibility with other techniques that can be integrated with it. In addition, OWL provides a larger function range than any other ontology language to date. For these reasons, OWL was used to define the ontologies described in this paper.

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 defined for classes in the form of restrictions. These specify the nature of a relationship between two classes. OWL also supports inheritance within class and property structures. The OWL Reference [6] describes the full range of language facilities.

OWL supports the sharing and reuse of ontologies through an import mechanism. Using this, definitions of classes, properties and individuals within an imported ontology are made available to the importing ontology. The ontological basis for APPEL exploits this, using multiple documents for different aspects of the core language and its specialisation for

call control. The use of ontologies is discussed in section III for call control policies, and in section IV for call conflict resolution policies.

E. Implementation of Ontology Support

An implementation of the approach has been created using Java as the programming language, Protégé as the OWL editor (http://protege.stanford.edu), Jena as the ontology parser (http://jena.sourceforge.net) and Pellet as the ontology reasoning engine (http://pellet.owldl.com). The work has been integrated into the ACCENT system. A major advantage has been generalisation of policy handling, notably in the wizard, allowing use of the same approach in a variety of applications.

The POPPET system (Policy Ontology Parser Program – Extensible Translation) has been designed to support ontology integration. POPPET runs as a stand-alone server. When invoked, it parses an ontology document at a given URL and reasons about its contents using the Pellet engine. A model of the ontology is constructed and stored for queries. A connecting application may then interrogate this stored ontology model using a variety of generic methods. Communication with the ACCENT policy wizard is achieved using Java RMI (Remote Method Invocation). The interaction between ACCENT and POPPET appears in Figure 2.

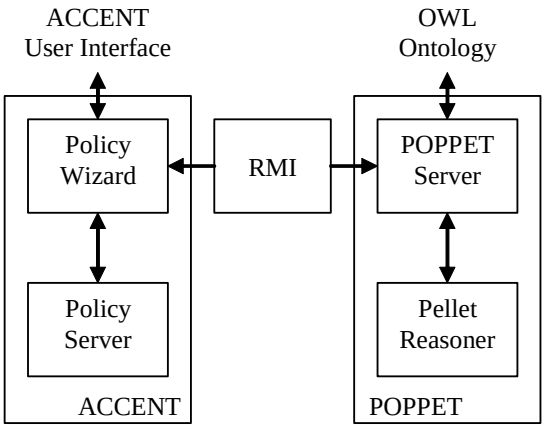

Figure 2. Ontology Integration using POPPET

Although implemented principally for use with ACCENT, POPPET is sufficiently generic that it may be used by other external applications that support RMI.

III. POLICY LANGUAGE FRAMEWORK FOR CALL CONTROL

Using OWL, a framework of ontologies was designed to describe the APPEL policy language – both the core language and its specialisations. The framework defines the language abstractly for generic policies and their use with the policy wizard. It also defines the specific extensions for call control.

A. Ontology Framework for Policies

Two common ontologies were developed using OWL. The first, named *genpol* (generic policies), defines the core constructs of APPEL. The second, named *wizpol* (wizard policies), extends this to capture specific facilities of the policy wizard. Crucially, *genpol* defines the concepts which describe policies in general. It is used as a starting point to specialise the policy language for any application domain. As OWL supports the sharing and reuse of ontologies by means of ontology importation, all definitions of classes, properties and individuals within an ontology may be used by the importer. The *wizpol* ontology imports *genpol*, extending it to provide additional user interface facilities not directly related to APPEL. Extending ontologies in this way results in the 'ontology stack' or layered model shown in Figure 3. On top of this, any domain-specific ontology may be defined and integrated with the ACCENT policy system.

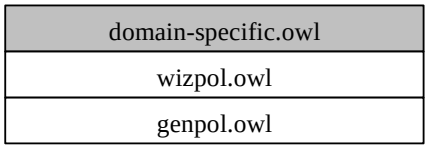

Figure 3. Policy Ontology Stack

The ontology framework describes the core language in an easily extensible manner, as well as reflecting the user interface support offered by the wizard. It defines only the structure of policy-related knowledge and not actual policies. The ontologies therefore contain no individuals or instances of ontology classes. Specific data values (e.g. trigger and action parameter arguments, condition values) are defined by the actual policies.

The ontologies *genpol* and *wizpol* are designed to be generic and reusable for any domain. Due to the transitive nature of OWL imports, a domain-specific ontology need only import *wizpol – genpol* is implicitly imported as well. The call control ontology extends the class hierarchy of *wizpol* to define additional subclasses and properties, together with applicable constraints. In particular, this includes the definition of domain-specific triggers, condition parameters, and actions – for call processing in the application described here.

To give a clearer understanding of the main components defined by the policy language, each ontology within the framework is described in the following subsections.

B. Generic Policy Language Representation

The generic policy language ontology, *genpol*, defines the core elements of the APPEL policy description language [8]. This ontology specifies a skeleton structure of classes and properties; this can be imported and extended within a domain-specific ontology. Contained within *genpol* 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 such as policy rules, events, conditions, actions, additional attributes, variables and operators. The relationships between these concepts describe named associations, inheritance properties and cardinality restrictions.

In outline, *genpol* defines the following main concepts and their relationships for call control policies:

- A *PolicyDocument* is the highest conceptual level of APPEL. It is defined to have zero or more *Policy* instances.
- A *Policy* is defined to have at least one *PolicyRule*, and must have *RequiredAttribute* instances. It may also have any number of *OptionalAttribute* instances.

- A *PolicyRule* may have zero or more *TriggerEvent* or *Condition* associations, but must have at least one *Action*.
- A *TriggerEvent* may be linked with a *TriggerArgument* using the *hasTriggerArgument* property restriction.
- A *Condition* must be associated with a *Condition Parameter*, *ConditionOperator* and *ConditionValue*. These are defined using the properties *hasConditionParameter*, *hasConditionOperator* and *hasConditionValue*, combined with a set of associated cardinality restrictions.
- An *Action* may be linked with an *ActionArgument* using the *hasActionArgument* property restriction.
- There are two types of operators in a policy: a *ConditionOperator* used within a *Condition*, and a *CombinationOperator* used to integrate two policy rules.

At the lowest level, *genpol* defines the minimum classes and properties required to create a domain-oriented specialisation of the policy language. In addition, automated ontology support is provided to the policy system. An in-depth description of *genpol* is presented in [4].

The policy system has many useful facilities related to policy definition, but which are not strictly part of the policy language. These additional constructs are modelled in the *wizpol* ontology as described in the next subsection.

C. Policy Wizard Representation

The ACCENT policy wizard supports a user-friendly means of creating and editing policies. Such a facility is key in supporting policy definition by non-technical users like ordinary subscribers. It is therefore an important aspect that must be captured by the ontology framework. The policy wizard incorporates a number of facilities that control and manipulate domain data prior to its display. Such facilities are not part of the policy language itself, but are useful in any domain-specific ontology intended for use with the policy system. This additional, wizard-related knowledge is defined in *wizpol* as a direct extension of *genpol*, thus specialising the core APPEL language for use with the policy wizard.

Examples of wizard-specific facilities include the categorisation of triggers, conditions, actions and operators. In addition, these are grouped by user level to match the subset of language functionality to the skill or authorisation level of a user. For example, administrative users see the whole of the language, while beginning users see a limited but useful subset. In outline, the extensions supported by *wizpol* include:

- Subclasses within each class hierarchy for the *genpol* classes *TriggerEvent*, *ConditionParameter* and *Action*. Four subclasses represent different user levels: *admin*, *expert*, *intermediate*, and *novice*. Another signifies *internal* policy system use.
- Subclasses *NamedTriggerEvent*, *NamedCondParam* and *NamedAction* for the *genpol* classes *TriggerEvent*, *ConditionParameter* and *Action* respectively, to support reasoning about the ontologies.
- Properties to associate categories with domain specialisations of triggers, conditions and actions, including *hasUserLevel* and *hasInternalUse*.
- Extensions to the list of operators defined within *genpol* according to the user level. For example, certain rule combination operators are relatively complex and are defined to be of use at *admin* or *expert* level only.

Collectively, *genpol* and *wizpol* form a base from which domain specialisations of the policy language can be defined.

D. Call Control Policy Language Specialisation

The call control ontology specialises the generic and wizard aspects of APPEL. In particular, the call control ontology defines the specific triggers, condition parameters and actions associated with call processing. The ontology for call control is described in detail by [3]. Figure 4 shows how *genpol* and *wizpol* classes are extended for call control.

In relation to specific extensions for trigger, condition parameter and action classes, the call control ontology also defines trigger and action arguments, status variables, and unit types (e.g. for cost or bandwidth). Whereas arguments and status variables are explicit language elements, unit types are intended for wizard display purposes. By incorporating unit type classes into the ontology, it is possible to describe how a value can be interpreted for the user. For example, a condition value such as bandwidth is measured using *KbpsUnitType*. Additionally, each trigger, condition and action is assigned various properties previously identified in *genpol* and *wizpol* for categorisation:

- The property *wizpol:hasUserLevel* associates each trigger, condition parameter and action with one or more user levels from *admin*, *expert*, *intermediate* and *novice*.
- The property *wizpol:hasInternalUse* defines certain triggers or actions as internal to the policy system. The *LogEvent* and *SendMessage* actions are examples.
- The properties *genpol:hasPermissibleParameter* and *genpol:hasPermissibleAction* are associated with each trigger to define which condition parameters and actions can be used in conjunction with the trigger in question within a policy rule. This ensures consistency of a trigger with its condition and action. For example, only a call trigger may have conditions on the caller and actions involving forwarding.

The effect of property restrictions on classes is that the categorisation of certain triggers, conditions and actions can be automatically inferred. As an example, the policy wizard can query the ontology to determine various triggers subsets: those available to expert users, those with a parameter argument, or those for use in conjunction with the *RejectCall* action. The ability to interrogate an ontology in this way offers more detailed knowledge than using a structural markup language like XML Schema to model the policy language.

Although the call control ontology is primarily intended to extend policy language constructs, unlimited additional knowledge can be included to describe aspects of call processing indirectly related to the policy language or wizard. Consequently, the ontology includes a variety of additional classes and properties to describe general telephony terminology. This includes the high-level concepts of *Call*, *CallAttribute* (e.g. topic, cost, type, priority), *CallType* (e.g.

international, emergency, conference, standard), *CallInitiatorAddress* and *CallDestinationAddress*. Such details also provide further insight into the call control domain when processed by non-policy system applications.

Generic Ontology Class	Call Control Ontology Class
genpol:TriggerEvent wizpol:NamedTriggerEvent	AddressAbsent, AddressAvailable, AddressPresent, AddressUnavailable, BandwidthRequest, Connect, ConnectIncomingCall, ConnectOutgoingCall, Disconnect, DisconnectIncomingCall, DisconnectOutgoingCall, ExternalGeneralEvent, NoAnswer, NoAnswerIncoming, NoAnswerOutgoing, Register, RegisterIncoming, RegisterOutgoing, StatusAway, StatusBusy, StatusFree, StatusHere
genpol:ConditionParameter wizpol:NamedCondParam	ActiveContent, Bandwidth, CallContent, CallCost, CallerCapability, CallerCapabilitySet, Callee, Caller, CallerDevice, CallerLocation, CallMedium,CallPriority, CallQuality, CallerRole, CallTopic, CallType, Date, Day, DestinationAddress, NetworkType, SignallingAddress, SourceAddress, Time, TrafficLoad
genpol:Action wizpol:NamedAction	AddCaller, AddMedium, AddParty, ConfirmBandwidth, ConnectTo, ForkTo, ForwardTo, LogEvent, NoteAbsent, NoteAvailable, NoteAvailability, NotePresence, NotePresent, NoteUnavailable, PlayAudioClip, RejectBandwidth, RejectCall, RemoveMedium, RemoveParty, SendMessage

Figure 4. Trigger, Condition Parameter and Action Classes

IV. Policy Conflict Detection and Resolution

Section II.B gave an overview of policy conflict in general. There follows a description of how ontologies support conflict handling within APPEL. It will be seen how this is modelled generically, and also specifically for call control policies.

A. Generic Policy Conflict Resolution

Conflicts among policies occur at run-time when simultaneously triggered policies propose conflicting actions. The process of detecting conflicts can be carried out statically (offline) or dynamically (online). Rather than hard-code policy conflict detection and resolution into the ACCENT system, APPEL deals with conflicts dynamically using resolution policies. This approach is far more complex and rigorous than any static, offline technique as it captures conflicts by analysing policies at run-time as they become eligible for execution. However, static handling of conflicts (such as at definition time within the policy wizard) is entirely feasible, although not currently implemented.

Detecting and resolving conflicts are separate steps, though they are both defined by resolution policies. A resolution policy is similar in structure to but different in content from a regular control policy. This subsection outlines ontology modelling of generic resolution policies, while subsection B demonstrates how this is extended for call control.

A resolution policy specifies what may trigger a conflict, any optional conditions, and resolving actions. The language for resolution policies follows the same structure as a regular policy, but with some small differences. Core resolution policy concepts are therefore defined within *genpol* (section III.B). In outline, a resolution policy is modelled as follows:

- A *ResPolicy* has zero or more *PolicyRule* instances.
- Each *PolicyRule* must have two or more *TriggerEvent* instances, zero or more *Condition* instances, and one or more *Action* instances.
- *TriggerEvent* instances in a resolution policy must be the *Action* instances of a regular policy, since conflict handling is triggered by the actions of regular policies.

Resolution policy actions can be generic or specific in nature. Generic actions apply to any domain. They resolve a conflict by choosing one of the conflicting actions, e.g. that of the superior user or of the earlier-defined policy. The policy server has in-built support for generic resolution actions such as *ApplySuperior* or *ApplyNewer*.

To help with conflict detection, *genpol* specifies a top-level class called *ActionEffect*. Subclasses in domain-specific ontologies (e.g. for call control) categorise regular policy actions using the restriction *hasActionEffect*.

B. Modelling Call Control Resolution Policies

The call control ontology specialises resolution policies through an extension of classes defined in *genpol*. In particular, it extends the list of resolution policy actions to include specific resolution actions for call control, e.g. *ApplyCaller* and *ApplyCallee*. In the event of conflict, these actions give priority to the policy associated with the caller or callee respectively. Specific resolution actions also include those of the application domain, e.g. forwarding or blocking for call control.

As resolution policy triggers are a combination of call control actions, the ontology creates subclasses of the

genpol:ActionEffect class to define specific categories for conflict handling. Each call control action is associated with one or more effect categories via the property restriction *genpol:hasActionEffect*. As an example, consider the actions and effects shown in Figure 5.

Action	Effect
AddCaller	PartyEffect, PrivacyEffect
AddMedium	MediumEffect, PrivacyEffect

Figure 5. Sample Effects in Call Control

Both the *AddCaller* and the *AddMedium* actions have a restriction linking them with *PrivacyEffect*. Therefore, it can be determined these actions may conflict as they share a common effect on the call environment. In separate work not reported here, this is used for automatic determination of conflict-prone policies.

V. Conclusion

The paper has outlined a novel approach to policy language definition using a framework of ontologies to model generic language constructs, as well as those specific to an application domain – call control. The approach has been used to support the ACCENT policy-based management system for handling call preferences. An ontology framework using OWL was designed to model APPEL, the policy description language used by the ACCENT system. The framework consists of two base ontologies, *genpol* and *wizpol*, together with a third ontology specific to call control.

The ontology framework describes the policy language in abstract terms. It has proven useful for two reasons. Firstly, modelling generic language aspects separately allows for easy extension of policy support for call handling, e.g. adding further triggers or actions without altering the core language. This saves time, promotes effective reuse, and gives greater scope for policy language revision. Secondly, the approach allows the policy system to be extended for new domains. The common ontologies (*genpol* and *wizpol*) may be readily used to create custom ontologies for new application areas.

The ontology framework also permits specialisation of conflict handling. Generic aspects of resolution policies are given by *genpol*, while domain-specific knowledge of conflicts is defined in specialisations of this – for call control here.

There are several ways the ontologies for call control may be used or extended, both within their intended field of policy-based call management and in other telecommunications contexts. The call control ontology includes call processing knowledge not directly related to the policy language. This information could be used in the ACCENT system by components other than the policy wizard, such as the policy server or the context system.

Due to the abstraction created by the ontology framework, generic aspects of the policy language and their specialisation can be developed independently. This enables greater scope for extension to both the policy wizard (within *wizpol*) and also to call control itself.

In related work by the authors and their colleagues, the approach is being extended to policy-based control of wind farms (http://www.prosen.org.uk), and to policy-based control of home care delivery (http://www.match-project.org.uk).

The call control ontology may also be used by other applications unconnected with the ACCENT system or even policies in general. OWL ontologies can be made available via a URL (http://www.cs.stir.ac.uk/schemas for the work reported here). As a result, the ontologies can be exploited by any application that can benefit from knowledge of call control.

Acknowledgements

Gemma Campbell was supported in this work by a studentship from the UK Engineering and Physical Sciences Research Council under grant C014804. The authors thank their colleagues on the PROSEN project for discussions that helped to shape the approach. Thanks are also due to the developers of the Protégé, Jena, Pellet and Racer Pro tools used in this work.

References

[1] ACCENT Policy-Based System. ACCENT project description http://www.cs.stir.ac.uk/accent, Nov. 2006.

[2] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell and K. Nahrstedt. A Middleware Infrastructure to Enable Active Spaces, *IEEE Pervasive Computing*, 1(4):74–83, Oct.–Dec 2002.

[3] G. A. Campbell. Ontology for Call Control, Technical Report CSM-170, CompSci & Maths, University of Stirling, Jun. 2006.

[4] G. A. Campbell. Ontology Stack for a Policy Wizard. Technical Report CSM-169, CompSci & Maths, University of Stirling, Jun. 2006.

[5] N. F. Noy and D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology, Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory, Mar. 2001.

[6] World-Wide Web Consortium. OWL Web Ontology Language Reference, Feb. 2004.

[7] World-Wide Web Consortium. Web Ontology Language Summary, Feb. 2004.

[8] S. Reiff-Marganiec and K .J. Turner. APPEL: The ACCENT Project Policy Environment/Language, Technical Report CSM-161, CompSci & Maths, University of Stirling, Jun. 2005.

[9] S. Reiff-Marganiec and K. J. Turner. The ACCENT Policy Server. Technical Report CSM-164, CompSci & Maths, University of Stirling, May 2005.

[10] K. J. Turner. The ACCENT Policy Wizard, Technical Report CSM-166, CompSci & Maths, University of Stirling, May 2005.

[11] K. J. Turner, S. Reiff-Marganiec, L. Blair, J. Pang, T. Gray, P. Perry and J. Ireland. Policy Support for Call Control, *Computer Standards and Interfaces*, 28(6):635–649, Jun. 2006.

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