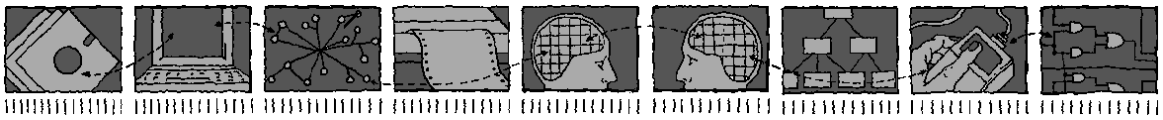
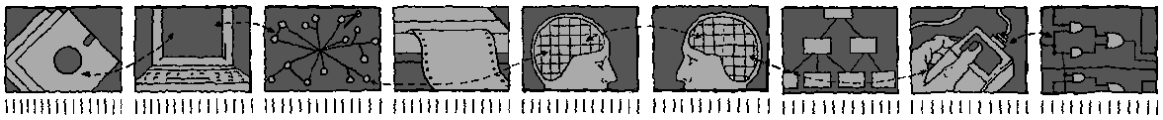


Department of Computing Science and Mathematics
University of Stirling



The ACCENT Policy Wizard

Kenneth J. Turner and Gavin A. Campbell

Technical Report CSM-166

ISSN 1460-9673

April 2014

Department of Computing Science and Mathematics
University of Stirling

The ACCENT Policy Wizard

Kenneth J. Turner and Gavin A. Campbell

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

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

Technical Report CSM-166

ISSN 1460-9673

April 2014

Abstract

The ACCENT project (Advanced Component Control Enhancing Network Technologies) developed a practical and comprehensive policy system for call control/Internet telephony. The policy system has subsequently been extended for management of sensor networks/wind farms and of home care/telecare.

This report focuses on a web-based policy wizard that acts as the primary interface between end users and the policy system. The policy wizard has an intimate knowledge of the APPEL policy language (Adaptable and Programmable Policy Environment and Language). The wizard allows end users to create policies using near-natural language without knowing or seeing XML, and to upload them to the policy system. The wizard also provides a number of convenience functions such as predefined policy templates, editing and activating existing policies, and defining policy variables.

Relative to the version of December 2005, this Technical Report has been updated as follows to reflect changes in the policy wizard:

- The whole report has been updated to reflect later work on the PROSEN and MATCH projects. As a result, the ACCENT and APPEL acronyms have changed. Call control, however, remains are the primary illustration of the approach in this report.
- Chapter 1 is now named 'Introduction', and a brief 'Conclusion' chapter has been added in section 4.
- Chapter 2 has been updated to sheet screenshots of the new policy wizard. The wizard now handles resolution policies.
- The wizard now makes use of ontologies, as described briefly in section 3.1. All domain-specific knowledge is held outside the wizard, so that largely common code can be used across all domains. As a result, the wizard configuration now also refers to the POPPET server.
- Section 3.6 describes a new code structure that allows different versions of the wizard to coexist.
- A brief explanation has been given in section 3.8 of what is involved in supporting a new application domain with the wizard.

Relative to the version of April 2009, this Technical Report has been updated as follows to reflect changes in the policy wizard:

- Section 1 has been revised to reflect the support of goals.
- Separate screenshots have now been provided for regular users and administrative users in sections 2.2 and 2.3. In particular, the definition of goals and prototypes has now been illustrated.
- Section 3.2 now mentions use with Pax Web/Jetty as well as with Tomcat.
- Section 3.5 now describes the use of templates for goals and prototype policies, and a new suffix for template resolution policies.
- Section 3.6 has been slightly revised to described the current structure of the code.
- Section 3.8 has been slightly revised to describe what is involved in supporting a new domain.

Relative to the version of June 2011, this Technical Report has been updated as follows to reflect changes in the policy wizard:

- Figure 2.4 in section 2.2 now has template policies in a different order.

- Section 3.2 now has a revised codebase for audio recording, compatible with later versions of Java.

Relative to the version of February 2013, this Technical Report has been updated as follows to reflect changes in the policy wizard:

- The mapping files described in section 3.3 are now always internal to the wizard.

Relative to the version of April 2013, this Technical Report has been updated as follows to reflect changes in the policy wizard:

- The new capability for checking policies is now mentioned in sections 2.2 and 2.3.
- The new *policy.message.port* property is now mentioned in section 3.3.

Contents

List of Figures

Chapter 1

Introduction

The ACCENT project (Advanced Component Control Enhancing Network Technologies, http://www.cs.stir.ac.uk/accent) developed a practical and comprehensive policy system for call control/Internet telephony. The policy system has subsequently been extended for management of home care/telecare on the MATCH project (Mobilising Advanced Technologies for Care at Home, http://www.match-project.org.uk). The policy system has also been extended for management of sensor networks/wind farms on the PROSEN project (Proactive Control of Sensor Networks, http://www.cs.stir.ac.uk/~kjt/research/prosen).

This report focuses on a web-based policy wizard that acts as the primary interface between end users and the policy system. The policy wizard has an intimate knowledge of the APPEL policy language (Adaptable and Programmable Policy Environment and Language, http://www.cs.stir.ac.uk/appel). The wizard makes use of domain-specific ontologies so that it can be used in any application. The wizard allows end users to create goals and policies using near-natural language without knowing or seeing XML, and to upload them to the policy system. The wizard also provides a number of convenience functions such as predefined templates, editing and activating goals and policies, and defining policy variables. Besides regular policies, the wizard supports prototype policies for underpinning goals and resolution policies for handling conflicts among policy actions.

[4, 5, 9] give some general background to the ACCENT project. There are technical reports describing the ACCENT Policy System [7], the ACCENT Policy Server [6], and the APPEL Policy Language [8]. Ontologies for the policy wizard are discussed in [1, 2, 3]. These and other papers are available from http://www.cs.stir.ac.uk/accent.

Chapter 2

Policy Wizard User Interface

2.1 General Principles

The policy wizard is not a wizard in the sense of a program that takes the user through a well-defined task such as creating a form letter or creating an Internet connection. However it is a wizard in that it provides a user-friendly interface to a complex technical task (defining goals and policies in XML form). The wizard is the primary interface to the policy system for ordinary end users. For the wizard described in this report, important aspects of its design include:

- The wizard is web-based. This means that it can be used from anywhere, including away from the user's normal base. Other wizards have been prototyped using voice-based input (VoiceXML) and digital pen-based input (Anoto, Logitech).
- The wizard supports multiple application domains (currently call control, home care and sensor networks). However, one instance of the underlying policy system (policy server, policy store) can support only one domain at a time. Each domain therefore needs a separate instance of the policy system (e.g. on different computers).
- The wizard is multilingual (currently English, French and German). This allows the user to use the policy system irrespective of the user's preferred language.
- The wizard supports multiple levels of expertise (beginner, intermediate, expert, administrator). This allows a beginning user to see the minimum of the policy language, but an expert to see the full depth of its capabilities.
- The wizard has extensive help when defining goals and policies. When a field has to be filled in, hints are provided. Hovering over a link provides a 'tool tip'. Online help is also provided in the user's preferred language.

Since the wizard is designed to be user-friendly, little needs to be done here to explain the interface. The wizard does not (currently) support 'undo'. It is therefore suggested that goals and policies be created and checked step-by-step. If a major error is made during editing, click Cancel and start again.

The wizard is domain-independent. However, for illustration the screenshots in section 2.2 for normal users focus on its use for call control/Internet telephony, while the screenshots in section 2.2 for administrative users focus on home care/telecare. Use of the wizard is very similar in all application domains.

2.2 Normal Users

- The login screen in figure 2.1 is straightforward.
- The main menu in figure 2.2 appears after logging in.
- Choosing 'Existing Policy' leads to figure 2.3, where an existing policy can be selected for editing, enabling, disabling, or deletion.

Figure 2.1: Login Screen

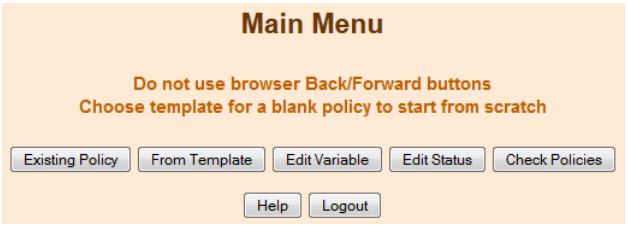


Figure 2.2: Main Menu

- Choosing 'From Template' leads to figure 2.4, where a number of predefined policies can be selected and edited. Note that a blank policy is created from a special template. Templates may contain values prefixed by '?'; these must be filled in by the user before the policy is saved.
- Choosing 'Edit Variables' leads to figure 2.5, where an existing variable can be selected for editing or deletion. Variables are created normally created in text format.

  For use in call control, audio clips are also supported as a special kind of variable value: WAV format, i.e. 16-bit uncompressed PCM (Pulse Code Modulation). Editing an audio clip is illustrated in figure 2.6. This provides buttons for record (red circle), stop (blue square) and play (green triangle). The meter bar shows the audio level during recording or the progress through the clip while playing.
- For call control, choosing 'Edit Status' leads to figure 2.7, where the user's availability, presence and profile can be edited. There are simple check boxes (ticks) for setting availability and presence. Alternatively, specific values can be filled in for these to indicate topics that the user is willing to be called about and the user's location. The profile is used to enable a group of policies quickly.
- Choosing 'Check Policies' leads to figure 2.8, where past and future policy execution can be checked. A policy can be selected from the drop-down box and then checked to see when it was performed. A trigger

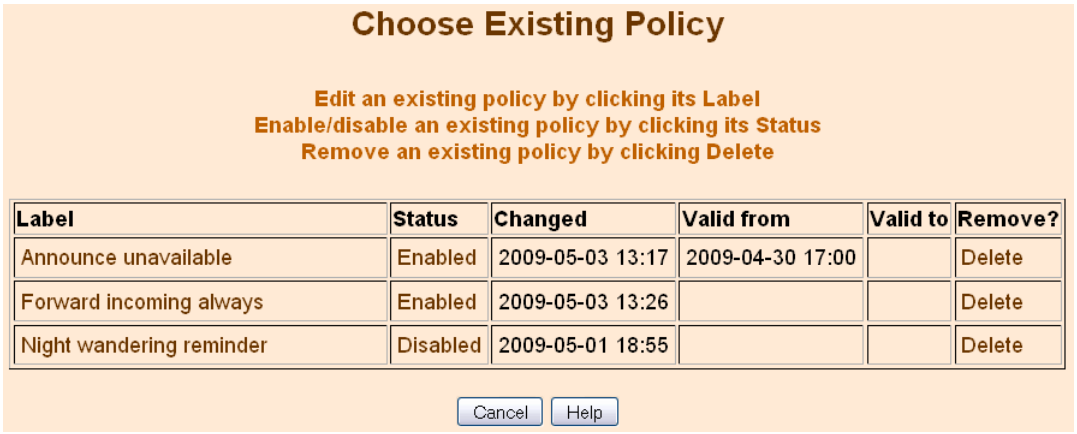


Figure 2.3: Existing Policies

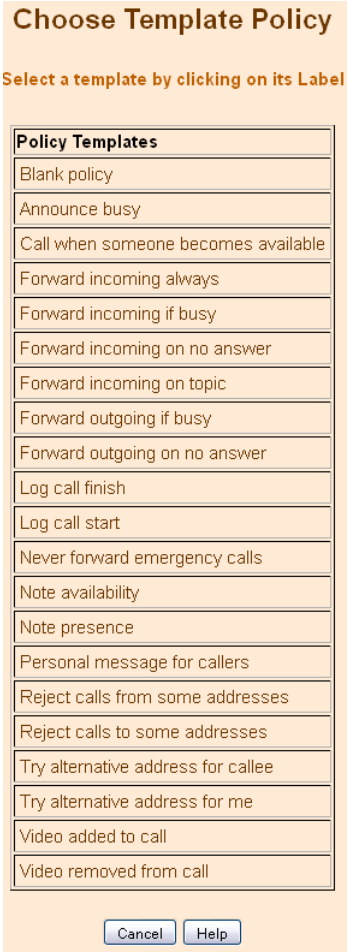


Figure 2.4: Template Policies

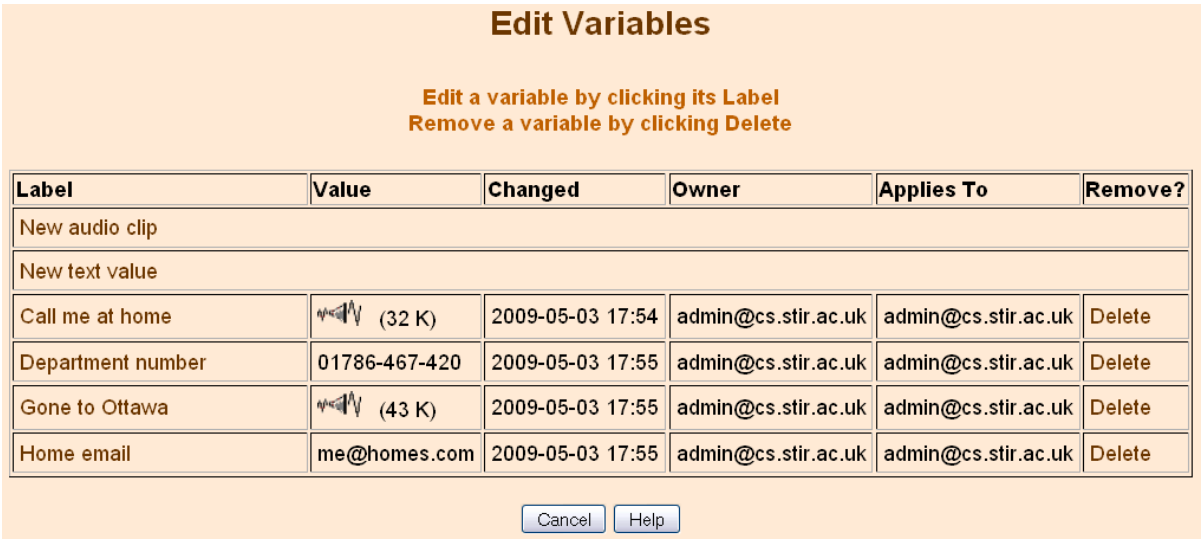


Figure 2.5: Edit Variables

Figure 2.6: Edit Audio Clip

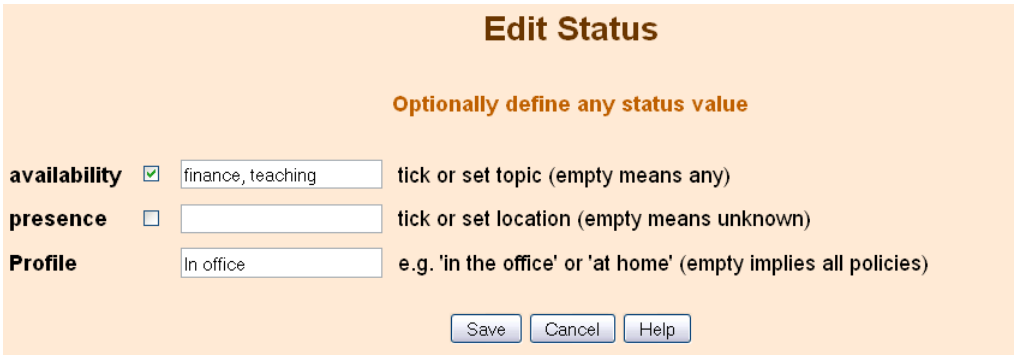


Figure 2.7: Edit Status

can be defined by clicking on the link. Optional variables can also be added for a trigger by clicking on the '· · ·' symbol; once a variable is defined it can be edited (but not deleted). The effect of a trigger can then be checked. An action can be defined by clicking on the link. If the absence of an action is to be checked, select 'not' from the drop-down list. Why an action was (not) performed can then be checked. In each case, an explanation is given in a pop-up window (that may need to be enabled in the web browser). *Note that only the immediately computable results of a policy will be displayed.* Output following expiry of a timer started by the policy will not be shown. However a policy can be triggered by setting date, time or time trigger variables.

- Choosing an existing or template policy leads to figure 2.9, where the policy can be edited. The applicability, preference and rules of a policy are independently edited. The only complex aspect is adding or deleting parts of a rule. The '· · ·' symbol after a trigger, condition or action allows a further trigger, condition or action to be added with a specified combination. The '· · ·' symbol after a rule allows a further rule to be added with a specified combination. In this way, complex tree structures can be created. To remove the first part of a combination (trigger, condition or action), set it blank. To remove the second part of a combination, click on the name of the combination and set it blank.

2.3 Administrative Users

- After logging in, an administrator sees the extended main menu in figure 2.10. This adds the options to edit users, resolution policies, goals and prototype policies.
- Figure 2.12 shows that an administrator also sees templates for resolution policies, prototype policies and goals.

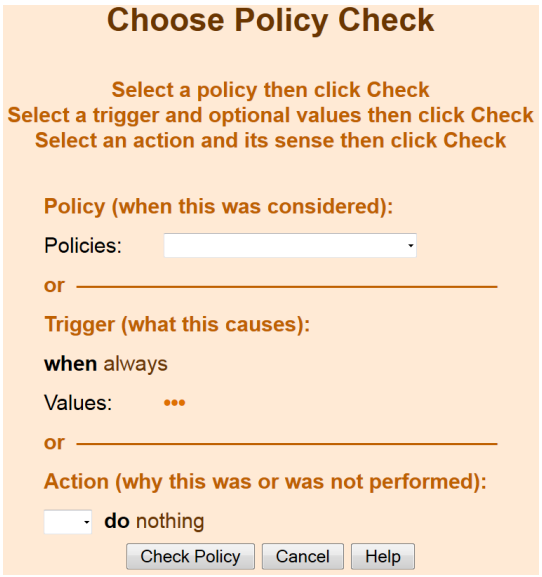


Figure 2.8: Check Policies

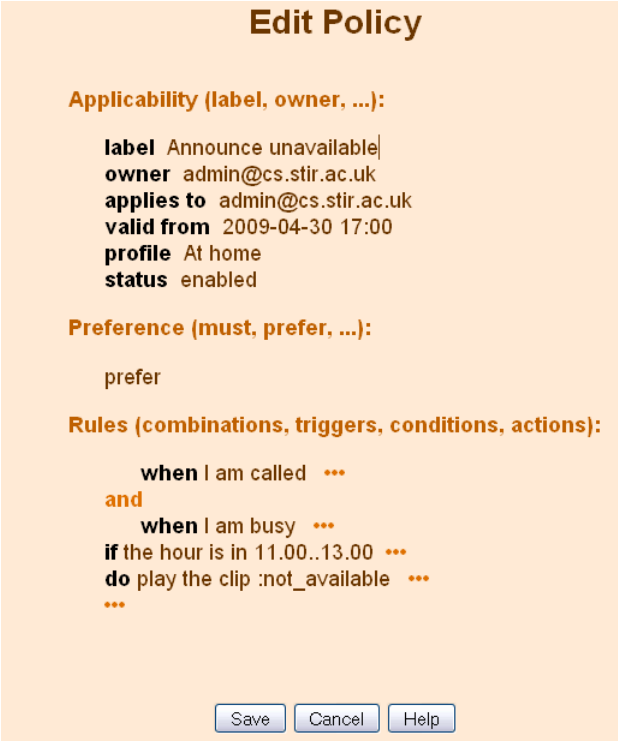


Figure 2.9: Edit Policy

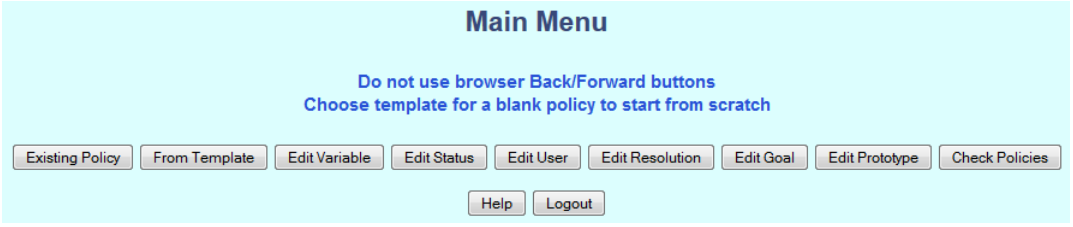


Figure 2.10: Main Menu

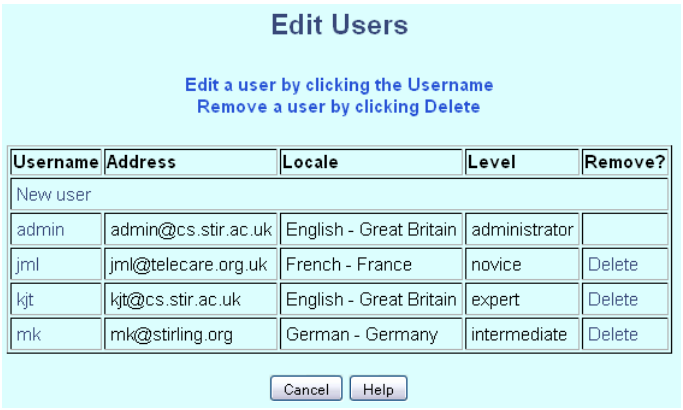


Figure 2.11: Edit Users

- An administrator can choose 'Edit Users', leading to figure 2.11, where user accounts can be created, edited or deleted. When a new user is created, policy variables are set to indicate the user is unavailable, absent, and has an empty profile. When a user is deleted, all policies and variables owned by this user are removed. The 'admin' account is required and cannot be deleted, although it can be edited.

- An administrator can manage resolution policies. This is broadly similar to editing a regular policy, as shown in figure 2.13. One difference is that *preference* and *variable* values are mostly used in place of concrete values. Resolution policies can also have generic resolution actions as well as regular policy actions.

- An administrator can manage prototype policies. This is broadly similar to editing a regular policy, as shown in figure 2.14. The main difference is that prototype policies have effects. These are individually added, edited and removed in much the same way as the rest of the policy wizard. In particular, the '$\cdots$' symbol should be clicked to add a new effect. Editing an effect is illustrated in figure 2.15, where the effect variable (system variable), operator and value are chosen. For an effect that must be exclusive of other simultaneous effects, the operator is 'increases/decreases alone by'.

- An administrator can manage goals. Initially this leads to a list of currently defined goals, as shown in figure 2.16. Because there can be multiple goals, each is given an individual weight (in the range 0.1 to 10.0). This is set by dragging the slider for the goal, followed by 'Save' for the whole set of goals.

  Clicking on the label of a goal allows it to be edited, as shown in figure 2.17. This is similar to editing a regular policy, but with simplifications. A goal does not have a profile or valid from/to dates. Goals have no triggers as they always apply. Goal conditions can use only general and environmental information such as the current day or interior temperature. A goal has only one action that optimises one or more goal measures. These are individually added, edited and removed in much the same way as the rest of the policy wizard. In particular, the '$\cdots$' symbol should be clicked to add a new goal measure. Internally, the policy wizard automatically calculates a weight for each goal measure.

  Editing a goal measure is illustrated in figure 2.18, where the measure variable (system variable), operator and optional value are chosen. A variable is defined as positive if it works towards goal, or as negative if it detracts from a goal. A threshold can be set if the variable affects a goal only above a certain value. Actual values above this threshold will affect the goal (e.g. a noise level above 70dB might be considered as negative). The ideal value for a variable can also be set. Actual values below or above this ideal will affect the goal (e.g. a temperature below or above 21C might be considered as negative).

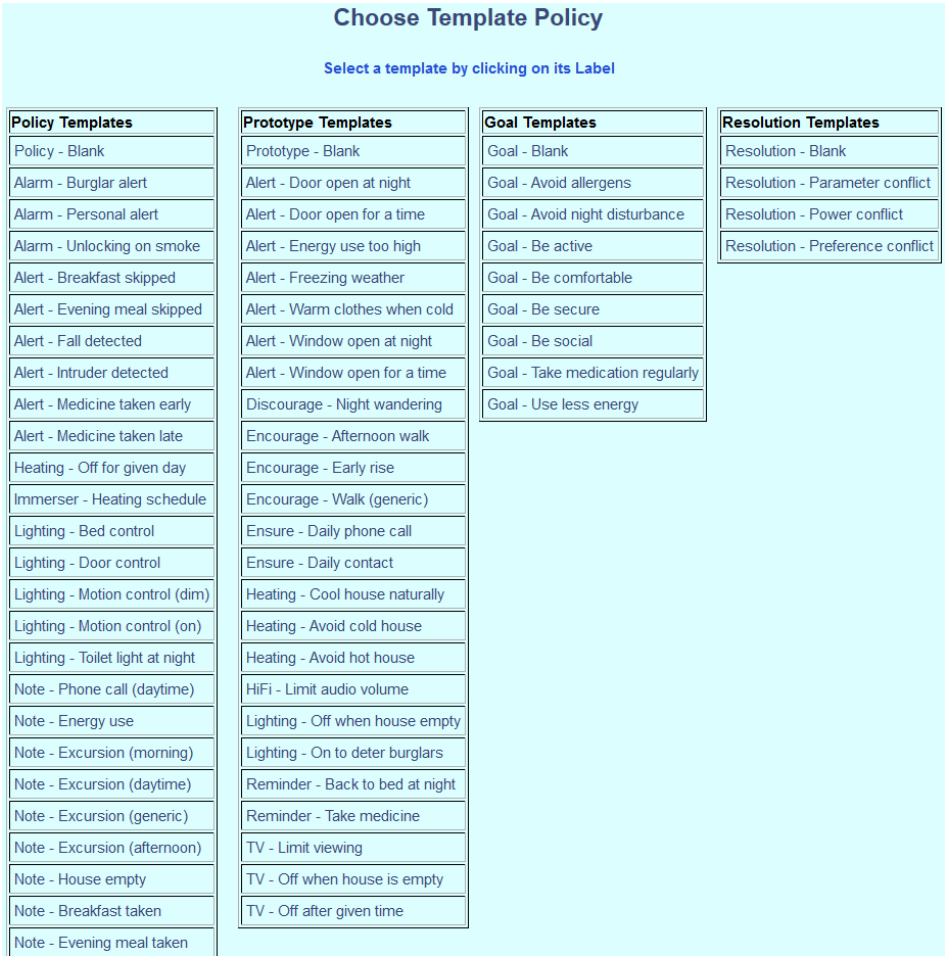


Figure 2.12: Template Policies

Figure 2.13: Edit Resolution

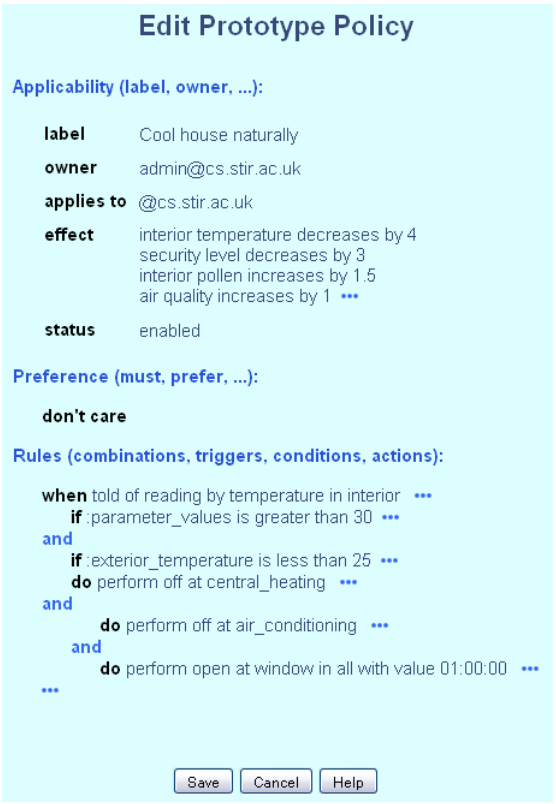


Figure 2.14: Edit Prototype

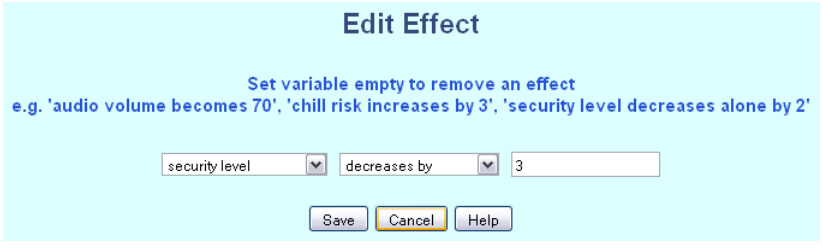


Figure 2.15: Edit Effect

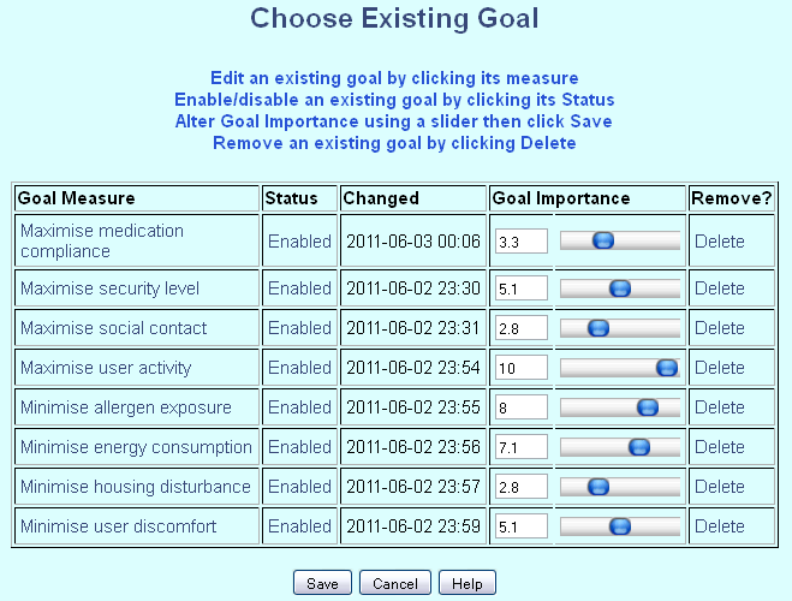


Figure 2.16: Edit Goal

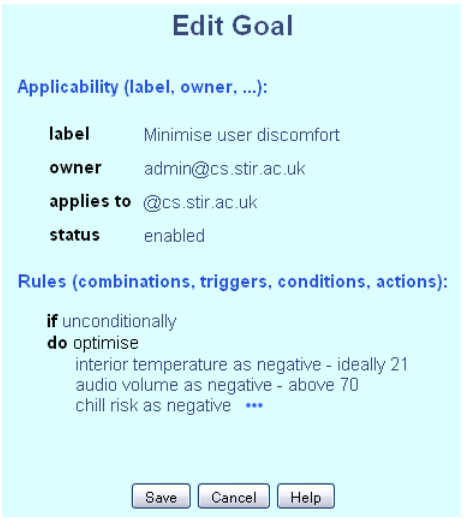


Figure 2.17: Goal List

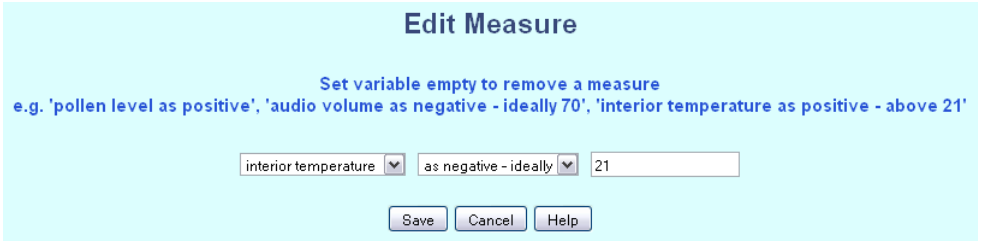


Figure 2.18: Edit Measure

Chapter 3

Policy Wizard Internals

3.1 Use of Ontologies

The wizard is designed to be used in multiple domains. It obtains domain-specific information from a separate ontology server to underpin operation of the wizard. POPPET (Policy Ontology Parser Program Extensible Translation [3]) provides an neutral interface for retrieval of information held in ontologies. Specifically, POPPET supports programmatic access to ontologies defined using OWL (Web Ontology Language [11]). Sample ontologies are described in [1, 2]. In fact a number of ontologies are used:

genpol: This is the generic policy ontology that applies to policies in any domain.

wizpol: This is the wizard policy ontology that contains additional information about how the wizard operates.

call_control, etc.: These are domain-specific ontologies that contain information about domain triggers, conditions, actions, units, user levels, etc.

When POPPET is instantiated for a particularly ontology, it builds a model of the ontology to support queries from external programs. This is performed through Java RMI (Remote Method Invocation), which enables the wizard to access the ontology model remotely. POPPET can be instantiated multiple times on the same system to support a variety of ontologies concurrently.

POPPET must be started before any attempt is made to use the wizard. As discussed below, the wizard has a configuration file that defines the POPPET server and ontology in use.

3.2 Integration with Other Tools

The policy wizard uses a set of JSPs (Java Server Pages). It therefore requires a servlet container such as Apache Tomcat (http://tomcat.apache.org), Mortbay Jetty (www.mortbay.org) or Pax Web (http://wiki.ops4j.org/display/paxweb/Pax+Web). Tomcat and Jetty are self-standing containers, while Pax Web bundles the Jetty container for OSGi (Open Services Gateway initiative, http://www.osgi.org). The wizard has been tested with Tomcat versions 4.1.27 onwards, Jetty versions 7.4.2 onwards, and with Pax Web 1.0.2 onwards. Tomcat requires an application context to be set up. In Tomcat 5.X, the file Tomcat/conf/Catalina/localhost/wizard.xml might have contents:

```
<Context path="/wizard" docBase="wizard installation directory"
 debug="0" reloadable="true" crossContext="false">
  <Logger className="org.apache.catalina.logger.FileLogger"
   directory="/WEB-INF/logs" prefix="wizard " timestamp="true"/>
</Context>
```

where the wizard installation directory is Tomcat/webapps/wizard, for example.

The JSPs are supplemented by Java code in package uk.ac.stir.cs.accent.wizard. Audio clips are handled by an adaptation of public domain code (http://dannyayers.com/2000/sound.htm).

To use the policy wizard requires a web browser with a recent version of HTML, CSS (Cascading Style Sheets) and JavaScript. The wizard has been tested with Microsoft Internet Explorer 6.0 onwards, FireFox 2.0 onwards,

Opera 7.5 onwards, and Safari 3.1 onwards. The web browser should be JavaScript-enabled (for checking and form-handling) and must be Java-enabled (if audio clips are to be used).

Multiple logins from the same web browser on the same client system are discouraged. As a result of browser/JSP limitations, this may result in the same session being used in all windows. With Internet Eplorer, it is possible to create a new session to avoid this. Although the policy wizard supports multiple domains, it is undesirable to do this with the same policy system instance. Goals, policies and variables created in different domains would then all be visible to the policy server and wizard.

If a user wishes to record audio clips using the policy wizard, the relevant Java security information must be set up. Java now requires signed applets to be used. Currently the clip recorder uses a self-signed certificate that must be imported before Java will trust it. A user with administrator privileges should do this by running the DOS batch script *security/import.bat* (having first set the relevant JRE location in this).

The policy wizard requires a table called users within the accent database of the policy database server. This table must contain at least an entry for the policy wizard administrator (named admin). Several users may be designated as administrators. The sample file lib/user_setup.sql is available from the ACCENT files as an example for this setup.

The policy wizard administrator requires an individual email address (e.g. admin@cs.stir.ac.uk) separate from any regular email address. The administrator creates policy system users, who then log in under their allocated username and password. These users would normally be from the same domain (e.g. cs.stir.ac.uk), but in principle could be from multiple domains.

3.3 Configuration

The policy wizard is configured by property files. The location of these is determined by whether the wizard is running as a bundle (system property *osgi.root* is defined) or as a normal webapp (this property is not defined). In the following, *root* might be *C:/usr/local/knopflerfish/osgi*, *domain* might be *home_care*, and *language* might be *en-GB*.

Bundle: The database property file is external to the wizard, while the mapping and wizard property files are bundled with the wizard and are therefore internal.

root/PolicyWizard.*domain*.database.properties
PolicyWizard/WEB-INF/lib/*domain*/mapping.properties
PolicyWizard/WEB-INF/lib/*domain*/*language*/wizard.properties

Webapp: The database, mapping and wizard property files are bundled with the wizard and are therefore internal.

PolicyWizard/WEB-INF/lib/*domain*/database.properties
PolicyWizard/WEB-INF/lib/*domain*/mapping.properties
PolicyWizard/WEB-INF/lib/*domain*/*language*/wizard.properties

The policy wizard uses a Java properties file database.properties to define information about various servers. A typical configuration file looks like the following; substitute locally meaningful values here. Note that ontology URLs must end with '#'.

```
# Home care wizard properties when running on local machine

# System administrator email address

admin.email             kjt@cs.stir.ac.uk

# Name of the database host (e.g. ″localhost″), username (e.g. ″home_care″) and
# password, name of users database table (e.g. ″home_care″)

users.host              localhost
users.username          home_care
users.password          -----
users.table             home_care
```

```
# Name of the policy server host (e.g. ″localhost″) and upload port number
# (e.g. ″9999″)

policy.host                 localhost
policy.message.port         9998
policy.upload.port          9999

# Name of the ontology server host (e.g. ″localhost″) and ontology name
# (e.g. ″home_care″)

poppet.host                 localhost
poppet.ontology.name        home_care

# URL for generic, wizard and home care ontologies (note: append '#' to URI)

ontology.policy.generic     http://www.cs.stir.ac.uk/schemas/genpol.owl#
ontology.policy.wizard      http://www.cs.stir.ac.uk/schemas/wizpol.owl#
ontology.policy.domain      http://www.cs.stir.ac.uk/schemas/home_care.owl#

# Prefix of system policies (instantiated prototypes) and system variables

system.prefix               !
```

The policy wizard uses a Java properties file mapping.properties to map policy language terms to policy wizard terms. A typical configuration file looks like the following.

```
absent                      policy.absent
active_content              policy.active.content
add_medium                  policy.add.medium
...
locale.de-de                language.de.de
locale.en-au                language.en.au
...
stage.0                     policy.novice
stage.1                     policy.intermediate
...
```

For each natural language, the policy wizard uses a Java property file wizard.properties to obtain information about the mapping from policy wizard phrases to the language. A typical configuration file looks like the following:

```
# The following are interpreted by the browser and so may use HTML entities for
# special characters

aspect.applicability        Applicability (label, owner, ...)
aspect.preference           Preference (must, prefer, ...)
aspect.rule                 Rules (combinations, triggers, conditions, actions)
...

# The following are interpreted by JavaScript and so cannot use HTML; escape
# a ″′″ character with ″\\″

error.Address               Define address in ″person@domain″, telephone number,
                            or ″:variable″ format
error.address               Define address in ″person@domain″ or ″:variable″ format
error.applies.to            Define ″applies to″ in ″person@domain″ format
error.audio                 audio
...
```

Note that some properties intentionally differ in their capitalisation (like error.Address, error.address).

The label for a particular trigger or action category can have a prefix character and optionally a suffix character. When the wizard displays a trigger or action, the prefix (and suffix) are transferred to the actual argument. For

Element	Novice	Intermediate	Expert
Modality	must, must_ not, prefer, prefer_ not, should, should_ not		
Combinator	*condition*: and, or *trigger*: else, or, sequential		*trigger*: and, andthen, guarded, orelse, parallel, unguarded
Trigger	absent(), available(), connect, connect_ incoming, connect_outgoing, disconnect, present(), unavailable()	absent(address), available(address), disconnect_ incoming, disconnect_outgoing, no_ answer(period), no_ answer_ incoming(period), no_ answer_outgoing(period), present(address), unavailable(address)	bandwidth_ request, event, register, register_ incoming, register_outgoing
Condition	caller, date, time, topic	call_ type, cost	active_content, bandwidth, call_content, callee, capability, capability_ set, destination_ address, device, location, medium, network_ type, priority, quality, role, signalling_ address, source_ address, traffic_ load
Action	forward_ to(Address), note_ availability(true), note_ availability(false), note_ presence(true), note_ presence(false), play_clip(audio), reject_call(reason), send_ message(address,message)	log_ event(message), note_ availability(topic), note_ presence(location)	add_caller(method), add_ medium(medium), add_ party(Address), confirm_ bandwidth, connect_ to(Address), fork_ to(Address), reject_ bandwidth(limit), remove_ medium(medium), remove_ party(Address)

Figure 3.1: Language Levels for Call Control

example, label *!warning* would cause argument *hazard* to be displayed as label *warning* with argument *!hazard*. Similarly, label *[list]* would cause argument *1,2 3* to be displayed as label *list* with argument *[1,2 3]*.

APPEL is translated into natural language in two steps. For example 'connect incoming' is first converted into 'policy.connect.in' using the mapping properties. This is then translated into (say) 'I am called' using the English wizard properties. A Login page is defined for each language in the wizard directory. One of these should be selected as the default for the locale, and linked or copied to index.jsp.

3.4 Language Levels

The policy wizard restricts the use of certain language features depending on the user's defined level (called 'stage' internally). These levels are domain-specific, and so are defined by the domain ontology. As an example, the levels for call control are shown in figure 3.1. [8] describes the meaning of these language terms.

The columns are cumulative, e.g. an intermediate user can do everything a novice can plus whatever is listed in this column. An administrator is the highest level. The only difference from expert level is that an administrator can see and alter the owner and applies_to fields, i.e. can define policies and policy variables for others. An administrator can also, of course, manage other users.

3.5 Templates

Template policies are XML files conforming to the APPEL schema, except that owner and applies_to are meaningless and are left empty. The files must be validated outside the policy wizard. Template policies exist in each locale directory (e.g. 'en-CA'). In principle they could be similar for each locale, though it is possible to have a different set according to local custom. The id values must be translated into the relevant language. In addition, the names of template variables must be rendered in the relevant language (e.g. '?address' in English, '?adresse' in French). If the language requires special characters (e.g. accented ones), be sure to save a template file in UTF-8 format.

Template regular policies are suffixed by the user level for which they are intended ('_0.xml' novice, '_1.xml' intermediate, '_2.xml' expert). Template goals are suffixed by '_g.xml', prototype policies by '_p.xml', and template resolution policies by '_r.xml'. Templates are sorted by filename, though their id is shown in the list. The filename should therefore closely match the id. Templates that should appear at the start of the list (e.g. a blank document) should have their filenames prefixed by '0'. Only templates appropriate to the user level are shown when 'From Template' is selected.

3.6 Code Organisation

The code is organised in the following way, conforming to the normal structure for servlets but split up for each application domain.

.classpath, .project: Eclipse build files

domain: sub-directories with JSP files, CSS stylesheet, wizard logo, slider files, and JAR file for audio clip classes (call control)

WEB-INF: all the supporting files

WEB-INF/classes: the compiled Java files in package uk.ac.stir.cs.wizard

WEB-INF/doc: JavaDoc for the Java source files in the clip and wizard sub-directories

WEB-INF/lib: the database properties file, the mapping properties file, JAR files, and application domain sub-directories

WEB-INF/libextra: extra JAR files for use in compiling (but not running) the code

WEB-INF/logs: container log files

WEB-INF/src: the Java source files in sub-directories uk/ac/stir/cs/clip (audio clips) and uk/ac/stir/cs/wizard (policy wizard); there is a simple build script to recompile everything

WEB-INF/web.xml: the web deployment descriptor

The *domain* and lib directories are subdivided according to the application domain: call_control, home_care and sensor_network currently. The application domains under lib are also subdivided according to language. For example, 'en-GB' ('English – Great Britain') contains the wizard properties file and predefined templates for this language.

3.7 Internationalisation

The policy wizard is designed to be multilingual. Suppose that it is required to add Swiss German ('de-CH') to the list of supported languages. The steps required are as follows. If the language requires special characters (e.g. accented ones), be sure to save the files in UTF8 format.

- Create the sub-directory de-CH in the relevant application domain sub-directory of lib. This must contain the file wizard.properties and templates for this language. If this is a variant on an already supported language (e.g. de-DE), the files from this can be copied and adjusted. If this is a completely new language, all terms required by the policy wizard will have to be translated. This can be tricky if the result is to be grammatically correct (e.g. nouns, adjectives and verbs agree). Judicious choices in the translation can make this possible for many (though not all) languages. The templates predefined for English can mainly be copied as they are. The main change required is to render the comments, keys and values in the new language.
- Create the file Login-de-CH.jsp in the relevant application domain sub-directory. This will be almost identical to the existing login files, but a couple of phrases need to be rendered in the new language. The locale also needs to be set as de-CH.
- Create the file Help-de-CH.jsp in the relevant application domain sub-directory. This may be a variation on an existing language help file (Help-de-DE.jsp for example), or a completely new translation.
- Edit the mapping properties file in the relevant lib sub-directory to add the new locale de-CH and its equivalent policy wizard term:

  locale.de-CH language.de.ch

  Note the use of '-' in locales but '.' in policy wizard terms.
- In the wizard properties file for *every* application domain and language, define the translation of language.de.ch. For example, in English this will be 'German – Switzerland', in French 'Allemand – Suisse', and in German 'Deutsch – Schweiz'.

3.8 New Domains

Extending the wizard for other application domains is a much more substantial exercise. The code of the wizard does not need to be altered, but the following need to be defined:

- An ontology for the new domain will be required. This might be based on one of the existing examples such as call_control.owl. An ontology editor such as PROTÉGÉ (http://protege.stanford.edu) is suggested for this.
- The JSPs must be modified to reflect the new domain. A new directory must be created at the top level to parallel those for existing domains.
- The database.properties configuration file must refer to the ontology created for the new domain.
- The mapping.properties and wizard.properties configuration files must be edited to refer to relevant terms in the new domain.

Chapter 4

Conclusion

The policy wizard described in this report has been successful in allowing non-technical users to define goals and policies. Being web-based, it can be used anywhere; for example, a user can remotely modify policies for handling calls. By using near-natural language, the wizard can support users in their local language. Other conveniences such as templates and policy variables make it possible for an administrator to customise the wizard for local usage.

The use of domain-specific ontologies allows the policy wizard to operate in a wide variety of applications. Currently, the policy wizard (and system) have been used to manage call control/Internet telephony, home care/telecare, and sensor networks/wind farms. It is believed that the approach is generic and can be applied in many other management applications.

Two other experimental versions of the policy wizard have been created, though they are not described here. One of these makes use of VoiceXML [10] to allow policy contents to be read out, and template policies to be instantiated with specific parameters. Another makes use of digital pen and paper (from Anoto or Logitech) to allow template policies to be completed through filling in paper forms.

References

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

[2] Gavin A. Campbell. Ontology stack for a policy wizard. Technical Report CSM-169, Computing Science and Mathematics, University of Stirling, UK, June 2006.

[3] Gavin A. Campbell. Overview of policy-based management using POPPET. Technical Report CSM-168, Computing Science and Mathematics, University of Stirling, UK, June 2006.

[4] Stephan Reiff-Marganiec and Kenneth J. Turner. Use of logic to describe enhanced communications services. In Doron A. Peled and Moshe Y. Vardi, editors, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE XV)*, number 2529 in Lecture Notes in Computer Science, pages 130–145. Springer, Berlin, Germany, November 2002.

[5] Stephan Reiff-Marganiec and Kenneth J. Turner. A policy architecture for enhancing and controlling features. In Daniel Amyot and Luigi Logrippo, editors, *Proc. 7th Int. Conf. on Feature Interactions in Telecommunications and Software Systems*, pages 239–246. IOS Press, Amsterdam, Netherlands, June 2003.

[6] Stephan Reiff-Marganiec, Kenneth J. Turner, Lynne Blair, and Feng Wang. The ACCENT policy server. Technical Report CSM-164, Computing Science and Mathematics, University of Stirling, UK, August 2013.

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

[8] Kenneth J. Turner, Stephan Reiff-Marganiec, Lynne Blair, Gavin A. Campbell, and Feng Wang. APPEL: Adaptable and Programmable Policy Environment and Language. Technical Report CSM-161, Computing Science and Mathematics, University of Stirling, UK, April 2014.

[9] Kenneth J. Turner, Stephan Reiff-Marganiec, Lynne Blair, Jianxiong Pang, Tom Gray, Peter Perry, and Joe Ireland. Policy support for call control. *Computer Standards and Interfaces*, 28(6):635–649, June 2006.

[10] VoiceXML Forum. *Voice eXtensible Markup Language*. VoiceXML Version 2.0. VoiceXML Forum, Piscataway, New Jersey, USA, January 2003.

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