Natural Language Expression of User Policies in Pervasive Computing Environments

Julie Weeds*, Bill Keller*, David Weir*, Ian Wakeman†, Jon Rimmer† and Tim Owen†

*Natural Language Processing Group
†Networks Laboratory
Department of Informatics, School of Science and Technology
University of Sussex, Brighton, BN1 9QH, UK
{juliewe,billk,davidw,ianw,jonr,timo}@sussex.ac.uk

Abstract

The pervasive computing environments of tomorrow will typically consist of a large heterogeneous collection of networked services. In an ongoing research project, we are exploring ways to enable non-technical users to configure their environment. Our architecture includes an ontology that precisely describes the available services, a formal language for defining user policies and a middleware implementation of formal policies. In this paper, we analyse how existing natural language technologies can be applied to bridge the gap from a natural language description of user policies to a formal representation.

1. Introduction

The pervasive computing environments of tomorrow will typically consist of a large heterogeneous collection of networked services. These could include services associated with physical devices such as printers, mobile phones, motion detectors, and household heating systems, services associated with non-physical resources such as personal calendars or pdf to postscript converters, and services associated with intelligent agents that could, for example, determine your current location, the location of your smart trousers, or identify items that you need to include in your next Internet shop.

We assume a model where service provision is mediated through a **broker** which implements a collection of **policies** that configure the actual services in the environment in a variety of ways. Policies can be used to automate routine tasks and may involve instantiating underspecified descriptions of events and/or combining different services. For example, a user may have a policy to always use the printer nearest to their current location. With knowledge of this policy, we would expect a broker to route an underspecified print request to the most appropriate printer.

Given the current state of the art, there are severe limits on the kinds of policies that it is currently realistic to expect a broker to implement, but there is clearly substantial scope for configuring the basic set of actual services. To some extent, it will be possible to preconfigure the services provided by a broker (for example, by providing defaults that can be used to instantiate underspecified service requests). However, there is clearly a need to allow users to add personalised policies. For example, you might want your phone calls forwarded to your answer machine when your calendar indicates that you are busy, unless the caller is a family member, and you aren't in a meeting with your boss.

Making it possible for non-technical users to configure their environment through a broker represents a major technological challenge. Users cannot be assumed to have significant technological expertise, indeed, they may not even be aware of the existence of some of the more *invisible* services in their environment. Furthermore, there is likely to be a significant gap between the system of rules and constraints that determine the behaviour of the broker and a user's conceptualisation of their environment. This paper arises from an ongoing research project in which we are exploring the role that natural language (NL) descriptions can play in bridging this gap. Our architecture includes an ontology that precisely describes the actual services, a formal language for defining policies, and a middleware implementation of formal policies[1]. In the remainder of the paper we present an analysis of how existing NL technologies can be applied to this problem.

2. An Abstract Policy Representation Language

In order to modularise the problem, we consider an architecture which has an (unambiguous) abstract policy representation language at its centre (see Figure 1). This language is "spoken" (or manipulated) by all of the agents in our system, which we envisage including the policy broker, a natural language user interface (NLI) and a graphical user interface (GUI). Given some unambiguous formal language, we are left with the smaller problems of developing the individual agents that can manipulate it. For example, the NLI, which is the focus of this paper, is required to "translate" from user policies expressed in natural language to user policies expressed in the formal language.

The design of the formal language is the subject of ongoing research within our project. Here, we will just give a flavour of what might be required. We will start by discussing the domain ontology, which is used to establish a shared terminology between the agents. We will then discuss how policies might be expressed with reference to concepts in the ontology.

This work is supported by EPSRC grant number GR/S26408/01.

[1]The latter is also being developed within our project (Owen et al., 2003; Robinson and Wakeman, 2003) but is outside the scope of this paper

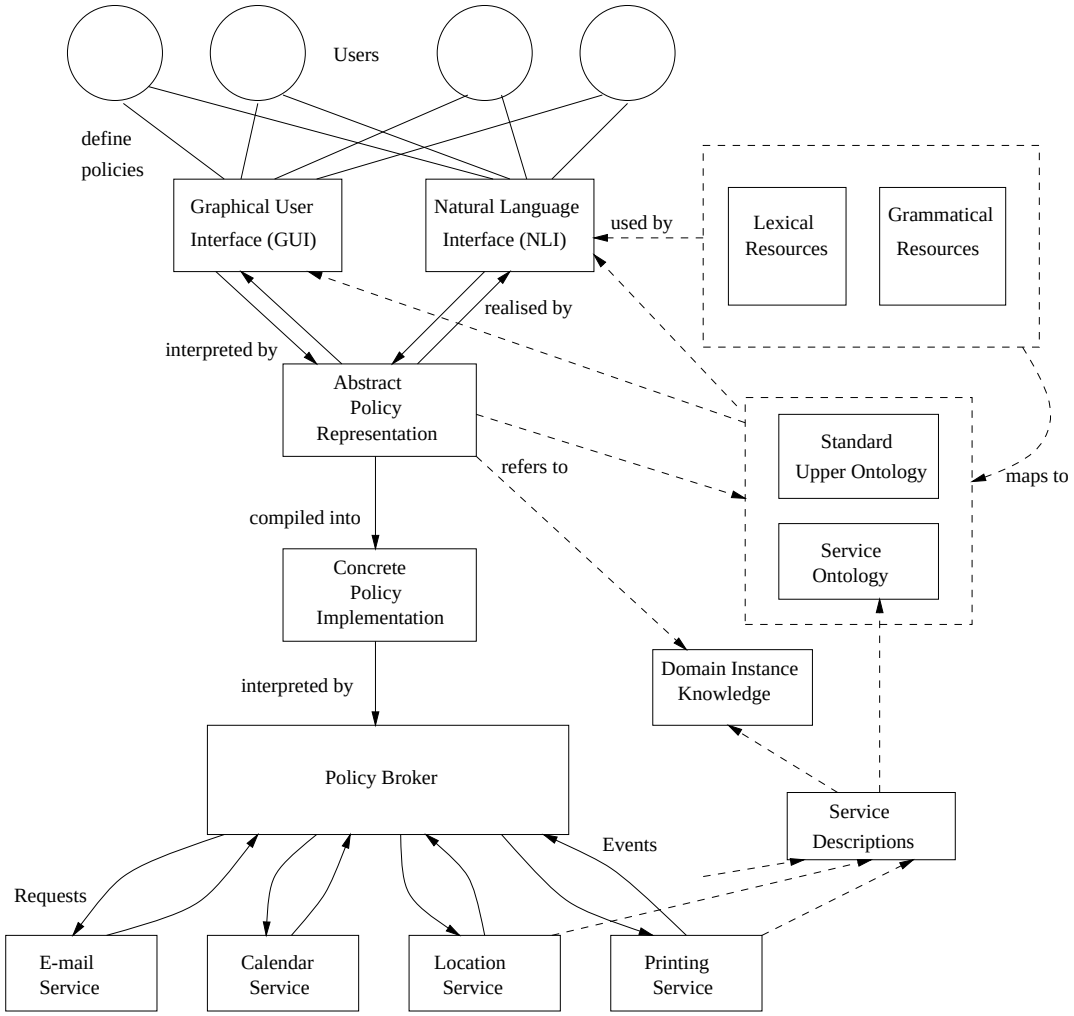


Figure 1: System Architecture

2.1. Domain Ontology

Throughout this paper we will use an office scenario to explore the issues in generating interpretations of NL policy descriptions. In this scenario there are classes of **objects**, corresponding to entities such as printers, scanners, documents and users and classes of **events**, which correspond to different types of services within the environment. For example, events may be user **requests**, e.g., "print document A". Other events may be system **notifications** of state changes, e.g., "user B is in the office".

The ontology is a shared terminology which captures the properties of and relationships between objects and events. For example, printers are devices with attributes including name, location and status; and print events are user requests that involve a user, a file and a printer.

Recent interest in the Semantic Web (Berners-Lee et al., 2001) has led to a proliferation of potential ontology representation formalisms. Baader et al. (2003) note that description logics (DLs) are ideal candidates for ontology languages since they provide a well-defined semantics and powerful reasoning ability. Further, as noted by Stevens et al. (2001), they are the underlying logical formalism of the web ontology languages OIL (Fensel et al., 2001) and DAML+OIL (Horrocks and Patel-Schneider, 2001), whilst being more intuitive to the human user.

Using DL, descriptions of concepts within the domain are built from atomic concepts (unary predicates) and atomic roles (binary predicates). Figure 2 shows a fragment of the concept hierarchy for our office scenario expressed in a typical DL. For example, the concept printer is declared as a subconcept of device and printers have attributes (functional roles) such as *colourness* and *status*. *colourness* is declared as a total attribute of the concept printer since every printer is related to exactly one ColourValue. Following Borgida and Brachman , we use the concept constructor **the** to declare total attributes.

Other roles, however, are relational rather than functional i.e., they do not define a one-to-one mapping. For example, the expression Email $\sqsubseteq \forall recipient.$User $\sqcap \geq 1\ recipient$ states that every recipient of an Email event is a User and that there is at least one recipient.

The relatively intuitive syntax of this DL makes it ideal for generating from natural language. For example, we can represent the natural language expression "an online colour printer" using the description:

$x \in$ Printer $\sqcap \exists colourness.$COLOUR $\sqcap \exists status.$ONLINE

In order to be able to distinguish between individual objects (e.g., a particular printer) and values (e.g. an integer

Object $\sqsubseteq$ **TOP** $\sqcap$ **the** $name$.**String**
PhysicalObject $\sqsubseteq$ Object
$\sqcap$ **the** $location$.PhysicalLocation
Device $\sqsubseteq$ PhysicalObject
Printer $\sqsubseteq$ Device
$\sqcap$ **the** $colourness$.ColourValue
$\sqcap$ **the** $status$.StatusValue
$\sqcap$ **the** $duplicity$.DuplexValue
$\sqcap$ **the** $idletime$.Duration
Scanner $\sqsubseteq$ Device
$\sqcap$ **the** $colourness$.ColourValue
$\sqcap$ **the** $status$.StatusValue
$\sqcap$ ....
User $\sqsubseteq$ PhysicalObject
ElectronicObject $\sqsubseteq$ Object
File $\sqsubseteq$ ElectronicObject
$\sqcap$ **the** $colourness$.ColourValue
$\sqcap$ **the** $security$.SecurityValue
$\sqcap$ **the** $owner$.User
$\sqcap$ **the** $format$.FormatValue
$\sqcap$ **the** $size$.Size
Document $\sqsubseteq$ File
$\sqcap$ **the** $version$.VersionValue
$\sqcap$ **the** $length$.Length
Event $\sqsubseteq$ **TOP** $\sqcap$ **the** id.**String**
Request $\sqsubseteq$ Event
Print $\sqsubseteq$ Request
$\sqcap$ **the** $agent$.User
$\sqcap$ $\forall patient$.File
$\sqcap$ $\geq 1\ patient$
$\sqcap$ $\forall target$.Printer
TurnOn $\sqsubseteq$ Request
$\sqcap$ $\forall patient$.Device
$\sqcap$ $\geq 1\ patient$
Email $\sqsubseteq$ Request
$\sqcap$ **the** $sender$.User
$\sqcap$ $\forall recipient$.User
$\sqcap$ $\geq 1\ recipient$
$\sqcap$ **the** $message$.String
$\sqcap$ $\forall attachment$.File
ClassObject $\sqsubseteq$ **TOP** $\sqcap$ **the** $name$.**String**
Relation $\sqsubseteq$ **TOP**
Nearest $\sqsubseteq$ Relation
$\sqcap$ **the** $objectclass$.ClassObject
$\sqcap$ **the** $location$.PhysicalLocation
$\sqcap$ **the** $individual$.Object

Figure 2: Fragment of the office scenario concept hierarchy

or the value "COLOUR"), we allow attributes to have values from concrete domains (Horrocks and Patel-Schneider, 2001). For example, the *length* of a Document is an **INTEGER** and is indistinct from another **INTEGER** with the same value. Similarly, the concept ColourValue is not defined in terms of other concepts and roles but in terms of the individual values:

$$ColourValue \equiv \{COLOUR, MONO\}$$

Having declared the primitive concepts and roles in our scenario, it is also possible to define new ones. For example, we might want to define a long document as a document with more than 10 pages:

$$\text{Document} \sqcap \textbf{the}\ long.\textbf{TRUE} \equiv \text{Document} \sqcap \exists length > 10$$

We also want to model superlative concepts e.g. "the longest document", "the busiest printer" and "the nearest printer to me". This type of information is most naturally associated with the entire concept rather than with each of its individual instances (Borgida and Brachman, 2003). However, as noted by Borgida and Brachman, DLs do not currently have the ability to be able to treat concepts as objects (as might be possible in some object-oriented systems). Thus it is necessary to create a meta-individual that is related to the concept by a naming convention. For example, we might create the individual PRINTER-CLASS-OBJECT (as an instance of the ClassObject concept) and attach the information regarding *busiest* and *nearest* as roles of this individual. However, certain relationships, such as *nearest*, will have to be reified (i.e. represented as a concept rather than a role) as they involve more than two objects. For example, Nearest is a ternary relation between a concept or class of objects, a physical location and an individual object.

Our discussion so far has focussed on the concept hierarchy. Most DLs also support the notion of a role hierarchy. In our example, there is a similarity between the *patient* role of a Print event (what is being printed) and the *message* role of an e-mail event (what is being e-mailed); and also between the *target* role of a Print event (where it is being printed) and the *recipient* role of an e-mail event (where it is being e-mailed). We will model these similarities by turning each pair of roles into sub-roles of a common super-role. Davis and Barrett (2002) note that very general roles prove useful for stating linguistic regularities in the linking between semantic roles and syntactic arguments, which is something we aim to exploit in our mapping between natural language and logical descriptions.

Finally, we note that our domain ontology is intended to encode only the concepts, objects and services specific to the user's environment. When new services are added, a description must be included by the service provider which will allow it to be incorporated into the ontology. Our ontology will be integrated with existing high-level ontologies such as the Suggested Upper Merged Ontology (SUMO) (Pease et al., 2002) to provide definitions of non-domain specific concepts such as time. Current research (Pease and Fellbaum, 2004) on integrating the machine readable dictionary WordNet (Fellbaum, 1998) with SUMO increases

User	Policy Set
1	*Normally, I use lja.* *If lja is off-line, I use ljx.* *If the document consists of more than 15 or 20 pages then I use ljb.* *If the document is in colour then I might use cclj.*
2	*I primarily use printer cork, because its near my desk.* *If the document is long and only a draft then I will use ljb.*
3	*If I am printing the final copy of a document which is in colour, then I will print it on cclj.* *Otherwise, if the document is long I will print it on ljb.* *Otherwise, I use the closest printer.*

Figure 3: Excerpts from user statements describing how they select what printer to use

its appeal for use with natural language. In order to integrate our ontology with SUMO, it will be necessary to provide a mapping between the DL representation of our ontology and a KIF or DAML representation.

2.2. User Policies

There is a range of different types of policies that a user might want to express:

default rules: filling in the gaps in an underspecified user request, possibly based on the characteristics of other objects involved in the event.

ontological definitions: defining a new concept or role in terms of other concepts and roles.

rewrite rules: changing a user request under some specified condition.

blocking rules: blocking a user request under some specified condition.

event generation: generating a new event on the basis of a trigger event.

In this paper, we focus on default rule policies, which have a first order logic (FOL) interpretation. Ontological definitions can also be expressed in FOL since they are assertions in English. We expect to be able to apply much of our work on default rule policies to ontological definitions. We also note that there is previous work (Pease and Murray, 2003) on the translation of ontological definitions from controlled English to logic. Rewrite and blocking rules present problems for any monotonic logic formalism, since the consequents of these rules may contradict other facts in the database (including the conditions of the rule). Event generation policies are similar in form to the complex sentences used in task-oriented dialogues (Balkanski, 1992) and their logical expression is the subject of ongoing research.

With a view to collecting real-life policies, we performed an initial study of how users manage a collection of available printers, and how they might express their policies using natural language. In the study, twenty-four users within a university department were asked to write down how they decide what printer to use. Figure 3 shows some typical examples extracted from their statements. In many cases, a default rule or policy could be applied which would, say, route a colour document to a colour printer if the target of the print event is unspecified. In principle, this rule could also be interpreted as a rewrite rule and apply when the broker receives a print request in which a specific printer is specified, but whether or not this is desirable, and how problems relating to this can be resolved is a complex issue that is outside the scope of this paper.

Another issue that is clear from our study is that policies cannot be interpreted in isolation. In addition to a policy about colour documents, a user may also have a policy to print long documents on a double-sided printer. One complication here is that *long* does not have a precise definition and thus a policy is required stating how it should be interpreted in this context. A second complication is the possible interaction or conflict between these two policies. What happens if the user attempts to print a document which is colour and in its final version and long? In this case, it may be possible to print the document to a duplex, colour printer. However, this may not be what the user wants and even if it is, it may not be possible given the actual services available i.e., there may be no double-sided, colour printer in the user's domain.

One possible solution is to interpret the semantics of user policies as preferences or soft constraints (e.g., Bartak (2002)) on the broker's reasoning. In the above example, a preference for a colour printer and a preference for a double-sided printer will be generated. We can also model a preference for a certain default printer, or an on-line printer, or the nearest printer to my current location in terms of constraints. The problem of finding an appropriate printer then becomes one of (possibly weighted) constraint satisfaction. Figure 4 shows constraints that we might wish to generate for a set of policies.

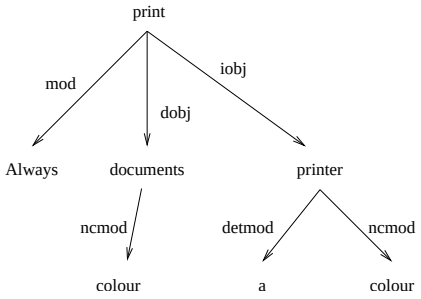
Another approach, which could be used alongside or instead of constraints, is to provide a multimodal interface which will allow the user to simulate and debug their own policies. For example, using the interface, a user could select which policy has highest priority. Using this type of model, the system can be thought of a tool for aiding the translation between different policy representations: NL, logical, graphical and software implementation.

3. From NL Descriptions to Constraints

Our general approach to obtaining constraints from NL policy statements is fairly standard (Allen, 1984) and involves mapping syntactic structure in the natural language to the semantic representation provided by the ontology. This section describes some of the more interesting details. In particular, we will discuss the recovery of syntactic dependencies using a shallow parser, extension of the lexicon using pre-existing lexical resources and distributional similarity methods, word sense disambiguation using knowledge of the semantic argument types from the ontology and recovery of implicit event participants using the ontology.

No.	NL Description	Policy Constraint	Strength
1	*Always print colour documents on a colour printer.*	$x \in$ Print $\sqcap\ patient.$(Document $\sqcap\ colourness.$COLOUR) $\rightarrow$ $x \in target.$(Printer $\sqcap\ colourness.$COLOUR)	Strong
2	*I usually print draft copies double-sided.*	$x \in$ Print $\sqcap\ agent.name.$\$Username $\sqcap$ $patient.$(Document $\sqcap\ version.$DRAFT) $\rightarrow$ $x \in target.$(Printer $\sqcap\ duplicity.$DRAFT)	Weak
3	*I never print confidential documents on lja.*	$x \in$ Print $\sqcap\ agent.name.$\$Username $\sqcap\ patient.$(Document $\sqcap\ security.$CONFIDENTIAL) $\rightarrow$ $x \notin\ target.$(Printer $\sqcap\ name.'lja'$)	Strong
4	*Never send documents to an off-line printer.*	$x \in$ Print $\sqcap\, patient.$Document $\rightarrow$ $x \notin\ target.$(Printer $\sqcap\, status.$OFFLINE)	Strong
5	*By default, I print documents on the closest printer.*	$(x \in$ Print $\sqcap\ patient.$Document $\sqcap\, agent.(name.$\$Username $\sqcap\, location.y))$ $\wedge\, w \in$ (Nearest $\sqcap\ objectclass.$PRINTER-CLASS-OBJECT $\sqcap\ location.y \sqcap\ individual.z) \rightarrow\ x \in target.z$	Weak

Figure 4: Examples of NL policies and corresponding constraints




```
(dobj, print:2_VV0, document+s:4_NN2)
(iobj, on:5_II, print:2_VV0, printer:8_NN1)
(ncmod, document+s:4_NN2, colour:3_NN1)
(ncmod, printer:8_NN1, colour:7_NN1)
(detmod, printer:8_NN1, a:6_AT1)
(mod, print:2_VV0, Always:1_RR)
```

Figure 5: Dependency analysis for the user policy "Always print colour documents on a colour printer"

3.1. Shallow Parsing

Shallow, dependency-based parsing can be used to determine the local, grammatical relations between the words in a sentence. These grammatical dependencies are closely related to the logical dependencies that hold between objects and events in our ontology. A key advantage of shallow parsing over deep syntactic analysis is its robustness. We aim to show that combining dependency-based parsing with the domain ontology will provide a robust and accurate approach to the interpretation of NL policy statements.

Our approach makes use of the RASP toolkit (Briscoe and Carroll, 2002), a pipelined, modular parsing system comprising separate processing stages for: tokenisation, part-of-speech and punctuation tagging, lemmatisation and shallow parsing. The output of the RASP parser is a dependency analysis of the input sentence, represented as a set of grammatical relations between lexical heads. An example of a dependency parse for the user policy *"Always print colour documents on a colour printer"* is shown in Figure 5, together with the corresponding set of grammatical relations that is output by the parser. The dependency structure shows that *"print"* is the lexical head of the whole sentence, and that it has a direct object (dobj) *"documents"*, an indirect object (iobj) *"printer"* and a modifier (mod) *"always"*. Further, the direct object *"documents"* is the lexical head of the sub-phrase *"colour documents"*, where *"colour"* is a (non-clausal) modifier (ncmod) of the head; and similarly for the indirect object *"printer"*.

3.2. The Lexicon

The words used by the user need to be mapped onto concepts in the ontology. We assume the existence of a **core lexicon** that associates a small number of words or phrases with each concept or class in the ontology. For example, the core lexicon might well assign the word *printer* to the concept or class `Printer` in the ontology.

There are, of course, many alternative ways that the same concept can be expressed, for example, by the use of synonyms or hypernyms/hyponyms. Rather than trying to include all of these directly to our lexicon, we are investigating how machine readable dictionaries (such as WordNet) and distributional similarity techniques can be used to overcome sparseness of the core lexicon.

This approach is bound to introduce a certain amount of noise, but the domain ontology can be used to resolve some of the potential uncertainty in how to map from lexical items into the ontology. For each word w in the dependency tree, a set of possible concepts in the ontology will be generated. Each possible concept c will have a **similarity score** c_w associated with it which indicates the similarity between the lexical item w and the core lexicon's entry for the concept c. For example, the lexical items associated with the `ColourValue` concept in the ontology are *colour* and *monochrome*. However, if a user uses the term *black-and-white*, we identify that the user means *monochrome* without explicitly stating this in our domain ontology since *black-and-white* is synonymous with *monochrome* in WordNet and can therefore be given a high similarity score. Similarly, we can determine that a user who refers to a *copy*, as in policy example 2, might be referring to a *document* since these are related in WordNet via their common hypernym *Writing, written material*. As will be discussed in Section 3.3., where the user refers to *draft copies*, there is further evidence for the document interpretation, since *draft* is a

possible value of an attribute of a document.

There are at least two problems with using WordNet to augment our domain ontology. First, a word may not exist in WordNet. For example, there is no entry for the word *double-sided* (in WordNet 1.6). Second, words tend to have multiple senses in WordNet, many of which are unlikely interpretations given the domain. To some extent, the domain ontology can be used directly for disambiguation. For example, we know that the most likely sense of *printer* is the *device* sense, rather than the *person* sense, since the entry for *printer* in the core lexicon maps it to the `printer` concept in the ontology, which is a subconcept of `device`. This idea can also be extended to words outside of the core lexicon. For example, there are four senses of the word *copy* in WordNet 1.6. In our office scenario, the highest similarity between concepts in WordNet occurs between the written material sense of *copy* and document, and so disambiguation is comparatively straightforward. However, as the scenario and the ontology are scaled-up, the process becomes more problematic.

These problems can be tackled using lexical distributional similarity methods (e.g., Weeds (2003)) to automatically generate thesauruses from domain-specific corpora. Such techniques can be used to find semantically similar neighbours of words not in WordNet. Further, it has been shown (McCarthy et al., 2004) that the most distributionally similar neighbours of a word can be used to select the most likely sense of a word in WordNet given the domain. There is also related work (Buitelaar, 2001) which uses a relative term frequency score to compute the domain relevance of a term and thus of a concept in a semantic lexicon such as WordNet. In both approaches, a domain specific corpus is required (either to derive reliable similarity scores or to compute domain relevance scores) and to this end, we are in the process of creating large domain-specific corpora consisting of text retrieved from the Internet using search engines given words in the core lexicon as queries.

In any case, the result at this stage will be a set of possible referents within the ontology for each word in the uttered policy together with an estimate of their plausibility.

3.3. From NL Dependencies to Logical Descriptions

Having mapped each lexical item onto a set of ontological concepts, the next step is to use the output of the shallow parser and the ontology to determine the most likely combination of concepts, and how these fit together. In general, we disambiguate the referents of each local tree of the dependency parse by finding the most coherant referents in the ontology: the most tightly located collection of elements in the ontology.

In policy example 1, where each word used is mapped to a single concept in the ontology, there is also a single path through the ontology that links these concepts in the way specified by the dependency tree. In general, we would expect there to be a mapping between the grammatical dependency relation and the semantic role in the ontology. In the dependency tree, the words *documents* and *printer* are the direct object and indirect object respectively of the verb *print*. The corresponding concepts in the ontology can be the patient and target arguments respectively of a `print` event. Similarly, the adjective *colour* modifies the nouns *documents* and *printer*, which maps to the ontological fact that `colour` is a value of a role that applies to both the concepts of `document` and `printer`. Accordingly, we can generate the following expression of the type of event described by this NL description:

$$\begin{aligned}\text{Print} \quad &\sqcap \quad patient.(\text{Document} \sqcap colourness.\text{COLOUR}) \\ &\sqcap \quad target.(\text{Printer} \sqcap colourness.\text{COLOUR})\end{aligned}$$

In general, the problem is much harder since each word used will map to a number of plausible concepts in the ontology. Thus, each combination of concepts will be scored according to their syntactic dependencies and semantic coherence within the ontology.

There are three clear benefits of using an ontology to generate logical forms. First, the ontology provides a certain amount of disambiguation. In policy example 4, we can determine that *send* is referring to a print event since it is applied to a document and a printer. This approach can be used to disambiguate the word *send* between the concepts `print` and `e-mail`. If we send a document to a printer than we are referring to a print action whereas if we send a document to a person then it is likely we are referring to an e-mail action.

Second, the ontology can be used to identify parsing errors. Although RASP is designed to be robust and accurate, its precision and recall of dependency relations is unsurprisingly less than human annotators. In particular, it has low precision and recall for the indirect object relation (Carroll et al., 1999). For example, the parser may incorrectly identify two words as having an indirect object relationship when they do not. We can identify this as a parser error if the words do not map to concepts that are related by the corresponding semantic role in the ontology. Further, we might be able to use our knowledge of expected concepts in particular roles to correct the parse or hypothesise syntactic dependencies missed by the parser.

Third, the ontology can be used to discover implicit arguments of events. In policy example 2, there is no explicit mention of a printer. However, we can introduce a printer into our logical representation of the event because the only path through the ontology from print to double-sided (assuming our word similarity method has returned a high similarity between *double-sided* and *duplex*) is through the concept of printer. Thus the following logical expression can be generated:

$$\begin{aligned}\text{Print} \quad &\sqcap \quad patient.(\text{Document} \sqcap version.\text{DRAFT}) \\ &\sqcap \quad target.(\text{Printer} \sqcap duplicity.\text{DUPLEX})\end{aligned}$$

3.4. Constraint Generation

The logical expressions we have generated so far describe an event but they do not express the desired constraint on the policy broker. In order to do this we need to be able to determine which parts of the expression make up the condition and which the consequent. We also need to be able to deal with the verbal modifiers such as *always*, *usually* and *never*.

In our printing policy examples, it is always the characteristics of the printer used in a print event that are determined by the characteristics of other objects such as the

document. In general, requests have arguments that are required (and therefore their characteristics will make up part of the condition) and arguments which may be underspecified (and therefore their characteristics will make up the consequent). This information is encoded in the *qualified number restrictions* in the ontological description of the event. For example, the *patient* role of the `Print` concept has the restriction ≥ 1, whereas the *target* role does not. We are also planning investigative work to discover whether the required information can be learnt from corpus data. We expect to find that the conceptual arguments that can be underspecified will correspond to the syntactic arguments that can be omitted.

The verbal modifiers are also of key importance in deciding the overall form of the constraint. However, we note that there are a relatively small number of them and therefore it is possible to enumerate them and their effects. For example, *always* produces a strong positive constraint, "usually" produces a weak positive constraint and *never* produces a strong negative constraint.

3.5. Eliciting Further User Input

As it stands, the system we propose generates a ranked set of possible constraints for each NL policy statement. Rather than trying to resolve any remaining ambiguity (which may in any case be due to a truly globally ambiguous policy), which could result in the broker acting in undesirable or unexpected ways, we envisage presenting the user with the set of ranked alternatives. The user can then select the desired logical form, which is a much simpler task than generating it from scratch. It will also be possible to present undefined concepts (such as "long") to the user and request clarification as to the definition of a "long document". These clarifications will be in the form of *definitional* policies, which extend the ontology.

4. Conclusions and Further Work

This paper describes ongoing research on the use of natural language to express user policies in pervasive computing environments. The central issue addressed in this paper is how the ontology is being used as the basis for interpreting NL descriptions, and in particular the referents of the lexical items in the description. We have presented an approach in which grammatical dependencies generated by RASP are mapped to ontological relations expressed in DL. Throughout the paper, we have highlighted areas and issues which require further investigation. In particular, we are investigating the range of possible user policies and their characteristics, so that we can constrain the natural language interpretations.

5. References

Allen, James, 1984. *Natural Language Understanding*. Benjamin Cummings, 1st edition.

Baader, Franz, Ian Horrocks, and Ulrike Sattler, 2003. Description logics as ontology languages for the semantic web. In *Lecture Notes in AI*. Springer.

Balkanski, Cecile, 1992. Logical form of complex sentences in task-oriented dialogues. In *Proceedings of ACL-1992*.

Bartak, Roman, 2002. Modelling soft constraints: a survey. *Neural Network World*, 12(5):421–431.

Berners-Lee, T., J. Hendler, and O. Lassila, 2001. The semantic web. *Scientific American*, 284(5):34–43.

Borgida, Alex and Ronald Brachman, 2003. *Description Logic Handbook*, chapter Conceptual Modelling with Description Logics. Cambridge University Press.

Briscoe, Edward and John Carroll, 2002. Robust accurate statistical annotation of general text. In *Proceedings of the 3rd International Conference on Language Resources and Evalutation (LREC-2002)*.

Buitelaar, Paul, 2001. Semantic lexicons: between ontology and terminology. In *Proceedings of OntoLex 2001*.

Carroll, John, Guido Minnen, and Edward Briscoe, 1999. Corpus annotation for parser evaluation. In *Proceedings of EACL-99 Workshop on Linguistically Interpreted Corpora*. Bergen, Norway.

Davis, Anthony and Leslie Barrett, 2002. Relationships between roles. In *Proceedings of OntoLex 2002*.

Fellbaum, C. (ed.), 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Fensel, D., F. van Harmelan, I. Horrocks, D. McGuinness, and P.F. Patel-Schneider, 2001. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45.

Horrocks, I. and P. Patel-Schneider, 2001. The generation of DAML+OIL. In *Proceedings of the 2001 Description Logic Workshop*.

McCarthy, Diana, Rob Koeling, and Julie Weeds, 2004. Ranking WordNet senses automatically. Technical Report TR 569, Department of Informatics, University of Sussex.

Owen, Tim, Julian Rathke, Ian Wakeman, and Des Watson, 2003. JPolicy: A java extension for dynamic access control. Technical Report 04-2003, University of Sussex.

Pease, Adam and Christian Fellbaum, 2004. Language to logic translation with phrasebank. In *Proceedings of the 2nd International WordNet Conference*.

Pease, Adam and William Murray, 2003. An English to logic translator for ontology-based knowledge rpresentation languages. In *Proceedings of 2003 IEEE Conference on Natural Language Processing and Knowledge Representation*.

Pease, Adam, Ian Niles, and John Li, 2002. The Suggested Upper Merged Ontology: a large ontology for the semantic web and its applications. In *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*. Edmonton, Canada.

Robinson, Jon and Ian Wakeman, 2003. The scooby event-based pervasive computing infrastructure. In *Proceedings of the 1st UK-UbiNet Workshop*. London, UK.

Stevens, R., I. Horrocks, C. Goble, and S. Bechhofer, 2001. Building a reason-able bioinformatics ontology using OIL. In *Proceedings of the IJCAI-2001 Workshop on Ontologies and Information Sharing*.

Weeds, Julie, 2003. *Measures and Applications of Lexical Distributional Similarity*. Ph.D. thesis, University of Sussex.