OBJECT-ORIENTED TECHNIQUES IN GRAMMAR AND ONTOLOGY SPECIFICATION

Matthias Denecke

Human Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
denecke@cs.cmu.edu

ABSTRACT

Currently, much effort is spent for the creation of linguistic resources such as grammars or domain models each time a new task oriented spoken language application is deployed. The resources for different applications, though different, resemble each other enough to warrant reusability of subcomponents. In this paper, we describe the application of techniques known from object oriented programming languages to grammar and ontology specification. To this end, we introduce a grammar formalism that is designed such that it supports the application of modularized development, multiple inheritance and deferred (or abstract) base specifications. In particular, we demonstrate how the combination of multiple inheritance together with deferred specifications and specification modules with name spaces facilitates rapid prototyping of ontologies and grammars for new domains and new target languages.

1. INTRODUCTION

Current spoken language applications are typically limited in scope to a semantically closed domain. The limitations allows the grammar writers to exploit assumptions to reduce the complexity and generality of the grammars. At the same time, the grammars written for one language but different domains exhibit similarities in rule structure as well as in coverage. For example, many dialogue systems cover utterance to start or end conversations, request help or request repetitions. In addition, the syntactic structure of the utterances contains similarities, such as phrases that start with fragments like `I would like ...` and so on. If one wanted to exploit the similarities of grammar fragments the need for modularity on a level below the grammar level is required. While there have been investigations to determine the possibility of using several grammars in parallel [8], few attention has been paid to applications of object oriented techniques to the specification of grammar and ontology modules. What is lacking is an approach to modularize grammar specifications in a way that is similar to the way in which objects in object-oriented programming languages are specified.

Typically, grammars for task-oriented spoken language applications are expressed in semantic grammars, in which the nonterminal symbols encode syntactic and semantic information. Thus, a nonterminal symbol can be seen as the result of a unification of two informational structures, one carrying the syntactic information, one the semantic information. If the grammar designer were to write a grammar in a new target language, he or she could only specify the syntactic information and could make use of the semantic information from the application in the first target languge . Similarly, if a word with the same syntactic distribution but different semantics were to be needed, only the semantics would need to be specified.

From a software engineering point of view, the described technique is an instance of multiple inheritance in that the resulting lexical entry inherits the constraints from the semantic and the syntactic specification. In this paper, we present multiple inheritance techniques that increase the reusability of semantic context-free grammars. Together with namespaces for grammar rules and semantic concepts, it then becomes possible to compile the final grammar from a set of partial grammar specifications.

The object oriented techniques are employed in the spoken dialogue system ARIADNE developed in the Interactive Systems Labs [2]. The object-oriented techniques allow a systems designer to specify subdomains of a spoken dialogue application and to compose an application of different dialogue packages at compile time. The dialogue packages, containing grammar and ontology specifications, are the equivalent of libraries in object oriented programming languages, containing classes and methods. The specification of the grammars and ontologies, together with specifications specific to the dialogue manager employed, is facilitated by an integrated development environment called CHAPEAU CLAC [3].

The paper is organized as follows. In section 2, we describe the application of object oriented techniques to domain models. In section 3, we describe the application of object oriented techniques to grammar specifications. In section 4, we conclude the paper with a discussion of the results and an outlook on future work.

2. OBJECT-ORIENTED TECHNIQUES IN ONTOLOGY SPECIFICATION

In task-oriented spoken language applications, the definition of the concepts used in the domain is part of the design process. Similar to interlingua specifications in machine translation systems [8, 9], we use a type hierarchy to enumerate the used semantic concepts and the relationships among them. In addition to semantic concepts, speech act types and syntactic types are represented as well. The semantics of utterances are represented in typed feature structures.

I would like to thank my colleagues at the Interactive Systems Laboratories for helpful discussions and encouragement. This research is supported by the Defense Advanced Research Projects Agency under contract number DAAD17-99-C-0061. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the DARPA or any other party.

2.1. Type Hierarchy

For the semantic domain model, we use a type hierarchy as is usual for typed feature structures. For more information, see [1], chapters 2 and 5.

2.1.1. IS-A *Relationships*

The type hierarchy forms a finite upper semilattice with subsumption as partial order, which means that for any subsets of types, there is a most specific lower bound, and, if there is an upper bound, then the least specific upper bound is unique. This requirement is necessary to ensure unification to be unique. The semantic components of the grammars are developed along the lines of the types in the type hierarchy.

2.1.2. HAS-A *Relationships*

In addition to the inheritance relationship, the domain model represents HAS-A relationships. Each type θ can take different features $f_1, \ldots, f_n$ that are appropriate for θ. The values of the f_i are restricted by appropriate types $Approp(\theta, f_i)$ in function of θ and f_i to ensure that the representations conform to the domain model. For each feature f, there is a least specific type $Intro(f)$ that introduces f; all types subsumed by $Intro(f)$ inherit f. The appropriateness conditions can also be used for type inference [1] which can also be of advantage in spoken dialogue systems [3]. Appropriateness conditions extend naturally to paths $\pi = f_1 \ldots f_n$. The IS-A and HAS-A relationships together describe the semantic model of the domain, and are thus domain specific. In addition, the type hierarchy encodes generic types used to express syntactic and speech act information [4].

2.2. Robust Processing

Moreover, we provide a mechanism to measure the gravity of incompatibilities. For a set of incompatible types $\{\tau_1, \ldots, \tau_n\}$, we determine their greatest lower bound $\tau^* = \sqcap_i \tau_i$ and set the degree of incompatibility to be the sum of the lengths of the shortest paths from τ^* to τ_i in the type hierarchy, or, more formally,

$$incomp(\{\tau_1, \ldots, \tau_n\}) = \sum_i len(path(\tau^*, \tau_i)) \qquad (1)$$

This measure is not associative and serves only to determine the quality of a representation after complete construction.

2.3. Object Oriented Techniques

2.3.1. *Namespaces*

As the type hierarchy for one specific application may consist of several subdomains each of which may be a reusable component in and by itself, the types and features are assigned a unique namespace to avoid naming conflicts. For an example, see figure 1. More details on the composition of type hierarchy packages are given in [4].

2.3.2. *Method Specification*

We extend the appropriateness specifications by *method specifications* to facilitate the interaction of the natural language processing component with a back-end application. A method specification $m = \langle n, C \rangle$ consists of a label n (the name of the method) and a set of constraints C. If a feature structure F fulfils the constraints in C, the method m will be invoked, passing the solution of the constraints on to the back-end application as parameters. Thus, method specification is a declarative and object-oriented specification of a guard that notifies the back-end application as soon as the informational content of the semantic representation reaches a certain saturation. The invocation is embedded in the standard type inference procedure of typed feature structures. A more detailed description of the method specifications as well as their combination with type inference procedures can be found in [5].

3. OBJECT-ORIENTED TECHNIQUES IN GRAMMAR SPECIFICATION

In standard approaches to robust parsing, conventional context-free grammars are used, together with an extended parse algorithm for robust processing. However, due to their lack of internal structure, standard nonterminal symbols only allow for equality checking and cannot be partially ordered. For this reason, we use *vectorized context-free grammars* for parsing. Vectorized context-free grammars are a straightforward generalization of context-free grammars where the nonterminal symbols are replaced by vectors of partially ordered elements. The vectorized context-free rules are annotated with fragments of typed feature structures that provide the semantic information of the utterance. For parsing, we use the SOUP context-free semantic parser [6] developed at Carnegie Mellon University. Since SOUP does not support unification based parsing, we compile the annotated grammar rules into data structures that are used internally by the parser.

3.1. Vectorized Context-Free Grammars

Vectorized context-free grammars are similar to standard context-free grammars in that they consist of a set of nonterminal symbols NT, a set of terminal symbols T, a set of rules R and a set of start symbols S. They differ, however, from standard context-free grammars in that the nonterminal symbols are vectors of partially ordered elements, and that nonterminal symbols on the right hand side of rules are annotated with semantic representations. The vectorized nonterminal symbols enable multiple inheritance and the annotations allow an automated creation of semantic representations.

3.1.1. *Vectorized Nonterminal Symbols*

While the nonterminal symbols of conventional context-free grammars do not provide enough internal structure for comparisons other than equality, and full fledged feature structure representations are too expensive in terms of specification and computation, we chose *vectorized nonterminal symbols* to represent nonterminal symbols in the rules. A vectorized nonterminal symbol in the annotated grammar rules is given by a vector $\langle \tau^1, \ldots, \tau^n \rangle$ where each of the τ^i is an element of a partially ordered set V^i (one of which is actually the type hierarchy representing the domain model introduced above). In tradition with the semantic grammars widely employed in spoken language applications, the vectorized nonterminal symbols contain syntactic and semantic information. As a notational convenience, we assume that all semantic information is represented by τ^1. To emphasize the particular role of the semantic

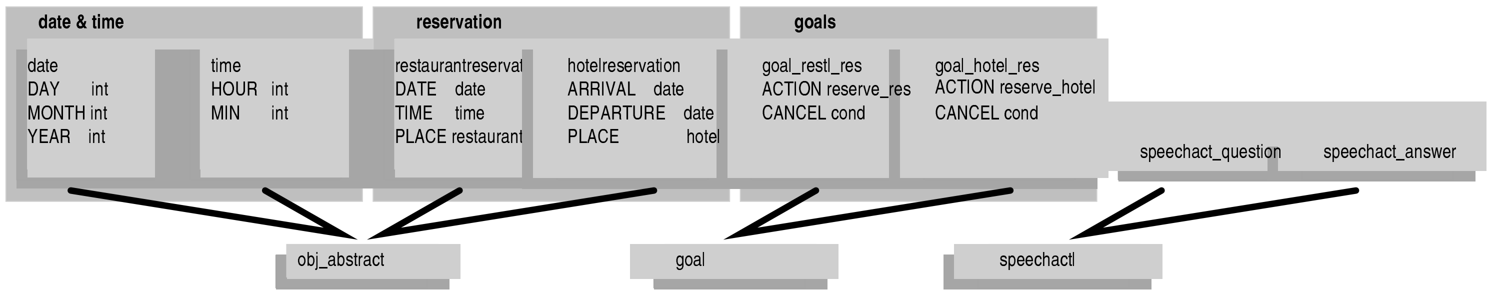

Figure 1: A part of the type hierarchy and its appropriateness conditions used in the travel application. Information increases from the bottom to the top. Two sub-domain models, called date & time and reservations are merged with the application-specific declaration of the types of the goals. The part of the hierarchy declaring the speech acts is domain-independent. Declarations are grouped in name spaces. This is a simplified presentation of the domain model actually used in the system.

type information in the vectorized nonterminal symbols, we underline the type symbol, as in $\langle \underline{\tau}^1, \ldots, \tau^n \rangle$. The major advantage of vectorized grammars over standard context-free grammars is that syntactic and semantic information can be specified separately in different τ^i, and that vectorized nonterminal symbols can be specified partially, to be combined later to more restrictive ones.

The combination of nonterminal symbols is the basis for multiple inheritance. In order to formalize our approach, we define unification, generalization and subsumption on the set of vectorized nonterminal symbols NT. For this to work out, we impose the same requirements on the V^i as on the type hierarchy described in section 2.1. In particular, we require the existence of a unique least upper bound of any set $V \subseteq V^i$ if any upper bound of V exists. The *unification* $\sqcup$ of $\mathbf{nt}_1 = \langle \underline{\tau}^1, \ldots, \tau^n \rangle$ and $\mathbf{nt}_2 = \langle \underline{\sigma}^1, \ldots, \sigma^n \rangle$ is given by $\mathbf{nt}$, where the ith component of $\mathbf{nt}$ equals the least upper bound of τ^i and σ^i if it exists. The unification of $\mathbf{nt}_1$ and $\mathbf{nt}_2$ is undefined otherwise. Similarly, the *generalization* $\sqcap$ of $\mathbf{nt}_1$ and $\mathbf{nt}_2$ is given by the generalization of their components, and is defined for all pairs of nonterminals. $\mathbf{nt}_1$ subsumes $\mathbf{nt}_2$ if and only if $\tau^i \sqsubseteq \sigma^i$ for all i. Two nonterminal symbols are called *incompatible* if their unification does not exist. Finally, we define the degree of incompatibility of two nonterminals to be equal to the degree of incompatibility of its components, or

$$incomp(\mathbf{nt}_1, \mathbf{nt}_2) = \sum_i incomp(\tau^i, \sigma^i)$$

Of course, we have $incomp(\mathbf{nt}_1, \mathbf{nt}_2) = 0$ if one of the two nonterminals subsumes the other.

In the current system, we chose the number of components in the nonterminal symbol vectors n to be equal to 3, where the first element is a semantic type drawn from the type hierarchy of the particular application, the second element the major syntactic category, and the third the minor syntactic category, chosen in function of the major category. In the examples, the use of an underscore denotes a *don't care* element, which is equal to specifying the bottom element of the corresponding partial order.

There is not a one-to-one correspondence between major syntactic categories and semantic types. As an example, consider the utterances `I would like to reserve a room` and `I would like a hotel reservation` . The derivations of these phrases contain the rules

$$\begin{array}{lll} \langle obj\_reservation, V, inf \rangle & \rightarrow & \text{reserve a room} \\ \langle obj\_reservation, N, \_ \rangle & \rightarrow & \text{hotel reservation} \end{array}$$

respectively[1]. The meaning of the example sentences is equal from a systems perspective and should thus be expressed by the same semantic representation. Were there different semantic types for the verbal and for the noun phrase, say, $act\_reserveroom$ and $obj\_roomreservation$, post-processing to arrive at equal representations would be necessary.

3.1.2. Vectorized Rules

To simplify the combination of various subgrammars, the set of rules R is divided into lexical rules and phrasal rules.

The Lexical Rules. The right hand side of lexical rules consists of a nonempty set of alternatives of string sequences, i.e. it is of the form

$$\begin{array}{lllll} \langle \underline{\tau}^1, \ldots, \tau^n \rangle & \rightarrow & w_{11} & \cdots & w_{1n_1} \quad | \\ & & \vdots & & \vdots \quad\quad | \\ & & w_{m1} & \cdots & w_{1n_m} \end{array}$$

Right hand symbols may be decorated with + and * symbols to simplify grammar writing. The information in the left hand symbol is specific enough to describe the concept expressed by the terminals on the left hand side, and their grammatical surface form, such as declination. In order to deploy an existing application in a new language, all processing that needs to be done for lexical rules is the translation of the terminal symbols and, if the new target language requires different syntactic categories, a modified specification of the syntactic information in the left hand symbol.

The Phrasal Rules. The right hand side of a phrasal rule r consists of a nonempty set of alternatives of nonterminal sequences. We term the semantic type τ^1 of the left hand nonterminal symbol the semantic type of the rule r. The nonterminal symbols on the right hand side are assigned a path π_{ij} of length $n \geq 0$ and a type τ_{ij}. The ith rule alternative, having k right hand symbols, of a rule is of the form

$$\begin{array}{l} \langle \underline{\tau}^1, \ldots \tau^n \rangle \rightarrow \\ \begin{bmatrix} \langle \underline{\tau}^1_{i1}, \ldots, \tau^n_{i1} \rangle \\ \pi_{i1} : \tau_{i1} \end{bmatrix} \quad \ldots \quad \begin{bmatrix} \langle \underline{\tau}^1_{ik}, \ldots, \tau^n_{ik} \rangle \\ \pi_{ik} : \tau_{ik} \end{bmatrix} \end{array}$$

[1]For the sake of example, the rules are simplified. In the system, the rules contain actually nonterminal symbols on the left hand side to take advantage of the ontological knowledge in the domain model.

where the $\mathbf{nt}_j = \langle \underline{\tau}^1_{ij}, \ldots, \tau^n_{ij} \rangle$ are the right hand side nonterminal symbols and the $\pi_{ij} : \tau_{ij}$ indicate the place of the semantic representations that are generated by the parse trees rooted at $\mathbf{nt}_j$. Furthermore, we require that $Approp(\tau^1, \pi_{ij})$ be compatible with τ_{ij} and with τ^1_{ij} in order to ensure the semantic representation of a parse to be well-typed. If $\pi_{ij} : \tau_{ij}$ is left unspecified, the empty path and $\tau_{ij} = \tau^1$ is assumed. If the rule is used in an acceptable derivation, the semantic representation of this rule is the unification of the feature structures defined by the paths and the types on the right hand side of the ith rule alternative, or more formally:

$$sem(r) = \bigsqcup_j \begin{bmatrix} \tau^1 & \\ \pi_{ij} & \tau_{ij} \end{bmatrix} \tag{2}$$

Since the nonterminal symbols are typed, we can detect the existence of the unification in equation (2) and determine the well-typedness of $sem(r)$ at compile time.

3.2. Robust Processing

Grammars for spoken language applications need to be tolerant against recognition errors and spontaneous speech. In addition to the features built in the parser, we support fault tolerance by the introduction of additional rules. In this section, we describe how the additional rules are generated and how their usage can be detected after parsing during semantic construction.

3.2.1. Rule Introduction

Robust parsing requires the relaxation of the constraints encoded in the grammar. Now, since the parser applies the rules always as a whole without modifications, type safety *within* a rules as required by equation (2) does not cause the grammar to become more brittle. On the other hand, we need to allow the parser to match ”closely resembling” nonterminal symbols. This corresponds to the equivalent of robust unification. For this reason, the rule compiler generates a rule of the form

$$\mathbf{nt}_1 \rightarrow \mathbf{nt}_2 \tag{3}$$

if $incomp(\mathbf{nt}_1, \mathbf{nt}_2) < c$ for some threshold c.

3.2.2. Semantic Construction

Since the vectorized rules are annotated with feature structures, it is possible to automatically generate a semantic representation of the utterance simply be recursively traversing the parse tree and unifying the feature structures. This is done only after parsing in order not to reduce the parsing speed.

The semantic representation of a derivation d of an input sentence is then determined by

$$sem(d) = \bigsqcup_j \begin{bmatrix} \tau^1 & \\ \pi_{ij} & (\tau_{ij} \;\dot{\sqcup}\; sem(d_j)) \end{bmatrix} \tag{4}$$

where d consists of the application of the rule r and the k right hand symbols of r are the roots for the derivations $d_1, \ldots, d_k$. In the above equation, $\dot{\sqcup}$ denotes the robust unification that is set to the greatest lower bound of the involved types in case the unification is not defined. Figure 2 illustrates equation (4).

In general, the resulting feature structure of a parse that makes use of one of the rules introduced by equation (3) cannot guaranteed to be well-typed,since the generalization of the types in case of failed unification can cause a type to be less specific than the type $intro(f)$ for a feature f.

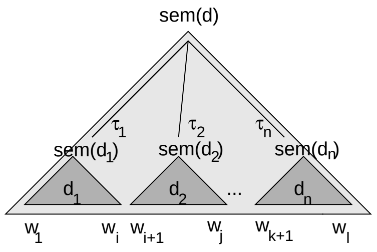

Figure 2: A derivation of an input sentence. The unification of the τ_i with the roots of the derivations of the sub strings need to be robust to enable robust parsing.

As the well-typedness of the rules can be detected at compile-time, we can expect generated semantic representation to be well-typed provided that no rules introduced in equation (3) have been used to derive the utterance. In the case where rules of the form shown in equation (3) are present in a derivation, we can detect the degree of discrepancy by the repeated application of equation (1) to those nodes in the semantic representation where the discrepancies occurred.

In order to achieve a normalized representation for dialogue processing, the generated feature structure is broken down into several structures such that each structure represents one object, one state or one action. This is similar to the representation of conditions and discourse referents in Discourse Representation Theory [7].

3.3. Object-Oriented Techniques

3.3.1. Namespaces

The grammars for an application are specified in several specification packages. In order to avoid naming conflicts, each grammar symbol is prefixed with the name of the package.

3.3.2. Multiple Inheritance

The compatibility relation of vectorized nonterminal symbols is one place where we can take advantage of the generalization to partially ordered elements. In context-free grammars, two nonterminal symbols $\mathbf{nt}_1$ and $\mathbf{nt}_2$ are compatible if and only if $\mathbf{nt}_1 = \mathbf{nt}_2$. In the case of vectorized nonterminal symbols, we can exploit the additional information given by the internal structure of the vectorized nonterminal symbols and the partially ordered sets from which the symbols' components are drawn.

In the current implementation, the vectorized grammar is compiled down to a standard context-free grammar that can be used by SOUP. Consequently, the relationships between the vectorized nonterminal symbols need to be compiled into the context-free grammar explicitly. For each two vectorized nonterminal symbols such that $\mathbf{nt}_1 \sqsubseteq \mathbf{nt}_2$ the compiler adds a context-free rule

$$\mathbf{nt}_1 \rightarrow \mathbf{nt}_2$$

to the resulting context-free grammar. For example, if a rule describes the distributional behavior of nouns that describe trips, the addition of a lexical rule

$$\langle obj\_flight, N, \_\rangle \rightarrow flight;$$

$$\begin{array}{llllll}
base_{ger,eng}: & \langle obj, N, \_\rangle = 1 & & & \\
\rightarrow & \langle det, \_, \_\rangle & \langle property, A, sup\rangle * = 3 & \langle property, A, prd\rangle * = 2 & \langle obj, N, \_\rangle = 1; \\
 & & & & \\
base_{fr}: & \langle obj, N, \_\rangle = 1 & & & \\
\rightarrow & \langle det, \_, \_\rangle & \langle obj, N, \_\rangle = 1 & \langle property, A, prd\rangle * = 2 & \langle property, A, sup\rangle * = 3;
\end{array}$$

Figure 3: Deferred rule specifications for German, English and French

$$\begin{array}{lllll}
base_{ger,eng}: & \langle obj\_restaurant, N, \_\rangle & & & \\
\rightarrow & \langle det, \_, \_\rangle & \langle prp\_spatial, A, sup\rangle * & \langle prp\_nationality, A, prd\rangle * & \langle obj\_restaurant, N, \_\rangle; \\
 & & \{\textsc{spatial}\ prp\_patial\} & \{\textsc{nationality}\ prp\_nationality\} & \\
 & & & & \\
base_{fr}: & \langle obj\_restaurant, N, \_\rangle & & & \\
\rightarrow & \langle det, \_, \_\rangle & \langle obj\_restaurant, N, \_\rangle & \langle prp\_nationality, A, prd\rangle * & \langle prp\_spatial, A, sup\rangle *; \\
 & & & \{\textsc{nationality}\ prp\_nationality\} & \{\textsc{spatial}\ prp\_patial\}
\end{array}$$

Figure 4: The instantiated rules.

Label	Deferred Spec.	Instantiated Spec.
1	$\langle obj, N, \_\rangle$	$\langle obj\_restaurant, N, \_\rangle$
2	$\langle property, Adj, prd\rangle$	$\langle prp\_nationality, Adj, prd\rangle$
3	$\langle property, Adj, sup\rangle$	$\langle prp\_spatial, Adj, sup\rangle$

Figure 5: A Virtual table as created as intermediate representation by the rule compiler. The information contained in column 1 and 2 is generated for a deferred rule specification. An instantiation of the rule adds the third column to the table.

is sufficient to correctly describe the distributional behavior of the noun *flight*, provided that the relation $obj\_trip \sqsubseteq obj\_flight$ is known to the type hierarchy. The same argument holds for the syntactic components of the vectorized nonterminal symbols. This shows how multiple inheritance can be used to facilitate the grammar specification.

3.3.3. Deferred Specifications

We proceed to develop a technique that allows the grammar writer to separate the specification of syntactic and semantic information. To this end, we employ the technique of multiple inheritance known from object oriented programming languages to grammar rule specifications. In particular, a rule specification can be declared *deferred* which indicates this particular specification of the rule to be incomplete. Deferred rule specifications are the equivalent of abstract base classes in object oriented programming languages and are typically to be found in a generic base package that serves as the starting point for the rapid prototyping of new applications.

Note that the nonterminal symbols are partially ordered by the subsumption relation defined above. The unification of two nonterminal symbols, defined to be their least upper bound, is then used to implement multiple inheritance. Currently, we use deferred rule specifications to abstract away the task dependent semantic information while encoding syntactic information. This requires us to provide deferred rule specification for each of the target languages. The nonterminal symbols that can be specialized at a later point are marked with the constraint $= l$, where l is a label that determines the identity of the nonterminal symbol.

As an example, consider the deferred rule specifications for English, German and French shown in figure 3. The rules are supposed to parse nouns that have at most one superlative adjective attached and at most one predicative adjective. In the travel domain, they parse fragments such as *the closest Italian restaurant* and *le restaurant le plus proche*.

For each deferred rule specification, the compiler creates internally a virtual table. The virtual table consists of an array of the deferred nonterminal symbols. The relationship between the identity of the nonterminal symbols in the deferred rule specifications and the entry in the virtual table is given by the label l in the deferred rule specification. An example for the virtual table of the rules in the above example is shown in figure 5 where the first and the second column describe the virtual table created for a deferred rule specification, and the third column designates the specification for the rule instantiation.

To instantiate a deferred rule specification, it suffices to specify a new nonterminal symbol $\mathbf{nt}_i$ such that $\mathbf{nt}_d \sqsubseteq \mathbf{nt}_i$ for each nonterminal symbol $\mathbf{nt}_d$ in the virtual table. To continue the above example, the specification

$$\begin{array}{ll}
\multicolumn{2}{l}{\text{instantiate } base_{ger,eng,fr} \text{ with}} \\
obj\_restaurant & = 1 \\
prp\_nationality & = 2 \\
prp\_spatial & = 3
\end{array}$$

causes the full specifications shown in figure 4 to be generated

It should be noted that changes in the deferred rule specifications cause changes in the instantiations. This again is similar to the behavior of abstract base classes in an object oriented programming framework. For example, if a developer adds a new rule alternative to a deferred rule specification, the deferred rule specification will cause the new rule alternative to be added to its instantiations at compile time.

Furthermore, it should be noted that semantic conversion information may not be part of a deferred rule specification. This is because it is unknown at the time of specification how the deferred specification will be instantiated and how the semantic information is to be used. An example will illustrate this point. Assume one wanted to instantiate the shown deferred rule specification $base$ to cover fragments such as `the cheapest non-stop`

$$\begin{array}{llll} base_{ger,eng}: & \langle obj, N, \_\rangle = 1 & & \\ \rightarrow & \langle det, \_, \_\rangle \quad \langle property, A, sup\rangle * = 3 & \langle property, A, prd\rangle * = 2 & \langle obj, N, \_\rangle = 1; \\ & \qquad\qquad \{\text{PROPERTY} = 4\} & \{\text{PROPERTY} = 5\} & \end{array}$$

Figure 6: Deferred rule specifications providing abstract conversion information

`flight` in a flight information application. The instantiation of $base$ would look similar to the following specification:

$$\begin{array}{ll} \text{instantiate } base \text{ with} & \\ \quad \langle obj\_flight, N, \_\rangle & = 1, \\ \quad \langle prp\_flighttype, A, prd\rangle & = 2 \\ \qquad \{\text{FLIGHTTYPE}\ prp\_flighttype\}, & \\ \quad \langle prp\_price, A, sup\rangle & = 3 \\ \qquad \{\text{PRICE}\ prp\_price\}; & \end{array}$$

The semantic conversion information specifies how the information on the property of the object provided by the adjective should be integrated in the representation of the object.

In the air travel package, the type prp_cheap will be appropriate for a feature PRICE while in the restaurant package, the type $prp\_nationality$ will be appropriate for a feature NATIONALITY. As the existence of these features cannot be assumed at the time of specification of $base$, this information needs to be provided at the time $base$ is instantiated.

In order to remedy this inconvenience one could introduce *feature hierarchies* which are the equivalent or role hierarchies known from description logics, and generalize the subsumption and unification procedures accordingly. Introducing the feature hierarchy

$$\begin{array}{lll} \text{PROPERTY} & \sqsubseteq & \text{PRP\_NATIONALITY} \\ \text{PROPERTY} & \sqsubseteq & \text{PRP\_SPATIAL} \end{array}$$

in the restaurant package and

$$\begin{array}{lll} \text{PROPERTY} & \sqsubseteq & \text{PRP\_FLIGHTTYPE} \\ \text{PROPERTY} & \sqsubseteq & \text{PRP\_PRICE} \end{array}$$

in the flight package, inheritance could be extended to features. In this case, one could specify the deferred rule specification as shown in figure 6, and together with an extension of the virtual table to features, one specialize the base rule according to

$$\begin{array}{ll} \text{instantiate } base \text{ with} & \\ \quad \langle obj\_flight, N, \_\rangle & = 1, \\ \quad \langle prp\_flighttype, A, prd\rangle & = 2, \\ \quad \langle prp\_cheap, A, sup\rangle & = 3, \\ \quad \text{PRP\_FLIGHTTYPE} & = 4, \\ \quad \text{PRP\_PRICE} & = 5; \end{array}$$

This would push the amount of reusability even further. However, the exact specification of feature hierarchies and its implementation remains future work.

4. CONCLUSION

We presented mechanisms to apply object-oriented techniques to the specification of grammar and ontology packages. Using he presented techniques, it is possible to partially specify different aspects of the finalized representations and to reuse the modularized specifications. The advantages of these mechanisms is that it speeds up the prototyping of new spoken dialogue applications and the deployment of existing systems in new target languages.

Future work includes the extension of the SOUP parse algorithm so that the vectorized rules can be processed directly and that the compile stage is not needed; and the extension of the underlying feature logic with feature hierarchies as outlined in section 3. Moreover, due to the limited domains, we have only a rudimentary coverage of syntactic information in the vectorized nonterminal symbols. We will extend the syntactic representations as it becomes necessary.

5. REFERENCES

[1] B. Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1992.

[2] M. Denecke. Informational Characterization of Dialogue States. In *Proceedings of the International Conference on Speech and Language Processing, Beijing, China*, 2000.

[3] M. Denecke. An Integrated Development Environment for Spoken Dialogue Systems. In *Workshop on Toolsets in NLP, Coling, Saarbrücken*, 2000. Available at http://www.is.cs.cmu.edu.

[4] M. Denecke and A. H. Waibel. Integrating Knowledge Sources for a Task-Oriented Dialogue System. In *Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Stockholm, Sweden*, 1999. Available at http://www.is.cs.cmu.edu.

[5] M. Denecke and J .Yang. Partial Information in Multimodal Dialogue Systems. In *Proceedings of the International Conference on Multimodal Interfaces*, 2000. Available at http://www.is.cs.cmu.edu.

[6] M. Gavalda and A. Waibel. Growing semantic grammars. In *Proceedings of the COLING/ACL, Montreal, Canada*, 1998.

[7] H. Kamp and U. Reyle. *From Discourse to Logic I.* Kluwer, Dordrecht, 1993.

[8] L. Levin, D. Gates A. Lavie, F. Pianesi, D. Wallace, T. Watanabe, and M. Woszczyna. A Modular Approach for Spoken Language Translation in Large Domains. In *Proceedings of the AMTA*, 1998. Available at http://www.is.cs.cmu.edu.

[9] M. Woszczyna, M. Broadhead, D. Gates, M. Gavalda, A. Lavie, L. Levin, and A. Waibel. Evaluation of a practical interlingua for task-dependent dialogue. In *Proceedings of the AMTA SIG-IL Third Workshop on Interlinguas and Interlingua Approaches, Seattle, Washington*, 2000. Available at http://www.is.cs.cmu.edu.