Speech Communication, vol. 48, no. 3-4, pp. 390-416, 2006

Rapid Development of Spoken Language Understanding Grammars

Ye-Yi Wang and Alex Acero

Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA

Abstract

To facilitate the development of spoken dialog systems and speech enabled applications, we introduce SGStudio (Semantic Grammar Studio), a grammar authoring tool that enables regular software developers with little speech/linguistic background to rapidly create quality semantic grammars for automatic speech recognition (ASR) and spoken language understanding (SLU). We focus on the underlying technology of SGStudio, including knowledge assisted example-based grammar learning, grammar controls and configurable grammar structures. While the focus of SGStudio is to increase productivity, experimental results show that it also improves the quality of the grammars being developed.

Key words:
Automatic grammar generation, context free grammars (CFGs), example-based grammar learning, grammar controls, hidden Markov models (HMMs), n-gram model, automatic speech recognition (ASR), spoken language understanding (SLU), statistical modeling, W3C Speech Recognition Grammar Specification (SRGS)

1 Introduction

While speech-enabled applications and conversational systems have been studied in research labs for many years, such systems have yet to become mainstream in the real world. One of the problems is the discrepancy between

Email address: {yeyiwang,alexac}@microsoft.com (Ye-Yi Wang and Alex Acero).

spoken dialog research and the reality in industry. Pieraccini (2004) comprehensively analyzed the "chasm" between SLU research and industrial applications. In SLU research, there are two paradigms. The first adopts a knowledge-based approach. Domain-specific semantic grammars are manually developed for spoken language applications. The semantic grammars are used by robust understanding technologies [Allen et al. (1996); Wang (1999); Bangalore and Johnston (2004)] to map input utterances to the corresponding semantic representations. Such implementations have relied on the manual development of domain-specific grammars, a task that is time-consuming and error-prone. It requires combined linguistic and engineering expertise to construct a grammar with good coverage and optimized performance. It takes multiple rounds to fine tune a grammar, and it is difficult and expensive to maintain the grammar and adapt it to new usages — an expert, ideally the original grammar developer, has to be involved in the adaptation loop. The second research paradigm adopts a data-driven, statistical modeling approach. While it alleviates the labor-intensive problem associated with the first paradigm, it requires a huge amount of training data, which is seldom available for industrial applications. In fact, Pieraccini (2004) lists these difficulties and the potential areas of improvement that the research community can provide:

(1) There is little data for training in the design/development phrase. Systems have to be developed with no data. This leaves the manual grammar authoring the only choice for initial system deployment. Therefore, tools for fast grammar handcrafting are very important. Other tools like those for content word normalization/speech-ification are also very desirable.
(2) There is a huge amount of data available after deployment. It is extremely difficult to manually analyze the data in order to find the problems in the initial deployment. Tools for automatic or semi-automatic adaptation/learning/system tuning are very useful for improving the system's performance.

In this article we introduce SGStudio (Semantic Grammar Studio), a grammar authoring tool that aims at closing the research/industry gap for SLU. We focus on the first problem of grammar authoring in this article, while we are still working on the second problem of dialog system tuning and adaptation.

The principles underlying the design of SGStudio include

(1) using prior knowledge to compensate the dearth of data, and making it easy to create such prior knowledge.
(2) making efficient use of data with supervised learning, and making it easy to annotate data for the supervision.
(3) providing users with a spectrum of different solutions according to data availability, with a possible tradeoff on understanding accuracy.
(4) tight coupling of ASR and SLU [Ringger (2000)], such that better SLU

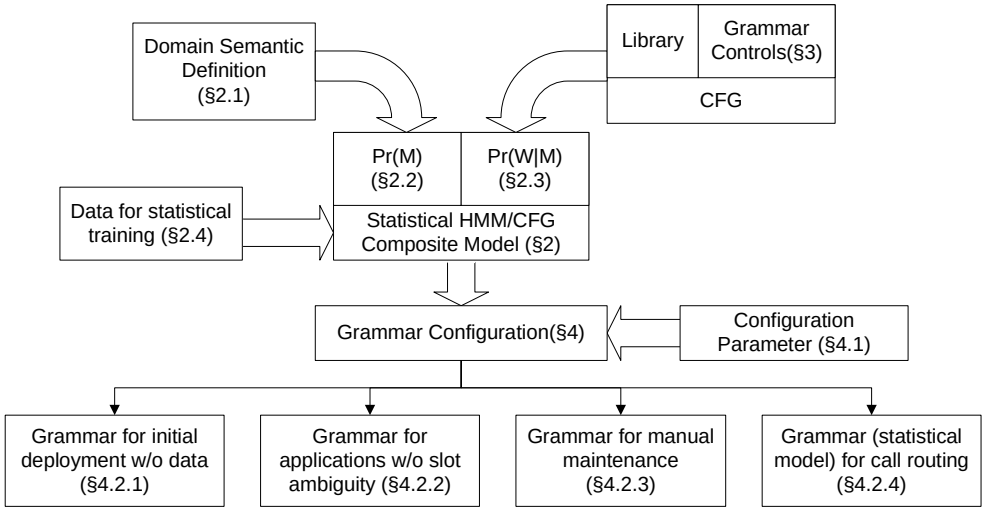

Fig. 1. SGStudio architecture. Each component is marked with the section number that describes it.

accuracy can be achieved by preserving the dependency between acoustic observation and meaning, and developers do not have to be trained to use two separate ASR and SLU recognizers.

Fig. 1 shows a realization of these principles. At the center of this architecture is a statistical model that adopts a pattern recognition approach to SLU. Given the word sequence W, the goal of SLU is to find the semantic representation of the meaning M that has the maximum *a posteriori* probability $\Pr(M|W)$:

$$\hat{M} = \arg\max_{M} \Pr(M|W) = \arg\max_{M} \Pr(W|M) \times \Pr(M) \tag{1}$$

Two separate models exist in this framework. The semantic prior model $\Pr(M)$ assigns a probability to an underlying semantic structure (meaning) M. The lexicalization model $\Pr(W|M)$ assigns a probability to the surface sentence (i.e., word/lexical sequence) W conditioned on the semantic structure. To overcome the data sparseness problem typically associated with statistical language modeling, a specific model under the framework of Eq. (1), the *HMM/CFG composite model* is introduced. It integrates the knowledge-based approach in the statistical framework in the following ways:

(1) The topology of the prior model, $\Pr(M)$, is determined by domain semantics.
(2) Probabilistic context free grammar (PCFG) [Jelinek et al. (1990)] rules that contain both domain-independent and domain dependent linguistic knowledge are used as part of the lexicalization model $\Pr(W|M)$).

To make it easy to include the prior knowledge, SGStudio adopts a simple frame-based semantic definition. It also provides a grammar library for domain-independent CFG rules, and it introduces *grammar controls* for the

easy creation of PCFG rules for domain-dependent concepts. Grammar controls can automatically generate high-quality knowledge-based ASR/SLU grammars from high level specifications.

SGStudio uses supervised learning, which requires annotated data, to make efficient use of limited amount of training data. To make the annotation easy to perform without the requirement of linguistic expertise, SGStudio limits the supervision at the semantic level and includes an annotation graphical user interface (GUI) that adopts a bootstrapping strategy (Section 2.4).

Often the amount of available data at initial system development varies across different applications. SGStudio can accommodate this difference with its configurable model structure. In fact, this flexibility goes beyond data availability. It also allows users to pick the model structure that best fits different application scenarios, including data availability, task complexity, and the availability of human resources in system maintenance, etc. The *grammar configuration* module (Section 4) of SGStudio customizes the general HMM/CFG composite model to fit these scenarios.

Fig. 2 shows the graphical user interface of SGStudio for prior knowledge definition. The pane on the left lets users define the frame-like domain semantics (Section 2.1) by adding frames (represented by the tabs in the GUI) and adding slots (represented by the textboxs in the left pane). On the right users can choose a filler grammar for the selected slot. The filler grammar can be selected from a grammar library, or a grammar control that is going to be defined in the "Grammar Control Definition" groupbox. In the example shown in Fig. 2, the user is using the city_name entries from a database table to populate the filler grammar, and use the city_code entries of the same table as the normalized semantics that is going to be returned upon successful recognition of a city name. The data table below shows the actual entries retrieved from the database. We will later show another screen shot of SGStudio for training data annotation in section 2.4.

SGStudio currently remains as a research prototype. It is mainly used inside Microsoft, with some limited exposures to Microsoft partners. Some of its component technologies will be included in the Resource Kit in the next release of Microsoft Speech Application Server. Communications with the speech product group in Microsoft are currently going on for future adoption of the technology in Microsoft Speech Application Software Development Kit.

The article is organized as follows. After a brief description of the related work in the remaining part of this section, the technical details of the HMM/CFG composite model, the grammar control technology and the grammar configurability issues are discussed. Fig. 1 includes the section numbers for these topics. Following that, ongoing and future work is discussed in section 5, and

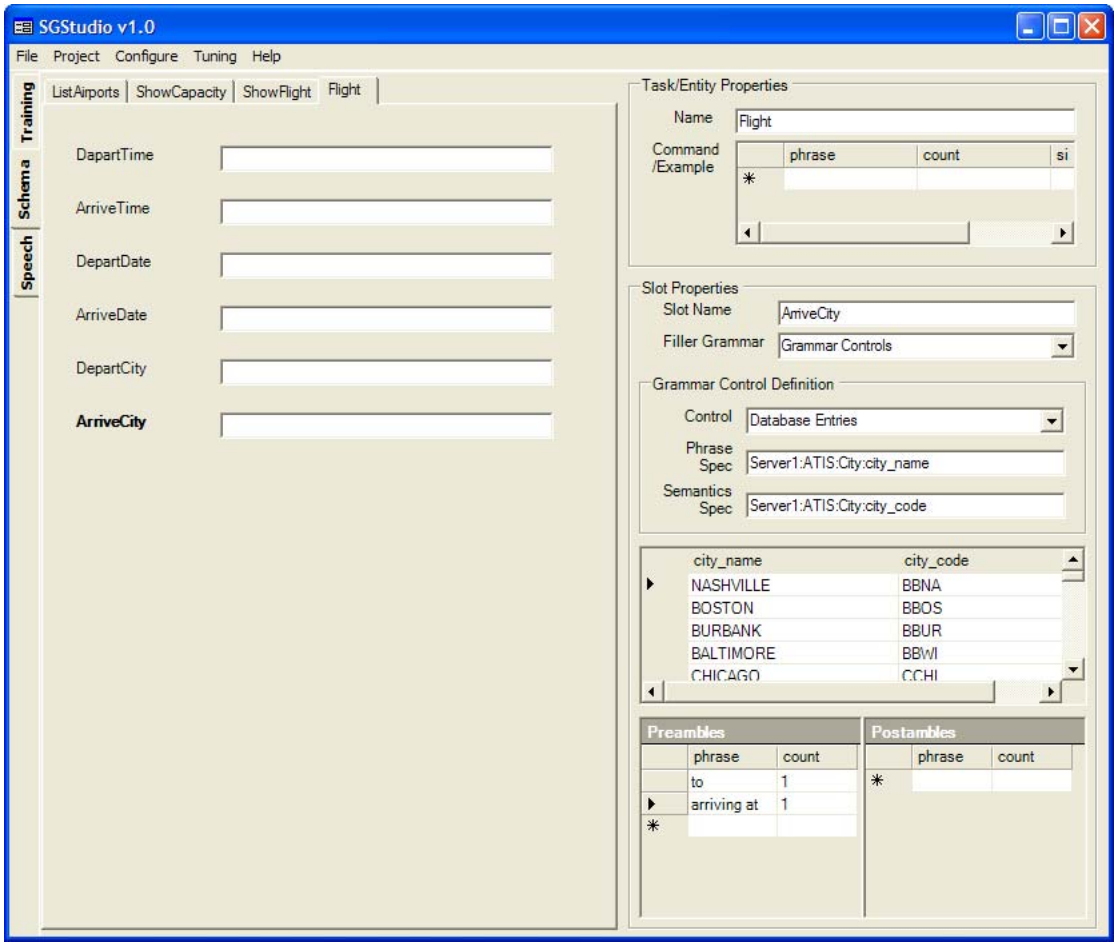

Fig. 2. SGStudio screen shot for prior knowledge definition.

section 6 concludes the article.

1.1 Related Work

While SLU research dates from the 70's [Woods (1983)], spontaneous spoken language understanding did not attract much attention until the early 90's, when multiple research labs from both academia and industry participated in the DARPA sponsored Air Travel Information System (ATIS) evaluations [Price (1990)]. The task in ATIS was to understand users' spontaneous spoken queries for air travel information. The systems investigated during this period fall into two different paradigms, knowledge-based and data-driven.

1.1.1 Knowledge-based paradigm

In the knowledge-based paradigm, Seneff (1992) alleviated the authoring difficulties by reusing the domain-independent part of a grammar. It was based on

the assumption that syntactic structures do not vary across different domains, and thus a high level syntactic context free grammar (CFG) could be shared by different applications. The domain specific knowledge was introduced to the model by replacing the low level syntactic non-terminals with semantic non-terminals. For example, noun phrases (NPs) could be replaced by domain-specific concepts like HOTEL_NAME. Dowding et al. (1993) adopted a similar idea with the unification grammar that includes some domain-specific semantic features. The difficulty with this approach is that the grammar developers must have in-depth knowledge of both the syntactic grammar and the domain. Ward (1994) introduced the Phoenix system, which adopted a different approach by modeling semantically. This limited the grammar rules that can be shared. However, developers could fine tune a grammar without any limitations imposed by a background syntactic grammar. The knowledge-based approach requires combined linguistic and engineering expertise to create high quality grammars.

1.1.2 Data-driven paradigm

In the data-driven SLU, most systems adopted a channel model that was widely used in ASR. Both Pieraccini and Levin (1993) and Miller et al. (1994) used a combination of hidden Markov models (HMMs) and n-gram models to robustly model spoken languages. The models were automatically learned from labeled training data. While the former used flat structures for meaning representations, the latter handled embedded structures. Della Pietra et al. (1997) introduced a model based on the study of statistical machine translation. Its distinct feature was that a concept in the semantic representation could generate non-consecutive surface observations (words). Macherey et al. (2001) introduced another SLU model based on statistical machine translation technology. He and Young (2005) investigated the use of HMMs in SLU, where the probability of a Markov transition between two states was decomposed into the probabilities of the stack operations that transformed one state to the other.

In addition to these statistical SLU efforts, grammar inference (or grammar induction) research also studies the problem of data-driven grammar creation. Fu and Booth (1975a) and Fu and Booth (1975b) surveyed the early work on automatic learning of finite state automata (FSA) from training data. Vidal et al. (1993) introduced an error-correction-based grammar induction algorithm and investigated its use for speech recognition. Stolcke and Omohundro (1994) attempted to use automatic induction of HMMs from training examples based on a Bayesian model merging algorithm. Wang and Waibel (1998) used iterative clustering and sequence building operations to find the common structures and applied them in a statistical spoken language translation system. Wong and Meng (2001) and Pargellis et al. (2001) used similar algorithms

to semi-automatically find language structures for SLU.

The data-driven approaches suffer from a data sparseness problem. The aforementioned statistical SLU systems address this problems with limited preprocessing that converts substrings to domain related concepts (e.g., replacing "Boston" with "Cityname" in the ATIS domain.) For those purely bottom-up, data-driven grammar inference algorithms, the quality of the inducted grammars could not be guaranteed, and language technology experts have to manually examine them and assign proper semantics for the automatically acquired structures.

1.1.3 Combining Knowledge-base and Data-driven Approaches

Recently there has been increased interest in combining the knowledge-based approaches with data-driven statistical learning in the field of human language technology. For language modeling, Wang et al. (2000) reported a unified language model that included CFG rules in an n-gram model. Bangalore and Johnston (2004) used domain knowledge as a supplement to the in-domain data, or adapted the out-of-domain data with the help of domain knowledge. Estéve et al. (2003) exploited the domain-dependent knowledge in the process of speech decoding. For SLU, Schapire et al. (2005) investigated the inclusion of prior knowledge in the statistical model for call-routing type of applications [Gorin (1995); Carpenter and Chu-Carroll (1998); Kuo et al. (2002); Hakkani-Tür et al. (2004)].

SGStudio combines knowledge-based and data-driven model for high-resolution SLU tasks, which need to identify many slots in comparison to the call-routing type of applications. Unlike the other statistical SLU models mentioned earlier that use the prior knowledge for data manipulation in a preprocessing step, SGStudio incorporates the domain knowledge as an integral part of the statistical model, and does so to a larger extent for every slot in the frames (Section 2.3). Because of this, it requires considerably less amount of training data.

Ringger (2000) discusses the issue of loose coupling and tight coupling in spoken language dialog systems. Tight coupling allows more inter-module (like ASR and SLU) dependency, but it is more difficult to engineer. Loose coupling makes more independence assumptions, which is practically more attractive, but potentially less accurate. Most research dialog systems adopt a loose coupling strategy, in which the dependency link between acoustic observation and semantics is broken. To compensate this, an intermediate process is often involved to correct ASR errors [Ringger (2000); Allen et al. (1996)] and/or robust parsing technology is applied to forced-match recognition results to domain semantics [Allen et al. (1996); Wang (1999); Bangalore and Johnston

(2004)]. In contrast, SGStudio's integration of prior knowledge in the model makes it possible for a tight ASR/SLU coupling, which is shown to improve the overall understanding accuracy.

2 Knowledge-assisted Example-based Statistical Modeling

The HMM/CFG composite model is a generative statistical model under the general framework of Eq. (1). Instead of writing grammars, developers are asked to prepare annotated examples, a task that does not require linguistic or engineering expertise. The HMM/CFG composite model learns from the examples and creates a grammar automatically. Unlike the grammar inference research that relies solely on the training sentences and infers the grammar bottom-up, it integrates the domain semantics in the topology of its prior model, and embeds the CFG rules, either from a grammar library or generated by grammar controls, in its lexicalization model. This prior knowledge greatly reduces the requirement of training data. In this section we first discuss the representation of domain knowledge, then explain how the knowledge is used in the prior and the lexicalization models. We will also present experimental results on the HMM/CFG composite model.

2.1 Domain Semantics

The semantic structure of an application domain is defined in terms of *Semantic frames*. Fig. 3 shows a simplified example of three semantic frames in the ATIS domain. Each frame contains several typed components called "*slots*." For example, the semantic frame "**Flight**" contains the slots "DCity" for departure city and "ACity" for arrival city. Both of them have the type "*City*", which means that the filler of the slots must be an object that has the "*City*" type, for example, a city name or a city code. The "*Void*" type of the "**ShowFlight**" and "**GroundTrans**" (for ground transportation) frames indicates that they are the top level commands called "tasks". The frame "**ShowFlight**" takes two slots, the "subject" slot has to be filled with an object that has the type "*Subject*" (e.g., FLIGHT or FARE), and the "flight" slot needs a filler with the type "*Flight*" — an instantiation of the "**Flight**" frame can be such a filler since the frame's type is "*Flight*".

Given a domain definition, the meaning of an input sentence is an instantiation of the semantic frames. Fig. 4 shows the meaning representation for the sentence "Show me the flights from Seattle to Boston." The meaning representation serves as an interface between linguistic modeling and application logic development. As soon as the domain semantics are determined, linguistic

<frame name="**ShowFlight**" type="*Void*">
<slot name="subject" type="*Subject*"/>
<slot name="flight" type="*Flight*"/>
</frame>
<frame name="**GroundTrans**" type="*Void*">
<slot name="city" type="*City*"/>
<slot name="type" type="*TransType*"/>
</frame>
<frame name="**Flight**" type="*Flight*">
<slot name="DCity" type="*City*"/>
<slot name="ACity" type="*City*"/>
<slot name="DDate" type="*Date*"/>
</frame>

Fig. 3. Examples of three simplified semantic frames defined in the ATIS domain. The type attribute of a slot restricts the type of its filler object. An instantiation of the "**Flight**" frame can be the filler of the "flight" slot of the "**ShowFlight**" frame.

<**ShowFlight** type="*Void*">
<flight frame="**Flight**" type="*Flight*">
<DCity type="*City*">Seattle</**DCity**>
<ACity type="*City*">Boston</**ACity**>
</flight>
</**ShowFlight**>

Fig. 4. The semantic representation for "Show me the flights from Seattle to Boston" is an instantiation of the semantic frames in Fig. 3.

modeling and application development can proceed in parallel.

2.2 Semantic Prior Model

The HMM topology and the state transition probabilities comprise the semantic prior model. The topology is determined by the domain semantics, and the transition probabilities can be estimated from the training data (Section 2.4).

Fig. 5 shows the topology of the underlying states in the statistical model for the semantic frames in Fig. 3. The left part of the diagram shows the top level network topology, and the right part shows a zoomed-in sub-network for state 2, which represents the embedded **Flight** frame for the "flight" slot of the "**ShowFlight**" frame. The initial state transition probabilities $\pi_1 = \Pr(\textbf{ShowFlight})$ and $\pi_5 = \Pr(\textbf{GroundTrans})$ comprise the prior distribution over the top level tasks. The transitional weights a_{12} and a_{13} comprise

the initial slot distribution for the "**ShowFlight**" frame. The transitional weights a_{56} and a_{57} comprise the initial slot distribution for the "**GroundTrans**" frame, and the transitional weights a_{C9}, a_{CA} and a_{CB} in the subnetwork comprise the initial slot distribution for the "**Flight**" frame.

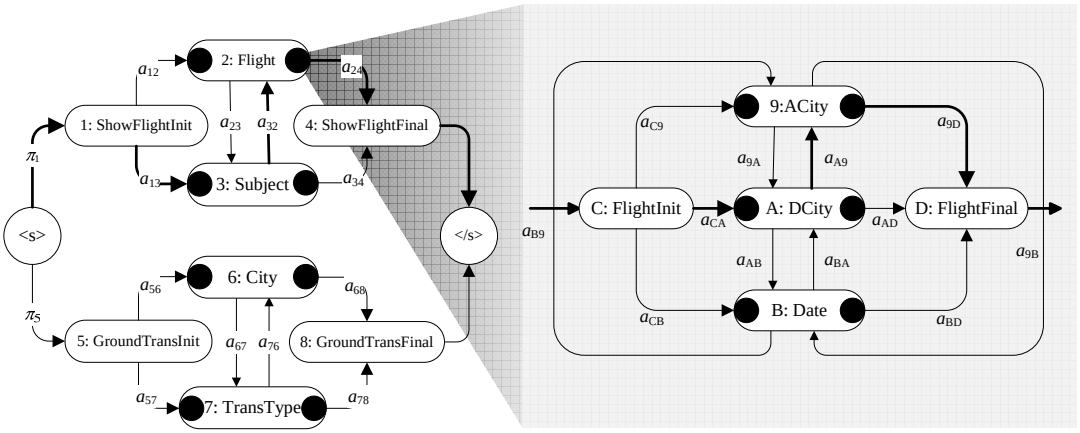

Fig. 5. The HMM/CFG composite model's state topology, as determined by the semantic frames in Fig. 3.

In Fig. 5, states 1 and 5 are also called the *precommands* for the **ShowFlight** and the **GroundTrans** frames, respectively. States 4 and 8 are called the *postcommands* for the **ShowFlight** and the **GroundTrans** frames, respectively. States 2, 3, 6, 7, 9, A and B represent slots. They are actually a three state sequence — each slot is bracketed by a *preamble* and a *postamble* (represented by the dots) that serve as the contextual clue for the slot's identity.

Given this topology, the semantic prior for the structure underlying the meaning representation in Fig. 4 is the product of the Markov transition probabilities across the different levels in the semantic hierarchy (the thick path in Fig. 5):

$$\begin{aligned}\Pr(M) = &\Pr(\textbf{ShowFlight}) \times \Pr(\text{Subject}|\text{<s>}; \textbf{ShowFlight}) \\ &\times \Pr(\text{Flight}|\text{Subject}; \textbf{ShowFlight}) \\ &\times \Pr(\text{DCity}|\text{<s>}; \textbf{Flight}) \times \Pr(\text{ACity}|\text{DCity}; \textbf{Flight}) \\ &\times \Pr(\text{</s>}|\text{ACity}; \textbf{Flight}) \times \Pr(\text{</s>}|\text{Flight}; \textbf{ShowFlight}) \\ = &\pi_1 a_{13} a_{32} a_{CA} a_{A9} a_{9D} a_{24} \end{aligned} \quad (2)$$

Generally,

$$\Pr(M) = \prod_{i=1}^{|M|+1} \Pr\left(C_M(i)|C_M(i-1)\right) \times \Pr\left(M(i)\right) \quad (3)$$

Here $|M|$ is the number of the instantiated slots in M. $C_M(i)$ is the name of the i-th slot in M, ($C_M(0)$ =<s> and $C_M(|M|+1)$ =</s> stand for the beginning and the end of a frame, respectively) and $M(i)$ is the sub-structure

that fills the i-th slot in M. Eq. (3) recursively calculates the prior probabilities of the sub-structures and includes them in the prior probability of the parent semantic structure.

2.3 Lexicalization Model

A fundamental problem in SLU is the tradeoff between model robustness and accuracy. A model needs to be relaxed, i.e., some restrictions imposed by the model may need to be loosened, in order to accommodate the extra-grammaticality that occurs in spontaneous speech. However, this may result in ambiguities and over-generalizations, which in turn may result in lower accuracy. The HMM/CFG composite model attempts to strike a balance on this issue. PCFG models, which impose relatively rigid restrictions, are used to model the slot fillers, which are more crucial for correct understanding. The knowledge introduced by the PCFG sub-models also compensates the data sparseness problem — there is no need for data to learn these ground level grammar structures. On the other hand, the sub-languages for precommands, postcommands, preambles and postambles, which glue different slot fillers together, are often domain dependent, subject to more variations and hard to pre-build a grammar for. For example, in ATIS domain, even though the task "ShowFlight" is a typical "Request for Information" type of sentence and can be naturally modeled with domain-independent rules such as "Show me ...", "I'd like to see ...", etc., there are also many domain dependent variations, e.g. "I want to fly from ...". The same problem exists for preambles and postambles. For example, it is natural to model a destination slot with the preamble "to" across different domain. However, in the ATIS domain, domain specific variations such as "the destination city is ..." and "arriving at ..." are also frequently observed. To cover these variations, we model these states as n-grams. N-gram models, when properly smoothed, are more lenient and robust. Fig. 6 shows a state alignment for the phrase "departing from Boston on Christmas Eve" according to the "**Flight**" network topology in Fig. 5. The original preambles and postambles are replaced by rounded rectangles that represent the n-gram models, and the slot fillers are replaced by rectangles that represent the PCFG models. Since a slot is always bracketed by its preamble and postamble (though both of them may cover empty strings), the probabilities for the transitions between a preamble and the corresponding slot filler and between a slot filler and its postamble are always 1.

Formally, the lexicalization probability in the composite model is

$$\Pr(W|M) = \sum_{\pi:|\pi|=|M|} P(W,\pi|M) = \sum_{\pi:|\pi|=|M|} \prod_{i=1}^{|\pi|} \Pr(\pi_i|M_i)$$

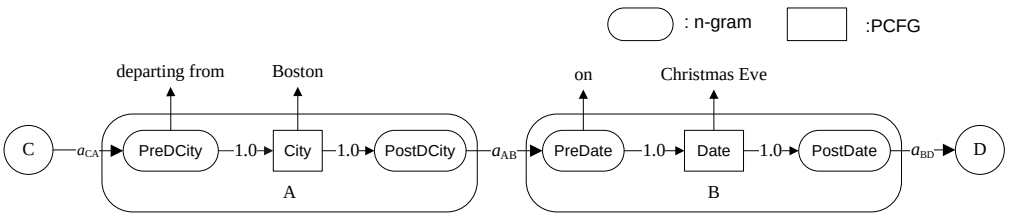

Fig. 6. A zoomed-in view of a state alignment for the phrase "departing from Boston on Christmas Eve" according to the network topology in Fig. 5. The output distribution of a rectangle state follows a PCFG, and that for a rounded rectangle state follows an n-gram model.

$$= \sum_{\pi:|\pi|=|M|} \prod_{i=1}^{|\pi|} P_{g_i}(\pi_i) \tag{4}$$

Here π is the segmentation that partitions W into $|M|$ segments [1], where $|M|$ is the number of states in the path that M intersects with the model, like the path in Fig. 6. Each segment π_i corresponds to a state in the intersected path. P_{g_i} is an n-gram model if M_i is a preamble or a postamble state, or a PCFG if M_i is a slot filler. It is possible that for some $i \neq j, P_{g_i} = P_{g_j}$. For example, the PCFG grammar for "cityname" is shared by the fillers for the DCity and ACity slots.

2.4 Model Training

For the prior model, the probabilistic distribution for the transitions from a state is estimated with maximum likelihood (ML) training by simply counting/normalizing the transitions in the annotated training data similar to the one illustrated in Fig. 4. A single prior count is used to smooth all distributions for the transitions from different states. This smoothing parameter is estimated with held-out data.

The n-gram lexicalization model can also be estimated with ML training. However, it is more complicated than a simple counting of the relative frequencies as in conventional n-gram training. For the n-grams of the non-filler states in the HMM/CFG composite model, the word sequences that a state emits are not marked in the training example for the sake of simplicity and efficiency. The only information that can be harvested from the training examples is the alignment of word sequences to state sequences as illustrated by Fig. 7, for the sentence "show me the flight that flies to Boston." The annotation for this sentence, as shown in Fig. 4, pegs the words "flight" and "Boston" to the specific states in the HMM topology. The alignment between the remaining words and model states is hidden and restricted by the annotation to small

[1] A segment may contain an empty string.

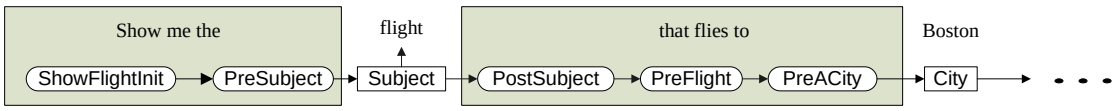

Fig. 7. The annotation restricts the word/state alignments in local regions. The exact word alignment to individual states can be learned with the EM algorithm.

regions. For example, the word sequence "show me the" has to be aligned with the state sequence "ShowFlightInit PreSubject". To obtain the alignment of words to an individual state (thus the aligned words serve as the example "sentence" to train the n-gram for that state), the EM algorithm [Dempster et al. (1977)] is applied, which enumerates all possible word sequences aligned to each state, computes the expected count for each of these word sequences with the current model parameterization, and use the word sequences with the expected counts to estimate the n-gram model for the state. The detailed mathematical derivation of the algorithm can be found in Wang et al. (2005).

The training of both the semantic prior model and the lexicalization model requires annotated data. SGStudio has a graphical user interface for easy data annotation, illustrated in Fig. 8. The tool works with a bootstrapping strategy. It uses the model (initially using the PCFG rules for the slots and the uniform distributions for the parameters) to analyze a sample sentence and output the hypothesized semantic structure. Users then make necessary modifications by clicking on a node for alternative interpretations. The models then get updated with the new samples. Fig. 8 shows that a user is changing the interpretation for "June fourteenth" from DepartDate to ArriveDate. With the tool, an experienced user can annotate ∼1500 samples in a working day. It is much faster than the speed (200 sentences/day) reported in Miller et al. (1994), which required the annotation of not just the semantic structures but also the alignments between words and preamble/postambles.

2.5 Converting the Model to the SRGS Grammar

Almost all statistically learned SLU models investigated in the research community require a specialized second pass analyzer, either called a parser or a decoder, to map the text output by the first recognition pass to a structured semantic representation. SGStudio, on the other hand, can convert the model it learned to the format of the industry standard described in the W3C speech recognition grammar specification (SRGS) [Hunt and McGlashan (2002)], such that the semantic representation can be constructed during the process of speech recognition. The grammar standardization makes it possible to create a single grammar for different speech recognition engines, and allows the dependency between acoustic observations and semantics (detailed discussion follows in section 2.6) in a tightly coupled ASR/SLU system.

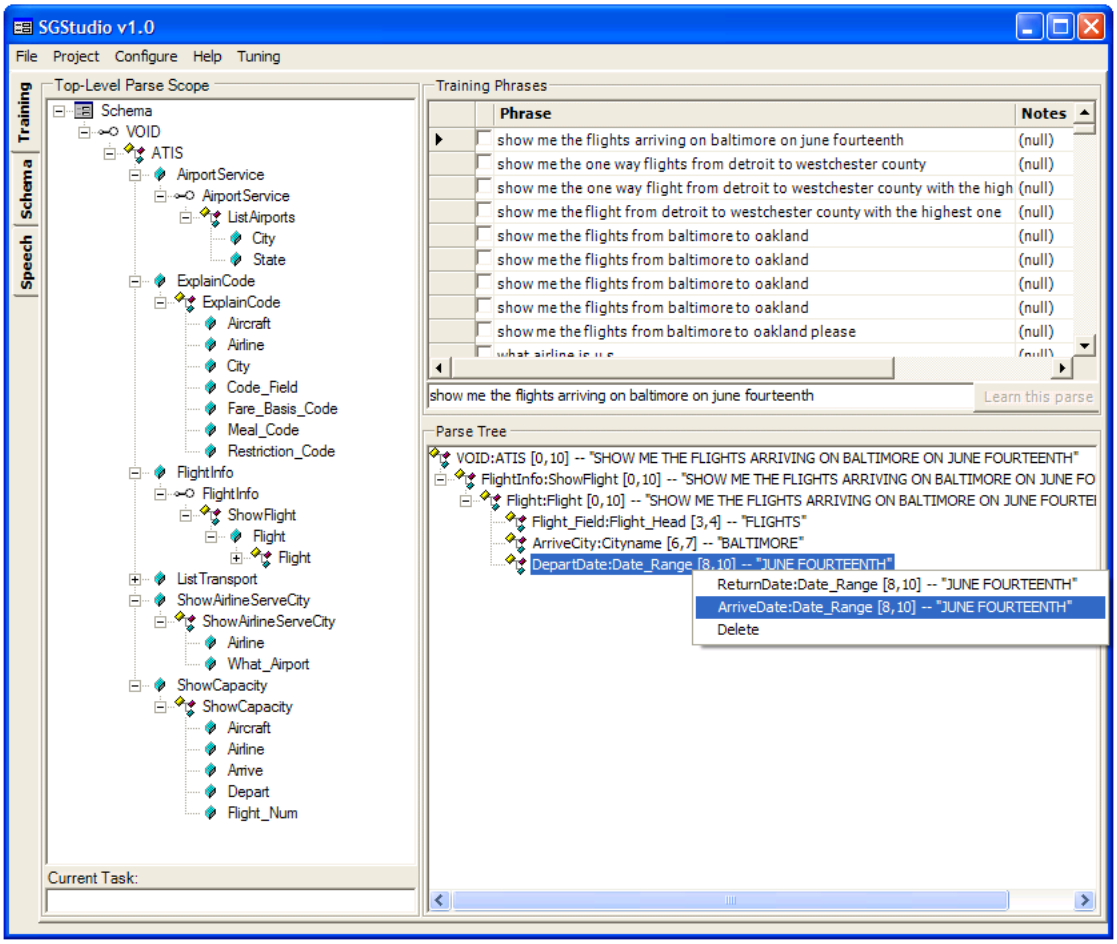

Fig. 8. SGStudio screen shot for example annotation.

2.5.1 The SRGS Grammar Format

The SRGS adopts an XML or a Backus Normal Form (BNF) representation format for grammars. Fig. 9 shows a simple example of the more frequently used XML representation. In the grammar, the items inside a <one-of> XML element specifies the alternates, with optional weights to specify the probabilities (or other weighting scores) for the alternates. The <ruleref> element makes reference to sub-rules. An important feature of SRGS is the semantic interpretation (SI) tag, which includes scripts inside the <tag> element that can be used to create semantic representations. In the SI tags, "$" represents the semantic structure that should be returned after the match with the current rule, and "$.foo" represents the "foo" field in the semantic structure for the current rule. "$$" represents the semantic structure returned from the sub-rule just being matched. The SI tags are ostensibly simple in this example. It becomes much more complicated when multiple SI tags from different sub-rules with alternative definitions work together to form the meaning of the current rule. The SI tags in the example grammar normalize a recognized text into its canonical semantic representation. Fig. 10 shows the representation

```
<rule id="ATIS" scope="public">
   <one-of>
      <item weight="0.8">
         <ruleref uri="#ShowFlight"/>
         <tag>$.ShowFlight=$$</tag>
      </item>
      <item weight="0.2">
         <ruleref uri="#GroundTrans"/>
         <tag>$.GroundTrans=$$</tag>
      < /item>
   < /one-of>
< /rule>
<rule id="ShowFlight">
   <ruleref uri="#ShowFlightInit"/>
   <ruleref uri="#Flight"/><tag>$.Flight=$$</tag>
</rule>
<rule id="Flight">
   from <ruleref uri="#CityName"/><tag>$.DepartCity=$$</tag>
   to <ruleref uri="#CityName"/><tag>$.ArriveCity=$$</tag>
   <item repeat="0-1"> on <ruleref uri="#Date"/>
                        <tag>$.DepartDate=$$</tag>
   </item>
< /rule>
<rule id="CityName">
   <one-of>
      <item>New York City<tag>$="NYC"</tag> </item>
      <item>Seattle <tag>$="SEA"</tag></item>
      ......
   < /one-of>
< /rule>
```

Fig. 9. A simplified and incomplete example of grammar for the ATIS domain.

for the utterance "Show me the flights from New York City to Seattle."

The SRGS grammar with SI tags allows tight ASR/SLU coupling — semantic structures can be constructed by the scripts in the SI tags in the process of ASR decoding. To be able to construct such semantic structures, the topology of the grammar must assign different structures (rules) to different meanings, so an appropriate SI tag can be associated with each rule. Because of this, n-gram language models for ASR, which do not model the structure difference, are not for tight ASR/SLU coupling.

```
<ShowFlight>
   <Flight>
      <DepartCity>NYC</DepartCity>
      <ArriveCity>SEA<ArriveCity>
   <Flight>
< /ShowFlight>
```

Fig. 10. The semantic representation created by the scripts inside the SI tags of the SRGS grammar during the recognition of the utterance "show me the flights from New York City to Seattle."

2.5.2 Model Conversion

SGStudio supports the SRGS grammar format. The HMM/CFG composite model is converted to the SRGS CFG as follows: the overall topology, as illustrated in Fig. 5, is basically a probabilistic finite state machine, which is a sub-class of the PCFG. The efficient conversion of the n-gram lexicalization model follows the work in Riccardi et al. (1996), and the CFG lexicalization model components can be directly included in the CFG language model. Furthermore, since each state in the model topology bears semantic meaning, the CFG rules for the state can be associated with SI tags as in the above example, such that a semantic representation can be automatically constructed during the process of recognition.

2.6 Experiments

We expect that the inclusion of the domain knowledge would reduce the amount of data required to train the proposed model. In the first experiment, we investigate the effect of different amounts of training data on the model's accuracy. We applied the model to the manual transcriptions of the ATIS 93 category A [2] test set. The standard ATIS evaluation for spoken language was performed and database query results were compared with the reference results. Fig. 11 plots the overall end-to-end system error rate of the HMM/CFG composite model on text input, with respect to the amount of the training data used.

Fig. 11 shows that it took about half of the annotated ATIS3 training sentences to achieve the accuracy close to what the system can achieve with more data. With a couple of samples per task, the system had a semantic accuracy [3] higher than 60%, and the error rate drops significantly for the first couple

[2] The utterances whose interpretations are independent of the context.

[3] The percentage of the returned flight information that with the fields fall into the "MinMax" set specified by the references, an ATIS specific evaluation method.

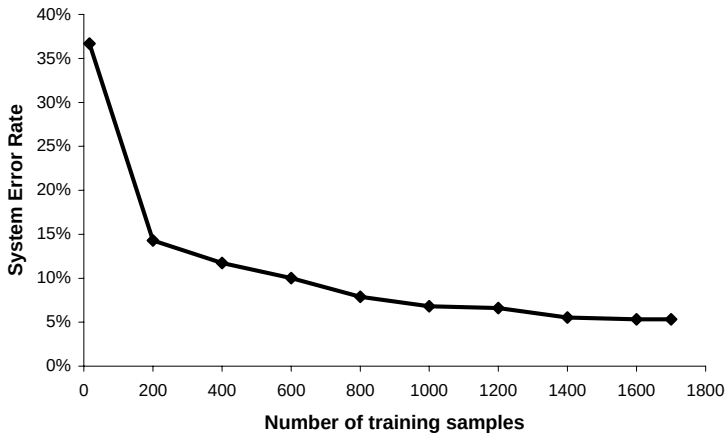

Fig. 11. End-to-end system error rate on text input vs. amount of the training data.

hundreds of samples. Using all the ATIS3 training data (∼1700 sentences), the end-to-end error rate is 5.3%. It is comparable to the performance of the best system (5.5%) in the 1993 ATIS evaluation, which used a semantic grammar manually developed by a spoken dialog expert who had access to the entire ATIS2 and ATIS3 training data (∼6000 sentences). As a reference point, Miller et al. (1994) reported that the Hidden Understanding Model, which is closest in spirit to the HMM/CFG composite model but used limited prior knowledge in a preprocessing step, had an error rate [4] at 26% with a model trained with ∼6000 sentences. This clearly demonstrates that the inclusion of domain knowledge in the statistical model reduces the requirement of training data.

One of the advantage of the HMM/CFG composite model over the previous methods is that the prior knowledge is incorporated directly into the model, and SRGS grammar can be obtained as described earlier for tight ASR/SLU coupling. In contrast, using the prior knowledge in a preprocessing step results in a model that can only "generate" the nonterminals in the preprocessing grammar instead of the terminal words that are covered by the nonterminals. This is not suitable for LM-based decoding in ASR. Hence most statistical learning systems use an n-gram language model in the first pass ASR, apply a preprocessor to map some substrings to domain concepts, and then apply a statistical SLU model to obtain semantic representations. This loose coupling approach can be formalized as the search for

$$\begin{aligned}\hat{M} &= \arg\max_{M} \Pr(M|\hat{W}) \\ &= \arg\max_{M} \Pr(M|\arg\max_{W} \Pr(O|W) \times \Pr(W)) \\ &= \arg\max_{M} \Pr(\arg\max_{W} \Pr(O|W) \times \Pr(W)|M) \times \Pr(M) \end{aligned} \tag{5}$$

[4] The paper claimed that around half of the errors were due to simple programming issues.

Table 1
The ASR word error rate and the SLU error rate (slot ins-del-sub) of the trigram model and the HMM/CFG composite model.

Recognizer		N-gram	HMM/CFG	Transcription
MS Commercial	WER	8.2%	12.0%	—
Recognizer	SLUER	11.6%	9.8%	5.1%
HapiVite	WER	6.0%	7.6%	—
	SLUER	9.0%	8.8%	5.1%

In this two-pass solution, the dependency link between the observed speech O and the meaning M via the word sequence W is broken. Instead, O is generated via a language model $\Pr(W)$ and an acoustic model $\Pr(O|W)$ that have nothing to do with M. In an extreme case where $\Pr(W) = 0$ whenever $\Pr(W|M) \neq 0$, no optimal solution can be found. A better solution should retain the dependency between O and M, as:

$$\begin{aligned}\hat{M} &= \arg\max_M \Pr(M|O) = \arg\max_M \Pr(O|M) \times \Pr(M) \\ &= \arg\max_M \sum_W \Pr(O, W|M) \times \Pr(M) \\ &= \arg\max_M \sum_W \Pr(O|W, M) \Pr(W|M) \times \Pr(M) \\ &\approx \arg\max_M \max_W \Pr(O|W) \times \Pr(W|M) \times \Pr(M) \end{aligned} \tag{6}$$

Hence, the SLU model $\Pr(W|M) \times \Pr(M)$ should be used directly as the language model in speech recognition.

In the second experiment, we used the SRGS grammar that was converted from the HMM/CFG composite model in a one-pass recognition/understanding experiment, and compared the word error rate and the SLU error rate with those of the two-pass systems. We used two speech recognizers, Microsoft commercial recognizer and HapiVite [Young (1993)], and we used the generic acoustic models that came with the recognizers instead of training an ATIS specific one. Table 1 shows the findings with the commercial recognizer and HapiVite. For the commercial recognizer, even though the composite model's word error rate is over 46% higher than the trigram model, its SLU error rate (measured as the slot insertion-deletion-substitution rate) is 17% lower. With HapiVite that is less aggressive in pruning, the word error rate of the HMM/CFG model is about 27% higher than the trigram model. However, the SLU error rate is 2.5% lower.

The results clearly demonstrate the sub-optimality of the two-pass approach for ASR and SLU. In this approach, the trigram model is trained to optimize the likelihood of the training sentences. If the test data is drawn from the

same distribution, the trigram model assigns higher likelihood to the correct transcription and hence reduces the word error rate. On the other hand, the objective of the HMM/CFG composite model training is to maximize the joint probability of the sentences and their semantic representations. Thus the correct representations of the test sentences will be assigned higher probabilities.

Similar findings on the dependency of acoustic observations on the semantics were reported in Riccardi and Gorin (1998). They interpolated a word n-gram with a language model containing phrases that were salient for a call-routing task, and observed that a slight word accuracy improvement resulted in a substantial improvement in understanding accuracy. Our finding is even more drastic here — we observed that the inclusion of semantic information in ASR language model resulted in worse word error rate but better SLU error rate.

It is also important to note that in this experiment the HMM/CFG composite model only used around 1700 training samples of the ATIS3 training data, while the n-gram solution used all the ATIS2 and ATIS3 training data, which consist of more than 6000 training samples. Theoretically this comparison is not fair to the HMM/CFG composite model. However, the comparison is informative in practice, because it is much easier to obtain unannotated data for LM training in a two-pass system than the annotated data required by the HMM/CFG composite model. We also conducted a theoretically fair comparison, in which only the 1700 training samples were used for n-gram LM training, which yielded 10.4% WER and 13.1% SLUER with the commercial recognizer, and 7.5% WER and 10.2% SLUER with HapiVite.

3 Grammar Controls

The HMM/CFG composite model exploits CFGs as the lexicalization models for slot fillers, which generally model a specific concept. The concept can be domain-independent, like date, time, or credit card numbers. In this case the CFG for it can be pre-built in a grammar library. The concept may also be domain-dependent, such as insurance policy numbers, or auto part numbers. In such a case, users have to either use the closest generic CFGs in the grammar library, for example, use the generic alphanumeric sequence grammar for a policy number, or build their own customized grammar. The first solution is not specific enough and thus has very high perplexity, which leads to high error rate. The second solution requires manual grammar development. Although it may be a trivial problem for dialog system/SLU experts, it is not as straightforward for regular developers. In reality, the norm for commercial systems is system-initiative, and the major grammar authoring task in building such systems is to model these simple concepts. In spoken dialog research, the norm is mixed-initiative systems, which often take the simple concept grammars for

granted. This leads many developers to an extreme conclusion that mixed-initiative system research is useless. Pieraccini (2004) also lists this problem as a research/industry gap.

As an anecdote that echoes the difficulty of authoring a customized grammar, one of the users of Microsoft Speech Server had built a system to recognize an alphanumeric concept with a very restrictive pattern — it always starts with the same letter sequence. Instead of writing a specific grammar for the particular pattern, the user opted to use the generic library rules, and then write a program to enumeratively correct errors like the confusion of 'A' with '8'. It is interesting for SLU researchers to understand what kept the developers away from writing their own customized grammar. According to discussions with the developers, there are four major areas of difficulty:

(1) It is hard to anticipate various expressions that refer to the same meaning. For example, "520" can be "five two oh", "five two zero", "five twenty", "five hundred twenty," etc.
(2) It is hard to normalize speech inputs (like all the variations for "520") with SI tags. To write correct SI tags, developers have to make sure that all the alternates in a rule return the semantics in the same format, and clearly understand what should be expected from a child node in every possible parse tree. This interdependency of SI tags across different structure levels makes it difficult to design the tags coherently.
(3) It is hard to optimize grammar structures for best recognition performance. Two grammars may accept the same language, but have significantly different impacts on the recognition speed. For example, a grammar that is not prefixed properly can result in a large fan-out at initial stages of decoding and significantly slow down the decoding speed. Developers need to have knowledge about the recognizer's search algorithm in order to develop a grammar that promises a high recognition speed.
(4) SRGS adopts XML as the language for grammars. While it has a lot of technical advantages, its verbosity is a source of errors in manual grammar development. An error analysis of a partner's grammars revealed that developers often forget to bracket alternates with the "<one-of>" tag, resulting in broken paths by cascading rather than listing from the grammars. To make matters worse, there are no debugging tools that can detect this type of errors easily.

These problems may not be difficult for SLU experts. However, it is expensive and often not viable to hire an expert to write a customized grammar. An alternative is to encapsulate the expert-level grammar implementation and hide these details from users. Using GUI programming as a metaphor, developers only need to specify what they need (e.g., OpenFile instead of OpenFolder) by selecting the right control (a prepackaged component) and customize the control to their own need with some parameters (e.g., set the initial direc-

tory shown in the GUI dialog). They do not have to know all the details about how an OpenFile dialog is drawn on the screen and how an event is thrown. For grammar development, controls can be provided for frequently used concepts. Users can customize the controls to their own needs with the control parameters. The implementation details, including the anticipation of different expressions, the SI tags, the grammar structure optimization and the SRGS syntax, are all taken care of by the language technology experts who encode their expert knowledge in the implementation of the controls. They are transparent to the grammar developers.

3.1 Basic Controls

Table 2 lists the basic grammar controls for some frequently used concepts. Wang and Ju (2004) contains the pseudo-code for the algorithm that generates the SRGS grammar for the ANC control.

Table 2
Basic grammar controls.

Name	Description	Parameter	Example
ANC	Alphanumeric	RegExp	ANC(\d{3}-\d{2}-\d{4})
CAR	Cardinal num.	Number range	CAR(1-31)
CAR	Cardinal num.	Number set	CAR(1, 2, 4, 8)
ORD	Ordinal num.	Number range	ORD(1-31)
ORD	Ordinal num.	Number set	ORD(1, 2, 4, 8)
LST	Item list	String items	LST(apple, pear, orange, peach)
LST	DB entries	Serv:DB:Tab:Col	LST(speech:ATIS:City:cityname)

The ANC control generates grammars for alphanumeric concepts, such as insurance policy numbers, auto part numbers, etc. Its parameter is a regular expression that describes the pattern of the alphanumeric string. Table 2 shows an example of the parameter that generates the grammar for U.S. social security numbers. Developers are instructed to describe the pattern as specifically as possible. If they know that a specific letter or digit must appear at a particular position, they should mark it in the regular expression such that the grammar is more constrained and has lower perplexity. The grammar created by the control compiler from a regular expression, for example, "AAA", has broader coverage than the finite state machine constructed from the same regular expression by the standard algorithm in the formal language and automata theory — the former automatically takes care of the linguistic variations and accepts both "A A A" and "Triple A" and transduces them to "AAA", while the latter only accepts "A A A". Another difference is that a

control created grammar also allows optional case marks (e.g., "Capital M C uppercase G R A D Y" for "McGrady").

The CAR and ORD controls generate grammars for cardinal and ordinal numbers (non-negative integers), specified by either a parameter for the range of the numbers or a parameter that lists the numbers in a set. The grammar is capable of modeling different ways to utter a number, for example, "five twenty" and "five hundred twenty" for number 520. In the implementation of this control, trivial listing of all possible ways to speak all the numbers is not the best practice. In order to make the grammar compact and efficient for speech recognition, a lot of structure sharing and prefixing need to be performed. Appendix A shows the SRGS grammar created with the basic control ORD(1-31).

The LST control generates the grammar for a list of items. The parameter specifies either the items in the list or a column in a database table. For many speech applications, there already exists a database and a lot of the information in the database can be used to produce a grammar automatically. The example in Table 2 shows the control that generates a City grammar from the "ATIS" database that resides on the server "speech". Items are taken from the "cityname" column of the "City" table in the database to populate the grammar. The control implementation is able to correctly handle the normalization issues for words like "Y2K" and "B2B".

3.2 Operations on Grammar Controls

The basic controls can be used to create grammars for simple concepts. However, developers often face more complicated tasks that the basic controls do not cover. In this section, we introduce several control operations that allow the generation of more complicated grammars. Table 3 lists the operations that were deemed useful when we attempted to use the basic controls as the building blocks to construct a variety of complicated grammars — for example, date and time grammars.

The concatenation operator combines two operand rules sequentially to form a more complicated rule. For example, "LST(April, June, September, November) $\otimes$ ORD(1-30)" generates the date grammar for the months with 30 days.

The paste operation pair-wisely concatenates the entries in the operand rules, rather than the rules themselves. The operand rules of this operation must contain the same number of entries. Assume that table T of database DB on server S contains a column for employees' first names and a column for their last names, then the paste operation "LST(S:DB:T:firstname) $\oplus$ LST(S:DB:T:lastname)" creates a grammar that correctly models all the legitimate employee names,

Table 3
Operations on grammar controls.

Operator	Operation	Description
$\otimes$	Concatenation	Concatenate two operand rules sequentially
$\oplus$	Paste	Pairwise concatenate the entries in the two operands
$\odot$	Normalization	Use the entries in the second operand as the semantic interpretations for the entries in the first operand rule
$+$	Union	Add up all the entries in the two operand rules
$\bullet$	Composition	Compose the two controls. Currently the first operand can only be the ANC control.

while the concatenation operation will result in a grammar that accepts any first name/last name combination.

The normalization operation associates an entry in the second operand rule as the normalized semantics of the corresponding entry in the first operand rule (via SI tags) in a pair-wise fashion. Assume that the database table T in the previous example also has an employee ID column eid, "(LST(S:DB:T:firstname) $\oplus$ LST(S:DB:T:lastname)) $\odot$ LST(S:DB:T:eid)" results in a grammar that accepts an employee's name and returns his/her employee ID. Similarly, the operation "LST(Monday, . . . , Sunday) $\odot$LST(1,. . . ,7)" accepts a weekday name and returns the corresponding number as its semantics.

The union operation simply joins the entries in two operand rules. For example, the operation "LST(January,March,May,July,August,October,December) $\otimes$ ORD(1-31) + LST(April,June,September,November) $\otimes$ ORD(1-30) + LST(February) $\otimes$ ORD(1-28)" produces a simple "date" grammar.

The composition operation is more restrictive. The left operand must be an ANC control. It will result in a spelling grammar for the entries in the second operand rule. For example, "ANC$\bullet$LST(speech:ATIS:City:cityname)" accepts utterances like "S E A T T L E" or "S E A double T L E" and returns "Seattle" as the semantics for the two utterances.

The combination of these operations, together with the basic grammar controls, provides a powerful way to generate various complicated speech recognition grammars.

We compare the speech recognition accuracy resulting from a customized grammar built by grammar controls with the accuracy obtained with the closest generic grammar from the grammar library of Microsoft Speech Application Software Development Kit (MS SASDK). This reflects how much improvement grammar controls can bring to a speech application over the best scenario without customized grammars. We focused our study on the most frequently used ANC control. Three different alphanumeric grammars were created for social security numbers (SSN), license plate numbers (LPN), and Washington State driver license numbers (WADL) from the ANC grammar control. Table 4 shows the parameterized controls that were used to generate the grammars.

Table 4
Parameterized grammar controls that generate the W3C speech recognition grammars.

Concept	Controls
SSN	ANC(\d{3}-\d{2}-\d{4})
LPN	ANC(\d[A-Z]{3} \d{3})
WADL	ANC([A-Z]{2}[A-Z*]{3}[A-Z][A-Z*]\d{3}[A-Z]{2})

220 samples were generated randomly for each concept from the regular expressions used by the grammar controls, except for the WADL. The Washington State driver license number starts with the five initial letters of a person's last name (filled with '*' if the last name is shorter), followed by the first initial and the middle initial ('*' if no middle initial), and then three digits and two letters. We randomly picked 220 last names from the US Census Bureau's 1990 census, took the first five letters from each name and appended it with a string randomly generated according to the regular expression "[A-Z][A-Z*]\d{3}[A-Z]{2}."

Speech data was collected from eleven subjects. The subjects were Microsoft employees, including five speech researchers, four software engineers/testers and two administrative members. The subjects were told what the numbers stood for, and they were instructed to read them out in the normal way they speak them in their daily life. Twenty utterances per concept were collected from each subject.

Microsoft commercial recognizer was used in the experiments, together with its default speaker independent acoustic model. For each concept, we used as the baseline the general fixed-length alphanumeric grammar from the grammar library. The same grammar library also contains the rule USSocialSecurity (Lib/SSN). We used it as an alternative (presumably better) baseline for the

Table 5
Character error rate (CER) and semantic error rate (SER) for the SSN recognition task. Digit9 is the grammar for the digit string of length 9. Lib/SSN is the SSN grammar in the grammar library. ANC/SSN is generated from the ANC grammar control in Table 4.

	Digit9	Lib/SSN	ANC/SSN
CER	17.8%	7.5%	1.8%
SER	33.2%	22.3%	13.2%

Table 6
Character error rate (CER) and semantic error rate (SER) for the license plate number recognition task. AN-7 is the grammar for the alphanumeric string of length 7. ANC/LPN is generated from the ANC grammar control in Table 4.

	AN-7	ANC/LPN
CER	10.8%	4.9%
SER	47.7%	22.7%

SSN task. The recognition outputs were then compared with the original numbers used in data collection. The statistics of the character error rates and the semantic error rates were collected — a recognition hypothesis is considered a semantic error if one or more character errors occur.

Table 5 shows the error rates of the three different grammars for the SSN task. The grammar generated by the ANC grammar control (ANC/SSN) cut the semantic error rate (row SER, column ANC/SSN) by more than 60% when it is compared with the fixed length digit sequence grammar (row SER, column Digit9), and over 40% when it is compared with the library SSN grammar (row SER, column Lib/SSN). The digit sequence grammar has the highest error rate because it doesn't cover the digit sequences that were read as numbers (e.g., "one hundred twenty three" for "123"). The library SSN grammar (Lib/SSN) has higher error rate than that of ANC/SSN. This is an artifact of the explicit read-out of the hyphens in SSNs by one speaker. The hyphen is not modeled by, and not straightforward to add to the library SSN grammar. This demonstrates that grammar controls effectively provide a customized alternative when the existing library grammar fails to model some speakers' peculiar utterances.

Table 6 compares the license plate number grammar created by the ANC grammar control (ANC/LPN) with the baseline grammar. Because ANC/LPN modeled the diversity of the digit sequence expressions, as well as the constraints on the locations where a digit or a letter is expected, it cut the error rate by more than 50%. The location constraints significantly reduce the fan-out at each grammar state (and hence the perplexity) and eliminate the frequently observed confusions between '8' and 'A'.

Table 7
Character error rate (CER) and semantic error rate (SER) for the Washington State driver license number task. AN-12 is the grammar for the alphanumeric string of length 12. ANC/WADL is generated from the ANC grammar control in Table 4.

	AN-12	ANC/WADL
CER	24.5%	23.6%
SER	81.4%	63.2%

Finally, Table 7 lists the error rates of the two WADL grammars. The grammar created from the grammar control (ANC/WADL) again has better accuracy. However, both grammars have a very high error rate for this task. An error analysis shows that the high error rates can be attributed to the following two reasons:

(1) There are more letters in a WADL number, and letters are acoustically more confusable than digits. The error analysis shows that the most confusable letters are 'S' vs. 'F', and then the E-set letters B, D, E, G, P, and T. This also explains why the license plate numbers has higher error rate than the social security numbers.
(2) WADL numbers often contain pronounceable substrings (last names or last name prefixes). Many subjects (often the speech researchers who tried hard to break the system) opted to pronounce the name instead of spelling out the letters. About 8% of the data was read out with the substring pronunciation, which is completely not covered by either grammar.

The results are not surprising since the grammars generated by the grammar controls are more constrained and therefore have lower perplexity. Nevertheless, it shows the benefit of grammar controls. Grammar controls provide an efficient way to create grammars customized to specific concepts in an application. This is not only helpful to regular developers without a background in dialog systems, who often find it difficult to create customized grammars, but also useful for spoken dialog experts to save time in grammar development. The tool has been well received by developers. As an example, a developer reported that he still could not make the SI tags work properly after spending a whole morning on a grammar for driver license numbers, and credited the tool with grammar controls for building the grammar for him in literally a couple of seconds.

4 Grammar Configurability

The HMM/CFG model was designed for mixed-initiative systems in which an object with a specific type can be the filler of different slots, e.g. a city

name can be the filler for either the departure or the arrival city slot. It still requires labeled training data even though the inclusion of the domain knowledge in the model has significantly reduced the requirement. In many different application scenarios, simplified model topologies that require even less or no training data are more suitable. In this section we illustrate that some configuration parameters of SGStudio can customize the HMM/CFG model topology to fit different application scenarios.

4.1 Configuration Parameters

We start with the introduction of the configuration parameters. In the next subsection we will show how they can be used to configure the model topology for different application scenarios.

Table 8 lists the configuration parameters. The "backbone" parameter controls the overall model structure. The value of the parameter is either "Template_HMM" or "Domain_Ngram". The difference between the two parameter values lies in the way the dependency for a lexical item is modeled. In the case of "Template_HMM" setting, the words that a state emits only depend on the state and the history of words from the same state. The first word of the state depends on the context cue of sentence beginning, "<s>". On the other hand, in the "Domain_Ngram" setting, a history may also include the previous state; or the words from the previous state when the previous state is modeled with a pooled (tied) n-gram (see the model type parameter in next paragraph). As an example for this setting, if preambles are modeled with a pooled n-gram, the slot filler state will depend on the words in the pooled n-gram as well.

The model type parameter applies to four different model states: slot preambles and postambles, and the precommands and postcommands for top level tasks. The parameter takes one of the five possible values. "None" means that the specific state should be omitted from the general model topology (e.g., slot postambles are often optional in English); "Wildcard" indicates that there is no specific language model for the state. A phone loop model should be used to accept any acoustic inputs; "PooledNgram" ties the model with the models of all the other "PooledNgram" states. The training data for all those states are pooled together. These three model types eliminate or reduce the requirement for training data. The value "Ngram" results in the standard composite model described in Section 2; and the "Rule" value lets the model use CFG rules instead of n-gram models for the particular type of states.

Table 8
The configuration parameters. The backbone parameter controls the overall model topology, while the ModelType parameters determine the types of statistical models used for the precommand, postcommand, preamble and postamble states.

Parameter	Application	Value	Description
Backbone	Model Topology	Template_HMM	Using the HMM with different preamble/postambles
		Domain_Ngram	Use the unified LM as the backbone of the model
Model Type	Preamble Postamble PreCommand PostComamnd	None	Don't model it
		Wildcard	Use wildcard model
		PooledNgram	Share an n-gram model
		Ngram	Use specific n-gram models
		Rule	Use specific CFG rules

4.2 Application Scenarios

4.2.1 Scenario 1: Key phrase spotting for system-initiated dialogs

We have shown that grammar controls can create grammars for system-initiated systems. The grammars model exactly what the system is expecting from a user. However, users' response may contain unexpected insertions. For example, in a pizza ordering application, the system may prompt a user for the size and toppings of the pizza. The user may reply "I want to have a large pizza with mushroom and cheese" or "medium pepperoni please". The SLU component needs to spot the word "large" and "medium" for size and "mushroom", "cheese" and "pepperoni" for toppings from the user's utterances. When there is no training data available, a wildcard model can be used for this task, with a phone loop model that picks everything except for the key phrases. This is shown in Fig. 12.

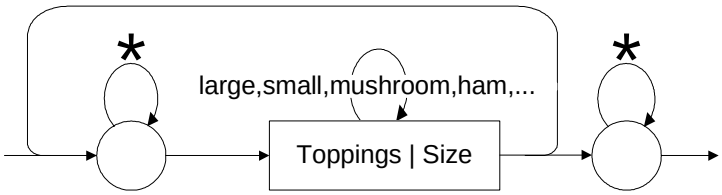

Fig. 12. The wildcard model: the key phrase model is bracketed by the wildcards (*) that can match anything a user may say. The task of recognition/understanding is to extract the key phrases from a user's utterance.

Fig. 13 shows the grammar configuration that generates the wildcard model.

```
<GrammarConfiguration>
    <Preamble>Wildcard</Preamble>
    <Postamble>None</Postamble>
    <PreCommand>None</PreCommand>
    <PostCommand>Wildcard</PostCommand>
< /GrammarConfiguration>
```

Fig. 13. The grammar configuration that creates the wildcard model. The precommands and the postambles of the general topology in Fig. 5 are omitted. All preambles are modeled with the wildcard so they collapsed into the left wildcard node in Fig. 12, and all postcommands collapsed into the right wildcard node.

While the model is robust and requires no training data, it often has a high insertion error rate. Because the wildcard model is very flat, it tends to assign lower probability to the matched acoustic feature frames. Therefore the input acoustics are more likely to be matched with the key phrase model, which results in false positive errors. This model can be used to deploy an initial system, and more sophisticated systems can be built as the data become available after deployment.

4.2.2 Scenario 2: Multiple slots without ambiguities/Unified language model

The wildcard model can be improved when data are available. If there is no slot ambiguity, i.e., the fillers for one slot cannot be the filler for another slot, then there is no need to have separate preambles and postambles for different slots as the contextual cues for slot disambiguation. We can lump together all the n-grams for the precommands, preambles, postambles, as well as the backbone slot n-grams (with the "Domain_Ngram" backbone setting) together. In doing so, the number of parameters is greatly reduced, and thus so is the requirement for training data. Note that there is no need for the postcommand states, since the words that occur at the end of a sentence are modeled by the postamble following the last slot, which is the same as the backbone n-gram model. The grammar configuration is shown in Fig. 14, and the resulting grammar structure in illustrated in Fig. 15.

The model in Fig. 15 forms the basis of the unified language model. It was first introduced in [Wang et al. (2000)] and had been further studied in [Dolfing (2004), Wang et al. (2004)]. The unified language model is a generalization of the class-based n-gram model, where PCFG non-terminals, as the word classes in the class-based n-gram model, are introduced into the n-gram model as "super-words". From the super-words, the terminal lexical items are generated according to the distributions defined in the PCFG. Let's consider the sentence "meeting at four PM with Derek." With the traditional n-gram model, the probability of the sentence is

```
<GrammarConfiguration>
    <Backbone>Domain_Ngram</Backbone>
    <Preamble>PooledNgram</Preamble>
    <Postamble>PooledNgram</Postamble>
    <PreCommand>PooledNgram</PreCommand>
    <PostCommand>None</PostCommand>
< /GrammarConfiguration>
```

Fig. 14. The configuration that creates the unified language model. The backbone of the model is an n-gram, which is tied with the preamble, postamble and precommand n-grams. The training data for all the n-grams are pooled together. This effectively results in a single n-gram model that predicts both the terminal words and the CFG non-terminals.

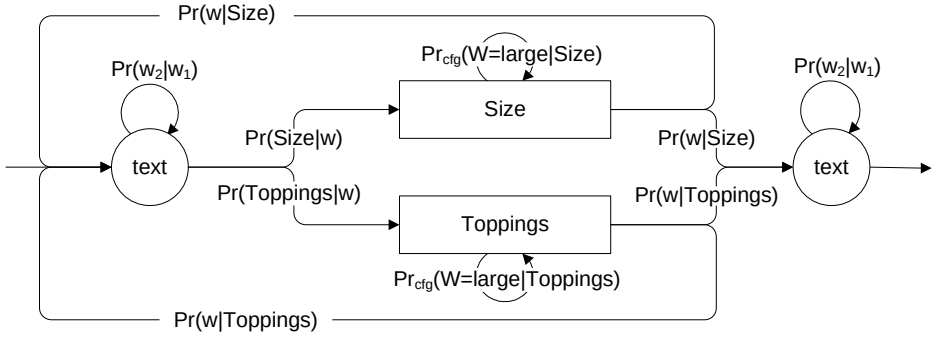

Fig. 15. Topology of the unified language model, in which words and the CFG non-terminals are modeled together in the backbone n-gram model.

$$\begin{aligned}\Pr(\text{“meeting at four PM with Derek”}) = \\ \Pr(\text{meeting}|\text{<s>}) \times \Pr(\text{at}|\text{meeting}) \times \\ \Pr(\text{four}|\text{at}) \times \Pr(\text{PM}|\text{four}) \times \Pr(\text{with}|\text{PM}) \times \\ \Pr(\text{Derek}|\text{with}) \times \Pr(\text{</s>}|\text{Derek})\end{aligned} \tag{7}$$

In the unified language model, the CFG non-terminals, for example, <Name> and <Time>, are introduced into the n-gram. The process that generates the previous example sentence is illustrated in Fig. 16. The probability of the example is

$$\begin{aligned}\Pr(\text{“meeting at four PM with Derek”}) = \\ \Pr(\text{meeting}|\text{<s>}) \times \Pr(\text{at}|\text{meeting}) \times \\ \Pr(\text{<Time>}|\text{at}) \times P_{cfg}(\text{four PM}|\text{<Time>}) \times \\ \Pr(\text{with}|\text{<Time>}) \times \Pr(\text{<Name>}|\text{with}) \times \\ P_{cfg}(\text{Derek}|\text{<Name>}) \times \Pr(\text{</s>}|\text{<Name>})\end{aligned} \tag{8}$$

The introduction of the CFG non-terminals makes the unified language model generalize better. For example, the sentence “meeting at noon with Peter Johnson” will share the same underlying structure as the one in Fig. 16. They

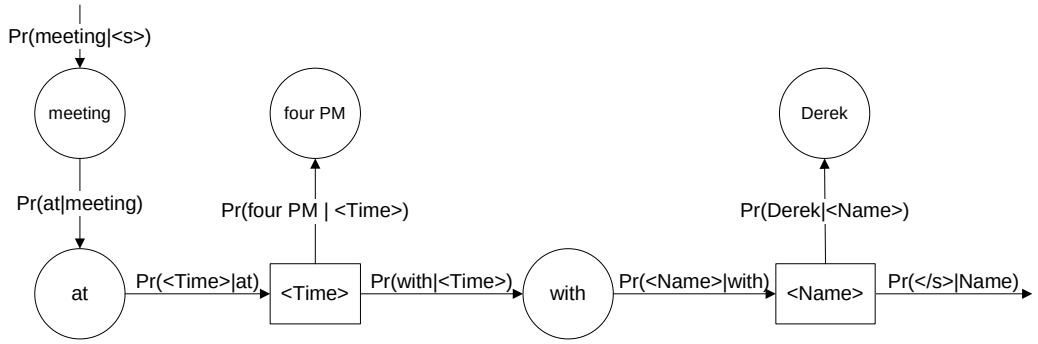

Fig. 16. The state sequence of the unified language model for the utterance "meeting at four PM with Derek". The circles represent the emissions of lexical items, and the rectangles represent the PCFG rules.

only differ in the emissions of the lexical items from the CFG states "<Time>" and "<Name>". Furthermore, the introduction of the non-terminals enables the speech recognizer to identify and output the slot semantics.

The absence of state specific preamble and postamble models make the model unable to use context to disambiguate the slots. For applications like ATIS where a city can be the departure, arrival or connecting city, and a time can be either a departure or a arrival time, the original HMM/CFG composite model is more appropriate.

4.2.3 Scenario 3: Grammar for Manual Maintenance

One disadvantage of the original HMM/CFG composite model is that the SRGS grammar created from it is not easy to read/understand due to the n-gram to FSA conversion. It is very difficult to manually modify the automatically learned grammar if the developers opt to do so. A compromise is to sacrifice the robustness for maintainability by replacing n-gram models with CFG rules for the precommand, postcommand, preamble and postamble states. Note that this does not affect the training algorithm very much. The only modification is to run a Viterbi segmentation at the final stage for all training examples and use the word sequence aligned with each state as a rule for the state. The probability for the rule is just the normalized expected count calculated by the EM algorithm described in Wang et al. (2005). Fig. 17 shows the configuration and the topology of such grammars.

While the resulting grammar is more maintainable, it is less robust because it requires the input sentences match the precommands, postcommands, and preambles exactly as seen in the training sentences. Normally this causes a 15∼20% (relative) increase in error rate.

```
<GrammarConfiguration>
    <Backbone>Template_HMM</Backbone>
    <Preamble>Rule</Preamble>
    <Postamble>None</Postamble>
    <PreCommand>Rule</PreCommand>
    <PostCommand>Rule</PostCommand>
</GrammarConfiguration>
```

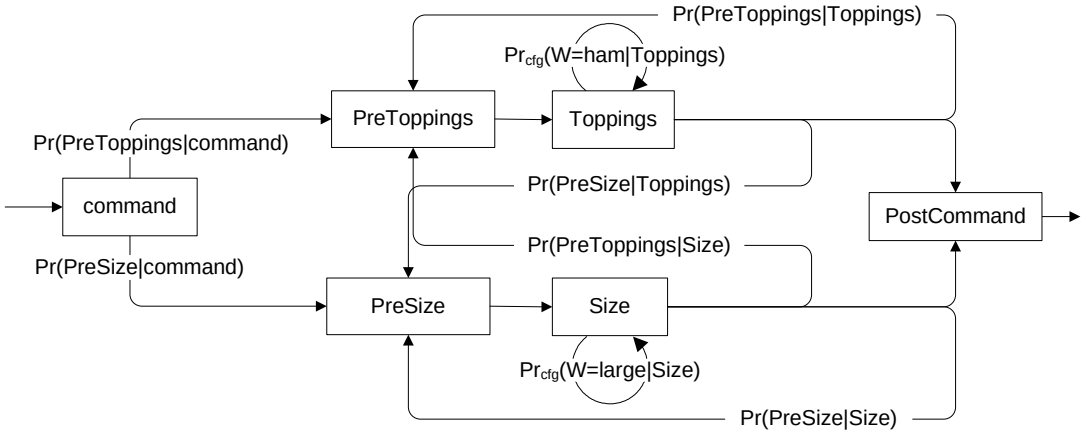

Fig. 17. The configuration and the topology of the more maintainable HMM/CFG composite model. The rectangles represent PCFG rules.

4.2.4 Scenario 4: Call Routing/Classification

The HMM/CFG composite model can also be configured for call routing applications.

If we omit all the preambles, postambles and postcommands, and use a separate n-gram model for the precommand of each of the top level tasks, as shown in the configuration in Fig. 18, then we end up with the topology shown in Fig. 19. In this case, each precommand n-gram is a unified language model — or a regular word n-gram model if there is no slot for the task frame. For an input sentence W, the recognizer picks the task that maximizes the *a posteriori* probability:

$$\text{Task} = \arg\max_{task_i} \Pr(task_i) \times \Pr(W|task_i) \tag{9}$$

This is the n-gram classifier described in Chelba et al. (2003).

Chelba et al. (2003) compared the model with other classifiers. The experiments were conducted with the ATIS data. It used all the ATIS2 and ATIS3 training data for training and ATIS3 test data ($\sim$ 900 utterances) for testing. The main database table name in the reference SQL query was taken as the class label for each utterance. There are 14 class labels.

Table 9 highlights some of their key findings on this data set with the ML

```
<GrammarConfiguration>
    <Backbone>Template_HMM</Backbone>
    <Preamble>NGram</Preamble>
    <Postamble>None</Postamble>
    <PreCommand>None</PreCommand>
    <PostCommand>None</PostCommand>
</GrammarConfiguration>
```

Fig. 18. The configuration and the topology for the call routing grammar.

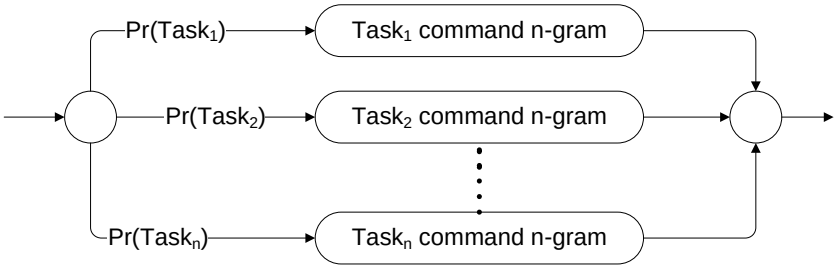

Fig. 19. HMM/CFG model's call-routing topology.

trained classification models, including a naive Bayes classifiers (NB) [Duda et al. (2001)], a two-pass trigram classifiers (2-pass Trigram), and a one-pass trigram (2-pass Trigram) classifier that is equivalent to the SGStudio generated n-gram classification model.

Table 9
Comparison of ML trained classifiers. All classifiers except for the last one operate in two passes. A trigram LM is used in the first pass ASR, and the classification models are applied in the second pass.

Classifier	WER	CER
NB	5.1%	9.7%
2 pass Trigram	5.1%	9.4%
1 pass Trigram	5.5%	9.0%

The one-pass trigram classifier works with the lowest classification error rate among the models, even though the word error rate is higher — this is another piece of evidence that reveals the the broken semantics/surface sentence/acoustic observation dependency chain results in a suboptimal solution.

The experiments were conducted in the ATIS domain simply due to the readiness of the data. Readers are cautioned that ATIS is not a good call routing application. We are in the process of obtaining more realistic call routing data to test the system. Also it has been shown that models trained discriminatively have better classification accuracy than ML trained models [Kuo et al. (2002); Hakkani-Tür et al. (2004)]. We are currently investigating discriminative training for the HMM/CFG composite model.

5 Ongoing and Future Work

This article has investigated the problem of grammar authoring for initial system deployment when little data is available. Potentially, the HMM/CFG composite model can also be used for system adaptation/tuning when data are available after the initial system deployment. In fact, we are currently working on a dialog system tuning tool that improves the model with system logged data.

While preliminary experimental results show that SGStudio can generate statistical models that accomplished good classification performance compared to other ML trained models, there is potentially a significant room for improvement with discriminatively trained models. We are currently investigating the discriminative model training, not just for call classification, but also for high resolution SLU tasks. We are also in the process of obtaining more realistic call routing data for evaluation.

SGStudio, like many other SLU systems, adopts a frame-based domain semantic definition, which is used to build the prior model topology. While the grammar topology and parameters are automatically learned, the model still depends on the developers description of domain semantics based on his/her domain knowledge. While the semantic frame is closely related to the application database schema, there is no systematic derivation of the semantic frames from the database schema. The quality of the description of domain semantics, to a large extent, depends on the developer's experience. Although semantic definition is a much simpler problem than grammar development, the systematic derivation of the semantic frame remains a very important and interesting research topic.

6 Conclusions

This article addresses the problem of grammar authoring in spoken dialog systems when the application developer has little expertise in human language technology. Previous data-driven grammar learning methods suffer from the problem that little training data is available at initial stages of system development. This practical problem has not been sufficiently addressed in the research community, and remains one of the major obstacles to the wide acceptance of spoken dialog technology. To close the gap, we introduced SGStudio, a tool that aims at enabling a regular developer to create high quality grammars for ASR and SLU. The major contributions include:

(1) The HMM/CFG composite model that includes domain knowledge as an

integral part of the statistical model in a principled way. It alleviates the data-sparseness problem and enables tight ASR/SLU coupling, and hence improves the overall understanding accuracy.

(2) The introduction of grammar controls and control operations. They provide a very powerful tool to create grammars for domain-specific concepts. This complements the data-driven statistical modeling approach.

(3) The combination of the HMM/CFG composite model, grammar controls technology and a configurable model structure results in a very practical tool for speech understanding grammar authoring. It increases the productivity of spoken dialog system development.

Experimental results show that the HMM/CFG composite model achieved better SLU accuracy on the ATIS task with much less training data, and the novel use of grammar control technology resulted in customized grammars that best fit users' needs. We believe that the marriage of spoken language modeling research and the best engineering practice presented in this article is a solid step towards closing the research/industry gap in spoken dialog systems, and it will promote broad adoption of speech technology.

7 Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions, and acknowledge the help from Ciprian Chelba and Mike Seltzer for their feedback on the manuscript.

A An Exemplar Grammar Control Generated Grammar

The following grammar is generated with the ordinal number grammar control ORD(1-31):

```
<?xml version="1.0"?>
<grammar xml:lang="en-US" xmlns="http://www.w3.org/2001/06/grammar"
root="root" tag-format="semantics-ms/1.0" version="1.0">
<! − −ORD(1-31)–>
  <rule id="root" scope="public">
    <tag>prefix=0</tag>
    <one-of>
      <item>
        <ruleref uri="#Ones_1_9th"/>
        <tag>$=$$</tag>
      < /item>
```

```
      <item>
         <ruleref uri="#Tens_10_29th"/>
         <tag>$=$$</tag>
      < /item>
      <item>
         <ruleref uri="#Tens_30_31th"/>
         <tag>$=$$</tag>
       < /item>
   < /one-of>
   <tag>$=$+prefix</tag>
< /rule>

<rule id="Ones_1_9th" scope="private">
   <one-of>
      <item>first <tag>$=1</tag></item>
      <item>second <tag>$=2</tag></item>
      <item>third <tag>$=3</tag></item>
      <item>fourth <tag>$=4</tag></item>
      <item>fifth <tag>$=5</tag></item>
      <item>sixth <tag>$=6</tag></item>
      <item>seventh <tag>$=7</tag></item>
      <item>eighth <tag>$=8</tag></item>
      <item>nineth <tag>$=9</tag></item>
   < /one-of>
< /rule>

<rule id="Tens_30_31th" scope="private">
   <one-of>
      <item>thirty first <tag>$=31</tag></item>
      <item>thirtieth <tag>$=30</tag></item>
   < /one-of>
< /rule>

<rule id="Tens_10_29th" scope="private">
   <one-of>
      <item>tenth <tag>$=10</tag></item>
      <item>eleventh <tag>$=11</tag></item>
      <item>twelfth <tag>$=12</tag></item>
      <item>thirteenth <tag>$=13</tag></item>
      <item>fourteenth <tag>$=14</tag></item>
      <item>fifteenth <tag>$=15</tag></item>
      <item>sixteenth <tag>$=16</tag></item>
      <item>seventeenth <tag>$=17</tag></item>
      <item>eighteenth <tag>$=18</tag></item>
      <item>nineteenth <tag>$=19</tag></item>
```

<item>twentieth <tag>$=20</tag></item>
<item>twenty <tag>$=20</tag>
<ruleref uri="#Ones_1_9th"/><tag>$=$+$$</tag>
</item>
< /one-of>
< /rule>
< /grammar>

References

Allen, J. F., Miller, B. W., Ringger, E. K., Sikorshi, T., 1996. Robust understanding in a dialogue system. In: 34th Annual Meeting of the Association for Computational Linguistics. Santa Cruz, California, USA, pp. 62–70.

Bangalore, S., Johnston, M., 2004. Balancing data-driven and rule-based approaches in the context of a multimodal conversational system. In: Human Language Technology/Conference of the North American Chapter of the Association for Computational Linguistics. Boston, MA, USA.

Carpenter, B., Chu-Carroll, J., 1998. Natural language call routing: a robust, self-organizing approach. In: International Conference on Speech and Language Processing. Sydney Australia.

Chelba, C., Mahajan, M., Acero, A., 2003. Speech utterance classification. In: IEEE International Conference on Acoustics, Speech, and Signal Processing. Hong Kong, China.

Della Pietra, S., Epstein, M., Roukos, S., Ward, T., 1997. Fertility models for statistical natural language understanding. In: 35th Annual Meeting of the Association for Computational Linguistics. Madrid, Spain, pp. 168–173.

Dempster, A. P., Laird, N., Rubin, D. B., 1977. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society B 39, 1–38.

Dolfing, H., 2004. Unified language modeling using finite-state transducers with first applications. In: International Conference on Spoken Language Processing. Jeju, Korea.

Dowding, J., Gawron, J. M., Appelt, D., Bear, J., Cherny, L., Moore, R., Moran, D., 1993. Gemini: A natural language system for spoken-language understanding. In: 31st Annual Meeting of the Association for Computational Linguistics. Columbus, Ohio, pp. 54–61.

Duda, R. O., Hart, P. E., Stork, D. G., 2001. Pattern Classification. John Wiley and Sons, Inc.

Estéve, Y., Raymond, C., Bechet, F., Mori, R. D., 2003. Conceptual decoding for spoken dialog systems. In: Eurospeech 2003. Geneva, Switzerland.

Fu, K. S., Booth, T. L., 1975a. Grammatical inference: Introduction and survey, part 1. IEEE Transactions on Systems, Man and Cybernetics 5, 85–111.

Fu, K. S., Booth, T. L., 1975b. Grammatical inference: Introduction and sur-

vey, part 2. IEEE Transactions on Systems, Man and Cybernetics 5, 409–423.

Gorin, A., 1995. On automated language acqusition. Journal of Accoustical Society of America 97 (6), 3441–3461.

Hakkani-Tür, D., Tur, G., Rahim, M., Riccardi, G., 2004. Unsupervised and active learning in automatic speech recognition for call classification. In: IEEE International Conference on Acoustics, Speech, and Signal Processing. Montreal, Canada.

He, Y., Young, S., 2005. Semantic processing using the hidden vector state model. Computer Speeech and Language 19 (1), 85–106.

Hunt, A., McGlashan, S., 2002. Speech recognition grammar specification version 1.0. http://www.w3.org/tr/speech-grammar/.

Jelinek, F., Lafferty, J. D., Mercer, R. L., 1990. Basic methods of probabilistic context free grammars. Tech. Rep. RC 16374, IBM T.J. Watson Research Center, Yorktown Heights, N.Y.

Kuo, H.-K. J., Zitouni, I., Fosler-Lussier, E., Ammicht, E., Lee, C.-H., 2002. Discriminative training for call classification and routing. In: International Conference on Spoken Language Processing. Denver Colorado.

Macherey, K., Och, F. J., Ney, H., 2001. Natural language understanding using statistical machine translation. In: Eurospeech 2001.

Miller, S., Bobrow, R., Ingria, R., Schwartz, R., 1994. Hidden understanding models of natural language. In: 31st Annual Meeting of the Association for Computational Linguistics. New Mexico State University.

Pargellis, A., Fosler-Lussier, E., Potamianos, A., Lee, C.-H., 2001. Metrics for measuring domain independence of semantic classes. In: Eurospeech 2001. Aalborg, Denmark.

Pieraccini, R., 2004. Spoken language understanding, the research/industry chasm. In: HLT/NAACL Workshop on Spoken Language Understanding for Conversational Systems. Boston.

Pieraccini, R., Levin, E., 1993. A learning approach to natural language understanding. In: 1993 NATO ASI Summer School. New Advances and Trends in Speech Recognition and Coding. Springer-Verlag, Bubion, Spain.

Price, P., 1990. Evaluation of spoken language system: the atis domain. In: DARPA Speech and Natural Language Workshop. Hidden Valley, PA.

Riccardi, G., Gorin, A. L., 1998. Stochastic language models for speech recognition and understanding. In: International Conference on Spoken Language Processing. Sidney, Australia.

Riccardi, G., Pieraccini, R., Bocchieri, E., 1996. Stochastic automata for language modeling. Computer Speech and Language 10, 265–293.

Ringger, E., 2000. Correcting speech recognition errors. Ph.D. thesis, University of Rochester.

Schapire, R. E., Rochery, M., Rahim, M., Gupta, N., 2005. Boosting with prior knowledge for call classification. IEEE Transactions on Speech and Audio Processing 13 (2), 174–181.

Seneff, S., 1992. TINA: A natural language system for spoken language appli-

cations. Computational Linguistics 18 (1), 61–86.

Stolcke, A., Omohundro, S. M., 1994. Best-first model merging for Hidden Markov Model induction. Tech. Rep. TR-94-003, International Computer Science Institute.

Vidal, E., Casacuberta, F., Garcia, P., 1993. Grammatical inference and applications to automatic speech recognition and understanding. Tech. Rep. DSIC II/41/93, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia.

Wang, N. J., Shen, J.-L., Tsai, C.-H., 2004. Integrating layer concept information into n-gram modeling for spoken language understanding. In: International Conference on Spoken Language Processing. Jeju, Korea.

Wang, Y.-Y., 1999. A robust parser for spoken language understanding. In: Eurospeech 1999. Vol. 5. ESCA, Budapest, Hungary, pp. 2055–2058.

Wang, Y.-Y., Deng, L., Acero, A., 2005. Spoken language understanding — an introduction to the statistical framework. IEEE Signal Processing Mazagine 22 (5).

Wang, Y.-Y., Ju, Y.-C., 2004. Creating speech recognition grammars from regular expressions for alphanumeric concepts. In: International Conference on Spoken Language Processing. Jeju, Korea.

Wang, Y.-Y., Mahajan, M., Huang, X., 2000. A unified context-free grammar and n-gram model for spoken language processing. In: IEEE International Conference on Acoustics, Speech, and Signal Processing. Istanbul, Turkey.

Wang, Y.-Y., Waibel, A., 1998. Modeling with structures in statistical machine translation. In: 36th Annual Meeting of the Association for Computational Linguistics/17th International Conference on Computational Linguistics. Montral, Qubec, Canada.

Ward, W., 1994. Recent improvements in the CMU spoken language understanding system. In: Human Language Technology Workshop. Plainsboro, New Jersey.

Wong, C.-C., Meng, H., 2001. Improvements on a semi-automatic grammar induction framework. In: IEEE Automatic Speech Recognition and Understanding Workshop. Madonna di Campiglio, Italy.

Woods, W. A., 1983. Language processing for speech understanding. In: Computer Speech Processing. Prentice-Hall International, Englewood Cliffs, NJ.

Young, S., 1993. The htk hidden markov model toolkit: design and philosophy. Tech. Rep. TR.153, Department of Engineering, Cambridge University.