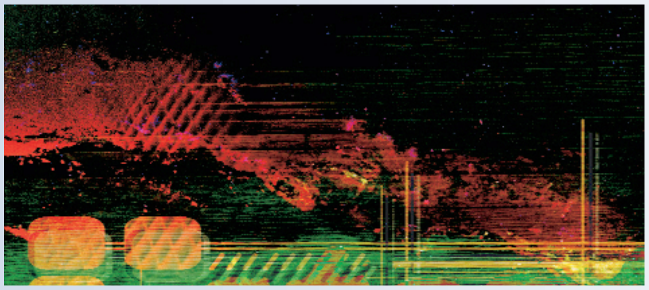
Modeling Complex Spoken Dialog

Increasing task complexity is challenging the models that underlie spoken dialog systems. A customized dialog language, evolved from building an actual system, uses core patterns that modelers can adapt to most applications.



Hans Dybkjær
Prolog Development Center

Laila Dybkjær
NISLab, University of Southern Denmark

Although not as complex as their research counterparts, commercial spoken dialog systems (SDSs) can handle increasingly advanced tasks. This relatively new complexity poses three immediate development challenges. The first is how to model the dialog when the user can select from many tasks. How can the system present the available choices and anticipate what the user will talk about next?

The second challenge is how to communicate with customers about SDS design. When the system is fairly simple, developers can easily learn how to handle the domain to be covered and have relatively little need to communicate with domain experts. When the domain is large and complex, however, developers are less likely to understand its nuances and thus must collaborate more closely with those who do. Thus, choosing the best communication method becomes another development task—a concern SDS developers have not had to deal with traditionally.

The final challenge is how to develop code efficiently for a large SDS when the design is likely to require updates throughout development. A complex application domain poses many language and flow problems, as the "Dealing with Complexity" sidebar describes.

We had to meet all these challenges in our effort to develop a commercial phone-based SDS that supplies information to employees about their holiday allowance. Key to this frequently-asked-questions (FAQ) system was our use of the Conceptual Dialog Language (CDL), a customized language that lets us express patterns specific to the system's dialog model while still providing a clear picture of the dialog model to domain experts and supporting model updates. CDL's power stems from its ability to capture the specifics of an SDS application while remaining flexible enough to support a range of modeling styles and requirements. To facilitate effective communication with domain experts, modelers can translate the dialog model in CDL to an HTML document, or they can compile it to a programming language for executable code.

FAQ SYSTEM

From September 2001 to December 2002, Prolog Development Center and NISLab developed a FAQ SDS that supplies holiday-allowance information on behalf of FerieKonto (www.feriekonto.dk), an institution that administers holiday allowances to Danish employees.[1]

As the right side of Figure 1 shows, the system is a phone-based SDS that uses the High-Level Dialog Description Language (HDDL)[2] and runs on ScanSoft's SpeechMania platform (www.scansoft.com). HDDL, a grammar and dialog description language that Philips developed in 1994, describes dialog in automatic inquiry systems that involve speech recognition. SpeechMania, a platform Philips developed and then sold to Scansoft, includes components for speech recognition, dialog and lexicon management, language modeling, and speech output. The package runs on a Windows PC with a telephone interface board.

Published by the IEEE Computer Society 0018-9162/04/$20.00 © 2004 IEEE

Dealing with Complexity

A well-known software-engineering challenge is how to deal with modeling complexity. Most practitioners use a divide-and-conquer strategy based on object orientation and other encapsulation. Others use communication abstractions in the form of patterns.

Spoken dialog system developers have similar solutions. Some have observed that concrete tasks, such as obtaining a date or a zip code, are common enough to be candidates for reuse as standard library subdialogs.[1] Commercial products like SpeechMania use similar ideas. The subdialogs encapsulate well-proven behaviors as long as the structured subdialog remains simple.

Unfortunately, a subdialog typically requires complex division into parameters, which is problematic because the subdialog is highly context sensitive. In Figure A, for example, the task is to turn "get a date value" into a subdialog and then call it twice to obtain a time interval. The subdialog would require at least three parameters: prompt, input grammar, and input interpretation. This is apart from the added problem of dialog flow.

At every user input, several error situations should give rise to different continuations. Including this as a parameter is difficult, but leaving it at the calling place could greatly increase its complexity.

Language

A common approach has been to divide grammars into subgrammars that correspond to subtasks and have active only the grammars that correspond to the tasks in focus. The drawback is that users and designers never work with the same division of the world. Some have attempted to add keywords to the active grammars to trigger tasks outside the focus. However, because speech recognizers become better, the trend is to have all grammars active all the time. Researchers are currently investigating how to combine subgrammars.[2]

```
[Assume that today is Monday.]
        S1.1: When do you leave?
        U1.1 Friday. [The user means the
first Friday after today]
        S1.2: Leaving Friday, when do you
return?
        U1.2: Wednesday. [The user means
the first Wednesday after Friday, not the
first Wednesday after today.]
```

Figure A. Example of how semantics depend on context, both on the immediate prompt and on the wider discourse and world state. Bracketed text is analysis, not dialog.

Flow

As Figure A shows, a period—in this case the time between Friday and Wednesday—is not simply a combination of two date subdialogs, even if you view the prompt text as a parameter. A full object model would require at least two dates (today, and a relative date), and the model still does not account for repair (what if the user had responded in U1.2 "No, Thursday," as a correction to "Friday"?). Clearly, using objects requires some level of sophistication.

Flow patterns are also important. Natural dialog tends to follow standard patterns that dialog management should support, not least to ensure that the system behaves uniformly so that the user can easily recognize what is going on.

References

1. D.G. Novick and S. Sutton, "Building on Experience: Managing Spoken Interaction through Library Subdialogues," *Proc. 11th Twente Workshop on Language Technology* (TWLT 11), S. Luperfoy, A. Nijholt, and G. Veldhuijzen, eds., University of Twente, 1996, pp. 51-60.
2. M. Rayner et al., "Plug and Play Speech Understanding," *Proc. 2nd SIGdial Workshop Discourse and Dialogue*, J. van Kuppevelt and Ronnie Smith, eds., 2001; www.sigdial.org/workshops/.

Information users need

In Denmark, all full-time employees have at least five weeks of vacation or holiday per year, but only those who have remained full-time workers the previous year can receive a salary during their vacation or holiday. Each month, employers contribute 12.5 percent of an employee's salary to FerieKonto, which saves it as holiday allowance to pay for that person's vacation the next year. Employees who continue in a permanent position receive their usual salary during holidays, and FerieKonto reimburses their employer. Employees who change to another position or have a temporary position, however, must fill in a holiday allowance certificate and submit it to FerieKonto. They then receive their saved holiday allowance instead of a salary.

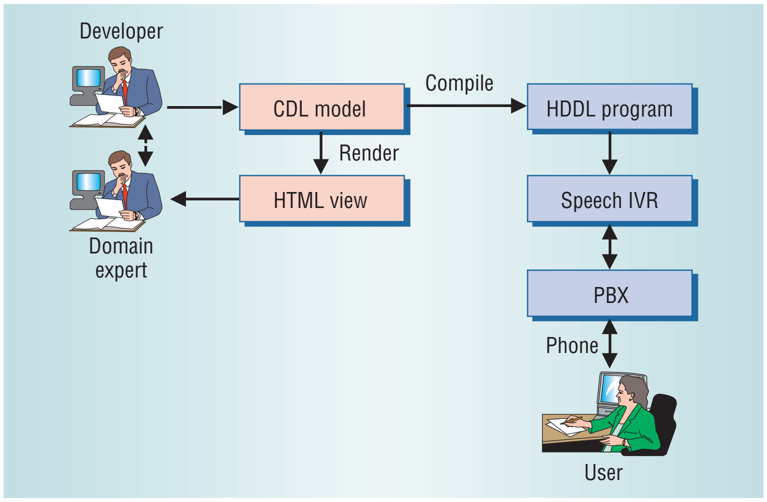


Figure 1. FAQ spoken dialog system (SDS) architecture. Danish employees call the SDS to get information about their holiday allowance. During system design, developers manipulated a CDL model, which they can translate to HTML for communication with domain experts or compile to a dialog program in HDDL, which the speech interactive voice response (IVR) then interprets through the switchboard (PBX).

```
                    [...]
System   13.2       - Do you want information about payment of
                    "small amounts" or "error in amount"?
                    - PAUSE1000
User     13.1       error in amount
Recognised(error in amount)
Concepts(amount/score 736, error/score 771)
System   14.1       - Please remember that tax and labor market
                    payments have been deducted from your holiday
                    allowance before it is paid to us.
                    - If you think the amount is wrong, please
                    contact your employer.
                    - Also contact your employer if a payment is
                    totally missing.
System   14.2       - If you think you have received a wrong amount,
                    please call again and talk to a case officer.
System   14.3       - Do you want information about "payment of
                    small amounts," "problems with the employer," or
                    to get a "phone number"?
                    - PAUSE1000
User     14.1       get a phone number
Recognised (got phone number)
Concepts(phone/score 1000)
System   15.1       - Our phone number is: 48 20 49 10
System   15.2       - You may also say "address," "email," "fax,"
                    and "web,"
                    - PAUSE1000
Event    10         hangup
```

Figure 2. Dialog fragment, translated from Danish. System pauses are in milliseconds. After each user utterance, the system records what it recognized and to what degree using a recognition score of 0 to 1000. The score represents the strength of a match to certain concepts that the developer specifies.

Because the rules governing holiday allowance earning and distribution are complicated, many people need information about it, and the FAQ SDS must model all possible concerns. Questions to FerieKonto can relate to a specific person, such as, "How much money (or how many days) do I have in my account?" The questions might also be more general, such as, "Is Saturday considered a holiday?" or "Can I transfer a holiday allowance to next year?"

Although the FAQ SDS recognizes personal questions, it deals strictly with nonpersonal questions, referring other inquiries to an operator or a Web page. Figure 2 illustrates the kind of dialog a person can have with the system. Typically, callers fall into some combination of four categories:

- They have a problem and want an answer to their question now.
- They don't know exactly how to describe their problem.
- They lack information about the holiday allowance domain in particular.
- They've never interacted with an SDS before.

Dialog model design

Because the holiday-allowance domain is large and complex, we knew that the dialog model design would be a central issue in developing the FAQ SDS. Consequently, we devoted much effort to structuring the large information space and presenting it to the user, as well as to designing the user introduction, which had to convey both the system's capabilities and how users could get information.

Although design issues for standard SDSs are fairly well understood from both general and concrete perspectives,[3,4] the FAQ SDS is far from a standard commercial system. The most obvious difference is that it covers a large domain comprising many tasks, any one of which might be in focus at a given time. It is thus impossible to know in advance which part of the system knowledge a specific user will find relevant. These numerous tasks naturally give rise to many domain concepts.

As the "Choosing the Right Dialog Model" sidebar describes, application domain and user ability largely determine the selection of dialog model or combination of models. The application can have a range of tasks, concepts, and values, and that range is part of what determines the most appropriate model. The FAQ SDS, for example, has many tasks and many concepts, but each concept has few values, and the users are primarily novices.

For the FAQ SDS dialog model design, our options were to use a natural language model or a combination of the menu and natural language models. A menu model alone would have been infeasible because the FAQ SDS domain has too many concepts.

Using the natural language model,[5] which in essence invites people to speak their minds after the system asks, “How may I help you?” was tempting, but we decided against it because users have only a limited understanding of what an SDS can do. Moreover, user tests during development indicated that giving the user too much freedom to initiate dialog was not compatible with how a FAQ system should work.

We confirmed these results later when we analyzed dialogs from the production system. Most users expected the FAQ SDS to guide them through the dialog. A few users took the initiative, using their own formulations to try to obtain answers. Still others seemed to ignore that they were speaking to a computer and provided too many details of the events leading to their current situation. Many users probably expected to speak to a human, so when they found themselves connected to a computer, they either hung up or did not speak but listened to the system’s introduction, perhaps to extract some general information.

These results convinced us that too many users wouldn’t know what to do, and only a few would take the initiative to use natural language and be able to focus their queries at the right detail level.

We ultimately decided to use a combination of menus and natural language to target the largest user group—people who didn’t know how to describe the problem but needed an answer—and still accommodate the second group—people with direct questions about their holiday allowance. The menu points cover only the most frequently asked questions and serve as examples of what the system covers and how the user should ask questions.

```
Story: Amount, Error
- error in amount

r04_04      Amount, Error
- Please remember that tax and labor market payments have been
deducted from your holiday allowance before it is paid to us.
- If you think the amount is wrong, please contact your employer.
- Also contact your employer if a payment is totally missing.

r07_14      Payment, Error
- If you think you have received a wrong amount, please call again
and talk to a case officer.

Qo: Offer   - Do you want information about ”payment of small
            amounts,” ”problems with the employer,” or to get a
            ”phone number”?
Small=Amount+Small | Amount=Amount+Small |
Problems=Problems+Employer |
Employer=Problems+Employer
Repeat=Amount+Error | Silence=Q: Instruction | (global)
```

Figure 3. HTML rendering of the dialog model. The links refer to approximately correct states. Fully correct transitions depend on dynamic information and need an interpreter. Overall, however, the HTML approximation is much simpler to read and explain to domain experts. A list of specialized links precedes the set of globally defined links.

CUSTOMER COMMUNICATIONS

Like any system, an SDS requires software engineering. As such, communication with the customer and domain expert is an important concern. For SDS development, the amount of required communication can differ considerably, depending on the system’s size and complexity.

Previously, the target user group got information from a simple voice-response system, a Web site, or a case officer, whom they must call during office hours. Consequently, FerieKonto and its domain experts had been using diagrams as dialog models.

For a small system, diagrams are useful in showing prompts and dialog flow, and they provide an adequate overview for nondevelopers. For a large, complex system like the FAQ SDS, however, diagrams do not work as well.

First, the domain experts must easily understand the dialog structure and be able to provide feedback on the correctness and appropriateness of holiday-allowance rule formulations and combinations. To allow detailed feedback, the domain experts should be able to move around in the dialog model in a way that corresponds to what the implemented system would allow and to see all output, including rules. Second, the model presentation must be easy to update, since changes to the system must be consistent with what nondevelopers see.

We found that representing the dialog model as an HTML document, an excerpt of which is shown in Figure 3, satisfied both these requirements. The HTML links let domain experts follow the essential dialog flow; dialog content remains clearly visible; and with support from CDL, experts can easily update the HTML model.

CONCEPTUAL DIALOG LANGUAGE

A key concern was how to create efficient code that would capture the large, complex domain in which the FAQ SDS had to operate. The dialog model and the grammar would likely require several iterations, and the evolving specification had to be consistent with the envisioned implementation. These constraints were our primary motivation for creating CDL. Our aim was to tailor it to the FAQ SDS and to base it on general notions so that modelers of other SDSs could use it.

Figure 4 gives part of the dialog model in CDL with an XML syntax. CDL represents input as concepts and output as phrases concatenated into prompts. The dialog structure is partly static, modeled as collections of prompts or stories, and partly dynamic, expressed through conditions and links that control dialog flow. Using the XML syntax is purely a matter of convenience. CDL’s combina-

Choosing the Right Dialog Model

The most common dialog models for commercial spoken dialog systems (SDSs) are command and control, menu, form-filling, and natural language—often in combination. As Table A shows, the exact model or combination of model elements depends on application complexity and the user's expertise with a particular SDS. Most current commercial SDSs belong to one of the first four rows in the table. Systems in the last two rows are still fairly rare.

Some researchers describe application complexity primarily in terms of language, but we view it in terms of tasks, concepts, and values. Language becomes abstract using domain concepts (station, departure and arrival times, and seat availability) that are definable as elements of meaning that the system needs to perform one or more tasks.

Linguistically, concepts correspond to subgrammars. A single task, such as obtaining information about train travel, can have a few concepts, and those concepts can have a few values (Is the seat available or not?) or many values (station names, times).

User expertise in the particular SDS under development seems to matter quite a bit. An expert user can be familiar with the domain and at least one SDS type, but even that user might have a hard time with a different SDS type. The user of a command-and-control system, for example, might use her background in the application domain and in using a natural language SDS to guess some of the functionality and some of the command words. But without knowing how this particular SDS works, she cannot fully exploit it. Thus, as a rule, the more novice the user, the more directive the dialog model.

Command and control

In this model, accepted system input is a set of predefined words and phrases, and the user initiates the interaction. Thus users must be familiar with the sys-

Table A. How application complexity and user expertise drive dialog classification.

Application complexity			User's domain and SDS expertise		
Tasks	Concepts	Values	Novice	Expert	Systems with these characteristics
One	Few	Few	Menu	Menu	Collect calling
One	Few	Many	Form filling	Form filling +*	Train information; airline ticket reservation
Many	Few	Few	Menu	Command and control	Simple car control, operating room
Many	Few	Many	Menu, form filling	Command and control, form filling +	Car control with other features, such as route planning
Many	Many	Few	Menu, natural language	Menu +, natural language	Simple FAQ
Many	Many	Many	Menu, form filling, natural language	Menu +, form filling, natural language	Complex FAQ, with accounts and names, for example

*A "+" indicates the use of speak ahead, in which users can provide the needed information in the order they choose, rather than waiting for the system to ask.

tion of abstraction and customizable compilations is what makes it work, not the use of XML.

Concepts as an input abstraction

The immediate description of user input is a grammar. Grammars are concrete and large—complexity we handle using concepts. In the FAQ SDS implementation, a *concept* is a semantic abstraction that HDDL maps to a concrete input grammar fragment.

Modelers can typically indicate to the speech recognizer what input they expect at a given point in the dialog by defining certain concepts to be active at that point. Having a small set of active concepts increases recognition quality, but only if the user says something within the set. If the user says something outside the set, the recognizer will not recognize it or will mistakenly view it as an active concept. In the FAQ SDS, we decided that the user will probably want to ask for everything at any time, so we made all concepts active all the time.

The FAQ SDS comprises 75 domain-related concepts, six user-provided metaconcepts—yes, no, help, repeat, bye, and other options—and four system-detected metaconcepts—barge-in, silence, toolong input, and not-understood input.

Structure from prompts and stories

After analyzing the domain and reviewing early drafts of the dialog structure, we identified four overall prompt types:

- *rule:* formal regulation governing holiday allowance;
- *information:* more details about holiday allowance or dialog continuation;

tem before they can really control it. The more command words the system includes, the more difficult it is to remember them all. Sample applications are voice control of operating rooms, such as Stryker's Endosuite, and in-car systems, such as those in some automobiles, in which drivers can control the CD player, radio, navigation system, and so on with voice commands.

Menu

In menu-driven systems, the user chooses from a list of items. This model is well suited for users who don't know the task and aren't familiar with the system. The system dictates to the user exactly what he can say, such as "Please say one of the following: pay bills, account balance, or transfer funds." Some systems offer skilled users the option of interrupting menu prompts (barge-in).

Menus appear in interactive voice-response or touch-tone systems and in their SDS equivalents. In these systems, dialog is always based on menus. Designers might also opt to use menus with other models, say, to let a user select a task from a voice portal. The selected task might then follow a form-filling structure.

Form-filling

In this model, the system has slots that the user must fill. A primitive form-filling model has only one slot, requesting a yes or no answer or a waybill or account number. More complex versions can require several pieces of information, such as the departure and arrival station and the arrival and departure times and dates for train travel.

This type of system typically guides the dialog, and the user needs no system or task knowledge. If the system dialog is poorly crafted, however, the user might not know exactly what to say and still have the system understand.

To accommodate users familiar with the task, SDSs using the form-filling approach often provide a speak-ahead option, where users can provide the needed information in the order they choose. In the train-travel example, a user could provide the departure and arrival stations, dates, and times in the same turn even if the system asked only for the departure and arrival stations.

Natural language interfaces

Interfaces that accept free natural language do not exist, but many systems allow limited natural language input, which the designer specifies. This kind of input fits well with the form-filling model because free input formulation can be a natural user choice. However, it is not a good fit with command-and-control systems, which are more restrictive.

Researchers are investigating conversational dialog that is not task oriented, but their efforts have yet to solve the problem of how to tell the user what the system can and cannot understand. The vast majority of changes to fine-tune a deployed system that allows some kind of natural language involve expanding the grammar. In the train example, the system must understand many formulations: "When is the next connection between Munich and Frankfurt today?" and "I need to be in Frankfurt at five pm tomorrow."

When the application domain has too many concepts, neither the menu nor the form-filling model is viable. Using full natural language queries would be optimal if SDS developers can overcome the limited subset of natural language that current systems are restricted to. This limitation might be acceptable for experts, but novice system users will always need help in determining what they can say and do.

- *question:* clarification of user issues; and
- *list:* options that the user can choose from.

These prompts are in essence reusable core patterns. Representing these patterns as a built-in syntax—basic building blocks—ensures that modelers can easily express them as needed and that the user sees a consistent structure.

We supplemented core patterns with patterns for specific situation types. A fixed repair pattern addresses any miscommunication: The SDS offers first instruction, then help, then the option of speaking with a case officer, and finally, if all else fails, politely closes the dialog. Another standard pattern is that after it gives the information, the system offers to provide more details related to that topic.

To allow for larger patterns and variations over prompt combinations, we introduced the notion of stories. A *story* combines several prompts into a larger structure. Figure 3, for example, is a partial story about errors in the amount paid. Stories are based on both rules and meta-issues, such as the user asking for help or other options or the system detecting silence or overly long input.

To compose stories, we used these informal design rules:

- Each single domain concept must have an associated story.
- If a concept concerns only one or two rules, the system provides these rules to the user immediately.
- If a concept concerns more than two rules, the system provides the most general rule and simultaneously indicates that more information is available on that topic.

Figure 4. Fragment of the dialog model in CDL, translated from Danish. Coding CDL in XML was purely a matter of convenience. The power of CDL lies in its ability to be at once application-specific and extendable.

```
<story topics="amount error" implicit="error in amount">
        <ruleRef rule="r04_04"/>
        <ruleRef rule="r07_14"/>
        <prompt type="Q" subtype="o">
                <phrase text="Do you want information about 'payment of small amounts,'
'problems with the employer,' or to get a 'phone number'?"/>
                <links type="meta">
                   <link ref="@story" cond="(got again)"/>
                   <link ref="QhI" cond="silence"/>
                </links>
                <links type="topic">
                   <link ref="amount_small" cond="(got small)"/>
                   <link ref="amount_small" cond="(got amount)"/>
                   <link ref="problems_employer" cond="(got problems)"/>
                   <link ref="problems_employer" cond="(got employer)"/>
                </links>
        </prompt>
        […]
</story>
[…]
<rule id="r07_14" topics="payment error">
        <phrase text="If you think you have received a wrong amount, please call again and talk
to a case officer."/>
</rule>
```

- If no rule is the most general, the system asks the user for more information.
- If related rules exist, the system offers them to the user after providing the information about the current story.

Flow via conditions and links

CDL captures the flow of prompts both within and between stories through links and uses conditions to control the flow of prompts. Every prompt has a list of links to other assigned prompts. We opted to have conditions on both links and stories and prompts to have greater flexibility in modeling style.

Link conditions are local in the sense that the system knows which prompt the dialog is in and what input it just received. Conditions on links enable a state-machine or flow-chart style—a traditional way to describe simple dialogs, such as those in voice-response systems. Link conditions answer the question, "Where should I go from here?"

Prompt and story conditions are global in that they are independent of local context. Conditions on stories and prompts enable a rule-based style more akin to AI systems. Prompt conditions answer the question, "Now that I'm here, should I enter?" The "talk to operator" story, for example, has two variants. If the operator is available, the system will tell the user that she is now being redirected to an operator. If an operator is not available, the system will tell the user when FerieKonto is open and explain how to consult a Web site for information.

Additional design support

In addition to typical dialog modeling and structuring tasks, CDL lets the modeler experiment with ideas that the programming language (HDDL in our case) might not support directly—ideas that could lead to a more varied and fluent dialog. For the FAQ SDS, we experimented with several ideas.

One is *reductive simplification*. Some stories have a partial overlap that is unavoidable because the same piece of information might be relevant in quite different contexts. To avoid redundancy, the system tracks which stories the user has heard. If the user has asked about payment of holiday allowance, the system will tell how this happens, but it will tell when the money will be paid only if the user did not already get this information from another story.

Another idea we experimented with was *barge-in control*. Barge-in is a technique that lets users interrupt system prompts. The system then stops talking and reacts to user input. Although barge-in is standard in SpeechMania, the platform does not include the ability to inform the system's dialog manager that a user has interrupted prompts. Consequently, the system never knows how much output a user actually heard before the interruption. Because part of our dialog design builds on knowledge about what information the user has already received, the FAQ SDS lets designers set barge-in as an event condition if desired.

The ability to *randomly switch among variations of frequent phrases*, such as "Are there other topics you want to ask about?" helps make the dialog less monotonous. Some variations that the FAQ SDS randomly switched were "Is there any other information you want?" and "Is there anything else you would like to know?" We generally have three variants of a frequent phrase.

The last idea we experimented with is *confidence control*. The speech recognizer provides a confidence measure between 0 and 1,000, which the developer can divide into three score ranges: low, medium, and high. We used 0-100, 101-200, and

```
COND ((V_currentName == MSN_amount_error__Qo)) {
       MO_comment("#COND ((V_currentName == MSN_amount_error__Qo))");
       M_setStory(amount_error);
       M_statPoint
       MC_preCond(MSN_amount_error__QhI, MSN_help__QhH, MSN_bargeIn__Qd);
       MC_preCond_Qo
       QUESTION (MH_allConcepts) {
             INIT {
                    MC_utterance_Qo
                    MP("-Do you want information about 'payment of small amounts,''problems
                    with the employer,' or to get a 'phone number'?");
                    MP_bargeIn_Qo
             }
             MC_events(MH_allConcepts);
       }
       MC_heard(V_amount_error__QoState);
       MC_postQuestion(3157, V_amount_error__QoState);
       MC_nextState(V_menu__more__QdState);
             IF ((^smallConcept && smallConcept.positive)) {
                    MO_comment("#(^smallConcept && smallConcept.positive)");
                    V_topicFound := TRUE;
                    MC_setState(V_amount_small__r07_03State);
                    V_catchLevel := V_catchLevel + 1;
                    MS_used(small, smallConcept);
                    BREAK;
             }
 […]
}
```

Figure 5. Machine-generated code for part of the fragment in Figure 4. The code, which constitutes the runtime version of the dialog model, is in the HDDL programming language. Macros (conventionally named with an initial M, MC, or MO) implemented much of the FAQ semantics. State names start with MSN. Variable names start with V.

201-1,000. The score value for any actual input depends on how well the acoustic input signal from the user matches the acoustic models of the concepts in the system's vocabulary.

A bad match can occur for several reasons, such as heavy background noise, a foreign accent or strong dialect, and out-of-vocabulary words. A low score causes the system to ask the user for confirmation of what she said: "Did you say maternity leave?" A medium score causes the system to repeat the key concept(s) the user mentioned before providing an explanation: "*You asked about maternity leave*. You cannot get your holiday allowance while you are on leave, but ...." The user then becomes aware of any misunderstanding as early as possible and can make a correction. A high score causes the system to give the user implicit feedback by mentioning in the information it outputs the key concept(s) it is responding to: "You cannot get your holiday allowance while you are on *leave*, but...."

IMPLEMENTED DIALOG MODEL

The final FAQ SDS dialog model comprises 85 concepts and 233 prompts, including 66 holiday allowance rules, structured into 102 stories. The dialog model is compiled either into human-readable form, the HTML in Figure 3, or into executable code, as in Figure 5. Although our executable code was in HDDL, it could also have been in VoiceXML, an upcoming standard for spoken dialog.[6] The model is about 2,400 lines of CDL (XML), which translates to nearly 50 pages of HTML description and 12,000 lines of HDDL code (compact, full of macro calls). The model also has 2,700 lines of grammar.

We could not have dealt with the SDS complexity or communicated the model to the domain experts without the CDL abstractions. So far, FerieKonto has not formally investigated users' satisfaction with the system and will find it difficult to do so, since the company is not allowed to log which users call the system. According to the FerieKonto case officer responsible for the system, which receives roughly 12,000 calls per year, they have received no negative user comments in the year and a half the system has been in use. The case officer also stated that the HTML version of the dialog model was easy for them to follow. Making changes and additions only in the CDL model guaranteed consistency between the HTML and HDDL versions.

The ideas of a customized language and core patterns apply well to SDS design. Widespread use of CDL would require a dedicated editor and integrated development environment, but for limited use, the solution we have described seems quite sufficient. ■

Acknowledgments

This project was supported by The National Agency for Enterprise and Housing and the Danish Ministry of Economic and Business Affairs. We thank the staff at FerieKonto for kind, competent, and enthusiastic cooperation.

References

1. H. Dybkjær and L. Dybkjær, "Experiences from a Danish Spoken Dialogue System," *Proc. 2nd Danish*

HCI Research Symp., DIKU tech. report 02/19, Univ. of Copenhagen, 2002, pp. 15-18.
2. H. Aust et al., "The Philips Automatic Train Timetable Information System," *Speech Comm.*, vol. 17, 1995, pp. 249-262.
3. N.O. Bernsen, H. Dybkjær, and L. Dybkjær, "What Should Your Speech System Say?" *Computer*, Dec. 1997, pp. 25-31; http://www.disc2.dk/tools/codial/index.html.
4. B. Balentine and D.P. Morgan, *How to Build a Speech Recognition System. A Style Guide for Telephony Dialogues*, 2nd ed., EIG Press, 2001.
5. A.L. Gorin et al., "Automated Natural Spoken Dialog," *Computer*, Apr. 2002, pp. 51-56.
6. N. Leavitt, "Two Technologies Vie for Recognition in Speech Market," *Computer*, June 2003, pp. 13-16.

***Hans Dybkjær** is a software developer at Prolog Development Center in Denmark, where he heads the speech group and works on interface specifications and the improvement of test procedures. He also led the development of the FAQ SDS described in this article. He received a PhD in computer science from the University of Copenhagen. Contact him at dybkjaer@pdc.dk.*

***Laila Dybkjær** is a professor at the University of Southern Denmark's Natural Interactive Systems Laboratory (NISLab), where her interests include development and evaluation of interactive speech systems, usability aspects in interface design, and dialog modeling and theory. She received a PhD in computer science from the University of Copenhagen. Contact her at laila@nis.sdu.dk.*

XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX

XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX