Natural Language Engineering **14** (3): 313–335. © 2007 Cambridge University Press
doi:10.1017/S1351324907004561 Printed in the United Kingdom

Bootstrapping spoken dialogue systems by exploiting reusable libraries

GIUSEPPE DI FABBRIZIO, GOKHAN TUR, DILEK HAKKANI-TÜR, MAZIN GILBERT and BERNARD RENGER

AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ 07932, USA
e-mail: {pino, gtur, dtur, mazin, renger}@research.att.com

DAVID GIBBON, ZHU LIU and BEHZAD SHAHRARAY

AT&T Labs—Research, 200 Laurel Avenue South, Middletown, NJ 07748, USA
e-mail: {dcg, zliu, behzad}@research.att.com

(*Received 22 December 2004; revised 24 January 2006; first published online 11 May 2007*)

Abstract

Building natural language spoken dialogue systems requires large amounts of human transcribed and labeled speech utterances to reach useful operational service performances. Furthermore, the design of such complex systems consists of several manual steps. The User Experience (UE) expert analyzes and defines by hand the system core functionalities: the system semantic scope (call-types) and the dialogue manager strategy that will drive the human–machine interaction. This approach is extensive and error-prone since it involves several nontrivial design decisions that can be evaluated only after the actual system deployment. Moreover, scalability is compromised by time, costs, and the high level of UE know-how needed to reach a consistent design. We propose a novel approach for bootstrapping spoken dialogue systems based on the reuse of existing transcribed and labeled data, common reusable dialogue templates, generic language and understanding models, and a consistent design process. We demonstrate that our approach reduces design and development time while providing an effective system without any application-specific data.

1 Introduction

Spoken dialogue systems (SDS) aim to identify intents of humans, expressed in natural language, and take actions accordingly to satisfy their requests. In a natural spoken dialogue system, typically, first the speaker's utterance is recognized using an automatic speech recognizer (ASR). Then, the intent of the speaker is identified from the recognized sequence, using a spoken language understanding (SLU) component. This step can be framed as an utterance classification problem for goal-oriented call routing systems (Gorin *et al.* 1997; Natarajan *et al.* 2002; Gupta *et al.* 2006). Then, the user would be engaged in a dialogue via clarification or confirmation prompts,

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

- **System:** How may I help you?
- **User:** Hello?
- *Call-type: Hello*
- **System:** Hello, how may I help you?
- **User:** I have a question.
- *Call-type: Ask(Info)*
- **System:** OK, What is your question?
- **User:** I would like to know my account balance.
- *Call-type: Request(Account_Balance)*
- **System:** I can help you with that. What is your account number?
- **User:** ...

Fig. 1. Natural language dialogue example.

if necessary. The role of the dialogue manager (DM) is to interact in a natural way and help the user achieve the task that the system is designed to provide.

In our case, we consider only automated call routing systems where the task is to reach the right route in a large call center, which could be either a live operator or an automated system. Note that we consider only systems where the ASR and SLU models are trained using transcribed and labeled data in contrast to simple interactive voice response (IVR) systems where a manually written grammar is used for both components. The greeting prompt is *How may I help you?*, which encourages the users to express their intents in natural language instead of saying one of the key phrases that the manually written grammar supports. An example dialogue from a telephone-based customer care application is shown in Figure 1.

Typically the design of such complex systems consists of several manual steps, including analysis of existing IVR systems, customer service representative (CSR) interviews, customers' testimonials, CSR training material, and, when available, human–machine unconstrained speech recordings. On the basis of these heterogeneous requirements, the user experience (UE) expert analyzes and defines by hand the system core functionalities: the system semantic scope (*call-types*) and the dialogue manager strategy that will drive the human–machine interaction.

The traditional and proposed application building procedures are given in Figure 2. In the original scheme (Figure 2a), once the data from the application domain is collected and the requirement analysis completed, the UE expert put together a *labeling guide* document. This includes at least the following parts: (a) detailed description of the application domain call-types, (b) clear rules and examples about how to categorize utterances into call-types, and (c) specific guidelines to properly identify and tag dysfluencies. The labeling guide is used by the labelers to consistently annotate the collected utterances. The UE expert next task consists in the human–machine interaction design that is based on the initial requirement investigation, the assessment of recorded human-human recording and various other customer/user business considerations. The outcome of this phase consists of a detailed and formal definition of the human–machine interaction (*call flow*), which includes the definition of the main dialogue strategy, disambiguation and dialogue recovery approaches, and effective system prompts scripting.

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

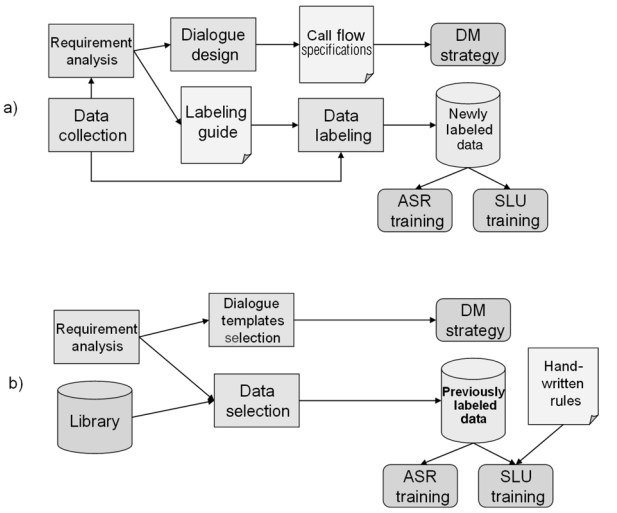


Fig. 2. The traditional (a) and proposed (b) application building procedures.

In a production environment, these steps are rather expensive and labor intensive. A typical data-collection session could take from 4 to 8 weeks with an average of 200–400 calls per day, depending on the amount of the real traffic the customer is willing to route to the automated collection system during the business hours. The labeling guide is typically written during the data-collection phase and requires several iterative revisions that could take from 2 to 4 weeks before reaching a stable version. The training of the labelers on the specific application domain starts only when the labeling guide is available. Then, the actual transcription and labeling begins which could last as long as 6 weeks.[1]

In the new procedure (Figure 2b), the UE expert has already available a large collection of previously transcribed and labeled data that are previously hierarchically organized by industry sector and dialogue acts labels. The UE expert can immediately star by *selecting data* from the library, excluding the labeling guide preparation and data-labeling phases all together. More specific application domain call-types can be initially covered by manually adding rules to the spoken language understanding module (typically CFG grammars or simple regular expressions) and manually augment the ASR language module with representative exemplars of the missing data. Dialogue interactions are also standardized (at least for typical call routing applications) in a subdialogue library that can be simply customized with the application-specific prompts and parameters. The newly defined process results in quicker deployment of new bootstrap applications by data reuse. Best practice within this framework shows that the traditional process defined above can be shortened to 2–3 weeks for a fully deployed *bootstrap* pilot system. This first approximation of the final system can be then used to collect application domain data on the field

[1] The estimates are based on average values for a medium complexity application.

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

with a system that is much closer to the desired application call router and thus more likely to capture a more natural human–machine interaction.

The organization of this article is as follows. Section 2 is a brief overview of previous work relevant to this article. Section 3 briefly describes the AT&T VoiceTone® Spoken Dialogue System, which we use in this study, and its main components, ASR, SLU, and DM. Section 4 provides detailed information about reusable library components. In Section 5, we present our method to bootstrap ASR, SLU, and DM for a new application. Finally, Section 6 presents our experiments using real data from a customer care application.

2 Previous work

There have been extensive contributions in the literature about toolkits for developing spoken dialogue systems and for streamlining the application building process (see McTear 2002 for an excellent survey of the current speech dialogue technology trends).

For instance, the CSLU Toolkit (Sutton and Cole 1998) from the Center for Spoken Language Understanding at the Oregon Graduated Institute of Science and Technology provides a graphical authoring tool (RAD) for designing and implementing spoken dialogue systems requiring limited knowledge about the underlying technology details. The modularity of the design encourages encapsulation and reuse through RAD objects and subdialogues libraries. The *RAD Object Tab* palette includes predefined objects such as numeric and alphanumeric input, keyword word spotting, elicit information requests, and generic dialogue repairs. Objects can be linked together with arcs and the interaction simulated on a desktop using the local microphone and speaker. In the CSLU toolkit, however, the speaker input is described by carefully handwritten rule-based grammars rather than stochastic models trained on a large amount of transcribed speaker utterances and it does not address issues related to cross-applications data organization, retrieval, and reuse.

The MATE Workbench framework (Dybkjr and Bernsen 2000), instead, offers generic annotation schemas and database capabilities to store and retrieve corpus data for spoken application purposes. This includes prosody, coreference annotation as well as dialogue acts annotation schemas and an easy-to-use interface. MATE is closer to the proposed data library tools, but it is missing the capability of organizing, generating, and evaluating ASR and SLU models and named entity (NE) grammars based on data extracted from the library.

In general, these toolkits and application frameworks offer a wide range of reusability options and easy-to-access language resources that help reduce the burden of dialogue systems development life cycle. They often provide organized speech files repository, several levels of annotations and transcriptions as well as general purpose grammars (e.g., date, time, digits, currency, credit cards, etc.), and reusable subdialogue libraries, but none of them to date have been focusing on large utterance collections semantically annotated by dialogue acts and domains (e.g., industry sectors).

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

In our previous work, we have presented active and unsupervised (or semisupervised) learning algorithms to reduce the labeling effort needed while building ASR and SLU systems (Riccardi and Hakkani-Tür 2003; Tur *et al.* 2005). There, the focus was on a single application, and only on the ASR and SLU components. In this study, we aim to exploit the labeled and transcribed data and common reusable dialogue templates obtained from similar previous applications to bootstrap the entire spoken dialogue system including ASR, SLU, and DM components, without extensive data collection, analysis, and labeling. Once the bootstrap system is deployed, application data can be collected and labeled to improve the system performance.

3 AT&T **VoiceTone® Spoken Dialogue System**

The AT&T VoiceTone® Spoken Dialogue System is part of the retail branded services offered to the AT&T's business customers. It provides network grade scalable services with large vocabulary speech recognition, natural language understanding, and dialogue automation technologies.

At a very high level of abstraction, the AT&T SDS is a phone-based VoiceXML product with specific extensions to address natural language needs. Typically, once a phone call is received, the dialogue manager prompts the caller with either a prerecorded or synthesized greeting prompt. At the same time, it activates the broad ASR language model trained with application-specific data. The caller speech is then translated into text and sent to the SLU, which replies with a semantic representation of the utterance. Based on the SLU reply and the implemented dialogue strategy, the DM engages in a mixed initiative dialogue to drive the user toward the goal. The DM iterates the previously described steps gathering information about the user's intent until the call reaches a final state (e.g., the call is transferred to a specialized CSR, or IVR or the caller hangs up).

In the literature, to collect some application-specific data to determine the application-specific call-types and train ASR and SLU models, first a "Wizard-of-Oz" data collection is performed (Gorin *et al.* 1997). In this approach, a human, i.e., wizard, acts like the system, although the users of the system are not aware of the human on the other side. This method turned out to be better than recording user-agent (human–human) dialogues, since the responses to machine prompts are found to be significantly different from responses to humans, in terms of language characteristics.

3.1 Automatic speech recognition

Robust speech recognition is a critical component of a spoken dialogue system. In this work, we use AT&T's Watson Speech Recognition Engine (Goffin *et al.* 2005), which supports any-vocabulary size task, and adopts continuous-density hidden Markov models for acoustic modeling and finite-state transducers for network optimization and search. In most state-of-the-art large vocabulary ASR engines, the two major components are acoustic and language models. In this work, we use trigram language models based on Variable N-gram Stochastic Automaton (Riccardi

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

et al. 1996). The acoustic models are subword unit based, with triphone context modeling and variable number of Gaussians (4-24). The output of the ASR engine (which can be the 1-best possibly with word confidence scores or a lattice) is then used as the input to the SLU component. The details of the recognizer architecture can be found in Goffin *et al.* (2005).

3.2 Spoken language understanding

In a natural language spoken dialogue system, the definition of "understanding" depends on the application. In this work, we focus only on goal-oriented call classification tasks, where the aim is to classify the intent of the user into one of the predefined call-types. As a call classification example, consider the utterance in the previous example dialogue, *I would like to know my account balance*, in a customer care application. Assuming that the utterance is recognized correctly, the corresponding intent or the call-type would be *Request(Account_Balance)* and the system would prompt for the account number and then indicate the balance to the user or route this call to the Billing Department.

Classification can be achieved by either a knowledge-based approach that depends heavily on an expert writing manual rules or a data-driven approach that *trains* a classification model to be used during run time. In our current system, we consider both approaches. Data-driven classification has long been studied in the machine learning community. Typically these classification algorithms try to train a classification model using the features from the training data. More formally, each object in the training data, $x_i \in \mathscr{X}_j$, is represented in the form (F_{ij}, C_{ij}), where $F_{ij} \subset \mathscr{F}_j$ is the feature set and the $C_{ij} \subset \mathscr{C}_j$ is the assigned set of classes for that object for the application j. Note that for the proposed approach in this article, the type of the classifier is irrelevant, as long as it can be trained with labeled data and augmented with rules. In this study, we have used an extended version of a Boosting-style classification algorithm for call classification (Schapire 2001) so that it is now possible to develop handwritten rules to cover less frequent classes or bias the classifier decision for some of the classes. This is explained in detail in Schapire *et al.* (2002). In our previous work, we have used rules to bootstrap the SLU models for new applications when no training data are available (Di Fabbrizio *et al.* 2002).

Classification is employed for all utterances in all dialogues as seen in the sample dialogue in Figure 1. Thus all the expressions the users can utter are classified into predefined call-types before starting an application. Even the utterances that do not contain any specific information content have special call-types (e.g., *Hello*). So, in our case, objects $\mathscr{X}_j$ are utterances and classes, $\mathscr{C}_j$, are call-types for a given application j.

3.3 Dialogue manager

In a mixed-initiative spoken dialogue system, the dialogue manager is the key component responsible for the human–machine interaction. The DM keeps track of the specific discourse context and provides disambiguation and clarification strategies

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

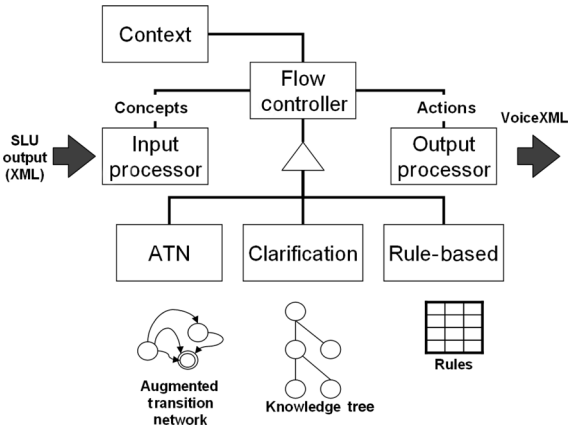


Fig. 3. Dialogue manager architecture.

when the SLU call-types are ambiguous or have associated low confidence scores. It also extracts other information from the SLU response to complete the information necessary to provide an automated speech service.

Previous work in AT&T on dialogue management (Abella and Gorin 1999) that applies to call routing shows how an object inheritance hierarchy is a convenient way of representing the task knowledge and the relationships among the objects. A formally defined *construct algebra* describes the set of operations necessary to execute actions (e.g., replies to the user or *motivators*). Each dialogue motivator consists of a small processing unit that can be combined with each other following the object hierarchy rules to build the application. Although this approach demonstrated effective results in different domains (Gorin *et al.* 1997; Buntschuh *et al.* 1998), it proposes a model that substantially differs from the *call flow* model broadly used by UE experts to specify the human–machine interaction.

Building and maintaining large-scale voice-enabled applications require a more direct mapping between specifications and programming model, together with authoring tools that simplify the time-consuming implementation, debugging, and testing phases. Moreover, the DM requires broad protocols and standard interface support to interact with modern enterprise backend systems (e.g., databases, HTTP servers, e-mail servers, etc.). Alternatively, VoiceXML (VoiceXML 2003) provides the basic infrastructure to build spoken dialogue systems, but the lack of SLU support and offline tools compromises its use in data-driven classification applications.

Our approach proposes a general and scalable framework for Spoken Dialogue Systems. Figure 3 depicts the logical DM architecture. The *Flow Controller* implements an abstraction of pluggable dialogue strategy modules. Different algorithms can be implemented and made available to the DM engine. Our DM provides three basic algorithms: *ATNs* (Augmented Transition Networks) (Bobrow and Fraser 1969), *clarification* based on knowledge trees (Lewis and Di Fabbrizio 2005), and *plan-based* dialogue manager drived by a rules engine. Traditional call routing systems are better described in terms of ATNs. ATNs are attractive mechanisms for dialogue specification since they are (a) an almost direct translation of call

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

flow specifications (Kotelly 2003), (b) easy to augment with specific mixed-initiative interactions, (c) practical to manage extensive dialogue context. Complex knowledge-based tasks could be synthetically described by a variation of knowledge trees. Plan-based dialogues are effectively defined by rules and constraints.

The Flow Controller provides a synthetic XML-based language interpreter to author the appropriate dialogue strategy. Dialogue strategy algorithms are encapsulated using object-oriented paradigms. This allows dialogue authors to write subdialogues with different algorithms, depending on the nature of the task and to use them interchangeably sharing the current dialogue state information through the local and global contexts. A complete description of the DM is out of the scope of this publication and it is partially covered in Di Fabbrizio and Lewis (2004). We will focus our attention on the ATN module, which is the one used in our experiments. The ATN engine operates on the semantic representation provided by the SLU and the current dialogue context to control the interaction flow.

4 Natural language reusable data components

This section describes reusable library components for boostrapping spoken dialogue systems. The library components include SLU models, ASR models, Named Entity (NE) grammars, user's utterance transcriptions, ASR transcriptions (when the data collection involved a human–machine interaction), call-type labels, audio data, dialogue-level templates, prompts, and other reusable data. Thus, we define the library as the collection of all data collected from the existing applications. The effort involved in maintaining libraries is justified by their many benefits. Defining an extensible taxonomy of call-type categories, for example, promotes uniformity and reduces the time and effort required when a new set of data is encountered. Libraries add organization that helps document the application.

In typical natural language applications, the data for new applications are collected and transcribed. A user experience (UE) expert starts a new application by evaluating an initial set of transcribed utterances and determining relevant labels or call-types for these utterances. The UE expert also selects positive (label applies) and negative (label does not apply) guideline utterances for each label (or call-type). These guideline utterances and descriptions of the labels are included in an annotation guide. The annotation guide is also organized by category areas where call-types with the same category are grouped together (e.g., "Billing Queries" might be one of the categories). A set of labelers uses the annotation guide to label additional transcribed utterances. These labeled transcribed utterances can be used to bootstrap future applications as described below.

4.1 Data organization

The data can be organized in various ways. For instance, the data can be organized by the industry sector. So if there is a new application in the Healthcare sector, the library components from this sector could be used to bootstrap the new application. Alternatively, the data can be organized by category (e.g., Service Queries, Billing

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

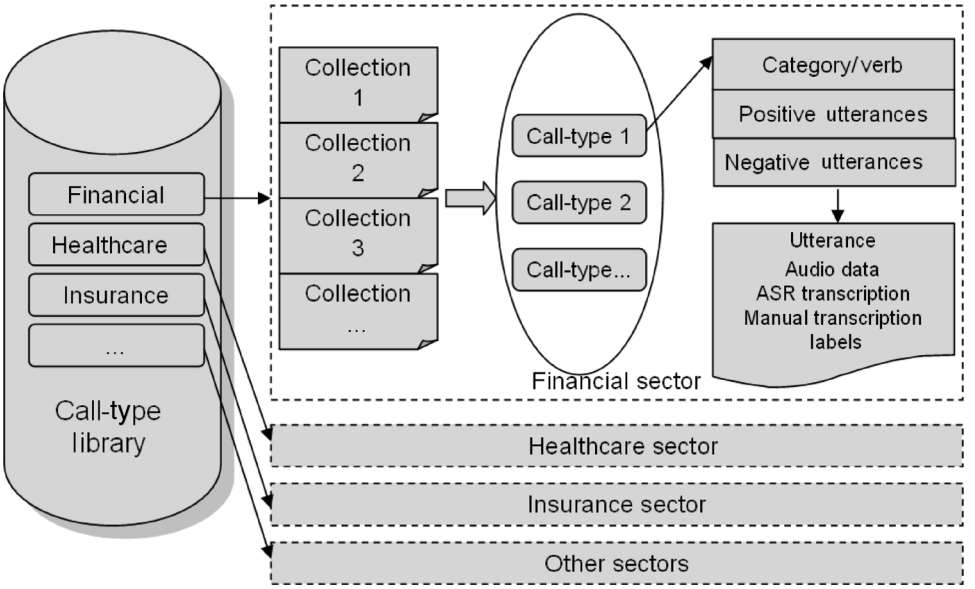


Fig. 4. Library architecture.

Queries, etc.) or according to call-types of individual utterances, or words in these utterances. Any given utterance may belong to more than one call-type. Call-types are given mnemonic names and textual descriptions to help define their semantic scope. Call-types can be assigned attributes that can be used to assist in library management, browsing, and to discipline the call-type design process. Attributes can indicate whether the call-type is generic, reusable, or is specific to a given application. Call-types could include the category attribute or at a lower level can be characterized by a "verb" attribute such as "Request, Report, Ask, etc." (Gupta *et al.* 2006). The notion of industry sector is not well defined; there are taxonomies for describing hierarchies of industry sectors (NAICS 2002), and a given call-type may not exclusively belong to a single sector. The UE expert will need to make a judgment call on how to organize the various application data sets into industry sectors.

4.2 Creating reusable library components

Figure 4 shows the architecture of the library. The elementary unit of the library is the utterance, and the key information stored with each utterance is its original audio recording, the ASR and manual transcriptions, and its call-type labels. The audio data and corresponding transcription can be used to train the ASR acoustic and language models and the transcriptions with the call-type labels are used for building new SLU models. Each call-type contains a set of utterances for training the call-type classifiers.

The labeled and transcribed data for each data collection are imported into separate data-collection XML databases. Our tools maintain the data for each collection in a separate file structure. For browsing the application data individually,

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

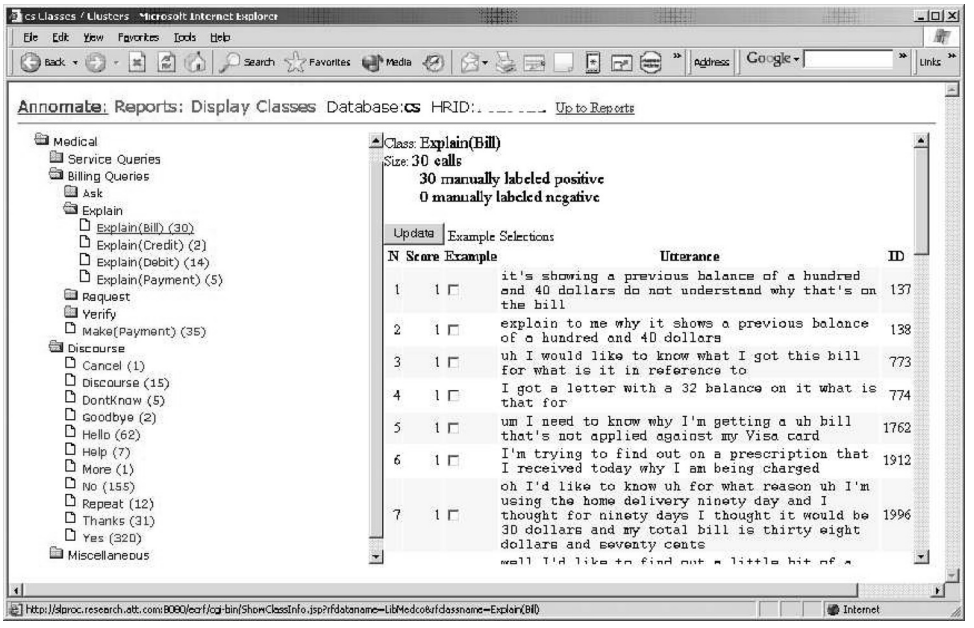


Fig. 5. Tool for extracting utterances from call-type libraries.

it is convenient to represent the hierarchical structure as a tree where each node stores several item attributes such as category, verb, call-type, and utterance transcription. A separate sector XML database stores the sector attribute for each data collection. This XML file also keeps track of the number of utterances imported for each application as well as the data-collection time stamps (when the utterances were captured from a live data-collection system). The call-type library hierarchy can be generated from the individual XML data-collection databases and the sector XML database. The call-type library hierarchy is organized using different views based upon industrial sector, data collection type, a general category description, verbs, call-types, and utterance transcription text. This provides a flexible way to browse a large amount of data that scales well for medium/large-size library. Currently it stores more than half million utterances, organized in five different sectors, twenty-eight data sets and around a thousand different call-types. However, users may be interested in retrieving call-types with more sophisticated query mechanisms that suggests that the library should be maintained in a relational database. By employing an XML approach, we can leverage the widely available tools that support XPath queries and XSLT to render interactive user interfaces with standard Web browser clients.

Once the call-type library tree is generated, the UE expert can browse and export any subset of the data as shown in Figure 5. Our tools allow the user to select utterances for a particular call-type in a particular data collection or the user can extract all the utterances from any of the data collections. The UE expert or the application developer can select utterances using various criteria and build ASR and/or SLU models. For example, one might select all the utterances from all the data collections in a particular sector. These data could be extracted and used to

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

generate SLU and/or ASR models for that sector. As new data are imported for new data collections of a given sector, better models can be built for each sector. In this way, the models for a sector are iteratively improved as more applications are deployed. The UE expert plays a large role in building the libraries since the UE expert will need to make careful decisions based on knowledge of the business application when selecting which utterances/call-types to extract.

In the next section, we will describe how the call-type library is used to bootstrap new applications in the same sector.

4.3 Using reusable library components

Historically, data collection consisted of recording human–human interactions or blindly recording initial customer requests before routing the call back to the default customer care toll-free number. An improvement on this involves building a small application that understands some generic requests since it is possible to build simple generic ASR and SLU models that apply across all sectors (generic call-types) (Di Fabbrizio *et al.* 2004). With reusable library components for each sector, we can use better ASR and SLU models that will cause more calls to be properly routed and ultimately give the callers more user satisfaction.

As described above, the sector ASR and SLU models are built using transcribed and labeled data from previous data collections. We can then use these models during the wizard data-collection phase for a new application (e.g., for a customer care application for a different company) in the same sector. Here, the real-time ASR transcription is used by the SLU classifier to determine the caller intent. The UE expert can also write handwritten rules to cover infrequent call-types or call-types not found in the existing SLU model.

In the next section, we present how we bootstrap spoken dialogue systems, using reusable library components.

4.4 Maintaining reusable library components

Our tools also allow the UE expert to modify the call-type library data. There are various reasons for needing this capability.

Call-type labels for the same caller intent may have different names across different data collections/applications. For instance, in some older VoiceTone® applications the call-type *Tellme(Balance)* captures the caller's request for account balance information. Some time later, the same caller intent was labeled *Request(Balance)*. Our tool allows the UE expert to easily map the old label to the new label so that utterances from both data collections can be combined to build a more effective sector SLU model.

Call-type labels do not always abide by the standard call-type label definitions. Our tools are used to create the call-type labels so the annotation guide generation is more error resistant. Since the annotation guide is used by the labelers to label the data (assign call-type labels to the utterances), the call-type labels are more consistent, which in turn makes the labeling process easier.

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

- Request(Call_Transfer_Generic)
- Request(Call_Transfer_Sales)
- Request(Call_Transfer_Spanish)
- Request(Call_Transfer_Marketing)

Fig. 6. Multiple call-types.

Call-types can be more granular for some data collections/applications. For some applications, there might be one call-type *Request(Call_Transfer)* that deals with call transfers to a customer representative. For other applications, the caller intent may be broken down into multiple call-types as shown in Figure 6.

These call-types can be collapsed back down to *Request(Call_Transfer)* using our tool although it is not clear that this would be the correct way to handle granularity in this case. The UE expert would need to understand what was important in the business application and will need to factor in the call flow for routing calls. For example, the *Request(Call_Transfer_Spanish)* might need to be routed to an entirely different destination.

5 Bootstrapping a spoken dialogue system

This section describes how we bootstrap the main components of a spoken dialogue system, namely the ASR, SLU, and DM. For all modules, we assume that no data from the application domain is available.

5.1 Unsupervised learning of language models

State-of-the-art speech recognition systems are generally trained using in-domain transcribed utterances, preparation of which is labor-intensive and time-consuming. In this work, we retrain only the statistical language models and use an acoustic model trained using data from other applications. Typically, the recognition accuracy improves by adding more data from the application domain to train statistical language models (Rosenfeld 1995).

In our previous work, we have proposed active and unsupervised learning techniques for reducing the amount of transcribed data needed to achieve a given word accuracy for automatic speech recognition, when some data (transcribed or untranscribed) are available from the application domain (Riccardi and Hakkani-Tür 2003). Iyer and Ostendorf (1999) have examined various similarity techniques to selectively sample out-of-domain data to enhance sparse in-domain data for statistical language models, and have found that even the brute addition of out-of-domain data is useful. Venkataraman and Wang (2003) have used maximum likelihood count estimation and document similarity metrics to select a single vocabulary from many corpora of varying origins and characteristics. In these studies, the assumption is that there are some in-domain data (transcribed and/or untranscribed) available and its *n*-gram distributions are used to extend that set with additional data.

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

In this article, we focus on the reuse of transcribed data from other resources, such as human–human dialogues (e.g., Switchboard Corpus; Godfrey *et al.* 1992), or human–machine dialogues from other spoken dialogue applications, as well as some text data from the Web pages of the application domain. We examine the style and content similarity, when out-of-domain data are used to train statistical language models and when no in-domain human–machine dialogue data are available. Intuitively, the in-domain Web pages could be useful to learn domain-specific vocabulary. Other application data can provide stylistic characteristics of human–machine dialogues.

5.2 Call-type classification with data reuse

The bottleneck of building reasonably performing (i.e., deployable) classification models is the amount of time and money spent for high-quality labeling. By "labeling," we mean assigning one or more predefined labels (e.g., *call-types*) to each utterance. In this project, we aim to use a call-type representation that can capture information that is sufficient to fulfill a customer's request. In our representation, users' utterances are labeled with the intent of the speaker as opposed to the action that the system must take in response. The goal of this study is having a framework, where the utterances are labeled consistently across applications, so that we can reuse them to bootstrap a system.

In our previous work, to build call classification systems in shorter time frames, we have employed active and unsupervised learning methods to selectively sample the data to label (Tur *et al.* 2005). We have also incorporated manually written rules to bootstrap the Boosting classifier (Schapire *et al.* 2002) and used it in the AT&T HelpDesk application (Di Fabbrizio *et al.* 2002).

In this study, we aim to reuse the existing labeled data from other applications to bootstrap a given application. The idea is to use the library of call-types along with the associated data and to let the UE expert responsible for that application exploit this information source.

Assume that there is an oracle that categorizes all the possible natural language sentences that can be uttered in any spoken dialogue application we deal with. Let us denote this set of universal classes with $\mathscr{C}$ such that the call-type set of a given application is a subset of that, $\mathscr{C}_j \subset \mathscr{C}$. It is intuitive that some of the call-types will appear in all applications, some in only one of them, etc. Thus, we categorize $\mathscr{C}_j$ into the following call-types.

1. *Generic call-types:* These are the intents appearing independent of the application. A typical example would be a request to talk to a human, instead of a machine. Call this set
$$\mathscr{CG} = \{C_i : C_i \in \mathscr{C}_k, \forall k\}$$
2. *Reusable call-types:* These are the intents that are not generic but have already been defined for a previous application (most probably from the same or similar industry sector) and already have labeled data. Call this set
$$\mathscr{CR}_j = \{C_i : C_i \notin \mathscr{CG}, C_i \in \mathscr{C}_k, \exists k \neq j\}$$

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

3. *Specific call-types:* These are the intents specific to the application, because of the specific business needs or application characteristics. Call this set

$$\mathscr{CS}_j = \{C_i : C_i \notin \mathscr{C}_k, \forall k \neq j\}$$

Now, for each application j, we have

$$\mathscr{C}_j = \mathscr{CG} \bigcup \mathscr{CR}_j \bigcup \mathscr{CS}_j$$

It is up to the UE expert to decide which call-types are reusable or specific, i.e., the sets $\mathscr{CR}_j$ and $\mathscr{CS}_j$. Given that no two applications are the same, deciding on whether to reuse a call-type along with its data is very subjective. There may be two applications including the intent *Request(Account_Balance)*, one from a telecommunications sector, the other from a pharmaceutical sector, and the wording can be slightly different. For example, while in one case we may have *How much is my last phone bill,* the other can be *Do I owe you anything on the medicine*. Since each classifier can tolerate some amount of language variability and noise, we assume that if the names of the intents are the same, their contents are the same. Since in some cases, this assumption does not hold, it is still an open question whether or not a selective sample of the data can be automatically reused.

Since $\mathscr{CG}$ appears in all applications by definition, this is the core set of call-types in a new application, n. Then, if the UE expert knows the possible reusable intents, $\mathscr{CR}_n$, existing in the application, they can be added too. The bootstrap classifier can then be trained using the utterances associated with the call-types, $\mathscr{CG} \bigcup \mathscr{CR}_n$ in the call-type library. For the application-specific intents, $\mathscr{CS}_n$, it is still possible to augment the classifier with a few rules as described in Schapire *et al.* (2002). This is also up to the UE expert to decide.

Depending on the size of the library or the similarity of the new application to the existing applications, it is possible to cover a significant portion of the intents using this approach. For example, in our experiments, we have seen that 10% of the responses to the initial prompt are requests to talk to a human. Using this system, we have the capability to continue the dialogue with the user and determine the intent before sending them to a human agent.

Using this approach, the application begins with a reasonably well working understanding component. One can also consider this as a more complex wizard, depending on the bootstrap model.

Another advantage of maintaining a call-type library and taking advantage of call-type reuse is that it automatically ensures consistency in labeling and naming. Note that the design of call-types is not a well-defined procedure and most of the time, it is up to the UE expert. Using this approach, it is possible to discipline the *art* of call-type design to some extent.

After the system is deployed and real data are collected, then the application-specific or other reusable call-types can be determined by the UE expert to get a complete picture.

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

5.3 Bootstrapping the dialogue manager with reuse

Mixed-initiative dialogues generally allow users to take control over the machine dialogue flow almost at any time of the interaction. For example, during the course of a dialogue for a specific task, a user may utter a new intent (speech act) and deviate from the previously stated goal. Depending on the nature of the request, the DM strategy could either decide to shift to the different context (*context shift*) or reprompt for additional information. Similarly, other dialogue strategy patterns such as *correction, start-over, repeat, confirmation, clarification, contextual help*, and the already-mentioned *context shift*, are recurring features in a mixed-initiative system.

Our goal is to derive some overall approach to dialogue management that would define templates or basic dialogue strategies based on the call-type structure. For the specific call routing task described in this article, we generalize dialogue strategy templates based on the categorization of the call-type presented in Section 5.2 and on best practice user experience design.

Generic call-types, such as *Yes, No, Hello, Goodbye, Repeat, Help*, etc., are domain independent, but are handled in most reusable subdialogues with the specific dialogue context. When detected in any dialogue turn, they trigger context-dependent system replies such as informative prompts (*Help*), greetings (*Hello*), and summarization of the previous dialogue turn using the dialogue history (*Repeat*). In this case, the dialogue will handle the request and resume the execution when the information has been provided. *Yes* and *No* generic call-types are used for confirmation if the system is expecting a yes/no answer or ignored with a system reprompt in other contexts.

Call-types are further categorized as *vague* and *concrete*. A request like *I have a question* will be classified as the vague call-type *Ask(Info)* and will generate a clarification question: *OK, what is your question?* Concrete call-types categorize a clear routing request and they activate a confirmation dialogue strategy when they are classified with low confidence scores. Concrete call-types can also have associated mandatory or optional attributes. For instance, the concrete call-type *Request(Account_Balance)* requires a mandatory attribute *AccountNumber* (generally captured by the SLU) to complete the task.

We generalized subdialogues to handle the most common call-type attributes (telephone number, account number, zip code, credit card, etc.) including a *dialogue container* that implements the optimal flow for multiple inputs. A common top-level dialogue handles the initial open prompt request. Reusable dialogue templates are implemented as ATNs where the actions are executed when the network arcs are traversed and passed as parameters at run time. Disambiguation of multiple call-types is not supported. We consider only the top-scoring call-type, assuming that multiple call-types with high confidence scores are rare events.

6 Experiments and results

For our experiments, we selected a test application from the pharmaceutical domain to bootstrap. We have evaluated the performances of the ASR language model, call classifier, and dialogue manager as described below. We have human–machine spoken language data from two AT&T VoiceTone® spoken dialogue applications

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

Table 1. *ASR data characteristics*

	App. 1	App. 2	Web data	Training data	Test data
No. of utterances	35,551	79,792	NA	29,561	5,537
No. of words	329,959	385,168	71,497	299,752	47,684

Table 2. *Style and content differences among various data sources*[*]

	App. 1 (%)	App. 2 (%)	SWBD (%)	Web data (%)	In-domain training data (%)
Percentage of pronouns	15.14	9.16	14.8	5.30	14.5
Percentage of filled pauses	2.66	2.27	2.74	0	3.26
Test set OOV rate	9.79	1.99	2.64	13.36	1.02

[*] Abbreviations: SWBD, Switchboard corpus; OOV, out-of-vocabulary.

in the library, whose data were collected and transcribed before (App. 1 in the telecommunication domain and App. 2 in the medical insurance domain). We have transcribed training and test data for this new application. We are trying to figure out what would happen if we did not have any data from this new application, and use the training data from this new application only to draw the upper bound. We use the in-domain test data to test the performance of alternative approaches.

6.1 Speech recognition experiments

To bootstrap a statistical language model for ASR, we used the App. 1 and App. 2 data. We also used some data from the application domain Web pages (Web data). Table 1 lists the sizes of these corpora. "Training Data" and "Test Data" correspond to the training and test data we have for the new application and are used for controlled experiments. We also extended the available corpora with human-human dialogue data from the Switchboard corpus (SWBD) (Godfrey *et al.* 1992).

Table 2 summarizes some style and content features of the available corpora. For simplification, we only compared the percentage of pronouns and filled pauses to show style differences, and the domain test data out-of-vocabulary word (OOV) rate for content variations. The human–machine spoken dialogue corpora include many more pronouns than the Web data. There are even further differences between the individual pronoun distributions. For example, out of all the pronouns in the Web data, 35% is "you," and 0% is "I," whereas in all of the human–machine dialogue corpora, more than 50% of the pronouns are "I." In terms of style, both spoken dialogue corpora can be considered as similar. In terms of content, the second application data are the most similar corpus, as it results in the lowest OOV rate for the domain test data.

In Table 3, we show further reductions in the application test set OOV rate when we combine these corpora.

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

Table 3. *Effect of training corpus combination on OOV rate of the test data*[*]

Corpora	Test set OOV rate (%)
App 1 + App 2 data	1.53
App 1 + App 2 + Web data	1.22
App 1 + App 2 + Web + SWBD data	0.88

[*] Abbreviations are explained in the first footnote to Table 2.

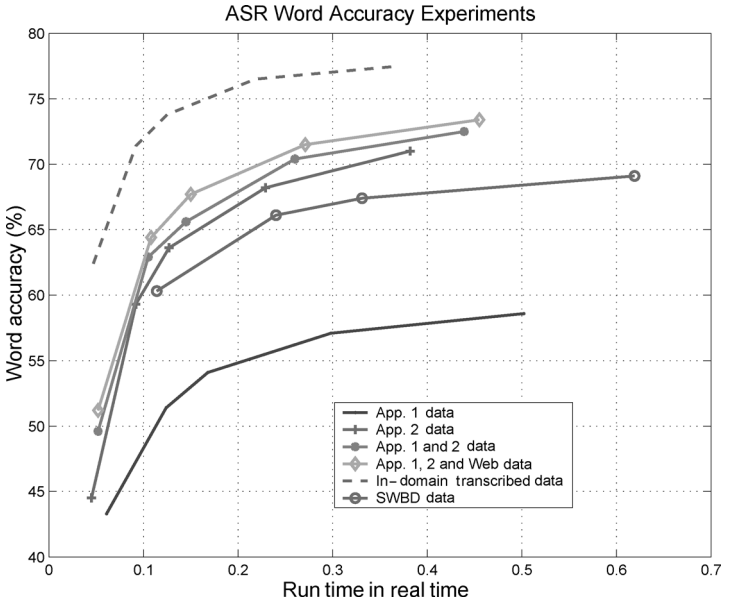


Fig. 7. The word accuracy of in-domain test data, using various language models and pruning thresholds.

Figure 7 shows the effect of using various corpora as training data for statistical language models used in the recognition of the test data. We also computed ASR run-time curves by varying the beam width of the decoder, as the characteristics of the corpora affect the size of the language model. Contentwise, as expected, the most similar corpus (App. 2) resulted in the best performing language model, when the corpora are considered separately. We obtained the best recognition accuracy, when we augment App. 1 data with App. 2 and the Web data. Switchboard corpus also resulted in a reasonable performance, but the problem is that it resulted in a very big language model, slowing down the recognition. In this figure, we also show the word accuracy curve where we use in-domain transcribed data for training the language model.

Once some data from the domain are available, it is possible to weigh the available out-of-domain data and the Web data to achieve further improvements. When we

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

lack any in-domain data, we expect the UE expert to reuse the application data from similar sectors and/or combine all available data.

6.2 Call-type classification experiments

We have performed the SLU tests, using the BoosTexter tool (Schapire and Singer 2000). BoosTexter is a *multiclass* (e.g., there are more than two categories), *multilabel* (e.g., the categories are not mutually exclusive so that the same text could appear in the data labeled with different categories) boosting-based text categorization tool that classifies the speech-recognized text into a certain number of possible semantic categories defined during the training phase. For all experiments, we have used word *n*-grams of transcriptions as features and iterated BoosTexter 1,100 times. In this study, we have assumed that all candidate utterances are first recognized by the same automatic speech recognizer (ASR), so we deal with only textual input of the same quality, which corresponds to the ASR 1-best hypotheses obtained using the language model trained from App. 1, App. 2, and Web data.

For the test set, we again used the 5,537 utterances collected from a pharmaceutical domain customer care application. We used a very limited library of call-types from a telecommunications domain application. We have made controlled experiments where we know the true call-types of the utterances. In this application, we have 97 call-types with a fairly high perplexity of 32.81.

If an utterance has the call-type that is covered by the bootstrapped model, we expect that call-type to get a high confidence. Otherwise, we expect the model to reject it by returning a low confidence or assigning the special call-type *Not(Understood)* meaning that the intent of the utterance is not covered. To evaluate this, we compute the *rejection accuracy* (RA) of the bootstrap model:

$$\mathrm{RA} = \frac{\#\text{ of correctly rejected utterances} + \#\text{ of correctly classified utterances}}{\#\text{ of all utterances}}$$

To evaluate the classifier performance for the utterances whose call-types are covered by the bootstrapped model, we have used *classification accuracy* (CA), which is the fraction of utterances in which the top-scoring call-type is one of the true call-types assigned by a human labeler and its confidence score is more than some threshold, among all the covered utterances:

$$\mathrm{CA} = \frac{\#\text{ of correctly classified utterances}}{\#\text{ of covered utterances}}$$

These two measures are actually complementary to each other. For the complete model trained with all the training data where all the intents are covered, these two metrics are the same, since the number of covered utterances will be the same as the number of all utterances and no utterance should be rejected.

To see our upper bound, we first trained a classifier, using 30,000 labeled utterances from the same application. The first row of Table 4 presents these results, using both the transcriptions of the test set and the ASR output with around 68% word accuracy. For the confidence threshold, the optimum performance was obtained with 0.3; therefore, we used it in all experiments. As shown in this table, 78.27%

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

Table 4. *SLU results using transcriptions and ASR output with the models trained with in-domain data, only generic call-types, and with call-types from the library and rules*

		Transcriptions		ASR output	
	Coverage (%)	RA (%)	CA (%)	RA (%)	CA (%)
In-domain model	100.00	78.27	78.27	61.73	61.73
Generic model	45.78	88.53	95.38	85.55	91.08
Bootstrapped model	70.34	79.50	79.13	71.86	68.40

classification (or rejection) accuracy is the performance using all training data. This reduces to 61.73% when we use the ASR output. This is mostly because of the unrecognized words that are critical for the application. This is intuitive since the ASR language model has not been trained with domain data.

Then we trained a generic model, using only generic call-types. This model achieved better accuracies as shown in the second row, since we do not expect it to distinguish among the reusable or specific call-types. Furthermore, for classification accuracy, we only use the portion of the test set whose call-types are covered by the model and, where the call-types in this model are definitely easier to classify than the specific ones. The drawback is that we cover only about half of the utterances. Using the ASR output, unlike the in-domain model case, did not hurt much, since the ASR already covers only the utterances with generic call-types with great accuracy.

We then trained a bootstrapped model, using 13 call-types from the library and a few simple rules written manually for three frequent intents. Since the library consists of an application from a fairly different domain, we could only exploit intents related to billing, such as *Request(Account_Balance)*. While determining the call-types for which to write rules, we actually played the role of the UE expert who has previous knowledge of the application. This enabled us to increase the coverage to 70.34%.

The most impressive result of these experiments is that we have a call classifier that is trained without any in-domain data and can handle 70.34% of the utterances with almost the same accuracy as the call classifier trained with extensive amounts of data. This is a deployable bootstrap SLU system performance. Noting the weakness of our current call-type library, we expect even better performance as we augment more call-types from ongoing applications.

6.3 Dialogue-level evaluation

Evaluation of spoken dialogue system performances is a complex task and depends on the purpose of the desired dialogue metric (Paek 2001). While ASR and SLU can be fairly assessed offline, using utterances collected in previous runs of the baseline system, the dialogue manager requires interaction with a real motivated user who will cooperate with the system to complete the task. Ideally, the bootstrap system has to be deployed in the field and the dialogues have to be manually labeled to provide an accurate measure of the task completion rate. Usability metrics also require direct

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

feedback from the caller to properly measure the user satisfaction (specifically, task success and dialogue cost) (Walker *et al.* 1997). However, we are more interested in automatically comparing the bootstrap system performances with a reference system, working on the same domain and with identical dialogue strategies.

As a first order of approximation, we reused the 3,082 baseline test dialogues (5,537 utterances) collected by the live reference system and applied the same dialogue turn sequence to evaluate the bootstrap system. That is, we simulated the human–machine interaction, using the dialogue traces from the existing real system logs. For each utterance in the dialogue, we submitted the user-transcribed utterances to the bootstrap SLU for classification. We replicated the dialogue strategy offline, artificially generated dialogue traces, and compared them with the original traces. According to the reference system call flow, the 97 call-types covered by the reference classifier are clustered into 32 DM categories (DMC). A DMC is a generalization of more specific intents, and it is typically used to make the decision about the next dialogue action.

The bootstrap system only classifies 16 call-types and 16 DMC according to the bootstrapping SLU design requirements described in Section 6.2. This is only half of the reference system DMC coverage, but it actually addresses 70.34% of the total utterance classification task. Compared to the fully deployed system, the bootstrap system will only cover a subset of the real call routing destinations. This corresponds to a smaller call-type subset that can be further expanded when the bootstrap system is deployed.

As previously mentioned, we simulate the execution of the dialogues, using data collected from the deployed reference system. For each dialogue turn, the utterance transcription, the audio recording and the labeled tags are available for evaluation. We adopted the following procedure.

For each dialog $\mathscr{D}_i$ in the reference data set, we classify each dialogue user's turn j with the bootstrap classifier and select the result with the highest confidence score c_j. We use two confidence score thresholds, T_{high}, for acceptance, and T_{low}, for rejection. The T_{high} threshold value (0.3) is selected on the basis of the optimal F-measure value. T_{low} threshold value (0.1) is an empirical number that better optimizes the dialogue confirmation strategies, and it is typically suggested by the UE expert. Call-types, whose confidence scores are in between these two thresholds, are typically validated by a confirmation subdialogue. The system will ask the caller a yes/no question to confirm the intention, for example, *Do you want a refill for your prescription?*). In our case, we assume perfect confirmation response (for this class of applications, semantic accuracy for confirmations is typically greater than 90%) and compare directly the manually labeled call-type with the classified one. Then,

1. the utterance j is considered as part of a correct dialogue trajectory if the following condition is verified:

$$\text{score}(c_j) \geq T_{\text{high}} \wedge c_j \in \mathscr{DMC}_j \wedge c_j \in \text{Concrete}$$

where T_{high} is the acceptance threshold, $\mathscr{DMC}_j$ is the manually labeled reference DM category set for the turn j, and Concrete is the set of concrete call-types;

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

Table 5. *DM evaluation results*

	DM category (%)		DM route (%)	
	Transcribed	ASR output	Transcribed	ASR output
Concrete	44.65	34.13	47.18	36.99
Concrete+conf	50.78	42.20	53.47	44.32
Concrete+conf+vague/generic	67.27	57.39	70.67	61.84

2. the utterance j is considered as part of a correct dialogue trajectory with confirmation if the following condition is verified:

$$T_{\text{low}} \leq \text{score}(c_j) < T_{\text{high}} \wedge c_j \in \mathscr{DMC}_j \wedge c_j \in \text{Concrete}$$

   where T_{low} is the rejection threshold;
3. if in any turn of the dialogue, a mismatching concrete call-type is found, the utterance j diverges from the reference dialogue trajectory and the dialogue is considered as unsuccessful:

$$\text{score}(c_j) \geq T_{\text{high}} \wedge c_j \notin \mathscr{DMC}_j \wedge c_j \in \text{Concrete}$$

4. if none of the conditions above are satisfied for the utterance j, we check for *Vague* and *Generic* call-types using the lowest threshold and assuming that the turn j does not contain any relevant user intention. Then the utterance j is considered as part of a correct dialogue trajectory if the following condition is true:

$$\text{score}(c_j) \geq T_{\text{low}} \wedge c_j \in \mathscr{DMC}_j \wedge c_j \in (\text{Vague} \vee \text{Generic})$$

We iterate each utterance in the reference dialogue till we reach the last turn. The dialogue i is then considered successful if we reach the final turn with one of the valid dialogue trajectory conditions (e.g., 1, 2, or 4). Notice that this empirical method, once the dialogue trajectory diverges from the reference path, marks the whole dialogue as unsuccessful and does not keep in consideration possible dialogue repairs that might bring the interaction back on track. For this reason, it can be considered as a lower bound dialogue success rate estimate.

A further experiment considers only the final routing destinations (e.g. specific type of agent or the automatic fulfillment system destination). Both reference and bootstrap systems direct calls to 12 different destinations, implying that a few DM categories are combined into the same destination. This quantifies how effectively the system routes the callers to the right place in the call center and, conversely, gives some metric to evaluate the missed automation and the misrouted calls. The test has been executed for both transcribed and untranscribed utterances. Results are shown in Table 5. Even with a modest 50% DM Categories coverage, the bootstrap system shows an overall task completion of 67.27% in the case of transcribed data and 57.39% using the output generated by the bootstrap ASR. When considering the route destinations, completion increases to 70.67% and 61.84%, respectively. This approach explicitly ignores the dialogue context, but it contemplates the call-type

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

categorization, the confirmation mechanism, and the final route destination, that would be missed in the SLU evaluation. Although a more completed evaluation analysis is needed, lower bound results are indicative of the overall performances.

7 Summary

This article shows that bootstrapping a spoken dialogue system reusing existing transcribed and labeled data from out-of-domain human–machine dialogues, common reusable dialogue templates and patterns, is possible to achieve operational performances. Our evaluations on a call classification system using no-domain-specific data indicate 67% ASR word accuracy, 79% SLU call classification accuracy with 70% coverage, and 62% routing accuracy with 50% DM coverage. Our future work consists of developing techniques to refine the bootstrap system when application domain data become available.

Acknowledgments

We thank Liz Alba, Harry Blanchard, Patrick Haffner, and Jay Wilpon for many helpful discussions. We are grateful to the anonymous reviewers whose comments were greatly helpful to improve the quality of this article.

References

Abella, A. and Gorin, A. 1999. Construct algebra: Analytical dialog management. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, Washington, DC, June.

Bobrow, D. and Fraser, B. 1969. An augmented state transition network analysis procedure. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 557–567, Washington, DC, May.

Buntschuh, B., Kamm, C., Di Fabbrizio, G., Abella, A., Mohri, M., Narayanan, S., Zeljkovic, I., Sharp, R. D., Wright, J., Marcus, S., Shaffer, J., Duncan, R. and Wilpon, J. G., 1998. VPQ: A spoken language interface to large scale directory information. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Sydney, New South Wales, Australia, November.

Di Fabbrizio, G., Dutton, D., Gupta, N., Hollister, B., Rahim, M., Riccardi, G., Schapire, R. and Schroeter, J. 2002. AT&T Help Desk. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, CO, September.

Di Fabbrizio, G. and Lewis, C. 2004. Florence: A dialogue manager framework for spoken dialogue systems. In *ICSLP 2004, 8th International Conference on Spoken Language Processing*, Jeju, Jeju Island, Korea, October 4–8.

Di Fabbrizio, G., Tur, G. and Hakkani-Tür, D. 2004. Bootstrapping spoken dialog systems with data reuse. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*, Cambridge, MA, April 30 – May 1.

Dybkjr, L. and Bernsen, N. 2000. The MATE workbench. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*, Athens, Greece, May.

Godfrey, J. J., Holliman, E. C. and McDaniel, J. 1992. Switchboard: Telephone speech corpus for research and development. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol 1, pages 517–520, San Francisco, CA, March.

Goffin, V., Allauzen, C., Bocchieri, E., Hakkani-Tür, D., Ljolje, A., Parthasarathy, S., Rahim, M., Riccardi, G. and Saraclar, M. 2005. The AT&T Watson Speech Recognizer. In

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press

Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Philadelphia, PA, May 19–23.

Gorin, A. L., Riccardi, G. and Wright, J. H. 1997. How may I help you? *Speech Communication* **23**: 113–127, October.

Gupta, N., Tur, G., Hakkani-Tür, D., Bangalore, S., Riccardi, G. and Rahim, M. 2006. The AT&T Spoken Language Understanding System. *IEEE Transactions on Audio, Speech and Language Processing* **14**(1): 213–222, January.

Iyer R. and Ostendorf, M. 1999. Relevance weighting for combining multi-domain data for *n*-gram language modeling. *Computer Speech & Language* **13**: 267–282, July.

Kotelly, B. 2003. *The Art and the Business of Speech Recognition—Creating the Noble Voice*, chapter 5, pp. 58–64. Addison-Wesley.

Lewis, C. and Di Fabbrizio, G. 2005. A clarification algorithm for spoken dialogue systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Philadelphia, PA, May 19–23.

McTear, M. F. 2002. Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys (CSUR)* **34**(1): 90–169, March.

NAICS. 2002. North American Industry Classification System (NAICS). http://www.census.gov/epcd/www/naics.html

Natarajan, P., Prasad, R., Suhm, B. and McCarthy, D. 2002. Speech enabled natural language call routing: BBN call director. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, CO, September.

Paek, T. 2001. Empirical methods for evaluating dialog systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL) Workshop on Evaluation Methodologies for Language and Dialogue Systems*, Toulouse, France, July.

Riccardi, G. and Hakkani-Tür, D. 2003. Active and unsupervised learning for automatic speech recognition. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland, September.

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

Rosenfeld, R. 1995. Optimizing lexical and *n*-gram coverage via judicious use of linguistic data. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, vol. 2, pp. 1763–1766, Madrid, Spain, September.

Schapire, R. E. and Singer, Y. 2000. BoosTexter: A boosting-based system for text categorization. *Machine Learning* **39**(2/3): 135–168.

Schapire, R. E., Rochery, M., Rahim, M. and Gupta, N. 2002. Incorporating prior knowledge into boosting. In *Proceedings of the International Conference on Machine Learning (ICML)*, Sydney, New South Wales, Australia, July.

Schapire, R. E. 2001. The boosting approach to machine learning: An overview. In *Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, March.

Sutton, S. and Cole, R. 1998. Universal speech tools: The CSLU toolkit. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Sydney, New South Wales, Australia, November.

Tur, G., Hakkani-Tür, D. and Schapire, R. E. 2005. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication* **45**(2): 171–186.

Venkataraman, A. and Wang, W. 2003. Techniques for effective vocabulary selection. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland, September.

VoiceXML. 2003. Voice extensible markup language (VoiceXML) version 2.0. http://www.w3.org/TR/voicexml20/

Walker, M. A., Litman, D. J. Kamm, C. A. and Abella, A. 1997. PARADISE: A framework for evaluating spoken dialogue agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)–Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Madrid, Spain, July.

https://doi.org/10.1017/S1351324907004561 Published online by Cambridge University Press