A Flexible Method for Manipulating Fields in a Directory Assistance Application

TR 29.3731
February 24, 2004

Vanessa V. Michelini
Ciprian Agapi
Michael H. Mirt
Fang Wang
James R. Lewis
Melanie D. Polkosky

IBM Pervasive Computing

Boca Raton, Florida

Abstract

This technical report describes a method to define, in a configuration file, the fields that describe a person in a directory database context. Each field will be described by a set of attributes that defines the field characteristics and behavior. One example of a field characteristic is whether the field is mandatory or optional in the application context. A company directory assistance application contains employee's names and information. Fields classified as mandatory in this context are first and last names. A field behavior is defined by how this field will be used within the application. In the same example, one of the behaviors of a telephone number is being a number where the application can transfer calls to, in other words, a "dial-enabled" number. The fields definition is used during the procedure to load the directory information into the application database, to generate grammars for the speech recognition application and in the call flow component.
In the last section we discuss how to apply this same concept in different applications.

ITIRC Keywords

Directory dialer
Dialer
Name dialer
Voice activated dialer
Speech enabled dialer
Directory assistance

Contents

Introduction

This report describes a method to define in a configuration file the fields that describe a person in a directory database. The fields definition will be used during the directory import procedure, grammar generation and call flow application. It is easy to add new fields using this method.

In Directory Assistance applications, one of the tasks in the design phase is to define the fields that will be supported by the application. The typical scenario is a person calling the Directory Assistance system to retrieve information about a person or listing. After requesting the desired person or listing, the system should be able to provide additional information. In a voice-enabled application, the fields are referred to by multiple components, such as grammars, databases, import files, etc. Here is a sample dialog:

System: Thank you for calling the Voice Directory. What is the name of the person you would like to contact?

Caller: Vanessa Michelini

System: Thank you. Vanessa Michelini's phone number is 561-862-2159. Would you like me to dial it now?

Caller: No

System: What contact information would you like to hear?

Caller: Mobile

System: The mobile number is 954-123-4567. More contact info for Vanessa Michelini?

Caller: Dial

System: Now transferring your call.

There are a set of basic fields that can be defined in advance, based on past experience of kind of information requested in this type of system. Depending on the customer, however, different fields may need to be added and this would imply code changes. This report describes a way to define the fields dynamically, so new fields can be added to the system without changing the code.

Design Approach

Field Definition

In this design approach, let's assume there is a set of special fields that are always defined. Let's use a directory of names as example. The system will recognize the requested person by first name and last name. In this case, the fields <gn> (given name), <sn> (last name) and <tn> (default telephone number) are defined as special and mandatory fields. When the caller says the person's name, the system retrieves the record information and reads back the default telephone number. If additional fields are defined, the system will allow the caller to ask for them. The fields are defined in a XML file. For each field, the following attributes are defined:

Attribute Name	Definition
<name>	Field name. A label to refer to this field inside the code
<audio>	The audio file to be played when this field is announced in a prompt
<tts>	The text to be used by TTS, if this field is announced in a prompt
<dialable>	Yes/No. If yes, this field is a telephone number that can be dialed to
<mandatory>	Yes/No. Indicates if this field is mandatory.
<spell>	Indicates the word (s) associated to this field
<outro>	Yes/No. Indicates if this field is part of the "outro" definition in the call flow
<contactInfo>	Yes/No. Indicates if this field can be retrieved in directory assistance mode

For example:

```
<field>
        <name>mobile
        <audio>mobile.wav
        <tts>"mobile number"
        <dialable>yes
        <mandatory>no
        <spell>mobile
        <spell>cell phone
        <spell>cell
        <outro>yes
        <contactInfo>yes
</field>
<field>
        <name>email
        <audio>email.wav
        <tts>"e-mail address"
        <dialable>no
        <mandatory>no
        <spell>e-mail
        <spell>e-mail address
        <outro>no
        <contactInfo>yes
</field>
```

Person Definition

Then, for each person in the directory, an entry is defined in the import file as follow:

```
<person>
        <gn>Vanessa
        <sn>Michelini
        <tn>561-862-2159
        <mobile>954-123-4567
        <email>vm@us.ibm.com

</person>
<person>
        <gn>Brent
        <sn>Davis
        <tn>561-862-2177
        <mobile>954-567-8901
        <email>bd@us.ibm.com
</person>
```

The first 3 fields are the special fields <gn>, <sn> and <tn>. The other 2 fields are defined in the fields definition file. The field name (mobile, email) is used as a tag to refer to that field. Multiple grammars are dynamically generated for this application. The first one is the names grammar:

```
<<start>> = <name> <outro>?
```

The Name Rule

The <name> rule is generated from <gn> and <sn> fields. In this design, the directory information is stored in the grammar file, as the annotation. So, as soon as a name is recognized, the information of that person is also retrieved. All the fields are appended, as follows:

```
<name> =
        Vanessa Michelini:      {tn=5618622159;mobile=9541234567;email=vm@us.ibm.com;}   |
        Brent Davis:            {tn=5618622177;mobile=9545678901;email=bd@us.ibm.com};
```

The Outro Rule

The <outro> rule is generated from the fields definition file, using all fields which outro=yes. The spelling (<spell> tag) is how the field can be referred to by the caller and the grammar annotation is the field name (<name> tag). In this example:

```
<outro> = (mobile | cell phone | cell): mobile;
```

Based on the example described above, the following combinations are accepted by the prompt "What is the name of the person you would like to contact?":

Command	Rule
Vanessa Michelini	<gn> <sn>
Vanessa Michelini mobile	<gn> <sn> <outro>
Vanessa Michelini cell phone	<gn> <sn> <outro>
Vanessa Michelini cell	<gn> <sn> <outro>
Brent Davis	<gn> <sn>
Brent Davis mobile	<gn> <sn> <outro>
Brent Davis cell phone	<gn> <sn> <outro>
Brent Davis cell	<gn> <sn> <outro>

The Contact Rule

The next grammar is the one loaded when the system prompts "What contact information would you like to hear?". In this grammar, all contact information fields (contactInfo=yes), including their different spellings, should be included:

```
<<contact>> = (mobile | cell phone | cell): mobile |
                (email | email address): email;
```

When the caller says "cell phone", the annotation "mobile" is returned and the system can search for the mobile information in the person's record. There are no hard-coded references in the system. When the system reads back "The mobile number is", the string "mobile number" come from the <tts> tag or, if a pre-recorded audio is defined, the system will play the <audio> tag for that field.

Adding a New Field

Let's say a new field will be defined:

```
<field>
        <name>secretary
        <audio>secretary.wav
        <tts>"secretary's phone number"
        <dialable>yes
        <mandatory>no
        <spell>secretary
        <spell>secretary phone
        <outro>no
        <contactInfo>yes
</field>
```

and this field is added to Vanessa's record:

```
<person>
        <gn>Vanessa
        <sn>Michelini
        <tn>561-862-2159
        <mobile>954-123-4567
        <email>vm@us.ibm.com
        <secretary>561-862-2189
```

```
</person>
```

The grammar with the name will remain the same, as the field secretary is not defined as outro=yes. The secretary phone number is appended in the annotation:

```
<name> =
        Vanessa Michelini:
{tn=5618622159;mobile=9541234567;email=vm@us.ibm.com;secretary=5618622189}   |
        Brent Davis: {tn=5618622177;mobile=9545678901;email=bd@us.ibm.com;secretary=;};
```

The grammar for contact information is generated as follow:

```
<<contact>> = (mobile | cell phone | cell): mobile |
                (email | email address): email |
                (secretary | secretary number);
```

So, without changing the code, the system is now able to accept the request for secretary telephone number. If the information is not available (e.g., Brent Davis doesn't have a secretary phone number), the code can handle the situation by presenting a message with options: "Secretary's phone number" is not available for Brent Davis, please choose "mobile number" or "email address". Note that all strings are extracted from the fields definition file.

Ease of Translation

This design is very flexible. As there are no hard-coded references of the fields in the code, the application can easily be translated to other languages. The fields definition file in this case will contain the translated string in the <spell> tag. For instance, let's use the mobile field as example in Portuguese language:

```
<field>
        <name>mobile
        <audio>mobile.wav
        <tts>"número do celular"
        <dialable>yes
        <mandatory>no
        <spell>celular
        <spell>telefone celular
        <outro>yes
        <contactInfo>yes
</field>
```

In this case, the <outro> rule will be created as follow:

```
<outro> = (celular | telefone celular): mobile;
```

The caller can say "Vanessa Michelini celular" to dial to Vanessa's cell phone. The contact information rule will be:

```
<<contact>> = (celular | telefone celular): mobile;
```

The system will use the <tts> and <audio> tag to read the information back. Assuming the prompts are also translated, the result would be:

O número do celular é 954-123-4567.

Discussion

The method described in this report is ideal for making applications flexible and minimizes the work required to extend the application to support additional fields and for translation into other languages. However, it adds complexity to the design and development phases, as there are no direct references to the fields in the code.

One key challenge is creating an appropriate speech user interface design. Although pre-recorded prompts can use the field names and references because they are easily replaced by new recordings when the application is extended or translated, the interface flow must allow extensions. One example is the employee contact information in a directory assistance application. Suppose the first version supports mobile number and department name as part of the employee contact information. The speech user interface must be designed to provide this information to the caller and, at the same time, be prepared to be extended in future releases to support additional fields, such as an e-mail address.
Consider the following two scenarios:

Scenario 1

[S]: Please select mobile number or department name.
[C]: mobile number
[S]: The mobile number is 561-123-4567
[S]: Please select mobile number or department name.

Scenario 2

[S]: What contact information would you like to hear? …..I can provide mobile number or department name. Which one would you like?
[C]: mobile number
[S]: The mobile number is 561-123-4567
[S]: What contact information would you like to hear?

In the first scenario the prompt immediately says the available options, which helps novice users. However, as the code loops back to the first prompt after playing back the information, the resulted call flow doesn't seems very smart and the caller can think "I just asked the mobile number…why this system is telling me to select mobile or department?".

In the second scenario, the system plays an open question (What contact information would you like to hear?). Expert users will know what to say. Novice users will probably hesitate and the help message will then play the available options (I can provide mobile numbers or department names. Which one would you like?). The resulted call flow seems much smarter in this scenario when the application loops back to the first prompt.

Thinking beyond the directory assistance context, this same method can be used in other applications with similar requirements. For example, consider an application that allows users to add upgrades in a computer configuration. Each upgrade is a field with characteristics and behavior. For each computer model, a different set of fields is supported. The same application can be used to support the various models, each one with a different fields definition.

Although the idea of separating the fields in a configuration file seems very trivial and good programming practice only, the actual implementation of the idea is very complex. On the other hand, it is worthwhile in situations where the application will be extended, translated or reused as this method will save future costs and time.