A VoiceXML Framework for Reusable Dialog Components

Jaroslav Gergic, Rafah Hosn, Jan Kleindienst, Stéphane H. Maes, TV Raman, Jan Sedivy, Ladislav Seredi

Presenting Author: Stéphane H. Maes, smaes@us.ibm.com

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA

Extended Abstract

1. Introduction

VoiceXML [1], or Voice extensible Markup Language, is a special markup language designed to facilitate the creation of speech application, especially interactive voice response (IVR) application. At the difference of conventional IVR programming frameworks, that involve proprietary scripts and programming languages over proprietary / closed platforms, VoiceXML provides a declarative, programming framework based on XML and ECMAScript, designed to run on web like infrastructures of web servers and web application servers (i.e. the Voice browser).

This paper addresses options to implement reusable dialog components within the VoiceXML framework as specified in VoiceXML 1.0 [1] and currently extended by the W3C Voice Browser working group [2]. The W3C has published reusable dialog components requirements [3].

In this paper, we present some suggestions and lessons learned so far in terms of: VoiceXML specifications and execution flow, including the form interpretation algorithm and reusable dialog components.

2. High level considerations on Reusable Dialog Components

From an application developer's point of view, it is important to provide pre-packaged reusable dialog components and sample code that can be use as libraries or as sample code / templates to build more complex applications and reusable dialog modules or customize them. It is especially important that these components allow users to learn VoiceXML from them and re-use them as they get deployed. For these reasons, it is critical to provide dialog reusable components authored in VoiceXML.

As we designed VoiceXML applications we have started to develop such reusable dialog component frameworks.

3. Reusable VoiceXML Dialog Components and Beans Framework

Reusable components are being worked on within the W3C voice browser working group. For a list of the requirements identified so far, see [3]. This paper details two frameworks for reusable dialog components built within the VoiceXML specifications [1].

- A client-side framework based on the <subdialog> tag and ECMAScript parameter objects to pass parameters, configuration and results. This solution is interpreted at the client side (VoiceXML browser).
- A server-side framework based on JSP (Java Server Pages) and beans that generate VoiceXML subdialogs. This solution can be evaluated at the server side. Other server side technologies could be considered.

These frameworks are not exclusive. They can be mixed and matched to provide the right solution. By design, the frameworks remain within the VoiceXML specifications and require no modification of the VoiceXML interpreter (except for dynamic compilation of grammars).

4. Reusable VoiceXML Dialog Components

The reusable dialog components are built using VoiceXML subdialogs and passing arguments through ECMAScript objects. This framework is compatible with any existing VoiceXML interpreter, independently of specific implementation platforms.

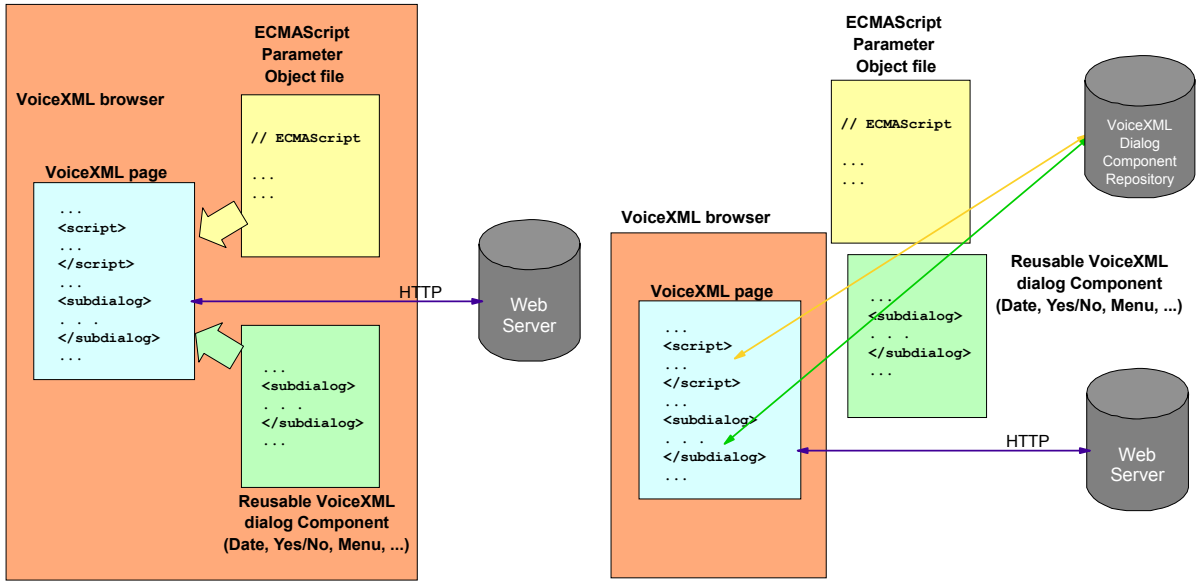

Figure 1illustrates client-side reusable VoiceXML dialog components with a local libraries or provided in a remote repository. Of course, application developers can and will mix and match these different approaches.

4.1 Skeleton specification

- An ECMAScript parameter object is created for each reusable VoiceXML dialog component.
- The parameter objects are populated with the appropriate parameters. For task reusable VoiceXML dialog components, all the parameters are optional. For template reusable VoiceXML dialog components, additional configuration parameters are mandatory (e.g. the collection of menu items which will be rendered by a navigation menu reusable VoiceXML dialog component). The ECMAScript parameter object is characterized by the use of ECMAScript parameter objects as containers that provide: default prompts and other object-specific resources, constructor that combines default and application-specific parameters, common methods for manipulating parameter content.
- reusable VoiceXML dialog components are invoked via the <subdialog> tag. Each reusable VoiceXML dialog component is passed its respective ECMAScript parameter object created as described above. Passing only one parameter in the <subdialog> simplifies the readability of the code since the amount and complexity of parameters even for a single reusable VoiceXML dialog component can be enormous. The execution follows the VoiceXML specifications [1].
- The reusable VoiceXML dialog component are implemented as standard VoiceXML documents referenced by the `src` attribute of the VoiceXML <subdialog>. The results of the reusable VoiceXML dialog components are returned in the return variable of the ECMAScript parameter object. It should easily fit the Natural Language Semantics Markup Language [4].

4.2 Reusable VoiceXML dialog component programming framework

The lessons learned of implementing reusable VoiceXML dialog components showed that two new requirements with respect to VoiceXML 1.0:

- **dynamic grammar generation**
  Until this capability is supported, the grammar must be dynamically compiled outside of the VoiceXML interpreter. For example, **`<grammar src="javascript:MP.grammar"/>`** in `simpComponent.vxml` can be replaced by: **`<grammar src="http://www.grammar_generator.example/dynamiccompiler?MP.grammarfURI">`** or **`<grammar src="http://localhost/dynamiccompiler?MP.grammarfURI">`** depending if the compiler is on a server or local. `MP.grammarfURI` is a javascript function that converts the grammar parameters in an URI compliant form. Other plug-in mechanisms can be considered. We recommend adding such a specification to the VoiceXML specification. Dynamic grammar generation is needed for template reusable VoiceXMXL dialog

components as illustrated by a `select menu`. There exist task reusable VoiceXML dialog components that may be hard to implement without dynamic-grammar support, e.g. `spoken-and-spelled name`.

- **ability to pass audio as variable**
  It is needed only if audio prompts are desired. In VoiceXML 1.0, the prompts must be generated outside the VoiceXML interpreter and passed by URI similarly to the dynamic grammar case.

The key advantages of the proposed approach are:

- The reusable VoiceXML dialog components are browser independent. They do not require any change to existing browsers, except for the new VoiceXML requirements and alternatives that we proposed.
- Authoring of complex reusable dialog components and modules are left to skilled VoiceXML programmers:
- Developers who use the reusable VoiceXML dialog components can use them as libraries or as sample code / templates to build more complex applications and reusable VoiceXML dialog modules or customize them. This has the same impact on easing learning and adoption as was seen in the world of HTML where users could see how a certain page was written when authoring their sites. This was instrumental in the exponential growth of HTML; the intent of our proposal is to do the same for voice applications.

The proposed reusable VoiceXML dialog components fulfills the gist of the W3C reusable dialog component requirements [3]. The requirement for simultaneous / parallel activation of the components is more tricky. It is currently an item beyond the scope of VoiceXML 10, currently addressed only through shared grammars across form items. To address it requires additions to the existing requirements [3] to support mixed initiatives: context sharing across reusable dialog components (subdialogs as well as objects) and modification of the VoiceXML execution model and form interpretation algorithm.

While developing client-side VoiceXML applications, we have observed that it can be sometimes difficult to build numbers of voice-based application without access to dynamic data sources like dynamic grammar compilation and dynamic access to data bases (via HTTP server or by ODBC, SQL, etc…). We will address this issue later in this document. Therefore, we believe components and mechanisms to provide access to dynamic data sources should also be addressed in these requirements.

5. Reusable VoiceXML Dialog Beans

5.1 Server-centric Reusable dialog components

Reusable VoiceXML dialog components and modules are not the only way to satisfy [3] within the existing VoiceXML framework and using existing VoiceXML browsers and web infrastructures. Server-side solutions with dynamic generation of the VoiceXML pages can immediately solve the issues of dynamic manipulation of prompts, dynamic grammar compilation and dynamic access to data sources.

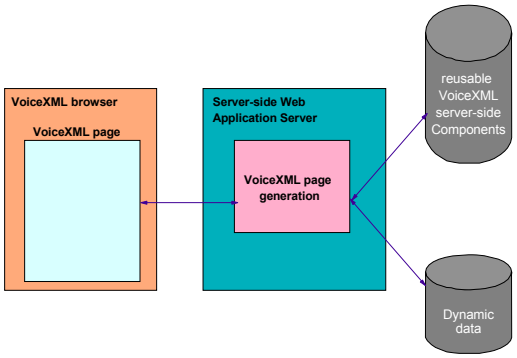

Figure 3 – generic framework for server-side reusable dialog components.

A common issue with server side reusable dialog components is the selection of a programming language - environment (Java, Perl, Python, PHP, C, VisualBasic, ...). The dialog component library should be programming language agnostic in order to achieve wide acceptance and re-usability.

The server-centric reusable dialog component generate VoiceXML code for the VoiceXML browser in a multiple step process: first, according to the description of the reusable dialog component, an intermediate VoiceXML code is constructed from the prepared reusable dialog components, then data from the dynamic data sources are inserted into it, finally the resulted pure VoiceXML code (with or without <object> or <subdialog> elements) is sent to the browser.

Figure 3 illustrates a generic server-centric framework for server-centric reusable dialog components.

We illustrate a server-centric solution based on Java Server Pages (JSP) with embedded reusable VoiceXML dialog beans (JavaBeans). There are numerous other simpler ways to implement server-side reusable dialog components. For

example it is possible to directly reuse the client-side reusable VoiceXML dialog components described in the previous section.

Accordingly, VoiceXML are dynamicallty generated by JSPs. The JSP call beans. Beans are subdivided into two categories: VoiceXML dialog beans that generate VoiceXML subdialogs. A simple framework can generate all the reusable dialog components enumerated in [3]; Service beans that perform other system task (dynamic access to backend (via HTTP server or by ODBC, SQL, etc...), determination of client properties, compilation of dynamic grammars etc…).

Reusable VoiceXML dialog beans are embedded in JSP pages that are responsible for bean creation and rendering. Since a full-fledged object-oriented procedural language (Java) is used for implementing the server-side reusable VoiceXML dialog beans, the speech objects are easily reusable through inheritance or aggregation.
The rendering follows the modified model view controller (MVC) principle, where the UI renderers, acting as views, encompass markup-specific rendering code (`renderFace`) and the model maintains the modality-independent state of the dialog bean.

In summary, we propose a VoiceXML beans framework where the beans can match the reusable VoiceXML dialog components or modules discussed earlier. A markup template processing engine (JSP, ASP, PHP) provide a convenient framework to dynamically generate the VoiceXML pages. In addition other service beans can be used to dynamically compile grammars and dynamically manipulate prompts.

5.2 Reusable VoiceXML dialog component programming framework

The key advantages of the proposed approach are:

- The reusable VoiceXML dialog beans are browser independent.
- Authoring of complex reusable dialog components and modules are left to skilled VoiceXML programmers: (Reusable VoiceXML dialog beans , Grammar files, etc…)
- Developers who use the reusable VoiceXML dialog bean just need to author VoiceXML templates (JSPs, ASPs, PHPs) that call the rendering code of the reusable VoiceXML dialog beans as they can involve other service beans.
- This framework exploits the servlet paradigm for web servers / web application servers. As such it directly fit into numbers of robust and scaleable web / e-business infrastructures.
- The use of the MVC principle enables extension of the application authoring to multi-channel and multi-modal applications [5].
- The beans framework and server side application can maintain context and enable context sharing and dialog flow control. Therefore, this framework is ready to support mixed initiative applications when available.
- The reusable VoiceXML dialog bean framework, satisfies [3].

6. Conclusions

We have illustrated how [3] can be satisfied within the VoiceXML framework [1] provided that the framework is extended to support dynamic grammar compilation and ability to pass audio as variable. We have provided solutions around these extensions and illustrated how server-side solution can also support simultaneous activation of reusable dialog components.

7. References

[1] VoiceXML forum, http://www.voicexml.org
[2] W3C Voice browser activity, http://www.w3.org/Voice/
[3] Reusable Dialog Requirements for Voice Markup Language, W3C Working Draft, 26 April 2000, http://www.w3.org/TR/reusable-dialog-reqs
[4] Natural Language Semantics Markup Language for the Speech Interface Framework, W3C Working Draft, 20 November 2000, http://www.w3.org/TR/nl-spec/
[5] S. H. Maes and T. V. Raman, Multi-modal interaction in the Age of Information Appliance, in Proceedings ICME 2000, July 2000, New York, USA.