VoiceXML Data Logging

Overview

VoiceXML Forum Tools Committee

Draft 0.3 - 20 August 2007

About the VoiceXML Forum:

Founded in 1999, the VoiceXML Forum is an industry organization whose mission is to promote and to accelerate the worldwide adoption of VoiceXML-based applications. To this end, the Forum serves as an educational and technical resource, a certification authority and a contributor and liaison to international standards bodies, such as the World Wide Web Consortium (W3C) IETF, ANSI and ISO. The VoiceXML Forum is organized as a program of the IEEE Industry Standards and Technology Organization (IEEE-ISTO). Membership in the Forum is open to any interested company. For more information, please visit the Website at www.voicexml.org.

Disclaimers:

This document is subject to change without notice and may be updated, replaced or made obsolete by other documents at any time. The VoiceXML Forum disclaims any and all warranties, whether express or implied, including (without limitation) any implied warranties of merchantability or fitness for a particular purpose.

The descriptions contained herein do not imply the granting of licenses to make, use, sell, license or otherwise transfer any technology required to implement systems or components conforming to this specification. The VoiceXML Forum, and its member companies, makes no representation on technology described in this specification regarding existing or future patent rights, copyrights, trademarks, trade secrets or other proprietary rights.

By submitting information to the VoiceXML Forum, and its member companies, including but not limited to technical information, you agree that the submitted information does not contain any confidential or proprietary information, and that the VoiceXML Forum may use the submitted information without any restrictions or limitations.

Abstract

The VoiceXML Tools Committee, under direction of the VoiceXML Forum, is developing specifications for building software related to VoiceXML. The specification described here is a data logging format and is called the Session Log Annotation Markup Language (SLAML). This document outlines requirements for the specification under development by the Data Logging Working Group.

Contributors

The VoiceXMLTools Committee comprises over 60 companies, many of whom contributed to the ideas presented in the specification. The effort was divided up into four teams:

1. **Application server**: France Telecom & Intervoice
2. **Voice browser**: Genesys & West
3. **Speech recognition server**: Nuance & Lumenvox
4. **Style guide & overview**: Genesys & SpeechPhone

Although many committee members were involved in designing the specification, primary authors for the documents were:

- Bogdan Blaszczak (Intervoice)
- Kyle Danielson (Lumenvox)
- Chung Lai (Lumenvox)
- David Thomson (SpeechPhone)
- Andrew Wahbe (Genesys)

Document List

This document provides an overview of the Data Logging Specification. The Data Logging Specification is organized with a separate document for each entity – initially the speech recognition server, the application server, and the VoiceXML browser – plus a set of SLAML documents that describe the format and syntax. In the future, the set of specifications will be expanded to include text-to-speech synthesis, speaker verification, web servers, and possibly database, development and/or analysis systems, and other transaction servers. The specification currently consists of five documents:

1. **overview.doc** – Introduction and high-level description.
2. **slaml/SLAML.html** - Main page for SLAML specification with syntax and rules
3. **slaml/Vxml-session.html** - VoiceXML Browser Data Logging spec.
4. **ASR_SLAML-wd-1.8.html** - Speech recognizer Data Logging spec.
5. **ASLS-SLAML-wd-1.5.html** - Application server Data Logging spec.

The key document is SLAML.html, which defines the style followed in the Data Logging Specification. SLAML.html is supported by documents that follow this style and provide details for logging data on specific entities (see Figure 1). These documents are found at http://www.voicexml.org/datalogging/.

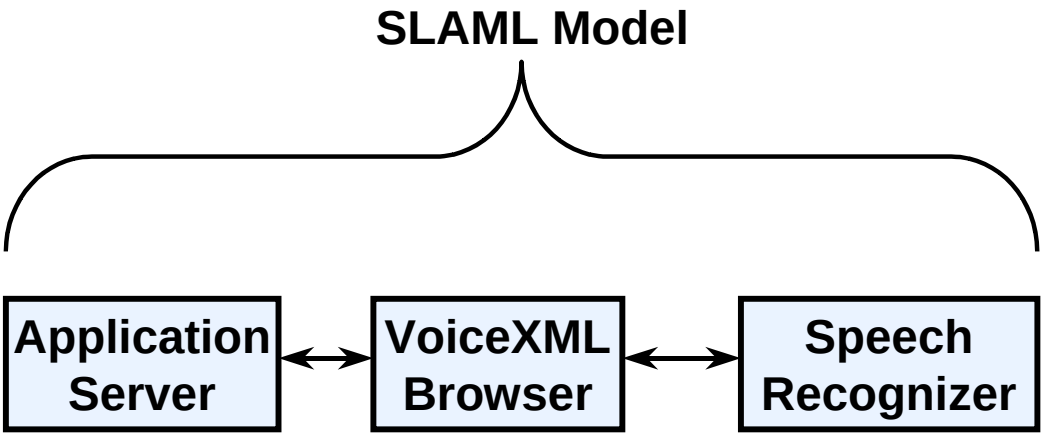


Figure 1. Draft documents exist for the overall SLAML model, the application server, the VoiceXML browser, and the speech recognizer.

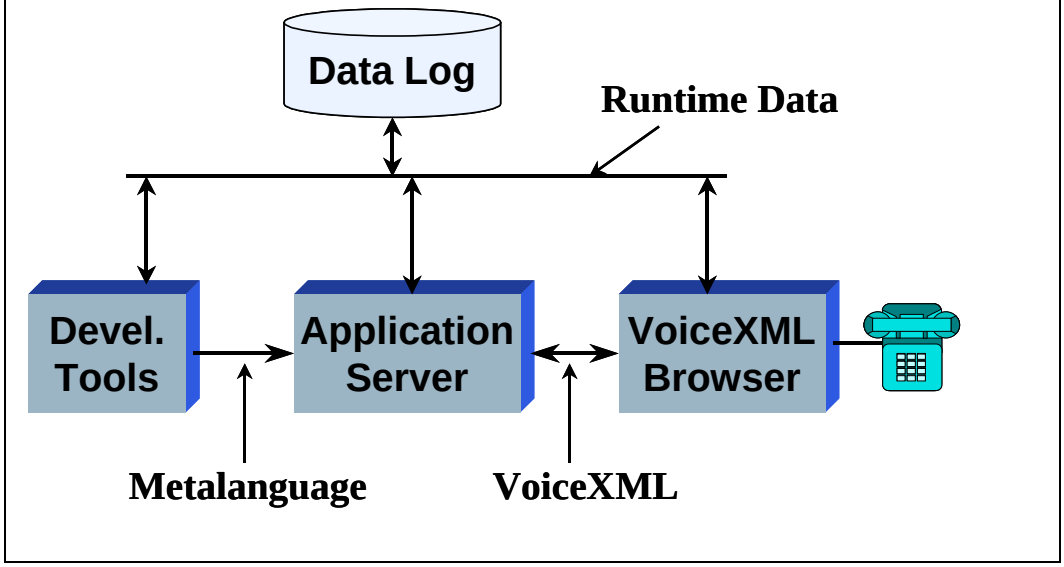
Introduction

The VoiceXML Tools Committee's objective is to improve time to market for voice-activated services by increasing the quality, availability, and interoperability of software for building and supporting VoiceXML-based speech services. We accomplish this objective by proposing standards that allow tools to be platform-independent and vendor-independent, so that tools created by different vendors are compatible. In particular, we define the protocols used for communication paths between tools.

Figure 2 shows a complete VoiceXML system divided into three parts, an application development environment (a.k.a. offline tools), an application server (a.k.a. online tools, sometimes called a VoiceXML page server or document server), and a VoiceXML browser (sometimes called a speech server or voice gateway). Service creation developers use application development tools to write the application. The application server takes the application specification as input and creates VoiceXML. The VoiceXML browser uses VoiceXML as input to provide the service.

Development tools include, but are not limited to, a grammar compiler; a table-based, GUI-based, wizard-based, or script-based call flow designer; a prompt and/or waveform editor; an expert system that assists the developer in making wise service design choices; error checking routines, and finite-state and n-gram grammar generation software. The output of the development environment may be a VoiceXML script or it may be metacode - a representation of the service call flow in a form that is later converted to VoiceXML pages by the application server.

Figure 2. High level view of a VoiceXML system showing three major building blocks.



During runtime, the application server sends VoiceXML documents to and receives documents from the browser in response to user input and other events. The browser executes VoiceXML code and uses touch-tone, recorded prompts, text-to-speech, and speech recognition software to communicate with callers.

Figure 3 shows a detailed view of the application development and application server blocks. While details may vary, tools created by most vendors use this same general architecture. The sections pertaining to Data Logging lie in the pink shaded area across the top. Even though the diagram depicts data logs stored in a central server, we do not specify whether data is logged centrally or within databases internal to the systems creating the data records.

The call flow designer helps the developer write an application, either via a text editor or a GUI (graphical user interface). The Call Flow Designer uses a grammar builder that creates grammar structures for use by a speech recognizer. Additionally, it may have pre-built high-level scripting objects (call flow subroutines in metacode or VoiceXML, pre-built grammars, etc.) for speeding development of common user interactions. The conversation manager receives the service description represented in metacode and generates VoiceXML pages (and accepts corresponding signals from the VoiceXML browser) during runtime. The conversation manager may be a state machine or other similar software. The conversation manager may have access to customer, service, and

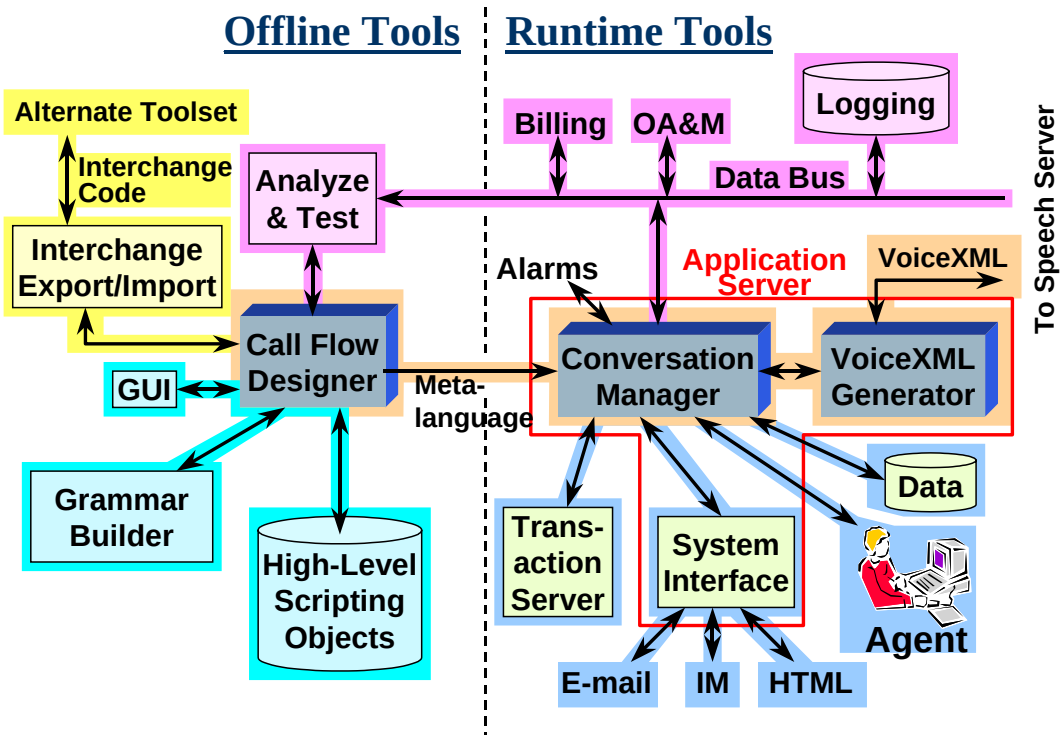


Figure 3. VoiceXML Tools Architecture Diagram.

other data. It may also access external systems such as e-mail, instant messaging, web pages, and even live agents when necessary.

An important characteristic of the service creation environment is the form of its output. While simple applications may be written directly in VoiceXML, many services (particularly complex services) are written in an intermediate form such as Java, C++, ASP, proprietary scripts, XML, etc., and are converted to VoiceXML by the application server. For our purposes, we call the intermediate software metacode and the language of this software is a metalanguage. Metacode created by development tools specifies the call flow or script (the sequence of events that occur during the call) and is used by the application server.

A related (and possibly identical) representation of the call flow and associated information is the interchange code, a common format used by tools built by different vendors to share application data. (We mention the interchange format only for completeness, it was previously a topic of discussion within the VoiceXML Tools Committee that has since been tabled.)

One feature of the application development tools section is service analysis and testing, where data collected from the application server and the browser during development, trials, and live service is used to improve future service quality. This runtime data, created primarily during operation, is an area for which few standards currently exist. The application server may generate runtime data related to caller actions, system parameters, or external information. This data, plus additional data received from the speech server, is stored in a logging database for use by billing, OAM&P (operations, administration, maintenance, and provisioning), service analysis, etc.

Runtime Data Logging

The focus of this document is in a standard format for Data Logging (the pink section in Figure 3). During testing and in a live service, runtime data is generated that provides date relative to quality of service, OAM&P, errors and exceptions, billing, and both individual and collective call traffic. The Data Logging specification describes runtime data and associated linkage information that is to be captured by each logging entity. In the future, it may also provide a data logging policy – a strategy for selecting subsets of data that are to be captured at a given moment.

Currently, data capture and storage/transport structure is not yet covered by industry standards, so each technology vendor uses a different approach for formatting the information. The intent of the Data Logging Working Group is to define a vendor-independent specification for such data. A few illustrative examples of data to be captured include:

- Conferencing a 3rd party
- Resetting a speech channel
- Telephony card failure
- Playback completed
- CPU idle percentage

- ASR version number
- TTS memory usage
- VoiceXML audio cache hits/misses
- Call duration
- VoiceXML session ID
- ANI
- Database response time
- Application-specific tags defined by the application

A successful runtime data standard will enable service providers to mix and match various development tools, application servers, and VoiceXML browsers from different vendors and still maintain consistent data logging. It will also allow interoperability between data analysis tools that help improve quality of service, support billing and maintenance systems, and provide real-time system monitoring. Complying with this specification will allow some or all of the sub-systems' logs to be aggregated into one consistent, well-structured log.

Consolidation

Data may be captured in each individual entity (speech server, etc.) or in a centralized data logging server. Either way, there needs to be a way to link data across systems so that it is possible to trace all activity related to a single call or group of calls. We recognize that entities pass messages and commands between each other and that these interactions need to be captured. Such linkages and messages are reconstructed from individual logs and logs from associated entities through use of interaction identifiers as described in SLAML.html.

Data Logging Working Group Charter

The Data Logging Working Group is chartered to define a methodology for collecting, storing, and retrieving runtime data. Runtime data includes information about the call, the platforms, the service, and human input and is generated by the application server and the VoiceXML browser. This data is used for billing, maintenance, service monitoring, and quality improvements.

Our intent is to define a comprehensive (to the extent possible) list of data elements to be captured, along with their units and meaning, and to specify a format for representing these parameters. In addition, we will define a style guide for new and/or special elements so that they may be added incrementally and safely without impacting the existing list of elements or the systems that use those elements. We do not specify how the data is analyzed; only that it is unambiguously readable and that it contains the necessary linkage between contained and containing subsystems.