Read my Feed – from RSS to SIP

Alessandro Falaschi[1], Emiliano Esposito[2]

[1,2]INFOCOM Dpt, Sapienza Universy of Roma, Italy

E-mail: [1]alef@infocom.uniroma1.it, [2]emiespo@tiscali.it

Abstract: **RRS web feeds are read by a synthetic voice, trough a SIP VoIP call, offering eye-free access to huge amount of classified textual content available on the Web. DTMF browsing allows to choose in between different RSS providers, and to cycle trough RRS titles, until the desired full article is selected, and read. The main content in the page is located by explicit parsing for known feeds, or by heuristic reasoning for new feeds and pages, which can be directly accessed by passing their URI as a SIP address parameter. Scalability is attained by caching of network and audio data instances, and expressivity improved by generation of SSML markup on the basis of the original HTML and CSS code. The whole system is made of a collection of Open Source components and public W3C standards, and the use of Festival for Speech Synthesis makes the service available for any supported language.**

Keywords: Voice over IP, SIP telephony, RSS, Voice XML, XSTL, Speech Synthesis, SSML, Multi Lingual

INTRODUCTION

RSS feeds [1] are with no doubt an XML-based pervasive format with meta-tagging capability, used by many news and blog publisher for indexing their content, allowing for easy retrieval and aggregation of past and recent additions. The idea behind our project is that RSS indexed content already contains all the necessary information for being accessed by other media delivery technologies, such as speech synthesis. Text to Speech conversion of retrieved text into voice is indeed an already existing commodity at the User Agent premise implemented by some PDA and cell phone [2], and server side solutions for automatic reading of stored information by IVR or CRM systems is absolutely not a novelty [3]. But in both cases, we are faced with closed, proprietary, and application-specific systems. On the converse, RSS is a container format ready to express quite any kind of web content, and a system entirely made of publicly available components allows for great flexibility and cooperative support among its potential users. Universality of the application domain is paired by full multi-lingual operation, thanks to the choice of Festival [4] as the Text to Speech engine; and the use of Asterisk as the media gateway, warrants the access to the system to the widest variety of user terminals, from PSTN to VoIP.

The rest of the work is organized as follows: first, a description of the components used, and of the way they communicate each other, is given. Then, a description of the interaction mode is given, and the mechanism by which RSS feeds and web pages are retrieved is discussed, together with the way they are converted in the audio brows-able VoiceXML format. Then, an insight about the way the speech process is operated is made, and the results of early experimentation given. Finally, some speculations about possible future uses and applications of the system are drawn.

SYSTEM ARCHITECTURE

Fig. 1 sketches how all the pieces (software entities, protocols, formats) fit together, and is the right place for listing them. All the components are released according to an Open License, and communication is performed by public standard and protocols.

On the left the VoIP PBX Asterisk [5] is shown, which terminates SIP and DTMF signaling generated by the user, and sends back the RTP streaming of media related to accessed content. But note that media should be not limited to audio – for instance, a virtual talking head could be added in the future.

Asterisk is under the control of the Voiceglue [6] Perl package, shown in the middle part of the figure, which adds to Asterisk the capability of correctly execute voice dialogs expressed by VoiceXML sheets. In particular, the Phoneglue sub-process uses the Asterisk Manager port for opening the FastAGI channels by which it is notified about the remote party interaction, and by which it requests the playback of audio files. Voiceglue itself is a separate process, written to be independent from the Asterisk Softswitch implementation, and which uses an ad-hoc crafted protocol called SATC (Simple ASCII Telephony Control) to communicate with PhoneGlue. Voiceglue bases its operation on some further components, such as

- the OpenVXI library [7], offering the services on which the VoiceXML browsing is based;
- the SSMLExtender module, expressly written by us, which adds support for the Speech Synthesis Markup Language (SSML) to the VoiceXML sheets dealt by VoiceGlue, offering a standard way of expressing prosodic variations.

In turn, the OpenVXI library used by VoiceGlue uses some more Open Source components, i.e.

- the Exerces XML parser, enabling OpenVXI to

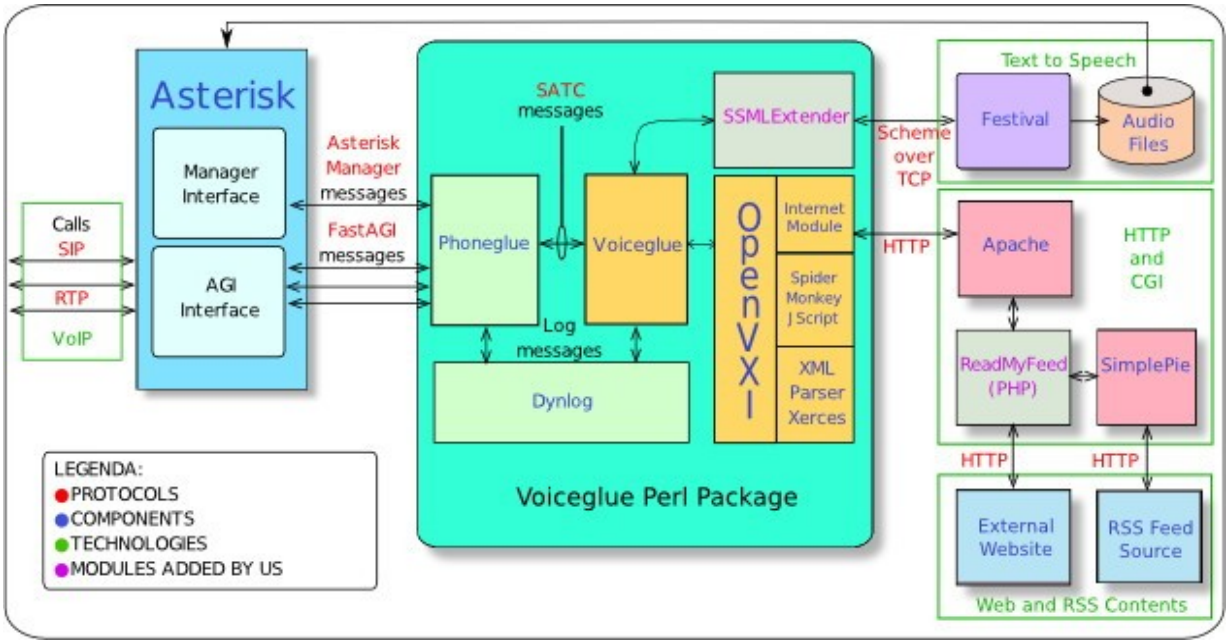

Figure 1: Overall System Architecture

interpreter VoiceXML sheets;

- the Spider Monkey JavaScript interpreter, as JavaScript is often used as an helper for audio navigation within VoiceXML sheets;
- the Internet module, used to fetch new VoiceXML pages from a web server.

At the right of Fig. 1 the source (RSS feed) and destination (Festival) of the retrieved content are shown, together with the ReadMyFeed PHP CGI (developed by us) which effectively builds up the VoiceXML sheets starting from RRS or HTML objects. Shown in the figure also are

- the SimplePie PHP component by which the RSS feeds are retrieved;
- the storage of audio files corresponding to synthesized prompts to be played by Asterisk when requested.

Not shown, is the three level cache made of

Figure 2: keys semantic assignment

- storage of already retrieved RSS feeds;
- storage of already retrieved HTML pages and their VoiceXML counterparts;
- audio files naming mechanism, which allows to synthesize recurrent phrases only once.

USER INTERFACE AND NAVIGATION

The structure of audio browsing is built by means of the expressive power granted by the VoiceXML [8] syntax, and our application is based on three chained VoiceXML sheet, i.e. one for the choice of a RSS channel, one for the choice of a title, and one for the playback of the selected content. As the user can interact only by DTMF signaling, we avoided as much as we can to build typical unfriendly patterns as "*Press one for this, two for that, three for the other, four for another one...*", so hard to keep in mind, and tried to maintain the semantic associated with the keys as stable as possible along the whole navigation history. In particular, key n. 5 always means "this", and selection of an item among a list of choices is performed by just "pressing five" after (or during) the utterance of the desired item: this behavior is obtained by building a new VoiceXML form for each item. For completeness Fig 2 shows a description of the general keys usage. Chaining and navigation in between the different VoiceXML sheets is made possible by sharing in between them the same *application root document*, which defines a common javascript namespace of session variables, and holds the forms to be executed as event handlers.

FROM RSS TO VOICEXML

Instead of having fixed, stored VoiceXML pages, they are generated on-the-fly, by execution of expressly written PHP code, named ReadMyFeed, and executed as a CGI by the web server Apache, after that Voiceglue has requested a new page.

The first VoiceXML sheet of the dialog is automatically generated by a first PHP script, starting from an XML configuration file, and implements a two-level decision tree by which an RSS channel is selected. A first menu allows to choose among some predefined channel genre (e.g. news, sports, blogs, whether..); then, a selection of known RSS channels is proposed, and one selected after the user input. Then, the retrieval of the titles associated to the requested channel is delegated to a second PHP script, which in turn passes the feed URI to the SimplePie [9] access filter, and formats its response by creating a new VoiceXML page. SimplePie supports a wide range of different (RSS and Atom) specifications, provides a convenient caching mechanism, and offers a nice set of parsing primitives. The resulting VoiceXML sheet is returned to VoiceGlue (and thus played to the user), allowing this time to cycle through the titles found in the feed, and choose one of them by pressing again the 5 key.

FROM HTML TO SSML

Selection of a title by the user, also identifies the *permalink* URI where the full article can be found, so that Voiceglue asks again our ReadMyFeed PHP script to generate a new VoiceXML sheet, containing the permalink URI content, to be used for its audio rendering to the user. This time the web page is retrieved directly, and needs substantial additional processing before of arriving to a useful representation, i.e.

1) if the page has been processed already, retrieve its VoiceXML counterpart from the cache, and go to step 7);
2) cleaning of badly-formed HTML elements;
3) parsing of the HTML page to a DOM representation;
4) location of the DOM node containing the main article content;
5) derivation of the SSML tagged text starting from HTML and CSS tags;
6) insertion of the SSML code within a new VoiceXML sheet and update of the VoiceXML cache;
7) return of the result to Voiceglue and then to the user.

Some insigth about the previous steps are given below.

Location of the main article contents

If the content provider of the *permalink* URI belongs to a set of known ones, step 4) is performed by using a set of hand-crafted rules which leverage on the peculiar page structure. But unknown sites should be dealt as well, because the user can bypass the first VoiceXML selection stage of the navigation, and directly request the reading of a particular RSS feed URI, as explained later. So, some heuristics have been added [10] for locating the main content for unknown channels.

XSLT conversion from HTML to SSML

Speech Synthesis Markup Language SSML [11] permits for addition of prosodic markers and cues to the text to be synthesized, allowing to enrich the resulting speech with variations representing intended semantic evidence. But HTML (and CSS) already contain such a semantic cues, which are inserted by the content creator for highlight some words with respect to others. So, in step 5) above we re-used an already available set of XSLT [12] rules, developed in the framework of the KTTS project [13], and use the PHP XSLTProcessor class for doing the transformation. The original *xhtml2ssml.xsl* rules where modified in order to fit well with the rest of the system. For instance, we started a new SSML <*prompt*> contest for many HTML tags, reducing in this way the speech rendering delay, as introduced by the execution pipeline described below.

SPEECH SYNTHESIS PROCESS

Once the VoiceXML sheets are returned to VoiceGlue, every prompt there contained must be converted to an audio signal to be played by Asterisk: here is it where the Text to Speech System Festival [4][14] plays its role. Festival is designed to be useful for conducting linguistic studies, as well as for ease the development of new synthetic voices for new languages, and for ease of adding TTS support to newly developed applications. Its behavior is almost completely programmable by a SCHEME [15] control language, whose statements can be communicated at run-time over a socket control channel.

SSML Extender Module

At this point, we developed some new Perl code which we called SSMLExtender, in order to make VoiceGlue capable of functioning with Festival in an efficient manner, and to make use of the SSML markup introduced into the VoiceXML sheets:

- while VoiceGlue originally spawned a new child process for every new utterance, we put a threshold on the maximum processing load, and created a queue based on semaphores, by letting an unique copy of Festival to run in server mode, controlling it by socket communications, and forcing it to serve no more than *N* concurrent requests;
- while VoiceGlue did not correctly honored SSML, trowing away most of the markup, we modified that behavior, so that SSML is now fully supported.

But these were not the only changes we took about Speech Synthesis. At first we tweaked a bit the Scheme configuration of the Italian voice, in order to improve the pronounce of foreign terms and Internet addresses. Then, we tackled the two following points.

XSLT conversion from SSML to SABLE

Festival do not currently understand SSML markup: instead, it honors the SSML predecessor named SABLE [16]. Therefore, we had to perform a new XML-to-XML conversion, by applying another set of XSLT rules for derivation of the SABLE markup which expresses the same variations indicated by the original SSML. Again, we started from the work done for the KTTS project [17], slightly modified for proper handling of the *language*

SSML attribute, and used it from within the SSMLExtender Perl module, by invocation of the XML::LibXSLT CPAN module [18] for performing the translation.

Audio Segmentation and Caching

The ReadMyFeed component splits a long VoiceXML sheet in a series of shorter prompts, thus avoiding an excessively long utterance to introduce great delay to the rendering of shorter ones, making the performances fair also in the case of concurrent processing, i.e. when different concurrent users access different, never requested pages. At the opposite, in order to simplify the audio prompts cache handling, and the subsequent playback through Asterisk, we concatenate all the prompts in the same VoiceXML page into an unique file, and perform an audio conversion of the result to PCM ulaw format. Finally, we compute an unique MD5 hash from the original VoiceXML page content, and use it as the name for the corresponding audio file, so that its synthesis can be avoided and the cached result used instead, when a new user asks again for the same content.

ON LINE TEST BED

The discussed system is operative and initialized with some representative feeds for news, sports, whether forecast and blog contexts. It can be reached by calling *sip:readmyfeed@ing.uniroma1.it*, and the content can be browsed and selected by DTMF signaling. Moreover, direct access to feeds (reachable or not by DTMF browsing), can be obtained by adding some optional parameters to the called SIP address, according to the general syntax

sip:readmyfeed@ing.uniroma1.it;parameters

where *parameters* can be the concatenation, separated by a semicolon (;), of some of the following elements:

- *feedurl=<URI>;* where <URI> is the RSS feed address. e.g. *feedurl=http://www.unita.it/drss.php*;
- *feedmax=<number>;* where <number> is how many titles should be listed, e.g *feedmax=5*;
- *readall=<boolean>;* where <boolean> determines if the page should be read entirely, or should be limited to the more relevant content, eg *readall=yes*.

RESULTS AND FUTURE TRENDS

Early tests shows that our audio user interface for RSS feeds browsing and access exhibits good usability and performances. Voice quality is acceptable, but the really wide lexicographical domain exposed by uncensored RRS channels, produced a moderate amount of mispronounced words, at least for Italian. This could be turned into an advantage, if feedback from the users is collected, and used as an help for enlargement of the actual pronounce exceptions dictionary. Next experiments could involve automatic language switching, either on the basis of explicit document metatagging, or by requesting a specific rendering language as a SIP address parameter, or by statistical text analysis. Finally, some experiments could be made about the mood selection for emotional speech synthesis, based on the analysis of the text.

References

[1] - RSS 2.0 Specs - http://www.rssboard.org/rss-specification
[2] - http://www.nuance.com/talks/
[3] - Loquendo VoxNauta Platform - http://www.loquendo.com/en/technology/voxnauta_platform.htm
[4] - Festival - http://www.cstr.ed.ac.uk/projects/festival/
[5] - Asterisk - http://www.asterisk.org/
[6] - VoiceGlue - http://www.voiceglue.org/
[7] - OpenVXI - http://sourceforge.net/projects/openvxi/
[8] - VoiceXML 2.1 - http://www.w3.org/TR/2007/REC-voicexml21-20070619/
[9] - SimplePie - http://simplepie.org/
[10] - http://w-shadow.com/blog/2008/01/25/extracting-the-main-content-from-a-webpage/
[11] - SSML - http://www.w3.org/TR/speech-synthesis/
[12] - XSLT - http://www.w3.org/TR/xslt
[13] - Gary Cramblitt - KDE Text-to-Speech System – file kdeaccessibility/kttsd/filters/xmltransformer/xhtml2ssml.xsl
[14] - Festival speaks Italian - http://www2.pd.istc.cnr.it/FESTIVAL/
[15] - SCHEME - http://groups.csail.mit.edu/mac/projects/scheme/
[16] - SABLE - http://www.bell-labs.com/project/tts/sable.html
[17] - Paul Giannaros - KDE Text-to-Speech System – file kdeaccessibility/kttsd/plugins/festivalint/SSMLtoSable.xsl
[18] - XML::LibXSLT - http://search.cpan.org/dist/XML-LibXSLT/LibXSLT.pm