REGULUS

Technical Report 2004-001

Writing Voice Navigable Formal Documents

Beth Ann Hockey, Kim Farrell, Manny Rayner,
Nikos Chatzichrisafis, Vladimir Tkachenko

Writing Voice Navigable Formal Documents

Beth Ann Hockey, Kim Farrell, Manny Rayner, Nikos Chatzichrisafis, Vladimir Tkachenko

Mail Stop T27A-2
NASA Ames Research Center
Moffett Field, CA 94035

{bahockey, kfarrell}@email.arc.nasa.gov,
{mrayner, nikos}@riacs.edu, vtkachenko@mail.arc.nasa.gov

Abstract

We describe how voice navigable versions of complex task-oriented documents can be written in an XML format that encodes extra information specifying how they are to be read by a voice browser. We focus in particular on the problem of creating XML representations that can be approved as voice versions of formal documents. The methods have been implemented in the context of a voice enabled procedure browser that will be deployed on the International Space Station late in 2004.

1. Introduction

There are many situations in which voice navigable documents would be useful, in particular when the user is visually impaired, or when the document describes a hands- and eyes-busy task that is to be executed concurrently. Early approaches to this problem focused on development of general voice browsers that could navigate through plain HTML, for example [1].

Experience rapidly showed, however, that task-oriented documents are difficult to read effectively without additional information explicitly related to the task described by the document. The task information is crucial. In reading out document, a human will typically say a lot of material in addition to what is written down; the source of this material is the human reader's understanding of the task. Additionally, some textual material may be paraphrased to make it sound natural in a spoken context. Without understanding what is being done with the document, it is not possible to know how to read it effectively.

The next wave of approaches consequently focused on this task information. Within the rapidly expanding voice industry, several standards have emerged: VoiceXML[2] permits direct definition of pure voice documents, and SALT[3] and X+V[4] of multi-modal documents. In the research community, there have been some fairly successful attempts to develop interactive multi-modal systems that can browse an abstract task description [5][6].

These second-wave approaches can be effective if it is feasible simply to replace a document with a task description. In many cases, however, this is not a realistic option. Here, we will describe work carried out under the Clarissa project[7][8], whose goal is to develop a useful voice browser for procedure documents used on the International Space Station (ISS). Early versions of Clarissa [7] used the RavenClaw dialogue manager [6], and replaced the target document with a RavenClaw task structure.

As the project matured and moved closer to being a fielded application (an initial field test is scheduled for Nov. 2004), the primary approach became increasingly strained. ISS procedures are critical formal documents that typically reflect hundreds or even thousands of person-hours of effort, and have gone through a lengthy approval process. It is not practically feasible to replace a document of this kind with a new structure; it is however equally impossible simply to read it out verbatim.

The rest of the paper describes the compromise solution we have developed to resolve this impasse. Clarissa documents are written in an XML format that contains all the text and layout information present in the original written procedure, together with additional information which specifies how the text is to be read out in the context of procedure execution. This XML is transformed automatically both into an HTML display document which will exactly mimic the appearance of the original paper document, and also into an annotated structure that can be followed by the dialogue manager and which will permit the text to be augmented and paraphrased where appropriate to enable it to be read aloud in a natural manner. A key point is that the XML encapsulates all of the procedure text verbatim as well as capturing the original formatting, making it reasonable to consider it to be an alternate form of the original document. This makes it possible for the XML versions of the procedures to be formally approved for use in the safety-critical environment of the ISS.

We give an overview of the Clarissa browser in Section 2, and describe the XML format and its interpretation in Section 3. Section 4 describes the design and approval process. Section 5 concludes.

2. The Clarissa procedure browser

Astronauts aboard the ISS spend a great deal of their time performing complex procedures. Crew members usually have to divide their attention between the task and a paper or PDF display of the procedure, or one crew member reads the procedure aloud, while the other performs the task. In either case, this is an extremely expensive use of astronaut time. The Clarissa Intelligent Procedure Assistant is designed to provide a cheaper alternative, whereby a voice-controlled system navigates through the procedure under the control of the astronaut performing the task [7][8]. The system functionality includes spoken dialogue control of navigation among procedure steps, coordinated display of the procedure text, ability to set alarms, recording and playback of voice notes, and a general mechanism for corrections and undos.

The system also includes several modes that are designed to address different tasks for which procedure documents are used. The system has a Challenge-Verify mode for careful procedure execution, when completion of every step needs to be tracked. Procedure documents are also used to skim through and refresh the user's memory of the steps. Clarissa's Terse mode supports this task by allowing the user to move through the procedure quickly, reading only part of what is written. While executing a procedure a user might also want to review previous steps, or preview upcoming steps. The review/preview mode supports this task by reading and navigating procedure steps without changing the current step or the record of completion of the procedure execution in progress. The use of conditional steps also supports the use of the same document for different tasks. Procedures can be written with a different set of steps to be used in different conditions. The same written document represents several different tasks, each using a different subset of the procedure steps.

For each procedure, the Clarissa system compiles both the HTML used for the display and a representation used for dialogue management from a single XML file. The dialogue manager is the core of the browser and is designed to be general in that it does functions and supports conversations applicable to procedures in general, such as navigation between steps, corrections, entering and exiting the various modes, recording alarms and voice notes, and querying the user for values. For some functions the visual display is designed to send the dialogue manager the same messages as would be derived from spoken input so the user's part of the conversation can be either speech or mouse clicks. All the procedure specific information is encoded in the XML. The browser uses the compiled XML procedures as data. This enables the ability to drop in an updated procedure without re-compiling the entire Clarissa system.

Clarissa currently handles five International Space Station procedures. These procedures are fairly elaborate; they average approximately 53 steps each and require an average of 980 lines of XML to represent them.

3. Voice navigable procedures

Our approach to voice navigable procedures requires the XML to encode the content and formatting of the original written procedure plus the material that needs to be spoken to perform the various tasks the document supports. The spoken version models what a human would read aloud while using the document to do a task. In some partmoocedures he uitten pd rpoken iorsionns t

[illegible]

of appropriate XML structures to encode them. Most of the important cases are illustrated in the example; discussion of them follows.

Invisible & inaudible: The spoken and visual versions of a procedure can differ for a particular item. In other cases, there could be material that should only be spoken or only be displayed. Attributes are used in the XML to mark material that should be displayed and not spoken (inaudible) or spoken and not displayed (invisible). All the value steps inside the table in step 3 of the example are invisible as are the questions to the user at the beginning of step 1. The Instruction in step 2 shows an inaudible-invisible pair.

Conditional text & speech: Some steps apply only when certain conditions are met. This structure supports selecting a subset of procedure steps to execute based on various conditions. In the example, Steps 1-3 are only read if at least one of the bags is being filled. The XML for step one has the tests for whether bag 1, bag 2 or neither are being filled enclosed in a conditional (<if>) structure. There is an ExecutePointer to step 4 in the case that neither bag is being filled. In this example all the conditional structures are invisible but this is not required. If a visible step is conditional, and its conditions are not met, the visual display grays it out, and it is not read.

Eliciting values: Written procedures have implicit and explicit conditions based on values that a human user will have access to, but a computer system may not. For example "If microbial sample bag is to be filled" in Figure 1. The corresponding conversation consists of two questions of the form "Is microbial sample bag X to be filled?" that elicit this information from the user. Values also need to be elicited conversationally if the results of measurements or readings need to be recorded, as in the table in Step 3 of the example. ValueStep is the XML structure designed to elicit values from the user. When a value is obtained from the user in a ValueStep, the user's response is assigned to a variable. The minimum and maximum approved values are tested by the system and the system is able to warn the user if a value is out of range.

Tables: In step 3, the written procedure has a table with row and column headings and several cells. The use of a table in executing a procedure is to collect the relevant values and record them in the appropriate cells. The XML for step 3 encodes, for each table cell, a question to the user eliciting the appropriate value. Each table cell question is part of an invisible ValueStep.

Comments on values: Step 3 gives additional information about a value beyond the min and max. The structures AddendumList and Addendum are used in ValueSteps to accomplish this. In step three of our example, the AddendumList tests for an increase relative to last week's measurement and then makes appropriate statements.

SpeechBefore: Supports material to be spoken in the introduction of a step, before the step number. In step 1 of the example SpeechBefore is used as a verbal marker of the scope of the initial "If microbial sample bag is to be filled", it is set to inform the user which steps apply to which bags.

Setting variables: Used to set variables needed in execution of steps. In the XML encoding of Step 1 of the example, ActionBefore sets the $info variable which needs to be spoken before the step text.

Labeling sections: Procedure text that applies to several steps is represented in written procedures by indentation and a vertical line. In the example, "If microbial sample bag is to be filled" is represented in the XML for display by LabeledSection and for the spoken version by a series of invisible value steps.

4. The design and approval process

The critical requirement when creating a voice version of a procedure is that the result should in a strong sense be the *same* document as the original. The procedure writing community at NASA is very focused on safety considerations, and any new version of a procedure has to go through a lengthy sign-off procedure. Extending this procedure to include voice documents has been one of the unexpected challenges of the Clarissa project.

During the design phase, our experience has been that it is unproductive to show the XML directly to the procedure writers, who generally do not have a background that makes it easy for them to work with this kind of format. Instead, we split the document into independent modules, and for each module write down scenarios exhaustively describing all possible ways in which the document module can be read out by the browser. Since the browser is interactive, these scenarios are in the form of interactive dialogues between the user and the system, like the one shown in Figure 2. The scenarios serve as specification documents in the interaction between procedure writers and voice system implementers, and are refined until they converge to a mutually agreed solution.

Using such written scenarios currently imposes practical restrictions on the types of structures that can be used in procedure documents, since overly general constructions are not amenable to exhaustive description in this form. Additionally, these scenarios are difficult to keep up to date when requirements and implementation details force changes. An interesting question for future research is how to develop ways to generate such scenario documents that both permit more expressive communications between document writers and speech developers and that are amenable to automation. Having a tool to automatically generate expected system dialogue for common scenarios would also benefit Quality Assurance teams who need to validate such spoken dialogue system code. For the Clarissa system, such validation is now done using manual testing, but this will not scale acceptably to a system that needs to handle thousands of possible procedure documents.

5. Summary and conclusions

We argue that the general case of voice navigation of documents is difficult because knowledge of how the document is being used is needed to understand what needs to be spoken. We have described the Clarissa system, and shown that it is possible to build a voice browser for an interesting and useful class of documents, using a single XML format that does not require knowledge of HTML or the inner workings of the dialogue manager.

An additional benefit is that different procedure writers would have different ideas on how to structure the dialog. Using a formal document description enforces uniformity in dialog style amongst documents.

6. Acknowledgements

We would like to thank Roger Lottridge, Mark Lipsky, and other members of the NASA Johnson Space Center Procedure Writing community. The greater part of this work was performed at the Research Institute for Advanced Computer Science (RIACS), under Cooperative Agreement NCC 2-1006.

7. References

[1] F. James, "AHA: audio HTML access", in *Computer Networks and ISDN Systems, Volume 29, Issue 8-13*, Sept. 1997

[2] VoiceXML Forum. http://www.voicexml.org, as of April 2004.

[3] SALT Forum., http://www.saltforum.org, as of April 2004.

[4] X+V, VoiceXML Forum. http://www.voicexml.org/specs/multimodal/x+v/12/, as of April 2004

[5] Bohus, D. and Rudnicky, A. "LARRI: A language-based maintenance and repair assistant". *IDS-2002*, Kloster Irsee, Germany.

[6] Bohus, D. and Rudnicky, A. "RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda". *Eurospeech-2003*, Geneva, Switzerland

[7] G. Aist, J. Dowding, B. Hockey, and J. Hieronymus, "An intelligent procedure assistant for astronaut training and support," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (demo track)*, Philadelphia, PA, 2002.

[8] M. Rayner, B. A. Hockey, J. Hieronymus, J. Dowding, and G. Aist, "An intelligent procedure assistant built using REGULUS 2 and ALTERF," in *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (demo track)*, Sapporo, Japan, 2003.

```
<LabeledSection display="If microbial sample bag is to be filled">
 <!-- ================      1st  STEP       ===================-->
  <Step> <Id>1</Id> <StepTitle>1</StepTitle>
   <StepContents>
     <ValueStep invisible="yes"><Id>1</Id>
       <Instructions>Is microbial sample bag one  to be filled?</Instructions>
       <Type>yesno</Type>
           <Var>$fillBag1</Var>
       </ValueStep><Id>2</Id>
     (... similar for bag 2 ...)
    <if invisible="yes" cond="($fillBag1 = no) and ($fillBag2 = no">
     <BasicStep invisible="yes"><Id>3</Id>
       <Instructions>Skipping steps 1 through 3</Instructions>
       <ExecutePointer>step 4</ExecutePointer>
     </BasicStep>
    </if>
    <BasicStep speechBefore="$intro"><Id>4</Id>
     <ActionsBefore>
       <SetVar invisible="yes" cond="($fillTBag1 = yes) and ($fillBag2 = no)">
         <Var>$intro</Var> <Value>"For test 1 only"</Value>
       </SetVar>
       ( ... similar for cases with bag 2 only and both bags ...)
              </ActionsBefore>
     <Instructions>Lock bag to hose</Instructions>
    </BasicStep>
   </StepContents>
 </Step>
 <!-- ===============     2nd  STEP      ====================-->
 <Step><Id>2</Id><StepTitle>2</StepTitle>
   <StepContents>
    <SubProcedureCallStep speechBefore="$intro"><Id>1</Id>
     <Instructions inaudible="yes">Perform {1.959  SAMPLE BAG FILL},
steps 6 to 17</Instructions>
     <Instructions invisible="yes">Perform procedure number 1.959, SAMPLE
BAG FILL, steps 6 to 17</Instructions>
     <ProcedureName>Sample Bag Fill procedure</ProcedureName>
     <ProcedureNumber>1</ProcedureNumber>
     <FromStep>6</FromStep>
     <ToStep>17</ToStep>
    </SubProcedureCallStep>
   </StepContents>
 </Step>
```

```
<!-- ==================      3rd  STEP     ====================-->
 <Step><Id>3</Id><StepTitle>3</StepTitle>
   <StepContents>
     <BasicStep speechBefore="$intro"><Id>1</Id>
       <Instructions>Measure  sample bag temperature</Instructions>
     </BasicStep>
   <!--============= Table =============-->
   <Table columns="3" rows="2">
     <TableRow>
       <TableHeader>TEST</TableHeader>
       <TableHeader>1</TableHeader>
       <TableHeader>2</TableHeader>
     </TableRow>
     <TableRow>
       <TableHeader>3. Temperature</TableHeader>
       <TableData>
         <if invisible="yes" cond="$filltBag1 = yes">
           <ValueStep invisible="yes"><Id>2</Id>
             <Instructions>What is the temperature for test bag1?</Instructions>
             <Type>temp</Type>
             <VarName>$bag1TempStep3</VarName>
             <Min>21</Min><Max>30</Max>
             <AddendumList>
               <Addendum cond="$bag1TempStep3 &gt; $bag1TempLastWk">
                 <AdTrueText>Increase from last week; nominal</AdTrueText>
                 <AdFalseText>no increase from last week;
                                off-nominal</AdFalseText>
               </Addendum>
             </AddendumList>
           </ValueStep>
         </if>
       </TableData>
       <TableData>
              (... similar for bag 2 ...)
       </TableData>
     </TableRow>
   </Table>
   <!--==============    END of Table     ==============→
   </StepContents>
 </Step>
</ClauseWithScope>
```

Figure 3:XML Representation for simplified procedure fragment