A FUNDAMENTAL ARCHITECTURE TO INTEGRATE CONVERSATION MANAGEMENT ENGINES WITH CONVERSATION DEVELOPMENT AND EVALUATION TOOLS

Emmett J. Coin, J. Qua

ejTalk Research
ejTalk.com

ABSTRACT

As the need for dialog design grows, developers will need to build more ambitious conversations more quickly. These conversations will be more complex, more numerous, and be subject to increasing economic pressure [3] to maintain and enhance them. Based on accepted data exchange standards, we propose a system that makes a sensible division of the conversation development [2], runtime and maintenance tasks, while also promoting the usability of autonomous, independent developments in conversation management.

1. INTRODUCTION

Building a conversational system, for even a small domain, is a very large task. There are two major problems that we address: 1) A major obstacle in the development process is understanding and quantifying what happens within a specific conversation as well as a statistical view of collections of conversations. This is critical since the spiral method of design requires that we find the biggest problem, try to fix it, and measure our fix to see how much better it is. Doing user trials is expensive and time consuming, so it is very advantageous to do as many design cycles as possible before the next trial is done. 2) Another obstacle is that current systems often weave together a tangled web of code that encompasses ASR, TTS, NLP, task schema, conversational strategies, knowledge stores, specific application code and more. This problem requires a "divide and conquer" solution. We will benefit if we divide the task into clear, distinct and isolated components based on specific disciplines.

Both of these issues are supported by an XML data exchange scheme: 1) Quantification and spiral methods benefit from viewing and analysis tools, but these tools are always secondary to the engine. Obviously the conversational system needs to converse before it can be measured. And, it is usually more interesting and intellectually rewarding to advance the engine technology. Changes to the engine lead to changes in the custom tools. Together these truths result in very few useful tools when they are needed. When transcript data is written as XML, the tools can be decoupled from the engine. New elements will not break the tools, yet simple presentation schemes (i.e. XSL) can quickly and easily provide new enriched views. 2) Using XML as a communication exchange between the isolated components will maintain clear boundaries between their distinct functionality. (Note: This is a data exchange paradigm, not an invocation paradigm. It will work well in a range of environments: procedural calls, COM, simple IP, etc.) It will also allow very specific, internal component information to be passed to the transcript with no impact on the system overall. Another powerful benefit is that components can communicate "suggestible" parameters to each other, even before the recipient of a suggestion has support for it.

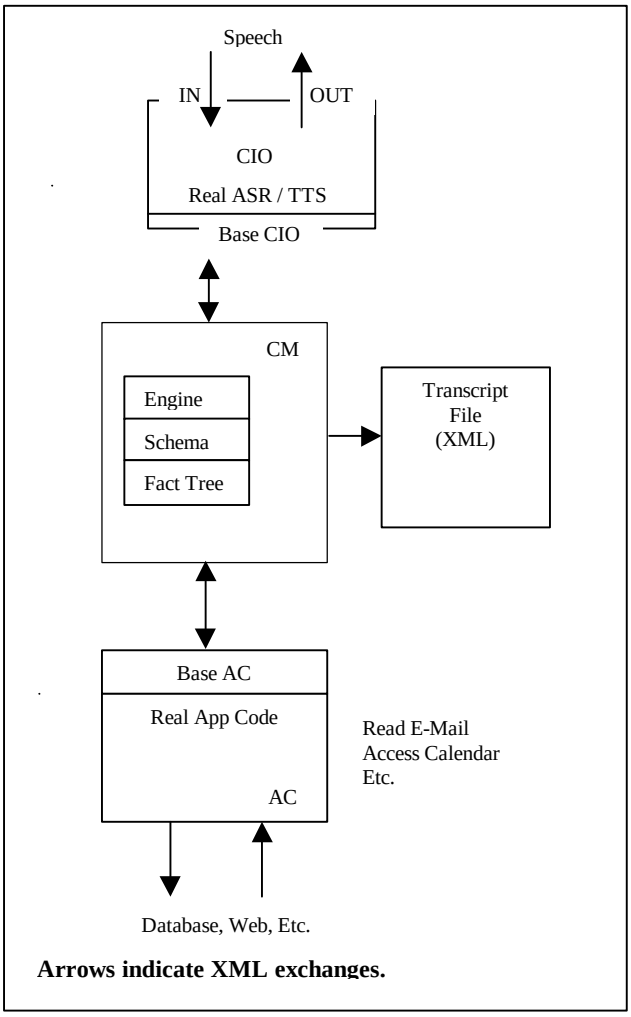

Figure 1. Block diagram illustrating the first tier components: Conversation I/O, Conversation Manager, and Application Code.

2. FIRST TIER COMPONENTS

These are the first tier components and the base XML exchanges that we identify:

2.1 Conversation I/O (CIO)

The CIO encapsulates the ASR and TTS components. We think of it as an autonomic speech center. Its job is to speak and listen to the human conversant. This component is independent, and can use different technologies to "listen" and "speak" (even different modalities as far afield as gesture, textin/textout, GUI, etc.)

The base exchange requirements are:

A request sent to the CIO --

```
<TO_CIO>
  <LISTEN>hello</LISTEN>
  <SAY> Hi, Robert here. Are you ready to
    get started?
  </SAY>
</TO_CIO>
```

A result returned from the CIO --

```
<FROM_CIO>
  <HEARD>WHY</HEARD>
</FROM_CIO>
```

2.2 Conversation Manager (CM)

The CM encapsulates the consciousness of the synthetic agent. It requests that the CIO speak and listen. It then expects the CIO to report what it heard. The CM has the job of understanding what the CIO heard, acting on the human's intentions, transitioning to an appropriate state, and advancing the conversation.

We obtain further benefit by dividing the CM into even smaller components, each of which is XML exchange based for their definition as well as inter-component communications (see Section 3). A reasonable segmentation is:

- ***Engine***
- ***Schema***
- ***Database***
- ***Log***

2.3 Application Code (AC)

The AC is the "muscle" of the system. It does something that is external to the conversation. The AC does some kind of real world activity that the CM "intends" to do.

It is "Real" code written by "Real" programmers. The application code communicates with the CM via an XML exchange.

A request sent to the AC --

```
<DO>
    <FUNCTION>LOGON_TO_EMAIL</FUNCTION>
    <INPUT>
        <ACCOUNT_NAME>
            emmett@ejTalk.com
        </ACCOUNT_NAME>
        <PASSWORD>
            ASRU99
        </PASSWORD>
    </INPUT>
</DO>
```

A result returned from the AC --

```
<DID>
    <FUNCTION>LOGON_TO_EMAIL</FUNCTION>
    <INPUT>
        <ACCOUNT_NAME>
            emmett@ejTalk.com
        </ACCOUNT_NAME>
        <PASSWORD>
            ASRU99
        </PASSWORD>
    </INPUT>
    <OUTPUT>
      <EMAIL_LOGON_SUCCESS>TRUE
      </EMAIL_LOGON_SUCCESS>
      <UNREAD_EMAIL_COUNT>7
      </UNREAD_EMAIL_COUNT>
    </OUTPUT>
</DID>
```

2.4 Tools

An infinite variety of tools are possible ranging from viewers to annotators to extractors.

Since all the components (and sub-components) are rooted in XML, it is easy to craft interactive, browser-based, visual tools to extract, examine, and display elements and relationships.

One of the first tools of interest is a transcript viewer: What happened in a particular conversation? It is a simple task using eight or so lines of XSL to view a basic transcript of:

```
…
MACHINE: Hello my name is Robert.
HUMAN: Hello.
MACHINE: What do you want to do?
HUMAN: Check my calendar.
…
```

Another line of XSL can show the current STEP name. And another line could make a direct link from that point in the transcript to the STEP viewer to immediately see that Schema definition.

One more line could show a debugging NOTE from the ASR (that the CIO inserted into the returned XML exchange before it returned to the CM). The CM engine would not need to be told it was there. And existing viewers would not break, although they would not display the NOTE.

We have only begun to explore the wide range of empowerment with this approach.

3. SECOND TIER COMPONENTS

Using the **ejTalk™** CM as an example, we can illustrate how a deeper layer of compartmentalization [4][1] benefits by the XML approach.

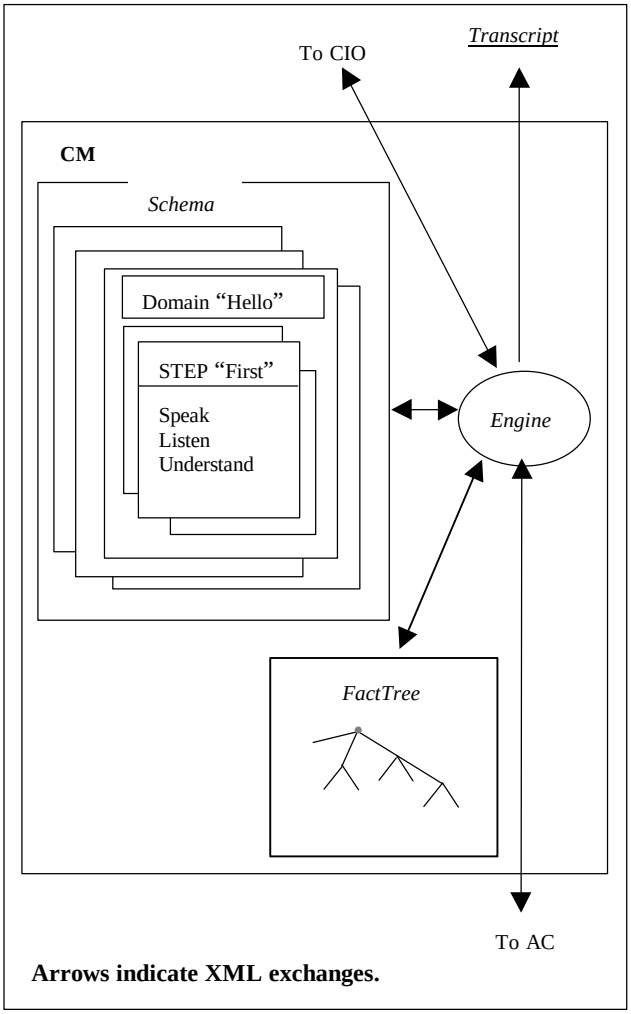

Figure 2. Overview of the **ejTalk**™ CM components.

3.1 Engine

The raw "brain" is a universal conversation machine waiting for a purpose (tabla rasa). It can communicate with the CIO, FactTree, Transcript, etc., but it doesn't know why. It is data driven via Schema definitions.

3.2 Schema

The "brain" software. It embodies the purpose, experience, preferences, goals, etc. Our engine uses an Object-Oriented DOMAIN/STEP state machine core. This methodology resolves a fundamental unit of conversation the FULL_STEP. A STEP includes**:**

- ***SPEAK - TTS generative grammar based utterance***
- ***LISTEN - ASR context, usually resolved at the DOMAIN level***
- ***UNDERSTAND - fits the ASR result to a semantic cluster in a STEP schema***
- ***ACT - sets Facts, generates transitional speech, loads any information destined to exit the CM, determines the next transition, etc.***

The XML representation of a simple **ejTalk**™ STEP: (Note: because of the OO methodology most STEPs would not contain all this explicitly, but would get much of its content implicitly from its derivation.

```
<STEP>
  <PROPERTIES>
    <FILE_NAME>first_entry</FILE_NAME>
    <PURPOSE>Start a conversation
    </PURPOSE>
    <AUTHOR> Emmett J. Coin</AUTHOR>
    <CREATION_DATE>
2/11/99</CREATION_DATE>
    <BASE_STEP>get_name</BASE_STEP>
  </PROPERTIES>
  <SPEAK>
    <S>
      <OR>
        <SYM>HELLO</SYM>
        <SYM>MY_NAME_IS</SYM>
        <PUNC>PERIOD</PUNC>
      </OR>
    </S>
    <HELLO>
      <OR>
        <TERM>Hi</TERM>
      </OR>
      <OR>
        <TERM>Hello</TERM>
      </OR>
    </HELLO>
    <MY_NAME_IS>
      <OR>
        <TERM>My name is Robert</TERM>
      </OR>
      <OR>
        <TERM>I am Robert</TERM>
      </OR>
    </MY_NAME_IS>
  </SPEAK>
  <UNDERSTAND>
    <YES>
      <SAY>
        <S>
          <OR>
            <SYM>OKAY_THEN</SYM>
            <PUNC>PERIOD</PUNC>
          </OR>
        </S>
        <OKAY_THEN>
          <OR>
            <SYM>OKAY</SYM>
          </OR>
          <OR>
            <SYM>OKAY</SYM>
            <TERM>then</TERM>
          </OR>
        </OKAY_THEN>
        <OKAY>
          <OR>
            <TERM>Okay</TERM>
```

```
    </OR>
          <OR>
            <TERM>Well</TERM>
          </OR>
        </OKAY>
      </SAY>
      <TO_STEP>get_name</TO_STEP>
    </YES>
    <HELLO>
      <SAY>
        <S>
          <OR>
            <TERM>It's a start</TERM>
            <PUNC>PERIOD</PUNC>
          </OR>
        </S>
      </SAY>
      <TO_STEP>get_name</TO_STEP>
    </HELLO>
  </UNDERSTAND>
</STEP>
```

3.3 FactTree

The FactTree defines a specific instance of the current Schema. It contains specific information about this synthetic agent as well as this specific conversation. In our model the FactTree persists and grows.

The Engine, Schema and FactTree can be thought of as layers. Each layer adds more color and texture to the synthetic agent.

As the name FactTree implies, it uses an XML document tree to store and retrieve data

3.4 Transcript

The transcript is a complete record of the happenings in one encounter between a human and the agent. It can contain as much detail as needed without concern for the burden on the analysis tools. An example for one FULL_TURN extracted from a transcript:

```
…
<FULL_TURN>
  <DOMAIN>hello</DOMAIN>
  <STEP>first_entry</STEP>
  <TIME>Sun Aug 01 00:04:03 1999</TIME>
  <STEP_COUNT>2</STEP_COUNT>
  <FROM_CMIO>
    <SAID>Why worry about reasons? Just do
      it. Hi there, anyone home?
    </SAID>
    <HEARD>WHY</HEARD>
    <UTT_RECORD>19990622_280443.wav
    </UTT_RECORD>
    <LAST_TTSLABEL>0</LAST_TTSLABEL>
  </FROM_CMIO>
  <UNDERSTAND>
    <FACTTREE>ESV_MEANING=[YES]</FACTTREE>
    <FACTTREE>YES=[YES_1]</FACTTREE>
    <FACTTREE>YES_1=YES</FACTTREE>
    <MEANING>YES</MEANING>
    <TO_STEP>get_name</TO_STEP>
    <TO_DOMAIN></TO_DOMAIN>
    <MEANING_DISCOVERED_IN>
      first_entry.step:
    </MEANING_DISCOVERED_IN>
  </UNDERSTAND>
  <TO_CMIO>
    <NOTE>
      STEP_INIT: Loading D:/vssdb/
      banterspec/Interface/domain/
      hello/get_name.step.
    </NOTE>
    <NOTE>
      STEP_INIT:Defer to base LISTEN.
    </NOTE>
    <LISTEN>hello</LISTEN>
    <SAY>Can you tell me your name?</SAY>
  </TO_CMIO>
  <TOTAL_TIME>88.89</TOTAL_TIME>
  <SEGMENT_TIME>43.25</SEGMENT_TIME>
</FULL_TURN>
…
```

4. SUMMARY

User competence in browser use is universal. XML is a simple, powerful, quickly mastered concept. The cognitive load for building tools and understanding how to use them is low.

All the CM, CIO, AC, Transcript, FactTree, Schema components use the same simple XML access methods. All the components can share data very easily.

The tool implementers can be drawn from the large pool of web developers.

We can all use more and better tools.

5. REFERENCES

[1] Brondsted T., Bo Nygarrd B., Olsen J. "The REWARD service creation environment, an overview" International Conference on Spoken Language Processing. Sydney, Dec 1998.

[2] Coin E. "Conversational Analysis:Tools" Second International Workshop on Human-Computer Conversation, Bellagio, Italy, July 1998

[3] Eckland W., Levin E., Pieraccini R. "User Modeling for Spoken Dialog System Evaluation" Proc of ASRU'97, Santa Barbara, Dec 1997, pages 80-87

[4] Coin E. "Managing Conversation: Bring a Human Touch to Speech Technology" CTI, July 1998, pages 115-118.