An Exploratory Case Study Using CBSP and Archium

Charles L. Chen, Danhua Shao, & Dewayne E. Perry

May 20, 2007
Introduction

Motivation

- How do you go from requirements to architecture?
- How do you capture the rationale for the architectural decisions that are made in this process?
- Two possible methods - CBSP and Archium

Background

- Evolving the CLC-4-TTS Suite
- Adding speech property support for FreeTTS
- First attempt fell short - problem with race conditions
- Very general idea of how to approach the problem; no specific designs for a solution
The CBSP Approach

⇒ Helps architects bridge the gap between requirements and architecture

⇒ Evaluate the relevance of a requirement along the 6 CBSP dimensions
   ⇒ Component, Bus, System, Component Property, Bus Property, System Property

⇒ Refine requirements into architecturally friendly CBSP artifacts
Using CBSP

Requirements

R1: Speech properties must be configurable for FreeTTS.
R2: Users must be able to interact with the system at all times.
R3: Speech may not have any long pauses.
R4: Equal priority messages should be spoken in the order they were received.
R5: High priority messages must be able preempt lower priority messages; preempted messages do not have to be saved.
R6: Speech properties from one message may not interfere with those from another.
Using CBSP

→ Refinement into CBSP artifacts

- R1 - R1_C: Set Properties interface to FreeTTS
- R2 - SP: Suggests multithreading
- R3 - Eliminated: Definition of “long pause” + queuing system
- R4 - R4_B: Queue for messages
- R5 - R5_C: Queue Manager
- R6 - BP: Suggests using a queue that can handle messages and associated properties
CBSP

→ **Strengths**
  - Structured process of going from requirements to CBSP artifacts
  - CBSP artifacts can be traced back to requirements

→ **Weaknesses**
  - Evaluations for trade-off choices focus on choosing an architectural style after having derived the CBSP artifacts, but not on deriving the CBSP artifacts

→ **Comments**
  - Rejecting the JSML Generator - had the idea, unsure where to put the rationale for rejecting it in the CBSP approach themselves
The Archium Approach

Architectural design decision includes:
- Rationale: why the change is made
- Design rules: what should be followed
- Design constraints: what should NOT be allowed
- Additional requirements: new requirements resulting from the change

Solution includes:
- Description
- Design rules
- Design constraints
- Consequences
- Pros
- Cons
Using Archium

➡ Problem
   ✧ The current interface to FreeTTS does not allow speech properties to be set

➡ Motivation
   ✧ The result of this is that Linux and Mac users are unable to experience the CSS speech property support being introduced

➡ Cause
   ✧ Speech property support has not been implemented yet

➡ Context
   ✧ Evolving the existing CLC-4-TTS Suite
Using Archium

→ Potential Solution #1: JSML Generator

- **Description:** Use JSML to encode the properties into a string along with the message. Pass the entire thing into FreeTTS.
- **Design rules:** All generated strings must be well formed JSML strings.
- **Design constraints:** Message needs to be put within tags that contain the properties; therefore messages and associated properties should be delivered at the same time.
- **Consequences:** CLC-4-TTS Suite is dependent on FreeTTS supporting JSML

→ Pros:
- + Easy to code (similar system exists for SAPI 5 already)
- + FreeTTS manages the queue
- + Easy to force FreeTTS to empty queue (for prioritization)

→ Cons:
- - FreeTTS does not yet support JSML; significant wait time expected as the FreeTTS project appears to be in hiatus (last update was in February 2005).
Using Archium

Potential Solution #2: Queue System

- **Description:** Create a queue system that will set the speech properties for FreeTTS, pass FreeTTS a message to be spoken, and then wait until it is ready for a new message with a different set of speech properties.
- **Design rules:** Must keep track of messages and associated speech properties
- **Design constraints:** Queue must not interfere with users' ability to interact with the system as whole; blocking is only to block the speech portion but nothing else.
- **Consequences:** CLC-4-TTS Suite is dependent on Java FreeTTS allowing the setting of speech properties.

**Pros:**

- Can be implemented immediately as Java FreeTTS already allows for the setting of speech properties.

**Cons:**

- Far more difficult than using a JSML generator
Archium

➡️ **Strengths**

- Alternate solutions and the reasons for choosing one solution over another are explicitly captured
- Pros and cons of a solution are documented as part of the solution

➡️ **Weaknesses**

- No real help given on thinking up the potential solutions

➡️ **Comments**

- Had the benefit of knowing the problem - not sure how easy Archium would have been to use otherwise since there is no guidance in arriving at potential solutions given the Problem, Motivation, Cause, and Context
Results & Conclusions

→ Solutions
  - Similar solutions with both methods
  - May be an artifact of both methods being used by the same person

→ Strengths and Weaknesses
  - CBSP and Archium each had their respective strengths and weaknesses

→ CBSP and Archium are complementary rather than competing
  - Structured process of generating architecture from requirements with CBSP
  - Capture of alternatives and rationale with Archium