Multimodal Framework Proposal

Ellis K Cave
Chief Scientist
Intervoice, Inc
October 5, 2007

The Multimodal Interaction Working Group's document "Multimodal Architecture and Interfaces" presents a powerful and flexible framework for building and running multimodal applications. In this document, Intervoice presents some simple use cases that could highlight areas in the specification which could benefit from additional functionality.

Concurrent Scriptable Functions

In the run-time MMI framework, the assumption is made that the Modality Components as well as the Interaction Manager are script-controlled entities. Modality Components are to be treated as a "Black Boxes" where the actual functionality of these components are hidden, and controlled by a script executed internally to the box. Another assumption is that when an MC is invoked, it will execute a script that involves a set of actions that will interact with a user.

In the MMI Framework, the functionality of a Modality Component can be invoked by the Interaction Framework, but the interaction level is fairly high, as can be seen from the list of Runtime Framework commands: NewContextRequest, Prepare, Start, Pause, Resume ClearContext, etc.. The high level of these commands purposely avoids defining modality-specific commands, thus not putting constraints on what level of scripting functionality resides in the Modality Components themselves.

However, there seem to be some instances when more modality-specific functions could be appropriately exposed and used by the framework. I will present a use case to demonstrate.

Lets take the example of a multimodal application which uses keypad and screen (stylus) input, with audio(speech) and screen output. The application is multimodal voicemail, and the user is currently browsing their voicemail messages. The user decides that he wants to skip ahead 10 seconds in the currently playing audio voice message. For this example, we will assume that all control of the audio stream resides in the Audio Modality Component, which is a voice browser running VXML V3, with the proposed media control capabilities.

The application's UI designer wants the application to behave the same whether the user touches the skip-ahead icon on the device's screen with his stylus, or presses a DTMF key on their phone (which was defined as the skip-ahead function in the application). The UI designer knows that there is a possibility that the user may not have a touch screen on his device. If the user chooses to press a DTMF key on the phone for the skip function,

the event would presumably get passed directly up from the DTMF detector within the Voice Browser Modality Component (or in the attached speech server), and be handled by the an event handler within the Voice Browser. All of the event action would be contained within the Voice Browser MC, and no events external to the VXML MC would be exposed.

However, if the skip-ahead event originates from the GUI Modality Component (MC), we face some interesting choices on how to handle the event. In the current proposal, all events from Modality Components go to the Interaction Manager (IM). So, the skip-ahead event gets passed to the IM. Logically, the IM needs to then re-route the event to the VXML MC, so that component can effect the skip-ahead audio function.

But wait! There is no skip-ahead function defined in the command set for the MC. We have Prepare, Start, Pause, Resume, etc. commands, but no Skip-ahead command in the MC. We can only invoke scripts. So, we need to write a skip-ahead script in VXML, so the user can invoke the skip-ahead function from the GUI Modality Component. Then, we will need to load the script into the VXML MC using the prepare function, so it will be ready for when the user needs it.

But wait! The application designer doesn't want to STOP the currently-running VXML script to run the skip-ahead-10 seconds-audio script from the GUI MC. The designer just wants the audio to skip-ahead 10 seconds, but keep the current voicemail dialog script document running. Keep playing the audio. Keep the speech recognizer running. Keep the DTMF detectors running. Keep all the same grammars active. Keep the volume the same. Just move the position of the current play pointer in the Prompt Queue ahead 10 seconds in the VXML MC's Prompt Queue resource, and keep everything else the same.

In the current architecture, if one wants to run a new script on an MC, one must Cancel the previously-running script, and save all the current dialog context (I assume the context is passed up to the IM in the Cancel Response event?). Then one must preload the new script, START the new script, which performs the "skip-ahead-10-seconds" action.

Once we have skipped ahead 10-seconds, we should get the DONE event back from the VXML MC, as it has finished running the Skip script. We then need to somehow restore the previous state of the original script (by passing the context down using the START event?). This process seems to imply rebuilding the previous prompt queue and all of it's associated state, which could be a daunting task. It looks like it will be to to the IM to do all of this context-saving, queue rebuilding, and restoring, as far as I can tell. It could mean quite a blizzard of events between the IM and the MC to get this all accomplished.

But wait! The user clicked the skip-ahead-10-seconds icon three times in quick succession. So, we need to go through this Start, Cancel, Save Context, Start, Done, Restore Context, and Start again sequence, three times, in quick succession.

For that matter, we will probably need a skip-back script, a volume-up script, a volume-down script, a speed-up-audio script, slow-down-audio script, a go-to-the previous-

prompt script, a go-to-the-main-menu-script, a get-an-operator-on-the-line script, and many more scripts for the various asynchronous audio and voice dialog functions that the designer may want to be a single event, invoked from a GUI MC, during application execution. Of course, one could probably combine all of these various asynchronous VUI functions into one big script with a selection parameter, but we still be starting and stopping scripts quite a lot, and the scripts could get quite big.

UI experts know that users rely on immediate feedback when invoking real-time control events such as skip-ahead, volume-up, slow-down, etc. If there is significant delay in the execution of the requested function (typically greater than 100-200 ms), users will over-compensate the specific adjustment by invoking additional actions (e.g. re-pressing the volume-up key), usually requiring corrections which will also be overcompensated. Response time is important on most common user asynchronous actions, and the currently-proposed complex stop-restart process probably isn't conducive to low-latency response performance required in most real-time user interactions.

The straightforward approach to a solution to this issue would seem to suggest a method that allows events external to an MC to directly invoke specific modality functions within the MC. So somehow the IM could send an event to the VXML MC that would allow the skip-ahead function to execute in parallel with whatever else is going on in the voicemail application dialog.

Unfortunately, this approach violates the stated design goal of encapsulation, as now other entities in the system would have to have some knowledge of the inherent primitive functions of a specific MC (such as skip-ahead-10-seconds), so that these functions could be directly invoked. We need a better plan.

To keep the encapsulation goal intact, we should have an architecture where scriptable functions can be loaded and invoked in the MC in a more flexible manner than the current strictly-sequential LOAD/DONE process. For example, if one could load both a normal voice dialog script, and a skip-ahead-10-seconds script, and trigger either script at any time, we could relieve the complex process of script stop/starting and context save/restoring. Of course, the developer would have the responsibility to deal with any resource conflicts that may occur when multiple MC scripts are running concurrently in the same MC, on the same set of resources. Surprisingly, in most real use cases, this doesn't seem to be a problem. Typically the concurrently-running scripts are dealing with very different functionalities.

Perhaps a new "basic design goal" should be added to the Overview (section 2) in the Architecture & Interfaces doc:

1) Concurrency – The architecture should allow for multiple instances of scripts to be loaded and executed in the Modality Components concurrently. This assumes the capability to load and prepare multiple scripts simultaneously or randomly within an application process, and then invoke those scripts asynchronously at any time, even when other scripts are concurrently running on the same MC.

There are many other use cases that benefit from this concurrent execution model. This concurrent functionality opens up a wide range of applications that were either not possible, or extremely complex to implement, with the current single-threaded MC model. I can present more of these use cases at the Japan MMI workshop, if it is felt that this proposal is worthy of consideration.

Ellis K Cave