Winthorpe – Let Your Applications Listen… And Talk

Contributors: Krisztian Litkey <krisztian.litkey@intel.com >, Jaska Uimonen <jaska.uimonen@intel.com>

Background

- Computing capacity of server/desktop/mobile CPUs still rapidly increasing
- Speech recognition has been improving and becoming increasingly viable
- Has been making inroads to a range of devices most of us interact with on a daily basis
- Dominant mobile platforms now all have their voice enabled personal assistants
  - iOS has Siri
  - Android has Google Now
  - Windows Phone has Cortana
- Many entertainment platforms include speech recognition capabilities
  - Both of the dominan next-gen console brands support voice control
  - Many of the latest Smart TVs support voice based navigation and search

Background

- Devices are shrinking, getting more powerful and intelligent
- Local computing capacity expected to keep increasing
- Online/cloud computation capacity expected to keep increasing
- Network quality and capacity of connected devices expected to increase
- => Both server based and offline speech recognition are subject to speed and accuracy improvements
- => Voice based interaction – IOW speech recognition for command and control, and synthesis for response and feedback – is becoming increasingly viable and important
- => It is becoming an additional interaction mechanism with the user in addition to the traditional keyboard/mouse/touchscreen

Background

- This opens up the possibility to run a whole range of new services on devices where this was not possible before either
  - Due to physical device limitations (eg. wrist watches, wearables), or
  - Because of safety and regulatory reasons (eg. cars and other vehicles)
- Voice based interaction not only feels more natural, when it works properly, it is usually less distractive and hence also safer

Silo Approach To Speech Recognition

- Traditionally platforms have usually taken a silo approach in speech enabling applications
- A selected few applications, the ones considered the most important by the platform vendor are integrated with a full stack of speech libraries
- The speech enabled applications are unaware of each other and when active they fully monopolize the speech capabilities of the system
- More often than not, the user needs to manually enable and disable speech control from the UI and can only interact with one application at a time

Dedicated Speech Application Approach

- Another approach, taken by Apple for its implementation of Siri
- It turns the silo approach in a way upside down
- Instead of making a few selected key applications speech aware, a dedicated speech assistant application is introduced to the platform
- The speech assistant application is enabled to control and interact with a few selected key applications
- Although it otherwise works nicely, when a new application is added to the system, and especially when a third-party application is added, the application cannot be easily speech enabled by bringing it under the control of the speech assistant
- The speech assistant needs to be modified to be aware of the new application

What Is Winthorpe

- A platform service for speech recognition and speech synthesis with a few main goals
- To provide a framework for speech enabling applications
- Keep speech enabling simple applications simple and straightforward
- Allow complex speech enabled applications to be implemented, too
- Let developers focus their full attention on how to improve the usability of their applications by utilizing the new speech based interaction mechanisms
- Offload the burden of the actual implementation of recognition and synthesis from the applications
- Avoid forcing either the silo or the dedicated speech application approach, allowing either or both to be used if one wishes to do so

Winthorpe Approach

- Clients declare their command set upon registration to Winthorpe
- Winthorpe notifies clients about recognized commands
- Clients request synthesizing a message (in a given language using a given voice)
- Winthorpe delivers notification events about the progress of synthesizing to the clients
- Winthorpe does not contain a speech recognition or synthesis engine
- Instead Winthorpe provides interfaces for plugging in existing speech recognition and synthesis engines
- Winthorpe comes with existing plugins for utilizing some of the existing OSS recognition and synthesizer engines
- Winthorpe can utilize Murphy, the resource policy framework in Tizen IVI, to arbitrate between speech enabled applications that are otherwise unaware of each other

Idealistic (And Maybe Naive) Long-Term Goals

- Build step-by-step a speech enabling framework for applications
- Use this framework to build a voice enabled personal assistant that
  - Is context/system state aware
  - Offers offline (local) speech recognition services
  - Can bridge to cloud speech services if necessary

Winthorpe Architecture

- [add an architectural picture here]

Winthorpe Architecture

- Winthorpe consists of a core daemon and a set of plugins
- The core provides only a minimal set of functionality
- The functionality can be extended by loading plugins to the daemon
- The core alone can't do speech recognition or synthesis
- The core alone can't communicate with clients outside of Winthorpe
- So a number of plugins must be loaded to achieve a reasonable working configuration

Functionality Provided By Winthorpe Core

- An IPC-neutral client API
- A speech recognition backend API
- A disambiguation API
- A speech synthesizer backend API
- A simple plugin infrastructure
- A simple infrastructure for handling configuration
- An event loop/mainloop for handling I/O, timers, etc.
- A logging and debugging infrastructure

Current Plugins Provided By Winthorpe

- Client API bindings for unix-domain and TCP/IP sockets
- Client API bindings for D-Bus
- Speech recognition backend based on CMU pocketsphinx
- A trivial disambiguation plugin
- A speech synthesis backend based on espeak/flite
- A speech synthesis backend based on festival
- A demo speech control plugin for MPRIS2-compliant media players
- A demo plugin for controlling an iPhone media player and delivering speech queries to Siri
- A demo plugin to execute simple speech queries using a browser and a search engine
- A demo plugin to provide speech control and synthesis support for the sample webruntime based media player in Tizen IVI

Client API

- Provide the basic interface clients use to interact with WInthorpe
- Core client API is fully IPC agnostic
- Consists of a set of functions clients use to request services from the daemon plus a set of callbacks the daemon uses to deliver events and notifications to clients
- Can be used to implement clients directly as Winthorpe plugins
- This is seldom desirable, as one usually wants an external process to act as a Winthorpe client
- This is usually accomplished using a plugin that binds the core client API to a suitable IPC mechanism and bridges between the external client and the core daemon
- Two such plugins already exists, one for unix-domain and TCP/IP sockets and another for D-Bus
- Internal and external clients look pretty much the same to the daemon

Client API / Registration

- During registration the client provides its
  - name: used in diagnostic messages
  - class: used for resource allocation through Murphy
  - set of commands to be recognized
- The class is the Murphy application/resource class the application belongs to from the speech recognition and synthesis point of view
- The class affects what policies Murphy applies to the application and consequently it largely determines under which conditions speech recognition and synthesis can be active for the application
- Deregistering an application removes all related commands and resources

Client API / Command Declaration

- The syntax for declaring a command is simply the sequence of tokens the command is made up from
- However, there a few tokens that Winthorpe treats specially:
    - Dictionary switch: allows one to switch the dictionary used by the recognizer backend at an appropriate point within the command
    - Wildcard (*): indicates that the remaining words/tokens in the spoken command sequence (utterance) should be collected and treated as part of the command
- Dictionary manipulation can be used to improve recognition accuracy by temporarily switching to a reduced dictionary after a unique prefix of the command has been recognized.
- A typical example is to switch to a digit-only dictionary when dialing a number by voice commands.

Client API / Command Declaration

- There are special tokens for both pushing and popping dictionaries to and from the active dictionary stack
- Once a command is fully recognized, or if recognition fails, Winthorpe automatically pops the stack and switches back to the originally active dictionary
- In the vast majority of cases there is just one dictionary switch necessary per command, so switching happens by including the push at the appropriate place and the pop is omitted altogether
- Deregistering an application removes all related commands and resources
- The wildcard token is useful for declaring commands that need to collect free-form text for further processing. For instance the demo search plugin uses this to collect the query which is then passed on to the search engine

Client API / Focus Handling

- Clients use voice focus requests to enable and disable their command set for recognition
- When a client gains (shared or exclusive) voice focus its command set is active
- When a client has no focus its command set is inactive
- Exclusive focus means, in principle, that no other client has focus at the same time
- Winthorpe uses the Murphy resource API to turn focus requests to respective Murphy resource requests (using an abstract Murphy resource dedicated to speech)
- The actual decision whom to give focus to, whether to grant exclusive or shared focus, and whom to withdraw focus from is actually done in and by Murphy according to the Murphy policy ruleset in use
- This allows altering/adjusting/adapting the behavior of the speech subsystem without touching the applications themselves

Client API / Focus Handling

- Focus state changes have identical semantics to Murphy resource state changes
- When the voice focus of a client changes, Winthorpe calls the focus notification callback of the client
- Focus state can change as a result of the client requesting focus state change
- Focus state can also change as a result of Murphy temporarily withdrawing and later reassigning voice focus to the client. This can happen because of other speech enabled applications activate / deactivate themselves or because of other changes in the overall system state, for instance a phone call getting accepted and activated.
- When requesting focus (shared or exclusive) clients should not assume anything about their focus state before they have received and actual focus state change notification
- When giving up focus (IOW changing focus state to none) clients do not need to wait for a notification, giving up focus always succeeds

Client API / Command Notifications

- When a client has voice focus, Winthorpe will deliver a command notification whenever a command registered by the client is recognized
- The notification consists of
  - The ID (index) of the recognized command
  - The tokens that make up the full command
  - Buffer of audio samples for the command
  - Location of each token within the sample buffer
- With this setup it is possible to split up the audio sample buffer of the command and do some post-processing on it, such as cut off the trailing part of the sample buffer and forward it for processing to an external service for further recognition

Client API / Command Notifications

- Currently the client API IPC bindings omit the audio buffer from notifications, so if you need the audio data, you have to implement your client directly as a Winthorpe plugin and use the core client API

Client API / Synthesis

- The synthesis part of the client API provides interfaces for
  - Querying all available voices or voices for a given langauge
  - Synthesizing a given message using a given voice
- Voice queries return the matching voices together with information about
  - Language and dialect,
  - Gender and age,
  - Name and verbose voice description
- The client can later use this information to select a particular voice it prefers to render a message with instead of using the default voice for the language used

Client API / Synthesis

- To render a message the client uses the synthesis request which includes
  - The message to render
  - The name of the voice to use for rendering
  - Rate and pitch to use for the synthesized message
  - Timeout and an mask of progress events to notify the client about
- The resulting synthesis ID can be used later to cancel the message at will and also to correlate progress event notifications with the request
- Winthorpe will synthesize the message and coordinate with the system policy, Murphy, to play out the message as soon as possible.

Client API / Synthesis

- The overall state of the system and other applications might prevent synthesis from taking place immediately
- In this case Winthorpe will queue the request for later execution
- The client can use the message timeout to control how long the request can be queued before it is automatically cancelled in case it cannot be executed in time
- The notification mask and callbacks can be used to monitor how the synthesis of the message is progressing
- This can be used, for instance, to synchronize the state of the UI of the application with the state of the synthesis

Client API / Synthesis

- Events are available for starting, completion, abortion, timeout, and progress of the synthesis
- Currently progress events are always delivered as percentages and milliseconds from the beginning of the message
- We are working on adding the more useful and reasonable character and word offsets to the progress events

Speech Recognition Backend API

- Instead of having a fixed built-in speech recognition engine, Winthorpe provides the necessary mechanisms for plugging in existing engines as backends
- The plugin infrastructure makes the backends pluggable while an abstract backend API provides the necessary infrastructure that allows one to adapt and dynamically register the recognition engine with the daemon
- The backend API consists of a set of functions Winthorpe expects every backend implementation to provide plus another set of functions Winthorpe provides for the backend
- The former set is used by Winthorpe to configure and control the backend while the latter is used by the backend to propagate recognition events to Winthorpe

Speech Recognition Backend API

- The functions the backend provides are used for
  - Verifying the usability of a decoder (dictionary)
  - Selecting/retrieving the active decoder to be used for recognition
  - Activating and deactivating the recognizer
  - Flushing/resetting the audio sample buffer
  - Rescanning a given part of the sample buffer
  - Copying a part of the sample buffer
- The concept of a decoder in this context is a backend specific combination of a language model, a dictionary, and potentially some other data which can be used by the backend to recognize a predetermined set of text in a particular language

Speech Recognition Backend API

- Upon completion of recognizing an utterance (a silence terminated sequence of speech) the backend uses the notification callback provided by Winthorpe to deliver a recognition event
- In the recognition event the backend passes to the core a set of utterance candidates
- Each candidate consist of a
  - Score (probability of correctness)
  - Length in the audio sample buffer
  - Set of recognizer tokens

Speech Recognition Backend API

- Each recognizer token in turn consists of
  - The recognized word
  - Its score (probability of correctness)
  - Location within the sample buffer
- The backend sorts an passes the utterance candidates to the core in decreasing probability of correctness to speed up disambiguation

Disambiguation API

- Winthorpe runs utterance candidates in a recognition event through the disambiguator
- The disambiguator may select one of these for further processing or it may also reject them all
- If one was selected the client which registered the mathing command receives a command notification
- The disambiguator may also decide to switch to a new decoder and reprocess part of the audio buffer with the new one
- This is how client commands involving a push are actually processed

Disambiguation API

- Similary to recognizer backends, disambiguators are also pluggable in Winthorpe
- The disambiguator API has three functions
  - Register a set of client commands to the disambiguator
  - Unregister a set of client commands from the disambiguator
  - perform disambiguation on a given set of utterance candidates
- The first two are used by the core to keep the disambiguator up-to-date about the active set of commands as clients come and go
- Currently voice focus state changes are not propagated to the disambiguator

Disambiguation API

- Winthorpe provides one trivial disambiguator which simply picks the first (most probable utterance candidate)
- It is a very good question whether the disambiguator should really exist as separate pluggable entity
- Perhaps a better alternative would be to fold this functionality into the recognizer backend (and also notify the backend about voice focus changes so that it could adjust its decoder accordingly if it is able to)
- We are experimenting with changes along those lines to see if it works better

Synthesizer Backend API

- As with recognizer engines, instead of having a fixed built-in one, Winthorpe allows plugging in exisiting engines as backends
- It defines an abstract synthesizer API which it expects every backend to implement
- The API has functions for
  - Registering/unregistering a set of voices to/from Winthorpe
  - Requesting the rendering of a given message with a given voice
  - Cancelling an ongoing rendering request
- Additionally Winthorpe provides a callback for the backend which this uses to deliver rendering progress notification events to the core which delivers them to clients

Plugin Infrastructure

- Winthorpe provides a simple plugin mechamis for loading components dynamically to the daemon
- Plugins in Winthorpe are used for
  - plugging in recognizers, synthesizers and disambiguators
  - providing IPC bindings to the client API
  - implementing (usually somewhat special) clients
  - or to extend the functionality of the daemon in some other ways
- Once a plugin is loaded it has full access to all the internal APIs of Winthorpe
- So plugins are able to perform all the actions the rest of the daemon is able to do

Plugin Infrastructure

- Winthorpe defines an abstract API it expects every plugin to implement, which consists of the following functions
  - Create: perform basic pre-configuration initialization
  - Configure: extract runtime configuration, preform basic sanity checks on it, adjust the plugins setup according to the configuration as much as possible
  - Start: perform any remaining steps to fully activate the plugin
  - Stop: deactivate the plugin, stop serving requests
  - Destroy: perform final cleanup, release all remaining resources

Configuration Handling

- Winthorpe uses a simple line-oriented text file for configuration
- Entries are simple assignments (key = value)
- Keys can be namespaced with an object-like dotted notation (foo.ba.r = xyzzy)
- Namespacing has no attached semantic meaning for the daemon
- Its only significance is that Winthorpe offers an interface for looking up all keys matching a prefix
- As another a syntactic sugar the configuration language offers a special syntax for grouping keys together with a common prefix into blocks delimited by curly braces

Configuration Handling

- The daemon allows setting any configuration key also from the command line which then takes precedence over the configuration file
- Simple functions are provided for
  - Setting configuration keys
  - Looking up all keys with a given prefix
  - Getting the value of a key as string
  - Getting the value of a key as boolean
  - Getting the value of a key as a signed or unsigned 32-bit integer

Configuration Syntax Example

Context-awareness via Murphy

- Winthorpe can use Murphy to coordinate with the rest of the system when speech recognition can be active and for which applications
- Murphy provides a flexible policy engine with scriptable policies, can be easily extended with plugins to track arbitrary aspects of the running system and take it into account for decision making
- Winthorpe uses Murphy resources for speech recognition and synthesis and pushes the arbitration and conflict resolution logic for this to Murphy
- It obeys the policy decisions of Murphy and activates/deactivates recognition and synthesis according to the decisions of the policy framework

Context-awareness via Murphy

- With Murphy integration we hope the speech services can blend in with and adapt to the state of the system more intelligently and naturally
- Winthorpe does allow for the usage pattern of the user explicitly controlling whether recognition is enabled or disabled both globally and on a per application basis
- However, ideally we'd like to get to the point where recognition can be left enabled for all applications (if they wish) and the system policy makes dynamically intelligent decisions, maybe occasionally assisted by the user, about which applications can utilize speech services and which cannot

Context-awareness via Murphy

- For instance, when a phone call is activated/accepted, the policy can be easily configured to withdraw the speech recognition rights from all applications, which in turn disables the recognition backend. Once the call is disconnected, recognition can be similarly reactivated by the policy.
- One could easily come up with quite a few similar reasonable automatic policies that make sense and feel natural to the user
- The Murphy integration is a work in progress, we have only taken the very first tiptoeing steps experimenting with ideas in this direction

Event loop, Loggin/Debugging Infrastructure

- Winthorpe reuses the policy independent Murphy common librarywhich among other things provides
  - An event loop with asynchronous I/O, timers, deferred callbacks, etc. one would expect to find in similar libraries
  - A configurable logging infrastructure with a chainable logger
  - A debugging interface with fine-grained control over what diagnostic messages are shown
- If a plugin relies on a library that requires glib, the event loop can be easily set up to be transparently pumped by a GmainLoop (by setting the configuration variable gmainloop to true)

Existing Plugins

- The current Winthorpe source tree contains several plugins that can be used to set up various working configurations
- Sphinx-speech: a speech recognition backend based on pocketsphinx. It is currently the only existing recognizer backend, so if you plan to play around with Winthorpe you will need this one
- Simple-disambiguator: a trivial and the only disambiguator that ignores all but the first utterance candidate from the recognition backend. So you need this to play around with Winthorpe.
- Espeak-voice: a speech synthesis backend based on espeak/flite. It is the most versatile and has the largest selection of languages and voices. It is also the only backend that supports pitch and rate control.

Existing Plugins

- Festival-voice: a synthesis backed based on festival. Otherwise on par with espeak-voice but festival usually comes with a smaller selection of supported languages. Also this backend does not support pitch or rate control.
- Native-client: socket based client API bindings for unix-domain and TCP/IP sockets. Comes with an accompanying client library that hides the details of communication from the client. The srs-native-client test binary uses this plugin and library to communicate with Winthorpe.
- Dbus-client: D-Bus 'bindings' for the client API. It is functionally equivalent to the native-client plugin but there is no accompanying library, client need to use D-Bus directly to talk to this plugin. The srs-dbus-client test binary uses this plugin to communicate with Winthorpe.

Existing Plugins

- Wrt-media-client: a proof-of-concept plugin with limited functionality that provides speech control and synthesis support for the HTML5 demo media player in the default Tizen IVI images
- Mpris2-client: a plugin that provides speech control for MPRIS2-compliant media players, such as Rhythmbox, available on most Linux desktops.
- Bluetooth-client: a proof-of-concept plugin that is able to bridge commands and queries to Siri on an iPhone, can command an iPhone to make phone calls using the contact list, and is able to control the AVRCP-compliant media player of the iPhone with speech commands.
- Search-client: a simple proof-of-concept plugin that might be able to execute your search queries using a web browser (uses chromium and google by default)

Existing Plugins

- Input-handler: a proof-of-concept plugin that can be used to give the user explicit control to disable and enable the recognition backend. It listens to input-layer events and toggles the recognition on/off when a configurable keypress (defaults to pause) is received

Future Work / Things we're working on

- Dynamic adjustment of decoders to the set of active commands. Currently the sphinx backend runs with statically pre-configured 'decoders'. This changes that to allow the backend to dynamically generate an FSG for the currently enabled set of commands (and also to notice when this is not possible due to some words missing from the dictionary)
- To be honest, Murphy integration in Winthorpe is quite in its infancy. We need to work on it, play around with it to better understand the inherent problems and start adding speech-specific parts to the ruleset.
- W3C Speech API for Crosswalk WRT using Winthorpe

Future Work / Things We Plan To Work On

- Another recognition backend based on Julius. This would be extremely important. Since we have only a single recognition backend, we don't know if the backend API is generic enough to easily support other backends than pocketsphinx. Even if it can, we need/want to change the API it to be less pocketsphinx-oriented.
- Is the division between disambiguation and the recognizer backend a good idea ?Or should disambiguation be the job of the recognizer backend ? Probably yes…
- Automatic grapheme-to-phoneme generation for unrecognized words. This is needed for things like using speech command to make phone calls or send an SMS to an entry in the users contact list for which the name is not in the dictionary so we need to make a guesstimate of the pronounciation. Phonetisaurus is a promising candidate…

Future Work / Things We Probably Should Work On

- Something which utilizes actively both recognition and synthesis. We need to gain better understanding
- Something the Linux desktop could benefit from… Maybe a simple assistant for evolution capable of things like: “list (or read) my 5 most recent e-mails”, “list my meetings for the following 2 hours”, “no, dismiss that meeting”
- VoiceML ? Or actually anything that would let applications implement (or actually instead of implementing it themselves just utilize) interactive voice-based menus with relative ease. Probably not VoiceML then…
- Lua bindings and runtime support for faster and easier prototyping ? A large part could be directly lifted over/reused by just refactoring and using the existing Murphy libraries

Future Work / Things We Probably Should Work On

- There might be apps that directly interact with Murphy using the resource interfaces… maybe we should prepare for that and let those handle the necessary speech resource interaction with Murphy themselves.
- Anything you can come up with ? Please, let us know…

Future Work / How Could You Help

- If you have experience or interest in working on speech-enabling applications, please join us in working on Winthorpe.
-

Resources

- Git repository
- Wiki
- Mailing list
- Project page
-

Backup Slides

Introduction

Background

Silo Approach

Dedicated App

Page title

- **Header bullet**
  - Sub-bullet 1
  - Sub-bullet 2
- **Second header bullet**

- **Highlight: read this, it's important**