Semantic Transcription Technology

Product Preview

December 2020

Originware Techology

Author: Terry Stillone (terry@originware.com)

Web: originware.com

Version: 1.2

Table Of Contents

1. Introduction

Introduction.

This document outlines the concepts and capabilities of Originware's "**Semantic Transcription Technology**". Here, the term "**Semantic Transcription**" refers to the derivation of the (intended) meaning of a given textual (English) sentence. The focal application domains for this technology are **vocal interactional systems** which serve domain knowledge, domain capability, disseminate information and control other devices and systems. The technology is a suited to apps, IOT devices, instruments, appliances, games, online conversational nodes and edge conversational nodes.

Supported conversational interactions include: interrogatives (questions), imperatives (directives), affirmations, demands, greetings and interjections. These interactions are represented as a semantic structure with full fidelity up to the limits conveyed in the textual content. Supported conversational scenarios include person to system (computer) and system to system .

It is important to point out that while person to system interactional queries generally convey clear intent, not all sentences are fully coherent and unambiguous. The intent of some sentences may require interpretation based on social attitudes, traits, customs or subject capabilities.

The transcription process resolves personal pronouns (you, me, I, we, us), mapping them to known participants but other pronouns (such as for example: him, her, this, that, it, person names, etc) must be resolved by the application to known personas and entities.

The technology discriminates between (**strict linguistic) semantics** and (**social) intent.** "**strict linguistic semantics**" honours the grammatical meaning. For example "*have you got the time*" has a linguistic semantic of the **affirmation:** "*<affirm> answer do you have the time, <yes or no>", w*here as the associated "**social intent**" is to "*<supply> the time"*.

Products and Broad Features.

The **Semantic Transcription** system is separated into two separate products:

- ▷ **Semantx**: The core semantic transcription system, includes grammar parsing, semantic parsing and intent parsing.
- ▷ **Converx**: The response framework which includes **Responder Node Trees** (routing based strategy for response resolution) and **Converx Nodes (**conversational network nodes which communicate conversational text and reply with conversational text).

Here is a list of product features:

Product	**Feature**	**Description**
Semantx	Grammar Support	Custom 40K word dictionary. Dictionary words tagged by **WRI** (Word Reference Identifier) which classifies their **POS** (Part of speech) and meaning category: e.g: /Noun/Thing/Animate/Fauna/Pet/("cat"). Multiple noun meaning-associations. Nouns can be tagged with multiple noun WRIs. Provision for automated editing of tokenized words, to correct speech recognition mismatches. Provision for the addition of custom words e.g custom domain specific terms.

Products and Broad Features (Cont.)

Product	Feature	Description
Semantx (Cont.)	Semantic Transcription Support	Semantic representation of conversational interaction is provided to the application with full linguistic fidelity.
	Intent Transcription Support	Intent representation of conversational interaction canonicalises the various ways of framing and expressing something to its pure intent form. Thus application response logic only has to case on intent (not on the various ways of expressing that intent). Intent representation is inherently lossy in fidelity.
Converx	Responder Node Trees.	*Use of hierarchal Responder Trees (authored by the client) that route requests to nodes that handle particular knowledge domains and topics. This partitions response logic along knowledge and topic domains and caters for configuration management of multiple target conversation nodes.*
		The supplied Responder Trees provide domain support for persona identity, function description, introduction, status, time, date, numeric and unit conversion.
		Use of responder identities for personalisation of replies.
		Application participates in the active resolution (routing) of requests to resolve a response.
		Support for reverse transcription from semantic to text (provides auto-generation of responses from the parsed intent or semantic).
	Converx Network Nodes	Edge based solution for routing queries and requests between network nodes (or network node hierarchies). Allows nodes to refer queries and requests that cannot be handled locally to online nodes for resolution and reply back.
		Nodes operate as Docker containers under BalenaOS (Linux), they can be resident on IOT (SOC) devices or hosted cloud systems.
	HTTP Support	Support for browser interaction with Converx Nodes and to also support custom HTTP APIs.
		HTTP requests can be passed to the Transcription Engine and routed through the Responder Node Trees to generate HTML responses.

Transcription performance is a key product feature, especially when fitted to vocal processing pipelines. The product performs well within human perceptual response times, see the appendix for performance benchmarks.

3. Semantic and Intent Representation.

Semantic and Intent Representation.

1. **Transcription** provides to the application logic, a representation of the sentence **Semantic** and **Intent.** These constructs are derived from the grammatical parsing of the sentence text. The grammar parsing itself, recognises the (grammatical) constructs of:
    - Interrogatives and Affirmations (questions).
    - Imperatives (demands and directives).
    - Greetings and Interjections.
    - Statements.
2. **Subjects** and **Actions** are the base constructs that describe the content (of the **Semantic** and **Intent**). Here, **Subjects** and **Actions** are broad encompassing representations, for example "*the house on the hill by the old grey oak tree past the stream"* is one single **<u>Subject</u>** and "*would quickly run around*" is one single **<u>Action</u>**.
3. **Semantic representation** is structured on three levels:
    - The **Framing** level, how the sentence is framed (interrogative, directive, affirmation etc).
    - The **Clausal** level, (describes the interaction(s) within the framing).
    - The **Content** level, (encompasses the **Subjects** and **Actions** that participate in the interaction).
4. **Clauses** to describe the interaction of subjects and actions. **Clauses** include two variants :
    - The **Subject Clause:** describes a subject in its own action: e.g "the ball flew".
    - The **Directed Clause:** describes an **actor** subject directing an action on an **object** subject and optionally directing the object to a **target** subject. e.g "he threw the ball to the dog".
    - The **Interrogative Clause** describes the event**:** e.g "when he had thrown the ball to the dog".
5. **Intent representation** in fine detai, needs to be custom for the application but on a high level, it does fall into the categories of:

To-request	To request the target persona to respond or perform something.
To-respond	To respond to the previous request.
To-state	To make a statement to the target persona.
To-greet	To greet the target persona.
To-interject	To interject.

The category *To-request* has sub-variants of:

- *To-provide* (provide a textual response) to a given target-persona (of type **Subject**):

To-identify:	Identify based on an interrogative clause (what, who, how etc).
To-supply:	Supply information based on a subject clause.
To-affirm:	Affirm (yes/no) with respect to the given clause to the given target.

- *To-perform* (to perform an action or to control a subject):
    - *To-trigger*: Trigger the given clause-action.
    - *To-apply*: Apply a given subject (measure) to a given target subject..

Intent Representation In More Detail.

Intent presumably operates on the state of components within the application. So the application must derive a series of models, that provides an operational pathway from **Semantic** to **Intent** to operability on the **Application State**.

These models include:

- The **Application Object State Model(s)**

  Models the state of operable objects within the application. These are the target objects to be manipulated by Intent. The application would normally already have these

- The **Subject Term Model (Optional)**.

  A model of the subjects that relates to the features and states of the **Object State Model.** This defines the subject matter "terminology". (The model also includes a mapping of **Subject** to **Subject Term).**

- The **Action Term Model (Optional)**.

  A model of the actions that relates to operations that operate on the **Object State Model.** This defines the action "terminology". **(**The model includes a mapping of **Action** to **Action Term).**

- The **Intent Op Model:**

  A model of how the **Object State Model(s)** are operated on by **intent**. This model will probably reference the **Subject** and **Action Term Models.**

In summary, the pathway from **Semantic** to **Application State** change invokes the mappings of:

- **Semantic Content** ⇒ **Subject Terms** and **Action Terms,**
- **Semantic + Terms** ⇒ **Intent-Op** ⇒ **Application State** change.

The complexity of these models depends on the diversity and extent of the **Semantic Expression** and **Content** matter required to control the application state.

Lets take a simple example, the control of a home lighting system. The lights are designated by their room and light in room. The **Object State Model** is thus:

Application Object State Model	
	struct Lighting { let zones: [Zone] }

Support	struct Zone { let designation : String; let lights : [Light] }
	struct Light { let designation : String; let isOn : Bool }

3. Semantic and Intent Representation.

The **Intent** models are:

	Subject Term Model	**Action Term Model**
Term Model	▪ enum eTerm ◦ case eZone(String) ◦ case eLight(String) ◦ case eOn ◦ case eOff	▪ enum eTerm: ◦ case eSwitch ◦ case eToggle
Term Mapping (Not all terms given here)	▪ Subject("on") ⇒ .eOn ▪ Subject("off") ⇒ .eOff ▪ Subject("kitchen") ⇒ .eZone("**kitchen**") ▪ Subject("general") ⇒ .eLight("**general**") ▪ Subject("accent") ⇒ .eLight("**accent**")	▪ Action("turn") ⇒ .eSwitch ▪ Action("switch") ⇒ .eSwitch ▪ Action("toggle") ⇒ .eToggle

Intent Model	
Light Reference Model	▪ enum eHouseLightingRef ◦ case eAllZones ◦ case eByZone(**zoneName**: String, eZoneRef)
	▪ enum eZoneRef ◦ case eAllInZone ◦ case eByLightType(**lightType**: String)
State Model	▪ enum eLightState ◦ case eOn ◦ case eOff
Intent Op Model	▪ enum eOp ◦ case eSwitchLight(**ref**: eHouseLightRef, **to**: eLightState) ◦ case eToggleLight(**ref**: eHouseLightRef)

4. The Semantic Transcription Process.

Here is an intent example: for the request: “what is the time”, the transcribed intent is:

To<request>/To<identify>:

- target persona: Subject(/Noun/Thing//Person/Identified/User)
- Interrogative clause: /Interrogative/What,

  subjectClause: Action(verb: /Verb/Auxiliary/ToBe/Is),

  Subject(/Noun/System/Time/Time, modifiers: [/Determiner/Article/Definite]),

The Operational Design of the Semantic Transcription System.

The diagram below depicts the design of the Semantic Transcription system with processing elements for each individual processing step:

▷ ***Step 1***. The **Tokenization** and **WRI** tagging of the sentence text to derive a **WRI** tagged, word token-sequence.

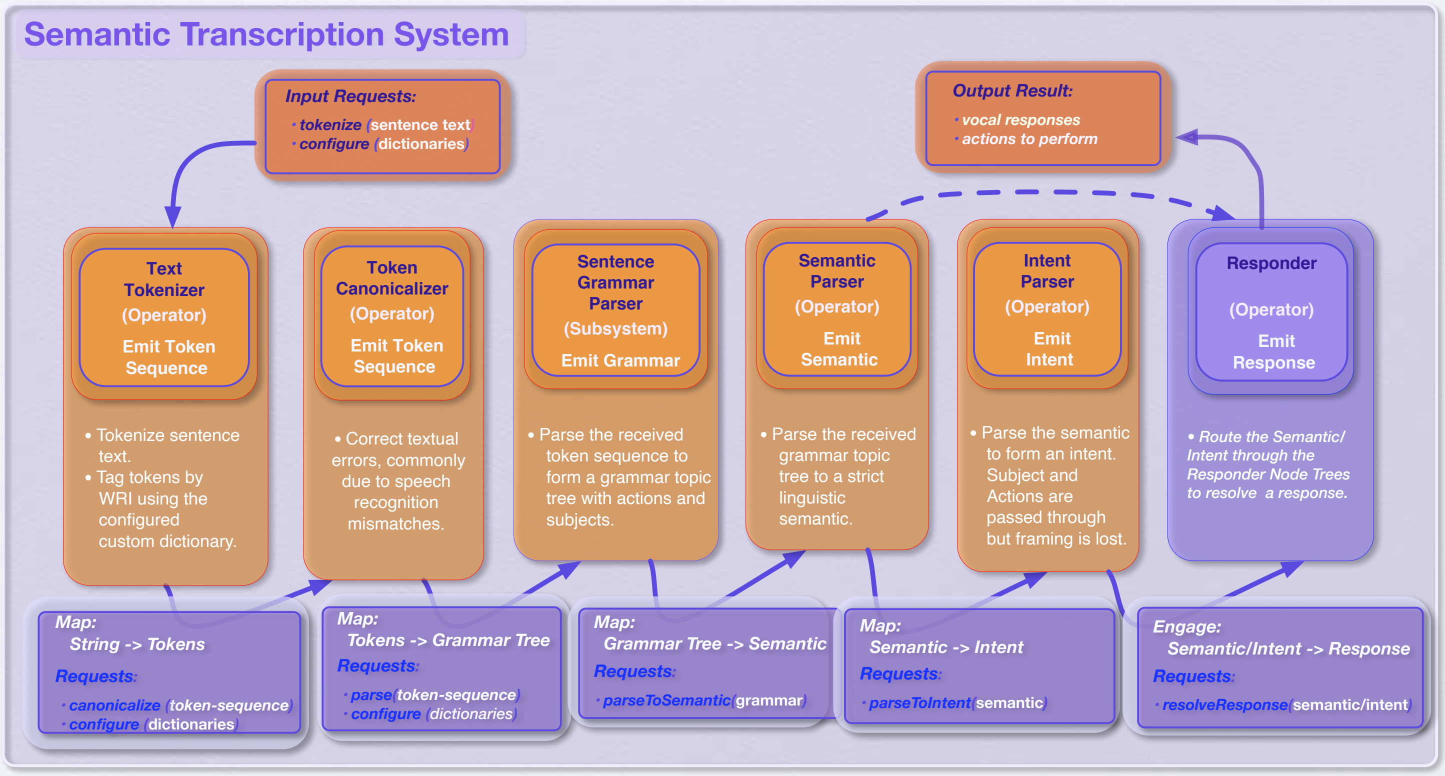
▷ ***Step 2***. The **canonicalization** of the word token-sequence (includes auto correction of speech recognition mismatches).

▷ ***Step 3***. The parsing of the word token-sequence to a **Grammar topic tree**. The tree includes Subjects and Actions within the sentence.

▷ ***Step 4.*** The parsing of the **Grammar topic tree** to a **Semantic** together with an optional addressee persona (who the sentence is addressed to).

▷ ***Step 5***. The parsing of the **Semantic** to **Intent**. This is optional, depending on whether the response generation system interprets the **Semantic** or the **Intent** or both.

▷ ***Step 6***. The resolution of the **Response** to the original request derived from the generated **Intent/Semantic**. The generated response typically includes vocal reply text and internal actions to be performed.


Semantic Transcription System
Input Requests:
• tokenize (sentence text
• configure (dictionaries)
Output Result:
• vocal responses
• actions to perform
Text Tokenizer (Operator) Emit Token Sequence
• Tokenize sentence text.
• Tag tokens by WRI using the configured custom dictionary.
Token Canonicalizer (Operator) Emit Token Sequence
• Correct textual errors, commonly due to speech recognition mismatches.
Sentence Grammar Parser (Subsystem) Emit Grammar
• Parse the received token sequence to form a grammar topic tree with actions and subjects.
Semantic Parser (Operator) Emit Semantic
• Parse the received grammar topic tree to a strict linguistic semantic.
Intent Parser (Operator) Emit Intent
• Parse the semantic to form an intent. Subject and Actions are passed through but framing is lost.
Responder (Operator) Emit Response
• Route the Semantic/ Intent through the Responder Node Trees to resolve a response.
Map: String -> Tokens
Requests:
• canonicalize (token-sequence)
• configure (dictionaries)
Map: Tokens -> Grammar Tree
Requests:
• parse(token-sequence)
• configure (dictionaries)
Map: Grammar Tree -> Semantic
Requests:
• parseToSemantic(grammar)
Map: Semantic -> Intent
Requests:
• parseToIntent(semantic)
Engage: Semantic/Intent -> Response
Requests:
• resolveResponse(semantic/intent)

5. Response Generation from Semantic/Intent.

Transcription (and response generation) equates to a series of functional transforms:

Input Text* ⇒ *Grammar* ⇒ *Semantic* ⇒ *Intent* ⇒ *Response

This varies to the typical **AI** solution which is end-to-end and equates to the monolithic transform of:

Input Text* ⇒ *Response

Any customisation or modification to this monolithic mapping requires access to the whole **AI** system.

The Originware **Transcription** system on the other hand is modular and flexible to cater for varied project requirements. Customization can be minimal, with the application suppling the mapping:

Intent* ⇒ *Response

Or it can include deeper participation in the dimensions of multiple personas, multiple knowledge domains, grammar inclusion (custom word terms), custom corrections (speech recognition mismatches), direct access to grammar parsing, semantic parsing, intent parsing etc.

The Responder Node Tree Architecture.

Response generation (from the **Semantic/Intent**) can be just as complex as the **Semantic Transcription** process itself. There are various knowledge and capability domains to cater for. The **Converx** product supplies a multi-tree-routing framework to route the semantic/intent to the relevant handling code. The process of **Intent/Semantic** to **Response** mapping comprises of a number of operational steps, given here:

- ▷ ***Step 1.*** Typically**,** resolution of **Pronouns** and named people or personas are performed but this can be deferred to Step 2.
- ▷ ***Step 2***. The routing of the **Intent/Semantic** through the collection of available **Responder Trees** to resolve a response that is relevant for the content and topics contained in the **Intent/ Semantic**. Supplied responder nodes include responder identity, expression of responder function, status and capability, time and date handling and unit conversion. The application participates in the determination of which responder nodes and responder trees are employed.
- ▷ ***Step 3***. If resolution is not satisfied, the request can be routed to other networked responder systems called **Converx Nodes**.
- ▷ ***Step 4.*** If resolution is still not satisfied, then a negative response can be auto generated from the original semantic. This step represents the default fail-over case for requests that the application does not handle.

Responder nodes respond with one of the following options:

- ▷ A **positive response** (that is relevant for the Intent/Semantic)
- ▷ A **negative response** (which is not treated as an error).
- ▷ An **error**.
- ▷ Otherwise a **nil response**.

As a general rule, requests are routed through responder nodes and trees a until positive response (or a hard error) is observed. Negative responses are collected and then used if no positive response is given. If no responses were collected at the final routing step then a nil response is given which equates a “No response” result.

The controlling application also has the option of auto generating negative responses.

6. Network Conversational Nodes.

Converx Nodes: Conversational Net-Nodes.

Converx Nodes (as opposed to Responder Nodes) are network agent-nodes that support a distributed conversational node-network. These **Conversational agent-nodes** communicate via plain conversional text (with a small amount of meta-data). The node network-hierarchy forms an edge system which distributes conversation processing, by assigning specific knowledge domains to specific net **Converx Nodes**. Net-route-through capability allows the network to collectively satisfy a greater scope of textual requests while not needlessly burdening higher level nodes.

Typically, a cloud based **Directory** node is deployed to route requests between processing **Converx Nodes (**and these individual processing nodes handle separate knowledge domains). Thus a local **Converx Node**, say on a small IOT device can route out of scope requests to cloud instances for resolution.

As an example, envision a medical instrument, with an internal dedicated **Converx Node,** where that node is only capable of handling verbal requests for the control of the instrument. But additionally, the node has the capability to net-route more complex queries to a cloud based **Directory Node**. So common use verbal commands for the instrument can be satisfied locally (and not overload the cloud **Directory Node)** but at the same time, the local node can address out of scope requests to the **Directory Node** for a greater, more comprehensive coverage of knowledge and capability.

Converx Nodes: Example Applications.

Lets extend our medical instrument example, and go through some more advanced scenarios. Lets say rather than the instrument having speech recognition capability it acts solely as a **ConverxNode** and expects WiFi connectivity to a user device (phone, tablet etc) which runs a specialised app to control the instrument. So speech recognition is employed on the user device and the device's internal **ConverxNode** routes those textual requests to the instrument. Also, the instrument controller hosts a small web-server that serves visual HTML pages for the state/operation of the instrument, that can be displayed on the device app when replying to requests.

The instrument no longer requires physical controls, the user speaks to their device App for operation, the instrument performs those actions, replies back verbal text that the Apps speech synthesis vocalises. The instrument also serves web pages, displayed on the user-device App to idicate the current visual-state/operation of the instrument.

Also, security can be provided so that only authorised devices and users can operate the instrument. The instrument may be configured to only accept control devices that have been given a pre-registered authorisation certificate that matches the device ID and user ID. The app on the control device may only operate after the user has passed appropriate security measures.

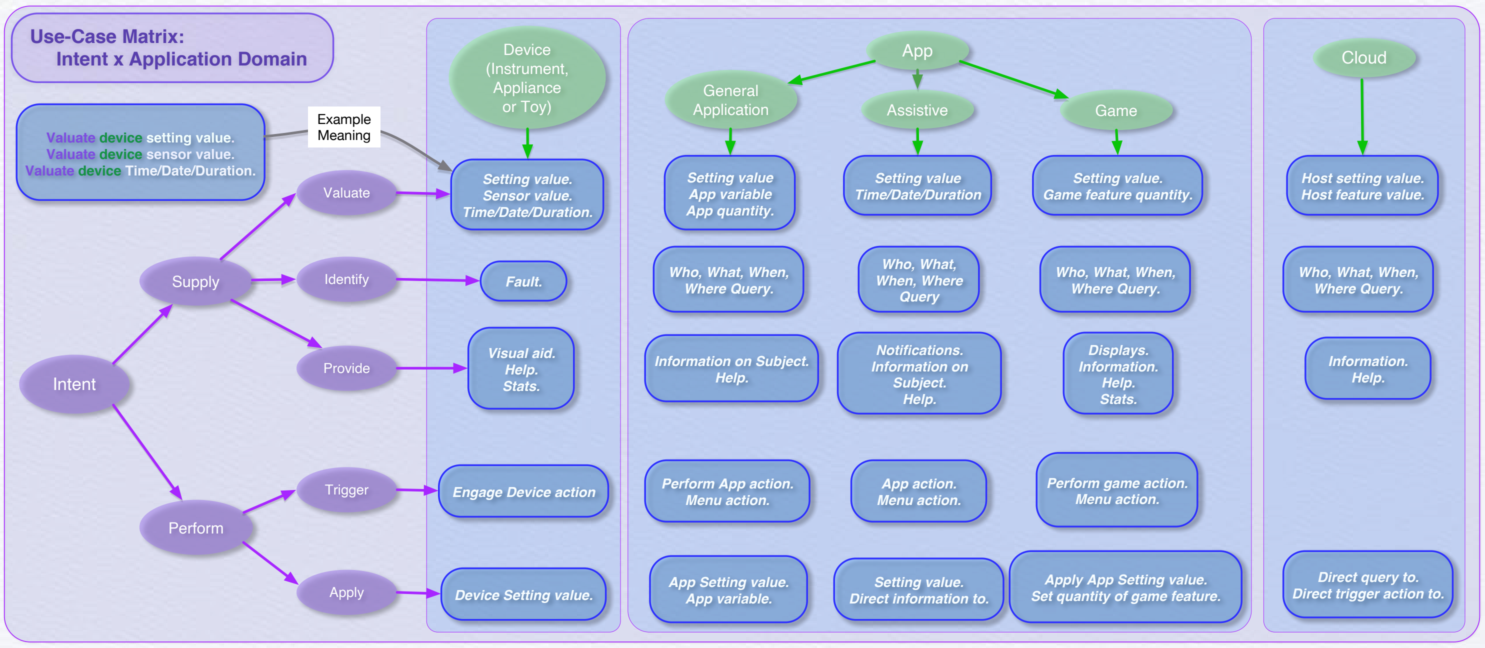
If the user experiences problems with the instrument, the **ConverxNode** app can be used to assist. The user can address the "Help desk" with a vocal query, which the **ConverxNode** app routes to the cloud based "Help desk" **ConverxNode**. The "Help Desk" **ConverxNode** in turn, performs triage on the request. It may route the request to the phone/tablet of a specific person or redirect it to the phone of the rostered help staff member. That person's **ConverxNode** app will inform them and they can vocalise a simple reply through the app or voice-call the user (with a vocal command to the app) or handle it personally.

Below is an example use-case matrix that organises **Application Domain** by **Intent**. It denotes what **intent** could be used for, within the specific **Application Domains**.

(Note the **To<affirmation>** Intent is not included in the matrix).

7. Product Applicability.

Applicable Use Cases.



Use-Case Matrix:
Intent x Application Domain
Valuate device setting value.
Valuate device sensor value.
Valuate device Time/Date/Duration.
Example Meaning
Intent
Supply
Perform
Valuate
Identify
Provide
Trigger
Apply
Device (Instrument, Appliance or Toy)
Setting value. Sensor value. Time/Date/Duration.
Fault.
Visual aid. Help. Stats.
Engage Device action
Device Setting value.
App
General Application
Assistive
Game
Setting value App variable App quantity.
Setting value Time/Date/Duration
Setting value. Game feature quantity.
Who, What, When, Where Query.
Who, What, When, Where Query
Who, What, When, Where Query.
Information on Subject. Help.
Notifications. Information on Subject. Help.
Displays. Information. Help. Stats.
Perform App action. Menu action.
App action. Menu action.
Perform game action. Menu action.
App Setting value. App variable.
Setting value. Direct information to.
Apply App Setting value. Set quantity of game feature.
Cloud
Host setting value. Host feature value.
Who, What, When, Where Query.
Information. Help.
Direct query to. Direct trigger action to.

Current Product Development State.

Both **Semantx** and **Converx** are currently in beta, migrating from the current version: **0.9** to the target version: **1.0**. The products are written in the Swift 5 language and are operational on the **Linux** (ARM), **MacOS** and **iOS** platforms with support for both 32bit and 64bit CPU architectures.

The major transitional work from 0.9 to 1.0 is majorly focused on the fine adjustment of the final Semantic **Expression** and **Intent** representations. This is to reduce the amount and type of matching that the response code must perform to determine the core **Intent** and also to grade nuances on the fine detail of the semantic.

- The current **Expression** representation is a little too general and is to be refined to the multi-clausal representation given earlier in this document.
- The current "**Semantic Pattern**" representation is to be migrated to a more generalised **Intent** form, stripping the semantic framing. **Actions** in the **Intent** are to be further abstracted from their semantic form by mapping their multi-auxilary-verb composites to tense tagging (e.g ***I had swum, I went swimming, I did go swimming,*** will be mapped to the same action **WRI** of the verb "swim" but tagged with their respective tense.
- Increase the gamut of less-used sentence construction/phraseology.

The major feature transitions for 1.0 to 1.1 are:

- Support conversational threads, that is, tracking the **To<Request>**, and **To<Reply>** sequencing to determine which conversations are closed or still active. This includes support for user **To<Reply>** intent handling, currently only system **To<Reply>** intent handling is supported.
- Support multiple grammar solutions (currently only the first resolved grammar solution is used).
- Support for the Apple WatchOS platform.

The long term roadmap is to port a Kotlin implementation and support Android platforms.

8. Additional Information.

For Additional Information.

Please see the Originware website: originware.com or email: Terry Stillone: terry@originware.com for more information and demonstrations of the technology.

There are also other software technologies available from **Originware** such as **Reactive Fabric**, a software design notation and incremental design methodology, please see:

originware.com/reactivefabric.html

(Please see the following Appendix for performance benchmarks)

Appendix A: Benchmarks.

Transcription Performance.

CPU Make	CPU Arch		Transcription Pass Time (ms)		Transcription Failure Time (ms)	
			Release	Debug	Release	Debug
Intel Core i7 2.3 Ghz	64 bit 32 KB L1	Min	**2ms**	3ms	**3ms**	3ms
		Av	**7ms**	15ms	**65ms**	200ms
		Max	**35ms**	115ms	**250ms**	650ms
Apple A10 2.34 GHz	64 bit 64 KB L1	Min	**2ms**	3ms	**20ms**	60ms
		Av	**15ms**	20ms	**70ms**	120ms
		Max	**55ms**	90ms	**135ms**	290ms
Raspberry Pi 3 Broadcom ARM Cortex A53 1.2 Ghz	32 bit 32 KB L1	Min	**15ms**	20ms	**220ms**	450ms
		Av	**50ms**	75ms	**400ms**	870ms
		Max	**215ms**	530ms	**1110ms**	2300ms

Notes:

- ▷ These performance results derived using a corpus of 5300 questions.
- ▷ The SDK is single-threaded, only one CPU is being exercised.

SDK Memory Footprint Metrics.

OS	Code	Release Memory Footprint
Linux	Standalone **Semantx** operation. (Library: libSemantx.a)	Text size: < 7Mb Data + BSS: ~300KB
Linux	Standalone **Semantx** & **Converx** operation. (Libraries : libSemantx.a + libConverx. + NIO)	Text size: < 10Mb Data + BSS: ~500KB
Linux	**Converx Node application** incorporating libraries: **Semantx, Converx and NIO**	Text size: < 12MB Data + BSS: < 600KB Swift support libraries: ~15MB Resident Memory Size: ~45MB

Appendix B - Source Code Metrics.

SDK Source Code Metrics (in swift code-only, lines of code).

Library	Section	Swift LOC
Semantx	Functional SDK	35K
	Initialization code	10K
Converx	Functional SDK	10K
	Example Responder Nodes	8K

Custom Dictionary (Metrics for Version 0.9)

Dictionary	Lemma/Word Count
Noun Lemmas	19.2 K
Verb Lemmas	4.5 K
Adverbal Words	1.6 K
Adjectival Words	7.3 K
Conjunctional Words	68
Prepositional Words	109
Total Lemmas + Words	32.8 K
Projected Word Count, by expanding lemmas	61 K