Anyone’s Guide to Conversation Service

Richard Steiger, Uniphore

Conversation Service V1

Document Version 0.4

**Contents**

[1 Introduction/Background 2](#_Toc118724228)

[1.1 Purpose and Scope 2](#_Toc118724229)

[1.2 Target Audience 3](#_Toc118724230)

[1.3 Stakeholders 3](#_Toc118724231)

[1.4 Working Document 3](#_Toc118724232)

[1.5 Evolutionary Path 4](#_Toc118724233)

[1.6 Conventions Used 4](#_Toc118724234)

[1.7 Some Overall Technical Considerations 4](#_Toc118724235)

[1.7.1 Entity Lifetimes and Ownership 4](#_Toc118724236)

[1.7.2 Datamodel is Reference-Oriented 5](#_Toc118724237)

[2 Key Concepts and Relationships 5](#_Toc118724238)

[2.1 Conversation/Core-Service Data Model 5](#_Toc118724239)

[2.2 Conversation 5](#_Toc118724240)

[2.2.1 Conversation States 6](#_Toc118724241)

[2.2.2 Channels 8](#_Toc118724242)

[2.3 Session and Journey 8](#_Toc118724243)

[2.4 Participant 8](#_Toc118724244)

[2.5 Business Context 8](#_Toc118724245)

[2.6 Recording 8](#_Toc118724246)

[2.7 AI Signal 9](#_Toc118724247)

[2.7.1 Named Entities 9](#_Toc118724248)

[2.7.2 Intents 9](#_Toc118724249)

[2.7.3 Sentiments 9](#_Toc118724250)

[2.7.4 Promises 9](#_Toc118724251)

[2.8 Flows and Flow Steps 9](#_Toc118724252)

[3 Use Cases 9](#_Toc118724253)

[3.1 Recording and Storing 9](#_Toc118724254)

[3.2 Retrieving 10](#_Toc118724255)

[3.3 Cataloging 10](#_Toc118724256)

[3.4 Querying 10](#_Toc118724257)

[3.5 Event Sourcing 10](#_Toc118724258)

[3.6 Flow Editing 10](#_Toc118724259)

[4 Storage and Persistence 11](#_Toc118724260)

[4.1 Storage Model Choice 11](#_Toc118724261)

[4.2 Relational Storage Design 11](#_Toc118724262)

[4.3 Open Issues 11](#_Toc118724263)

[5 References 11](#_Toc118724264)

# Introduction/Background

## Purpose and Scope

This purpose of this document is to provide as an informal overview of the emerging Core Platform Services, this version focuses on the ***Conversation Service*** [CS], the first ***Core Platform Service***.[[1]](#footnote-2)

Along the way, key terminology used in Uniphore products, and in the Conversational Automation and AI (CAAI[[2]](#footnote-3)) field, are informally defined, intended to help clarify terms, some of which are still tribally defined, hence can be used inconsistently across product lines, and their teams.[[3]](#footnote-4)

The scope of this document, while primarily intended to help guide development of the CS in early versions by normalizing our language and concepts, is equally intended to capture broader architecture and design patterns, conventions, and terminology applied to the Platform as a whole.

## Target Audience

This document is initially written for anyone involved or interested in building the CS, and any of its required infrastructure, so includes those having development, product management, operations, and project management roles.

## Stakeholders

At this point, the CS has the following stakeholders

* U-Assist (In Call & ACW)
* U-Analyze
* U-SelfServe
* Beehive

## Working Document

This document is a living and working document, and at this stage, isn’t a definitive specification nor a set of explicit architectural guidelines and recommendations. Instead, it’s intended as a snapshot of the Conversation Service team’s rapidly-evolving vision, currently focused on

* discovering and elucidating those parts of the overall Core Service data model and service APIs,
* understanding how conversation and related data would best be organized,
* distilling core APIs and data representations that span and unify existing U-Product lines,
* enabling a more declaratively configurable, flexible, and faster way to share data across product components and engagement solutions,
* providing a pragmatic target for integrating our product lines with the Core Services platform, and
* aspirationally, enabling building low/no-code graphical flowgraph, rule orchestration, and other editing tools, based on the Beehive platform.

This document is also based on the CS teams’ current and rapidly evolving understanding of the Conversational Automation domain. In particular, much of the terminology in this document is tentative, essentially placeholders for concepts for which we aren’t yet aware of existing terms, or have yet to be invented.[[4]](#footnote-5)

We’ve gone as far as is sensible without knowing more about all of our product lines, which are the intended sources of functionality and data that Core Services are intended to integrate and unify.

SMEs in various product teams will notice that there are some big gaps between what’s proposed here, and the product datamodel specs, such as <https://uniphore.atlassian.net/wiki/spaces/PE/pages/1195966465/Real-Intent+Data+Model+and+Topics>. Such gaps are to be expected given our early stage gathering and attempting to merge them into the Core Services Data Model, and we look forward to working with y’all to bridge the gaps.

All inaccuracies and confusions are the author’s sole responsibility. I look to others to help make this document accurate, and help make it useful over the course of Core Service development, Uniphore product integration, and eventually, to serve as a technical guide and on-boarding aid for those new to Uniphore’s products and the conversational automation and AI domain.

## Evolutionary Path

In section 2, I describe our working proposal for the ***Core Services v1 data model***, which comprises conversation and related entity types. [[5]](#footnote-6) The data model has been reverse-engineered from CS’s rapidly evolving v1 implementation, specifically the data protobuf specification [1] [[6]](#footnote-7).

I’ve deliberately held-off describing CS’s APIs, for the obvious reason that our current state of knowledge about Uniphore’s product line architectures, and present and anticipated use-cases, is still too vaporous to put forth.

Our plan going forward is as follows:

* meet with product PEs to understand current functionality, data models, APIs, dataflows/pipelines, end-to-end scenarios and block diagrams, challenges, and opportunities;
* identify core architectural patterns that will help us achieve plug-and-play modularity at data, api, and pipeline levels, giving us maximum flexibility to build
* custom solutions
* develop more advanced analytics
* develop and incorporate new AI services
* enable rapid experimentation across Uniphore
* be increasingly agile, hence competitive; and
* define and build CS 1.0, including datamodels, APIs, storage, and flexible and modular flows.

## Conventions Used

On initial mention of any concept, its name is written in ***bold italic*** to call-out its following definition. Entity types are written capitalized in this font when introduced, for example Conversation. To make such introductions flow more easily, types may be implicitly introduced when an instance is first referenced, in which case the instance is the type’s name in lowercase in the same font, having singular or plural number (e.g., participants).

## Some Overall Technical Considerations

### Entity Lifetimes and Ownership

As of this writing, there isn’t a clear convention about exactly which entities’ lifetimes are tied to a conversation, and which exist independently of conversations.

There are some fairly obvious lifetime dependencies, for example,

* as currently conceived, participant lifetimes are scoped under their “parent” conversations, while
* placeholder concepts BusinessContext, Recording, Transcript, and Turn seem better treated as anchored under Recording (the current equivalent) independently of conversations.

I’ve deferred attempting to attribute type-subsystem ownership boundaries until more of the Core Services have come into view. Similarly, entity lifetimes and which storage subsystem they’re stored are strongly dependent, and unknown at this time.

### Datamodel is Reference-Oriented

While runtime data representations using foreign keys in-place of references or good-ole pointers is very common, and essential in many low-level implementations (such as in database engines), I’m of the school that wires between boxes is an undeniably clearer and more concise conceptual representation at higher design and architecture levels. In the same way, I believe that leaning on protobuf specs or similar wire protocols as specs of record is an anti-pattern, and that the latter should be forward-engineered from the former. If you have doubts about this position, please see [8].

# Key Concepts and Relationships

## Conversation/Core-Service Data Model

Given their central role, conversations provide access to entities that conceptually reside in other core services.

The UML Class Diagram on the following page is one possible view of the v1 data model, providing an overview of entity types and their relations.

In the diagram, placeholder types have red borders. As already mentioned, these will be the focus of discovering what equivalent data already exist and in which product line. The one value I’ve intended to capture in this model is latent abstractions, such as AiSignal, allowing aggregation and share processing pathways of very similar entities.

It’s all up for debate!

In the following subsections we describe each entity type and closely related type groups.

## Conversation

CS’s data model’s primary entity type is Conversation. which provides access paths to related entities.

A Conversation instance (simply, conversation) is a record of an interaction between 2 or more participants, either of which may be human or machine [2], within the time-scope of a session.

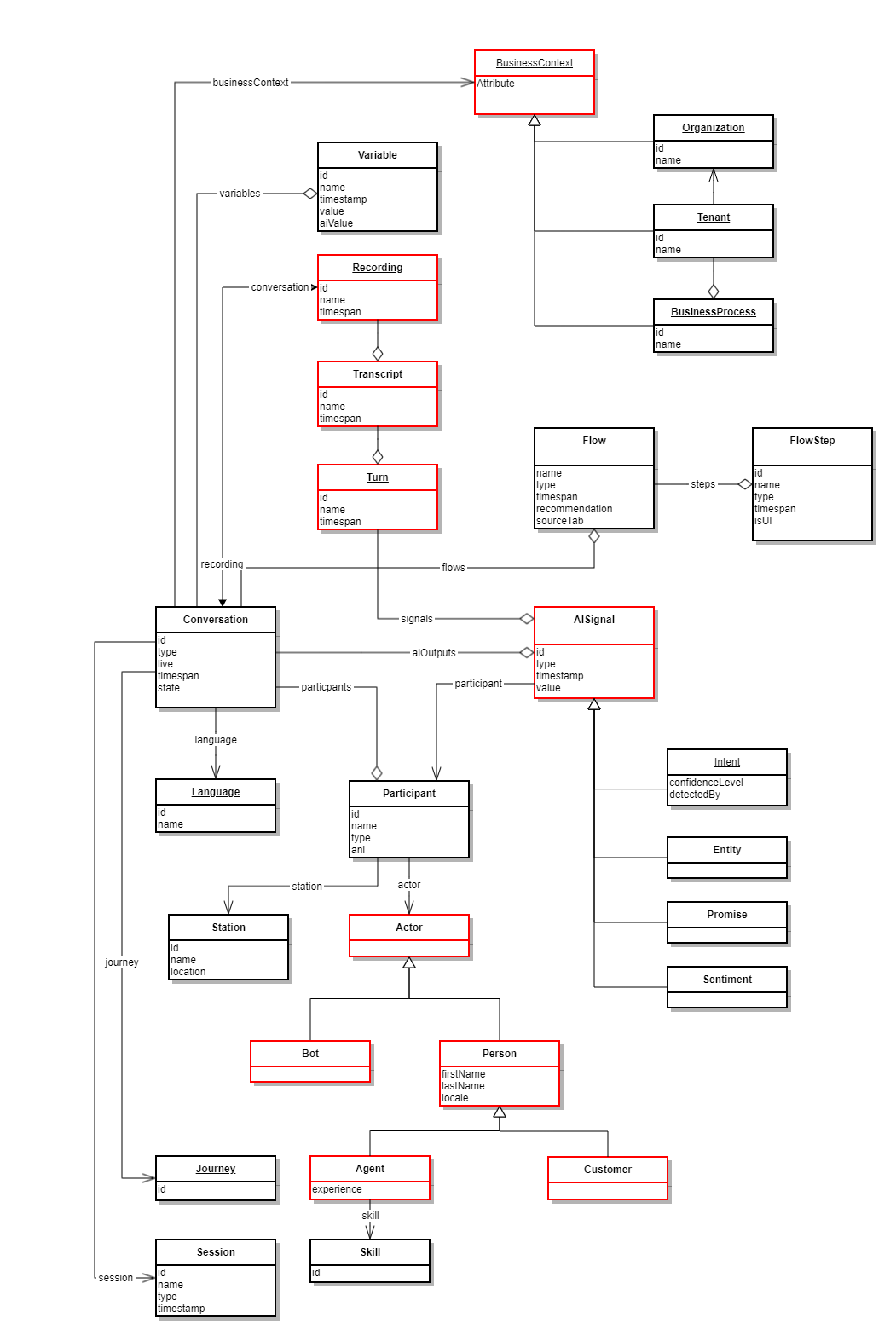
Mirroring how human-human conversations occur in the real world, conversations are naturally structured as a sequence of turns, in which one participant (the speaker) emits voice, gestural, keystroke, or other ***signals***, and the other participant (the hearer or listener) attends to and (hopefully) comprehends the signals[[7]](#footnote-8).

### Conversation States

Each conversation has an associated state machine, having the following states:

* INVALID
* ACTIVE
* ON\_HOLD
* FINISHED
* TRANSFERRED

Each conversation’s state is maintained by CS, based on events received from other subsystems.



### Channels

Implicit in the above definition is the fact that a conversation may be carried over various channels, each of some medium. The following are the media used most often in conversational automation:

* audio (telephone, VOIP, other real time voice transport)
* video (video transport, e.g., Zoom, Teams, Slack)
* typing and pointing (with mouse, pen, hand trackers)

For any media configuration (aka a “modality”), all channels time-synchronously record their stream, allowing later replay that faithfully replicates the original experience.

## Session and Journey

TBS, included in the v0 protobuf spec, will research and update.

## Participant

In a typical conversation, 2 participants engage in dialog, with one being a (human) end-customer voice-calling in to a customer service call center, the other being a (human) customer service agent, or an agent bot handling the call.

The Participant type is a relationship between an actor and a station or channel endpoint over which the actor communications, for the duration of the owning conversation’s timespan. As shown, an actor may either be a person or a bot, and a person may be either a customer or an agent.

## Business Context

We’ve *introduced BusinessContext as an abstraction of the organizational setting where conversations take place. As shown, a businessContext may be an organization, or such an organization’s tenant on the Platform Cloud, or a specific businessProcess for that tenant.*

## Recording

*A Recording is a record of the* various media signals that occurred over the span of a conversation. I’m assuming that recordings may be born before or after the originating conversation, but once both exist, they belong in a one-to-relationship.

A ***Transcript*** is a text document that’s been extracted from a voice recording, from a single turn within the recording, or even from a turn fragment in an active recording. Like other products in the CAAI marketplace, Uniphore uses automatic speech recognition (ASR) to extract transcripts from conversations. As with human speech recognition, ASR isn’t perfect, and transcript use cases exist where a human will correct a transcript, making transcripts potentially mutable. As of this writing, storage, identifier generation, mutation, and other aspects of transcript management is outside the scope of CS, but needs to be addressed in future versions.

## AI Signal

Various Uniphore products provide various AI and NLP processing services on transcripts, and in some cases, audio recordings, to extract rudimentary human-understood meanings from conversations, in the form of AiSignals.

An aiSignal [[8]](#footnote-9)is any event indicating the occurrence of an utterance within a conversation having semantic significance to participants and the business. As of this writing, an aiSignal may be the detection of an intent, a namedEntity, a promise, or a sentiment.

### Named Entities

A NamedEntity is an utterance[[9]](#footnote-10) refering to some [real-world object](https://en.wikipedia.org/wiki/Real_world_object), such as a person, geolocation, organization, product, data, street address, etc., which can be denoted with a [proper name](https://en.wikipedia.org/wiki/Proper_name). Named entities are the key nouns in a conversation, referring to subjects of relevance to the caller’s issue. A given conversation may contain any number named entities.

### Intents

An Intent is an utterance indicating the speaker’s purpose or central reason for initiating the conversation.

### Sentiments

A Sentiment is an utterance indicating the speaker’s emotional state in simple emotional valence terms.

### Promises

A Promise is an utterance indicating that a CX agent has made a commitment to the customer that they will see that the customer’s concerns are addressed and resolved.

## Flows and Flow Steps

TBS when we better understand this area.

# Use Cases

This section covers the CSs use-cases.

## Recording and Storing

**Use case name:** Recording a conversation in durable storage

**Primary Actor:** Any Uniphore product engineer working on conversation capture.

**Description:** I need to submit individual conversations as single dataset.

**Preconditions**:

* I have required access permissions
* I’ve generated the dataset in the required format[[10]](#footnote-11)
* I have the CS endpoint[[11]](#footnote-12)

**Postconditions**:

* The dataset has been stored.

## Retrieving

**Use case name:** Retrieving entire conversations for a post-call process

**Primary Actor:** Any Uniphore product engineer or an integration engineer needing conversation access.

**Description:** I need to obtain entire conversations as a single unit.

**Preconditions**:

* I have required access permissions
* I have the conversation’s UID[[12]](#footnote-13)
* I have the CS endpoint

**Postconditions**:

* I have a dataset containing the desired conversation.

## Cataloging

TBS.

## Querying

TBS.

## Event Sourcing

TBS.

## Flow Editing

TBS.

# Storage and Persistence

## Storage Model Choice

What storage models needed for persisting conversation entities, one from {columnar, relational, document, blob}?

Based on the conversation data model’s simplicity, my initial take is that storing it in an RDBMS is at least as functional and performant as the other storage models and is well-supported by OSS ORM packages.

In [4], Justin Harringa explored using Redis as an alternate storage and message-assembly mechanism, more discussions are needed understand the tradeoffs.

Events are stored in some columnar storage subsystem.

## Relational Storage Design

Given the above suggested adoption of an RDBMS to store conversations, it seems to me natural to adopt one of two ORM mappings:

* **table-per-type** – each of the (currently) 10 entity types is mapped to its own table
* **type-tagged table** – all types are mapped to a single table, with a type-discriminator column
* **hybrid** – types would be grouped by attribute similarity, and each group mapped to a type-tagged table.
* one such grouping would put types having occurrence instants together, versus types having occurrence intervals
* another grouping would be to put AI data in one table, and flow-related data in another table

## Open Issues

* What SLAs or other KPIs should be provided?
* What DR should be provided? Can it be covered by Platform’s more general DR provisions?
* Does CS need its own aging (downsampling, summarizing) and archiving facilities, or does Platform provide a solution?
* Volumetrics: for what flow volumes should we design?

# References

[1] V1 conversation common protobuf spec <https://gitlab.com/-/ide/project/uniphore/cloud/uniphore-protos/tree/master/-/uniphore/conversations/v1/common.proto/>

[2] Jurafsky, D. and Martin, J., *Speech and Language Processing*, 2nd Ed, Pearson Prentiss-Hall

[3] *Named Entities,* <https://en.wikipedia.org/wiki/Named_entity>

[4] Lenny Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*, 1st Edition, Addison-Wesley, ’14

[5] Harringa, J., Call Service Proposal, Uniphore internal, <https://uniphore.atlassian.net/wiki/spaces/PlatEng/pages/2330001414/Call+Service+Proposal>

[6] Giroux, M.A., *On GraphQL-to-SQL*, <https://productionreadygraphql.com/blog/2020-05-21-graphql-to-sql>

[7] Freeman, P., *Building a GraphQL to SQL Compiler on Postgres, MS SQL and MySQL*, <https://hasura.io/blog/building-a-graphql-to-sql-compiler-on-postgres-ms-sql-and-mysql/>

[8] Gorski, D.M., *The type system is a programmer's best friend*, 11/1/22, <https://dusted.codes/the-type-system-is-a-programmers-best-friend>

1. This doc builds on [6] and other docs in-flight within the architecture guild. [↑](#footnote-ref-2)
2. Is there already such an acronym used in product, engineering, or marketing? [↑](#footnote-ref-3)
3. I’m still a noobie to CAAI, and I encourage you, the reader, to keep in-mind that this entire document is a living work-in-progress, subject to rapid and significant changes. I’m also relying on you, gentle reader, and other Uniphorians to help shape the language and content based on their much deeper and broader experience and expertise. [↑](#footnote-ref-4)
4. If presented with any alternate synonymous terms already in use within Uniphore’s product teams, or more widely within the industry, I’ll happily adopt them in place of any I’ve invented. I’m looking at you, Recording [↑](#footnote-ref-5)
5. We adhere to standard IT industry use of the term ***entity*** to denote a possibly persistent object or data structure having a unique identity, within some durable namespace or scope, in contrast to a ***named entity***, frequently shortened to *entity* in Uniphore documents and NLP papers and the like. See [2] for additional background. [↑](#footnote-ref-6)
6. See the References section near the end of this document. [↑](#footnote-ref-7)
7. aka “says”, including gestures and other signals [↑](#footnote-ref-8)
8. called ai\_value in the protobuf spec; this is a case where we’ve adopted an alternate term that hopefully better captures the semantics [↑](#footnote-ref-9)
9. a spoken word or phrase [↑](#footnote-ref-10)
10. has this been defined? [↑](#footnote-ref-11)
11. what if any client libraries do we want to provide to streamline the API? [↑](#footnote-ref-12)
12. UID = *unique identifier*, applies to any data entities [↑](#footnote-ref-13)