Dialogue & Discourse 16(3) (2025) 25–59 doi: 10.5210/dad.2025.303

A modular architecture for creating multimodal embodied agents with an episodic Knowledge Graph as an explainable and controllable long-term memory

Thomas Baier T.BAIER@VU.NL
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Selene Báez Santamaría SELENE.BAEZ.SANTAMARIA@GMAIL.COM
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Piek Vossen P.T.J.M.VOSSEN@VU.NL
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Editor: Hendrik Buschmeier

Submitted 11/2023; Accepted 04/2025; Published online 12/2025

Abstract

How can flexibility and control over the interpretation of multimodal signals by embodied agents be balanced? Flexibility means that agents respond fluently in any context, whereas control means that responses are transparent and faithful to goals and principles that are explicitly defined. This paper describes a modular platform to create multimodal interactive agents using an event bus on which signals and interpretations are posted as a sequence in time, but also provides control options to drive the interaction given specific intentions and goals. Different sensors and interpretation components can be integrated by defining their input and output *topics* in the event bus, which results in an open multimodal sequence-driven workflow for further interpretations. In addition, our platform allows us to define higher-level intents that control sequence patterns to achieve a goal. A key component is an episodic Knowledge Graph (eKG) that acts as a long-term symbolic memory to aggregate and connect these interpretations. This eKG establishes coherence and continuity across different interactions. Intents and the eKG make it possible to define different (embodied) agents and compare their behavior without having to implement complex software components for multimodal sensor data and design the control over their dependencies. In this paper, we explain the broad range of components that we developed and integrated into various interactive agents. We also explain how the interaction is recorded as multimodal data and how it results in an aggregated memory in the eKG. By analyzing the recorded interaction, we can compare agents and agent components and study their interactive behavior with people and other agents.

Keywords: Multimodal agent interaction; Modularity, flexibility, and control; Data sharing and technology sharing.

1. Introduction

Interaction among agents, both humans and AI, is especially complex when it occurs in real world contexts. It involves a complex psychosocial process between agents with intentions, capabilities, and

roles, and is related to the shared environment and space within which the interaction occurs. Recent approaches to conversational AI and robots are based on encoder-decoder architectures in which end-to-end systems learn to respond to encoded multimodal signals (Huang et al., 2020; Miyazawa and Nagai, 2023). Fu et al. (2022) mention two main issues with such encoder-decoder systems: they lack interpretability and control, resulting in unwanted behavior (hallucination, biases) that is difficult to repair.

Two things are essential to study interactions by different agents and processing components: 1) we need to be able to record the interaction as multimodal data so that we analyze, evaluate, compare and re-use interaction data, and 2) we need to be able to freely experiment with different components to test their impact on the interaction. This is specifically challenging because multimodal signals, such as images, sound, speech, faces and their expressions and gestures, might not be aligned, are noisy, and can be interpreted in many different ways, partially dependent on each other (Miyazawa and Nagai, 2023). Whereas other platforms such as PSI Bohus et al. (2017) and ROS Macenski et al. (2022), focus on handling streams of signals through open configurations of processing components, we augment such a platform with higher levels of abstraction and reasoning over these multimodal streams. To make sense of this multitude of signals and possible interpretations, a critical component is an episodic long-term memory in which interpretations are stored over time and which forms the basis for reasoning over and responding to new input. Such a sense-making module is essential to create transparent, trustful, and functional agents.

Previously, Vossen et al. (2019); Báez Santamaría et al. (2021) introduced a) EMISSOR to store sequences of multimodal signals with their annotations and b) episodic Knowledge Graphs (eKG) to store the accumulation of interpretations as symbolic triples. EMISSOR registers interactions as scenarios in which multimodal signals align using a temporal ruler to mark signals' start and endpoints. Segments within each signal are automatically annotated with interpretations, such as detected objects or faces through cameras, or entities and properties mentioned in speech. For example, recorded images in time can be annotated as a human face that is next identified as "Carl", who is the speaker of an audio signal (human voice), annotated as a text signal "I am from Amsterdam", which is further processed for its content. A visualization of an interaction represented in EMISSOR is included in the appendix A.

The eKG represents the accumulation of interpretations of signals over time as RDF[1] triples consisting of subject, predicate, and object identifiers (Internationalized Resource Identifiers or IRIs), e.g., the speaker *Carl* claiming: [`leolaniWorld:Carl`, `n2mu:live-in`, `leolaniWorld:Amsterdam`]. Such triples combine into more complex and rich knowledge structures and can easily be connected with other external knowledge such as DBpedia[2] specifying what is *Amsterdam* through another triple. By interpreting conversations and perceptions as triples, the agent not only records signals but also learns about people and the world over time through interaction. By representing interpretations as claims of sources as defined in the GRaSP model van Son et al. (2016); Fokkens et al. (2017); Vossen and Fokkens (2022), the eKG forms a so-called Theory-of-Mind Gopnik and Wellman (1992) by accumulating perspectives on claims in which possibly conflicting information holds relative to the sources of these claims. As a result, the eKG can deal with conflicting claims when sources disagree and perspectives on claims that change over time.

EMISSOR and the eKG together do not yet make an agent. An agent needs to pick up signals and call the necessary modules to process these before the interpretations can be rendered to EMISSOR

1. Resource Description Framework, `https://www.w3.org/RDF/`
2. `https://www.dbpedia.org`

or the eKG. Furthermore, an agent must respond to this input by generating output signals and actions. This paper describes a platform for creating different interactive agents whose interactions can be registered in both EMISSOR and eKG and can integrate any component to generate responses. The core of the platform is an **event bus**, which can be considered a *multi topic* input/output queue that connects time-based signals with interpretation components as separate software modules. Component APIs define the expected input and output event types, and the topics that a component is connected to -such as "audio-in" or "text-out"- can be defined freely and specified in the configuration of an agent. Components can be replaced against competing implementations as long as their APIs match. Likewise, different agents can be created easily by assembling components and configuring their connection to the event bus.

The primary signals originate from back-end components that read data from sensors at time points and publish them as events to a particular topic in the event bus. Different interpretation components consume these signal events (defined by their input topic) and publish the result of processing the input event(s) to their output topic(s) in the event bus as new events. These components can annotate segments from signals with interpretations, such as human speech or text, objects, and people perceived, ultimately pushing these interpretations to the eKG. Any agent is therefore triggered by the incoming signals and their processing by the integrated components. We call such an agent a **signal-driven agent**.

Although event-driven architectures are not new as such, our event bus platform differentiates itself by adding higher-level components that try to steer and control the signals by **intervention**. This can be done by 1) the so-called intention modules that define the goals as dialog states to be satisfied before moving on and 2) by pushing high-level interpretations of these signals to the eKG for reasoning within a Theory of Mind (ToM). The reasoning in the eKG results in knowledge states that are assigned to actions of the agents to improve the knowledge states. Whereas intents act as dialog states similar to a dialog management component, the eKG more openly and proactively drives the interaction.

Adding control mechanisms turns a **signal-driven agent** into a **control-driven agent**. Because of these control options, it is also more easy to incorporate (third-party) encoder and decoder modules into our platform to interpret signals or verbalize responses without suffering too much from hallucination and generalized biases. Different variants of agents will still render compatible multimodal data from their interactions within our framework as we store their interpretations and responses in EMISSOR and the eKG. Therefore, our platform can be used effectively to create and compare agents by analyzing the data rendered, both in EMISSOR and as an eKG (Báez Santamaría et al., 2022).

Although we published before about EMISSOR and the eKG, we never described how these are combined with an event-driven run-time system, which is the topic of this paper. This paper explains the general architecture and implementation of the Leolani platform, as opposed to the Leolani agent, which is one of the agents that could be built on the platform.[3] We also explain how different interactive agents can be created within the same platform using various components and demonstrate how we can add control to a **signal-driven agent**. Currently, we implemented a

3. Originally, Leolani was developed as a specific agent application to get to know you. In recent years, Leolani has evolved into a platform that can be used to create many different agents. We now use Leolani for the platform and Leolani agent for the implementation that uses the widest range of components currently available. See our Github repository for more details and instructions: `https://github.com/leolani/leolani-mmai-parent`

wide range of agents with no controlling options such as Eliza agent[4], Blenderbot (Shuster et al., 2022), or Llama Touvron et al. (2023), and with maximized control over the interaction such as the multimodal Leolani agent (Vossen et al., 2019), agents that play an "I spy" game or SPOTTER, which is a multimodal game to analyze referential expressions Kruijt et al. (2024).

The paper is further structured as follows. We first position our platform on related work in Section 2. In Section 3, we explain the overall architecture of the platform and its connection to EMISSOR and the eKG. Section 4 describes the components that are currently available and Section 5 describes the functions that are available to evaluate interactions. We report on the methods for analyzing and comparing interactions in Section 5. Section 6 explains how you can create your own agent either by combining existing components or by defining your own component. Finally, we present a technical specification on processing requirements, scalability, runtime latency, and real-time processing in Sections 5 and 6. All our code is available on Github under the Apache2.0 open source license from: `https://github.com/leolani`.

2. Related work

Recent advances in Deep Learning resulted in unprecedented performance in various technologies, also necessary for building intelligent interactive agents, ranging from language understanding, speech recognition, conversational skills, vision, to audio processing. Furthermore, fusion models combine multimodal input that is otherwise processed independently into a unified framework (Gao et al., 2020). The latest versions of OpenAI's ChatGPT (Wu et al., 2023) and Google Gemini (Team et al., 2023) combine these capabilities with conversational skills in longer contexts. Despite their impressive capabilities, and in addition to well-known problems such as hallucination and biases imposing unwanted interpretations and responses, these models are, however, still far from dealing with the complexity of physical real-world situations in which robots need to perform. Miyazawa and Nagai (2023) give an overview of how deep learning models are integrated in robot applications and research. They show on the one hand that state-of-the-art research in robotics relies more and more on leveraging these models but also that interacting with real-word settings or virtual simulations of these settings still requires additional modeling and integration for controllable behavior of agents.

Miyazawa and Nagai (2023) limit their survey to what we call **action robots**, which are agents that need to understand instructions and communicate about their task success with human instructors when dealing with some world. This greatly limits communication and social complexity, i.e. in most simulated worlds virtual agents do not encounter avatars representing real people or users. In the case of social agents and specifically **social robots** whose embodied realization needs to interact with people in the real world, situations are extremely complex and it is also a challenge to design agents that behave appropriately. The complexity of designing such agents is clearly demonstrated by the complex dialog flow of the agent designed by Stange et al. (2022) whose purpose is to generate verbal explanations for decisions in relation to its intentions. Besides a wide range of different components for understanding, perception, memory, planning and acting (both physically and verbally), their architecture also shows complex dependencies and decision points connecting these components through classical AI components such as a Dialog Manager, Decision Engine and Strategy Manager. Remarkably, their design does not consider the integration of Deep Learning technology in the architecture, which would add another layer of nontransparency to the decisions made by the agent due to their lack of predictability as black-box models.

4. `https://github.com/leolani/eliza-parent`

In order to be able to design and build agents with embodiment that can act physically and socially in real-world settings, a platform is needed in which researchers and developers can freely experiment with integrating any component for interpreting multimodal signals, building memories, making decisions in relation to intentions, and rendering physical and social actions. Microsoft created a Platform for Situated Intelligence, PSI[5] for this purpose (Bohus et al., 2017). PSI offers multimodal data visualization and annotation tools, as well as processing components for various sensors, processing technologies, and platforms for multimodal interaction. PSI models multimodal situations and interactions within and comes close to a comprehensive solution. PSI is a software integration platform through which developers can share modules using a streaming architecture for signal annotation. However, interactions are not stored in a shared representation, and the platform cannot be used to share experimental data independently of the platform itself.

In the context of (spoken) dialog systems, a generic model for incremental processing was proposed by Schlangen and Skantze (2011). ReTiCo (Michael, 2020) is a Python implementation of this incremental model as a framework for creating systems and simulations of spoken dialog. It provides a range of incremental modules based on services like Google ASR, Google TTS and Rasa NLU. Although generic and open with respect to the integration of processing modules, these approaches model incrementality as a sequence of units that trigger each other. They do not consider higher-order modules that "oversee" such sequences and define patterns for the purpose of achieving intentions as we do in our platform enabling cross-platform comparison.

ReTiCo focuses on spoken dialogs in which audio signals form a natural sequence. If other modalities are considered, such as image or video, temporal alignment of signals is necessary as provided in PSI. Kennington et al. (2017) therefore combine ReTiCo and PSI for spoken dialogue systems in visual contexts. In their approach, sequential incremental interpretation by different modules is provided by a mixture of modules in both frameworks. Whereas Kennington et al. (2017) combine two different frameworks, PSI implemented in C# for Windows platforms and ReTiCo in Python for Linux, we provide an integrated framework that can run on any platform while using a distributed server architecture. Furthermore, their approach requires a dialog management component that needs to take decisions at many modules-interface points in the interaction, which puts a heavy burden on the design of an agent. In our approach, we allow for both low-level and high-level control over multimodel signal interpretation, both in terms of agent intent reasoning as well as in terms of an episodic Knowledge Graph that creates coherence and continuity across encounters. Controlling the flow of interaction can be defined easily in many different ways without changing the code and interaction between different processing components.

An older comprehensive robot platform worth mentioning here is openEASE[6], which is a web-based knowledge service between a robot and a human that comprises episodic memories (Beetz et al., 2015). It produces semantically annotated data of manipulation actions, including the agent's environment, the objects it manipulates, the task it performs, and the behavior it generates. It is provided with a query language and inference tools that allow reasoning about the data and answering queries regarding what they did, why, how, what happened, and what they saw. EASE uses the so-called NEEMS (Narrative Enabled Episodic Memories) as episodic memories. NEEMS consists of a video recording by the agent of the ongoing activity. These videos are enriched with stories about actions, motion, their purposes, effects, and the agent's sensor information during the activity. Knowledge is represented through prolog predicates as annotations of sequences. EASE is not

5. https://github.com/Microsoft/psi
6. https://www.open-ease.org

data-centric, but a service platform that uses a knowledge database as a back-end. The database can be explored through prolog queries. The focus of openEASE is on physical interactions and not on conversations with complex referential relations between expressions, situations, and episodic knowledge of past experiences.

Recently, open distributed platforms for the creation of multimodal agents have been proposed by Kennington et al. (2020) and Chiba et al. (2024). These platforms allow for the incorporation of new (multimodal) modules for signal processing that can also be distributed, making the architecture flexible and expandable. These approaches come close to our approach but lack the incorporation of an episodic Knowledge Graph that provides control as in our system.

None of the above platforms provides an easy way to define control over the incremental signal processing nor do they allow for easy integration of state-of-the-art Deep Learning technology. Our approach distinguishes itself by pairing open integration and processing of multimodal streams of signals in a single framework with options for adding higher-order intents for dialog management and episodic memories for incremental reasoning, long-term coherence, and continuity. By modular integration of such **controlling** modules, our platform makes it easy to experiment with Large Language Models and Fusion models without giving up control. Similar to Retrieval Augmented Generative approaches, the eKG and the intents can be used to drive the interpretation and also the generation of signals by these models.

3. event bus architecture

The architecture of our platform is built around two central concepts, one being EMISSOR as a shared framework to represent data in a multimodal interaction and the other the communication between the modules through an event bus using a publish/subscribe model. This provides a high level of decoupling between modules in our system, as they only need to be concerned about their individual input and output data, ensuring it is in the right format and not about the context they are used in. This, on the one hand, allows us to use them as flexible building blocks for applications. On the other hand, integration of external software components into our platform can be achieved by adding a thin integration layer (or wrapper) that only needs to take care of converting input and output data to the format and implement their connection to the event bus. As an example, an object detection module that is integrated in the platform would take raw image data from a received event with an EMISSOR *ImageSignal*, perform object detection on these raw data, and publish an event containing the detected objects as an EMISSOR *Annotation* on the original *ImageSignal*, making them available for further processing to other modules. Depending on the used event bus implementation, this can even connect different run-time systems, allowing one to connect components across different platforms.

Figure 1 shows a schematic representation of the event bus architecture. The central component is the event bus itself, to which signals and interpretations are pushed either by back-end components (on the left side) or interpretation components (on the right side). The event bus is represented as a vertical (gray) bus to emphasize that it has a temporal dimension where events are processed in a sequence. The labels on the arrows between the event bus and the components denote the type of event transmitted. The vertical lines in the event bus symbolize different topics used by the individual components to define the event flow through the application. The topic labels to which components are connected are denoted in italics next to the arrows. By defining the input topic and the output topic for a module, we can condition which components become active for the event bus (input topic

specification) and what the potential components are that follow their output topic. This allows us to define the possible interactions from the bottom up.

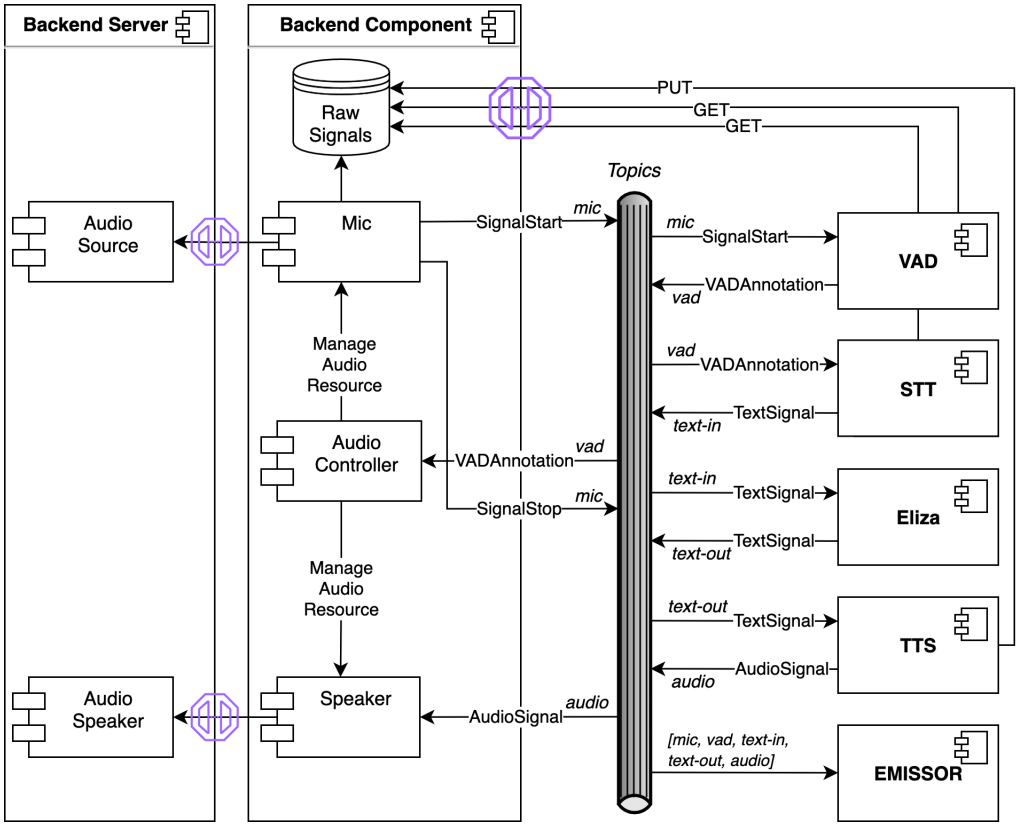

Figure 1: Schematic representation of the event bus architecture showing backend components to read from sensors and fill the queue in the event bus and interpretation components for annotating published signals. In this schema, only audio signals are processed with an Eliza module for generating a response. Interaction data is recorded from events by the EMISSOR component.

Examples of back-end components are microphones, audio controllers, and cameras. Sensors connected to back-end components can be directly linked to the system or obtained from remote servers. This schema shows how a microphone produces an audio signal with a start and a stop point. Some examples of interpretation components are shown on the right side of the event bus. The VAD component at the top applies voice-activity detection to an audio signal published by the microphone, which reports back to the event bus whether the audio is human speech, annotating the corresponding segment in the audio. The Speech-to-Text (STT) component applies speech recognition to any audio signal annotated as human speech and publishes the transcript as a text signal to the event bus. In this example, the text signal is picked up by an implementation of the Eliza chatbot (Weizenbaum, 1966), which checks it for triggers and returns a response as another text signal. Instead of Eliza, any other module can be used to generate a response either directly to the text signal or indirectly through interpretations of these signals. Finally, the text-to-speech (TTS) component picks up the

text response and converts it to an audio signal. The back-end speaker listens to the event bus for any audio signal to be output. An audio controller between the microphone and the speaker mutes the mic when speaking and vice versa. Any data produced by the sensors or the components as signals is recorded in an EMISSOR data structure.

3.1 Enabling reasoning

The main application can function with just the event bus and some simple components, taking in signals and generating signals as a response. However, the behavior of such a **signal-driven agent** is, however, primarily **reactive** and not **reflective** of the information conveyed during an interaction. Examples of such agents are Eliza which responds to trigger words or generative Language Models such as Blenderbot, DialogGPT or Llama that generate the most-likely continuation of the dialog through implicit reasoning. To turn such an agent into a **control-driven agent**, we need to interpret the signals at a more abstract symbolic level. This can be done in the form of annotation of (segments of) signals or by the explicit knowledge conveyed and the implications derived so that the agent can reason over it. The reasoning results in a decision to create a response. Symbolic signal annotations can directly be used by higher-level intentions that check for such occurrences to move from one dialog state to the next; e.g., obtaining your name means we can now chat. In the case of our eKG the response is based on reasoning over the knowledge quality in the eKG.

Figure 2 shows the interconnections across the three components: event bus, EMISSOR, and eKG. The three components are aligned so that each interaction in the event bus generates a corresponding multimodal signal representation in EMISSOR and, if the interpretation yields knowledge, also an RDF triple in the eKG.

EMISSOR stores interactions as **scenarios** with metadata in JSON files for signals in each modality: images, audio, text, and RDF-triples; further details are described in Báez Santamaría et al. (2021). The metadata grounds the signals to a temporal ruler and defines any annotation of **segments** within a **signal** as interpretations. The raw signals themselves are stored as separate files on disk.[7]

In the eKG model, interactions are initialized as instances of situational `eps:contexts` grounded in time and space. Contexts correspond to scenarios in EMISSOR. Within a context, a series of `sem:events` are represented, which are grounded in time like the signals in EMISSOR. We distinguish between conversation and perception events:

Conversation: Conversation events in the eKG are structured as `grasp:Chat` instances containing `grasp:Utterances` in which participants in the interaction mention `grasp:Statements` or formulate questions. The information mentioned in `grasp:Statements` is stored in a `gaf:Claim`, which is a named graph attributed to a speaker (`grasp:wasAttributedTo`) and that hold certain perspectives (`grasp:Attribution`). These perspectives are properties such as `grasp:CertaintyValue`, `grasp:PolarityValue`, `grasp:SentimentValue`, `grasp:EmotionValue`. Currently, we included modules that extract such perspective values from the text generated by speech-recognition as well as modules that extract these values from facial expressions.

Perceptions: In contrast, perception events in the eKG are structured as `grasp:Visual` instances containing `grasp:Detectionss` which could be objects or people. These detections are considered to be instances of `grasp:Experiences`. Similarly to conversation events, the infor-

7. We also consider the RDF triples extracted from text as a modality stored on disk in EMISSOR.

mation obtained from these `grasp:Experiences` is stored in named graphs as `gaf:Claims` with triples for the labeled output of object/face-recognition components. In this case, the information is `grasp:wasAttributedTo` sensors, and the `grasp:Attributions` are restricted to `grasp:CertaintyValues` only (e.g. based on confidence scores). Perception events therefore reflect the awareness of an agent of objects and people in the context. Perceptions correspond to image signals in EMISSOR.

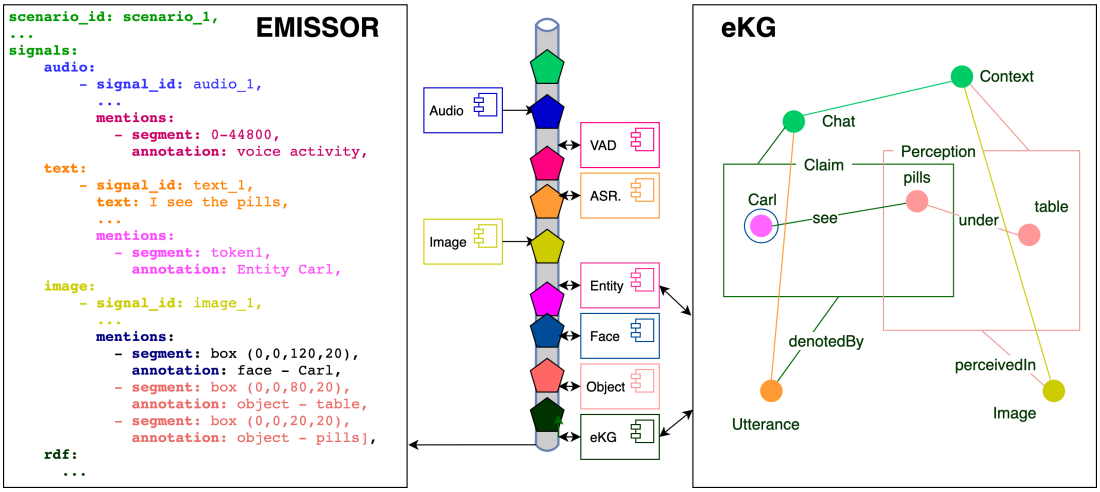

Figure 2: Schematic overview of relations across data structures in EMISSOR, the episodic Knowledge Graph and the event bus.

As shown in Figure 2, the event bus connects the specific components that make up an agent and at the same time displays the data for EMISSOR and for the eKG. These components can also push data, such as annotations or triples, to the event bus itself so that other components can take these as input for further processing. The event bus is thus populated not only by the sensors but also by the interpretation components. In the next section, we give an overview of the main components in our platform and explain in more detail how they interact (for a complete overview, we refer to our Github repository). In the next Section 5, we explain how interactions can be analyzed, compared, and evaluated, and, in Section 6, we explain how new components can be added and how you can create your own agent.

4. Components available

Interaction task is broken down into a large number of smaller tasks that operate on specific input events taken from the event bus and push their output back onto the event bus. So far, we have developed the following components either by creating a wrapper for third-party systems or by developing components in-house. We grouped the components by their underlying input signal:

1. Audio signal

   voice activity detection [8] classifying an audio signal as speech, output is an annotated audio signal pushed to the event bus and saved to EMISSOR as an audio annotation.

   automatic speech recognition [9] transforming speech to text, various models can be chosen which generate a text output signal to the event bus and EMISSOR, among them Whisper (Radford et al., 2023).

   speaker identification (under development) identifying the speaker based on a sample of their voice, output is a speaker's identity (as an IRI and their name), and an audio signal annotated with the source pushed to the event bus and EMISSOR but also registered in the eKG as an encounter.

2. Text signal

   mention detection [10] detecting names and potential objects in a text signal using spaCy[11], output is an annotated signal with named entities and object mentions, pushed to the event bus and EMISSOR.

   mention identification [12] identifying instances in the knowledge graph for mention annotations in a text signal, output is an identifier (IRI), pushed to the event bus and EMISSOR. Corresponding RDF triples (IRI `gaf:denotedBy` `gaf:utterance#offset`) are pushed to the eKG.

   triple extraction [13] detecting RDF triples in a text signal; various implementations can be selected; output are RDF triples pushed to the event bus, EMISSOR and the eKG. Several in-house systems have been developed (among which a conversational extractor that considers sequences of turns) but also Stanford's Open IE[14].

   query extraction [15] detecting a question in a text signal and converting this into a SPARQL query; output is a SPARQL query sent to the eKG and possibly other knowledge graphs.

   emotion detection [16] detecting basic emotions in a text signal; output is an emotion label[17] pushed to the event bus and EMISSOR as an annotation and possibly as a perspective to the eKG.

   dialog act detection [18] detecting the dialog act for each utterance through two different modules, outputting one of many different conversational dialog acts to the event bus and EMISSOR (Yu and Yu, 2019; Chapuis et al., 2020).

8. https://github.com/leolani/cltl-vad
9. https://github.com/leolani/cltl-asr
10. https://github.com/leolani/cltl-knowledgeextraction
11. https://spacy.io
12. https://github.com/leolani/cltl-knowledgelinking
13. https://github.com/leolani/cltl-knowledgeextraction
14. https://nlp.stanford.edu/software/openie.html
15. https://github.com/leolani/cltl-questionprocessor
16. https://github.com/leolani/cltl-emotionrecognition
17. BERT trained with the GO annotations (https://huggingface.co/bhadresh-savani/bert-base-go-emotion) and RoBERTa find-tuned with MELD and WASSA data Kim and Vossen (2021)
18. https://github.com/leolani/cltl-dialogueclassification

3. RDF signal

   knowledge representation for claims [19] posting RDF claims to the eKG; output is JSON-LD as responses to the changes in the eKG by digesting new claims pushed to the event bus.

   knowledge representation for queries [20] posting SPARQL queries to the eKG or other graphs; output are RDF triples as the result of the query pushed to the event bus.

   response generation [21] verbalising qualitative reflections on changes in the eKG (responses); output is a natural language response as a text signal, pushed to the event bus and stored in EMISSOR.

   language generation [22] verbalising SPARQL query results; output is natural language as a text signal, pushed to the event bus and stored in EMISSOR.

4. Image signal

   object recognition [23] detecting multiple objects in images using Yolo[24]; output is bounding boxes and object labels as an annotated image signal, pushed to the event bus and stored in EMISSOR.

   face detection [25] detecting human faces in an image, the output are bounding boxes with the estimated age and gender as an annotated image, pushed to the event bus and stored in EMISSOR.

   face identification [26] identifying people from their face, the output is an identifier (IRI and a label as name), pushed to the event bus and stored in EMISSOR. Encounters are also registered in the eKG as perception events.

   face emotion detection the same component that does text emotion detection also includes a module "emotic" Kosti et al. (2019) for contextual face emotion detection.

4.1 Processing audio signals

The components that process the audio signal were discussed in detail in Section 3 when explaining Figure 1. In a nutshell, the **voice-activity-detection** component detects portions of human speech in audio, which the **automatic-speech-recognition** transcribes. Eventually, these components push a text signal to the event bus, which is rendered from an audio signal for text processing components.

4.2 Processing text and rdf signals

The components that operate on text signals are more complex. They generate different interpretations and types of outputs and combine with components that operate on RDF triple signals.

19. https://github.com/leolani/cltl-knowledgerepresentation
20. https://github.com/leolani/cltl-knowledgerepresentation
21. https://github.com/leolani/cltl-languagegeneration
22. https://github.com/leolani/cltl-languagegeneration
23. https://github.com/leolani/cltl-object-recognition
24. https://github.com/ultralytics/yolov5
25. https://github.com/tae898/age-gender
26. https://github.com/leolani/cltl-face-recognition

Various Natural Language Processing (NLP) modules are called within different components, among which spaCy(Vasiliev, 2020), NLTK, StanfordNLP (Angeli et al., 2015) and various transformer models. In addition to their basic processing, the **triple extraction** components output various annotations, for example, mentions of people and objects for which referring expressions must be resolved to known people and objects. Referring expressions can be names (Carl), common noun phrases (the waiter), or ambiguous pronouns. By reasoning about the context and previous encounters, **mention-identification** components aim to establish the referent of these expressions. Resolved expressions are converted to IRIs incorporated in triples. These can be registrations of mentions, e.g., *Carl talking about Carla*, or as part of triples expressing a property of an individual: [`leolaniWorld:Carla`, `leolaniWorld:live-in`, `leolaniWorld:Amsterdam`] . Pronouns such as "I" and "you" are resolved to the speaker and the addressee.

Whenever a `gaf:Claim` is posted to the eKG, the **knowledge-representation-for-claims** component generates a reflective response to the claim. This involves running various precoded SPARQL queries to detect knowledge gaps, conflicts, uncertainties, novelties, analogies, and possible generalizations. The **response-generation** component selects a result to formulate a response to try to improve the eKG, for example, to fill gaps or resolve conflicts and uncertainty. The triples are verbalized as natural language text in a text signal. Depending on whether the eKG responder is chosen within the framework, the response is converted into a new text signal attributed to the agent.

The previous route through the event bus applies to signals classified as statements. If a text signal in the event bus is classified as a question, the **query-extraction** component extracts a SPARQL query from the text signal that the **knowledge-representation-for-queries** component posts to the eKG (or any other triple store). The **language-generation** component takes the query's result (one, many, or no triples) to verbalize it as a text signal. Different answers are generated according to the type and number of query results. Furthermore, answers consider the status of the knowledge, e.g. how certain, who is the source, and when was it mentioned. In addition to querying the eKG directly, we also implemented alternative components that can query a database of responses (questions about the agent), external resources (consulting Google search, Wikipedia, news, the weather or Wolfram Alpha), or visual information that was perceived by the agent.

In addition to processing statements and queries, some components only annotate mentions or emotions expressed in the text (without necessarily being involved in a triple). The **mention-detection** component detects things in the text that (can) exist in the physical world, which are object that the object recognition can detect. Furthermore, it detects named-entity expressions in the text. These mentions are saved as annotations of the text signal in EMISSOR and pushed to the event bus for further processing. Next, the **mention-identification** component picks up entity mentions and tries to resolve their identity given the entities registered in the eKG, e.g. by checking names, the contexts, or resolving pronouns' co-reference. Identities are pushed to the event bus and registered as `gaf:Mentions` in the eKG. Possibly, a new identity can be established and a new IRI is stored in the eKG with the properties that can be inferred.

The **emotion-detection** component interprets a text signal and pushes an emotion to the event bus as well as its annotation to EMISSOR. The emotions expressed by the interlocutor represent potential perspectives of the source on the current situation or the information expressed.

4.3 Processing image signals

Finally, the **object recognition** and **face-detection** components take image signals as input to detect objects and faces with bounding boxes. These are saved as annotations of image signals in EMISSOR and pushed to the event bus. From the event bus, the **face identification** component establishes the identity of people by comparing it with the faces of known people. Unknown faces are saved as new identities with inferred properties, such as sex and age. A new identity is pushed to the event bus after which it is picked up by a component that asks for a person's name. The image-signal-processing components either produce annotations pushed to the event bus and/or encoded in EMISSOR or produce an IRI with properties: name, age, gender, and registered perception event in the eKG.

4.4 Sample input-output signal sequences

To illustrate how sensor signals are processed, we show some schematic representations of typical sequences of input and output events with their payload type and input/output topics denoted above the arrows[27].

1. audio (signal) $\xrightarrow{\text{mic}}$ audio (annotation) $\xrightarrow{\text{vad}}$ speech (annotation) $\xrightarrow{\text{asr}}$ text (signal) $\xrightarrow{\text{text-in}}$ mention (annotation) $\xrightarrow{\text{entities}}$ IRI (annotation) $\xrightarrow{\text{linking}}$ **claim triple** (eKG) $\xrightarrow{\text{triples}}$ response triples (eKG) $\xrightarrow{\text{response}}$ text (signal) $\xrightarrow{\text{text-out}}$ audio (signal)

2. audio (signal) $\xrightarrow{\text{mic}}$ audio (annotation) $\xrightarrow{\text{vad}}$ speech (annotation) $\xrightarrow{\text{asr}}$ text (signal) $\xrightarrow{\text{text-in}}$ mention (annotation) $\xrightarrow{\text{entities}}$ IRI (annotation) $\xrightarrow{\text{linking}}$ **query triple** (eKG) $\xrightarrow{\text{triples}}$ response triples (eKG) $\xrightarrow{\text{response}}$ text (signal) $\xrightarrow{\text{text-out}}$ audio (signal)

3. audio (signal) $\xrightarrow{\text{mic}}$ audio (annotation) $\xrightarrow{\text{vad}}$ speech (annotation) $\xrightarrow{\text{asr}}$ text (signal) $\xrightarrow{\text{text-in}}$ response (**Agent response database**) $\xrightarrow{\text{response}}$ text (signal) $\xrightarrow{\text{text-out}}$ audio (signal)

4. audio (signal) $\xrightarrow{\text{mic}}$ audio (annotation) $\xrightarrow{\text{vad}}$ speech (annotation) $\xrightarrow{\text{asr}}$ text (signal) $\xrightarrow{\text{text-in}}$ response (**Eliza**) $\xrightarrow{\text{response}}$ text (signal) $\xrightarrow{\text{text-out}}$ audio (signal)

5. audio (signal) $\xrightarrow{\text{mic}}$ audio (annotation) $\xrightarrow{\text{vad}}$ speech (annotation) $\xrightarrow{\text{asr}}$ text (signal) $\xrightarrow{\text{text-in}}$ response (**Llama**) $\xrightarrow{\text{response}}$ text (signal) $\xrightarrow{\text{text-out}}$ audio (signal)

6. audio (signal) $\xrightarrow{\text{mic}}$ audio (annotation) $\xrightarrow{\text{vad}}$ speech (annotation) $\xrightarrow{\text{asr}}$ text (signal) $\xrightarrow{\text{text-in}}$ response (**Wolfram Alpha**) $\xrightarrow{\text{response}}$ text (signal) $\xrightarrow{\text{text-out}}$ audio (signal)

7. audio (signal) $\xrightarrow{\text{mic}}$ **speaker** (annotation) $\xrightarrow{\text{speaker}}$ **IRI** (annotation) $\xrightarrow{\text{identity}}$ perception (eKG)

27. Topic names are exemplary and can be configured in the application

8. image (signal) $\xrightarrow{\text{cam}}$ **face** (annotation) $\xrightarrow{\text{face}}$ **IRI** (annotation) $\xrightarrow{\text{identity}}$ perception (eKG)

9. image (signal) $\xrightarrow{\text{cam}}$ **object** (annotation) $\xrightarrow{\text{object}}$ **IRI** (annotation) $\xrightarrow{\text{identity}}$ perception (eKG)

4.5 Responding

Components that respond to processed signals in the event bus play an important role in defining the functionality of the agent. It is possible to integrate existing agents as response components, as long as you can define a wrapper that converts events from the event bus into the necessary input and the output of the agent into a signal that can be rendered by the embodiment. Currently implemented examples are Eliza, Blenderbot, and Llama that only require text signals as input and create a text signal as response. Others could be task-based, such as recommender systems, Q&A (Wolfram Alpha) or e-commerce services.

A more advanced responder is the eKG itself, which has been extended with modules that assess each change to the graph as a result of the interaction and generate a possible response as a result of this assessment. Since the graph is updated for a large variety of interpretations coming from different modalities, the agent is also sensitive to many different aspects of the interaction. We defined a wide range of graph updates to which the agent could respond. Typical examples are knowledge gaps in case the agent hears about things it does not know yet, uncertainty that is either expressed by sources or inferred from the data, conflicts when sources claim different things, mentions and perceptions of relevant things, analogies, novelty relevant for certain people, and many more. Assessments are called **thoughts** and are represented as the output of specific queries on the graph. Given the type of thought, we define different drives to respond to each, which is formulated as a text response given the output of the query as input Báez Santamaría et al. (2021). Typically, a single turn can thus produce a large number of thoughts, and specific strategies can be deployed to select the most effective thought, e.g., using reinforcement learning. What is effective also depends on the knowledge exchange during the interaction and hence on the user as a source.

The eKG-based responder can be adapted and extended by defining additional queries, specifying selections of thoughts to respond, or by reinforcement learning. In general, a wide range of thoughts and drives results in surprising and spontaneous behavior, whereas a small range results in systematic and rigid behavior that could be focused on specific tasks only.

4.6 Crossmodal awareness

In addition to multimodal signal processing, the Leolani platform provides various ways for cross-modality processing:

Alignment of segments Segments of signals in different modalities are defined using the same temporal ruler and likewise can be related temporarily: disjoint, adjacent, overlapping, or inclusive. These relations define contexts for cross-modal interpretations.

Sharing interpretation labels to the extent that mentions and perceptions share their annotation labels, e.g. **person** as a perceived image and as a named-entity category in text, this can create further coherence relations on top of coherence through segment alignments.

Sharing of identification labels components that produce identities for different modality signals can share these identities through the eKG. For example, face recognition is associated with an

eKG identifier and a name as a label, while the same name mentioned in text can be mapped to the same identifier in the eKG, resulting in all interpretations being aggregated across modalities to the same identity.

Consider the following example to demonstrate the cross-modal alignment. The agent is communicating with a person identified as `leolaniWorld: Carl`. During this interaction, the agent perceives an image of another person that is unknown. Close in time, preceding, overlapping, or following this image signal, the agent receives an audio signal from Carl. Processing the audio signal yields the text "That is my sister Carla". The object recognition produces the label **person** for the image and the named entity recognition detects a **person** entity in the text. Further text processing will pick up the deictic reference "that" and the identity expression "is my sister Carla". If no person exists in the eKG with the name Carla, a new identity `leolaniWorld:Carla` is created with the relation `leolaniWorld:sister-of` to `leolaniWorld:Carl`. The face identification module is then activated to create a visual representation that identifies the new unknown face. In this example, the temporal segment alignment of the image and the audio/text signal, as well as the shared interpretation labels, create an alignment across the modalities that an agent could use to combine all three cross-modal interpretations to yield one coherent result. Other scenarios can be easily modeled, e.g. if Carla was mentioned long before she was perceived or the other way around. If other people with different visual properties are named Carla, this could trigger a disambiguation intention and initiate communication by the knowledge linking module, depending on how similar or different the visual information is and or the signals are disjoint.

Besides the cross-modal interpretation of signals, we are working on components to aggregate these interpretations into overall situations, called contexts, that assume certain degrees of coherence and consistency. In every interaction, a new instance of a context is created in which time and space are defined Vossen et al. (2019). These contexts exist for the duration of an interaction and provide the basis for cross-modal consistency and permanence. Within a context, people and objects are perceived and mentioned within the same location as in a space. The location will be identified using the people and objects within it, in addition to a visual spatial map, but the objects, and potentially also the people, can also be identified given the location. Locations and objects/people are in a dualistic relation. Finally, the same location can have multiple interactions with the same or different people. Each interaction represents a different scenario following a specific conversational flow and logic, while they can take place in the same physical location, with some objects remaining the same while others change.

Context Overarching data structure for defining situational awareness.

Location and object identity Locations are identified across encounters using visual maps that include the identities of objects and people within and vice versa.

Scenarios Multiple interactions can take place within the matching contexts, each defining a unique scenario but possibly sharing the same location and objects within.

The identities of locations and objects within contexts are strongly intertwined Vossen et al. (2019). Different locations may look very similar, and the same holds for different objects. Across contexts, identifying the location and objects will involve matching properties as perceived in earlier encounters. In case of a strong match, identity can be assumed, in case of a moderate match, the

agent may ask for confirmation, and in the case of no match above a threshold, identity needs to be established with the help of the human user through communication.

A specific location in which the agent is activated could be an office space. The agent creates a new context instance in the eKG to aggregate interpretations within this context. After scanning the space and objects and people within, the agent compares this information with previous contexts in the long-term memory of the eKG and the corresponding sensor data (images) saved in EMISSOR. These memories are based on all previous encounters and the interpretations of the sensor data, e.g. repetitive interactions taking place in the same office identified in the past. The agent tries to align the identities of the location, people, and objects in the current context with the contexts of the past. Likewise, the agent may recover the identity and name of the office with various properties such as who owns it, what specific objects are still there, which ones are missing, and which ones are new. Object identity is complex, as many phones, chairs, tables, laptops look very similar, and some things move easily, whereas others do not. If there is not sufficient evidence for a match, a location needs to be considered as a new space. The matching and corresponding identities have a considerable impact on the dispersion and fusion of information in the eKG. With each perception and interpretation, a choice needs to be made as to whether perceptions are different, the same, or vary depending on how permanence is defined across locations. The platform allows for different approaches to establish location and object identification in relation to the eKG in terms of different granularities that can vary on a spectrum from all different to all the same. In future work, we will develop components that find an optimal balance between these extremes. Finally, it should be noted that the agent, at any point, could communicate uncertainty about identities to people to obtain confirmation.

Contexts, signals segments, and their annotations are represented in EMISSOR data structures, but also in the eKG. Multimodal agents can be designed to use either representation to arrive at specific cross-modal interpretations or ignore it. However, the more specific and aligned the interpretations are, the more dense and rich the information is for an agent to respond.

4.7 Higher level intentions

The above components become active when the input topic requirements are met, making a **signal-driven agent** that always responds to the presence of input topics in the event bus. However, our event bus architecture allows one to define higher-level intentions that represent the behavior of the agent to fulfill an end task or goal, that is, a **control-driven agent**. These intentions can be prioritized and remain active until resolved. For each intention, we can specify which components are actively listening to the event bus and which follow-up intentions are published once a task is completed or a certain goal is reached. The above components are thus activated only within these intentions. Intentions provide dialog management control over the interaction.

The SPOTTER agent (Kruijt et al., 2024) is an example in which the dialog flow for playing a reference game is fully controlled by intents that define dialog states. In this game, references to people need to be resolved to position them correctly in a row between a human and an agent. In Appendix 6, we show the dialog flow diagram for the game taken from Kruijt (2025) where the control is defined by higher-level intentions using Conversational States and Dialog Management decisions.

Another example is given by Vossen et al. (2024), in which an agent communicates with a person to learn about their daily activities to fill the knowledge gap between the previous conversation and the current, given the long-term history of the person. The agent uses the eKG to learn what is

not yet known, what is expected, and what is possible, which primarily drives the communication. The purpose of this activity monitoring is to obtain a picture of a person's well-being over time. Again, intents are used as Conversational States for the degree of saturation to be reached (how many activities and how much information about each should be known) and whether there is a further need to ask questions, see the Appendix 7.

Below is an overview of different high-level intentions on the platform.

Getting-to-know-you After detecting a human face in an image signal, the agent tries to identify the person to great him/her by name or to get to know a new person by asking for a name. Adding this identity to the eKG triggers more questions to learn about this person.

Giving consent Ask people permission to keep the data and share them for research. If no consent is given, the agent will remove any data (EMISSOR and eKG) and stop the interaction. This scenario can precede the get-to-know-you scenario.

Leolani-agent An agent that interacts freely with users relating interpretations to identities (IRI) and triples (Vossen et al., 2019). The Leolani-agent can be configured to follow certain thoughts and drives, such as filling knowledge gaps, and will remain proactive until these are resolved.

Eliza Takes text signals as input and generates a text response using Eliza trigger patterns and hard-coded responses. Eliza has no other goals than to respond to triggers endlessly.

Blenderbot Process text signals with Blenderbot (Roller et al., 2020). Blenderbot is a generative model trained with different conversational data to generate a human-like response. Blenderbot has no other goal than to respond to a user, just as Eliza, and will not initiate communication.

LLama Process text signals with Llama (Touvron et al., 2023). Any type of instruction can be given to adapt Llama. Also, Llama has no other goal than to respond to a user, just as Eliza, and will not initiate communication.

AboutAgent Process text signals by mapping questions to a database with information about the agent. AboutAgent has no other goals except to answer user questions about the agent.

Spotter A game played over several fixed rounds during which two players (a human and a robot) need to communicate about the differences in pictures visible to each player separately. The goal is to align the images and complete the game with a high score. Referring expressions are analyzed in relation to the common ground that is created between the players Kruijt et al. (2024).

Getting-to-know-more-of-you An agent that converses with people to learn about their activities of daily life. The agent uses the episodic Knowledge Graph to reason about activities from the past and the period between the current and the previous conversation to fill the gap Vossen et al. (2024). The goal is to monitor people's life and well-being over time.

Standup-comedian An agent that interacts through scripted protocols to act as a stand-up comedian pulling jokes and riddles from Llama but following the rules of the game.

Goodbye When the text signal contains a goodbye cue, it ends the scenario and says goodbye to the participant. The agent can start a new scenario with any person present.

Similarly to the above intents, it is easy to add any task-based intent to fill slots in a form, to retrieve information from an external database, or play a game. Both components and higher-level intentions can be created and combined freely. In part, this provides control over certain goals or tasks to be completed, but it also allows flexible and spontaneous behavior. In the next section, we describe several ways in which interactions can be analyzed and evaluated. We also provide details about the scalability and run-time latency of agents developed in the platform. Finally, we explain how to design your own agent by adding components and high-level intentions. All the code for the platform with the currently developed components is available on Github: `https://github.com/leolani` under Apache2.0 license. A good starting point for building an agent is the repository: `https://github.com/leolani/cltl-combot`.

5. Interaction evaluation

Our platform provides various ways to analyze and evaluate interactions, either through the recordings by EMISSOR or the representation in the eKG. First, EMISSOR captures multimodal streams of signals by storing raw signals (audio and image) and basic metadata on their temporal alignment. However, many modules from the Leolani platform can also be used to annotate interactions outside the event bus framework, such as object and face detection, speech recognition, emotion detection from face or from turns, dialog act classification, entity mentions in text, and triples expressed through turns. What is needed is to represent the sequence of signals in EMISSOR. When annotated, we can analyze the sequence of signals in terms of its annotations. If the triples are also added to an eKG, we can measure the incremental growth of the graph during the interaction. Báez Santamaría et al. (2022) describe such an analysis for a diverse set of interactions by students with an implementation of Leolani (human-agent), Blenderbot interacting with the same Leolani implementation, Blenderbot interacting with Eliza (agent-agent) and scripted human-human interaction between actors in the Friends sitcom. The EMISSOR and eKG representations are either created during the interactions or extracted a-posteriori, which resulted in comparable representations showing differences and similarities across the different interactions. Likewise, Krause et al. (2023) applied this analysis to a wide range of existing conversational datasets, ranging from task-oriented conversations such as MultiWoz (Budzianowski et al., 2018) to fully open conversations between people as in the Personal Events in dialog dataset (Eisenberg and Sheriff, 2020).

In addition to a structural and graph-based analysis of interactions, it is possible to add reference responses to replace system responses and apply standard metrics, such as BLUE, ROUGE, METEOR, and BERTScore as provided in the Google evaluate package.[28] Following Mehri and Eskenazi (2020), transformer models, such as BERT, RoBERTa, or their fine-tuned USR model, can be used as a reference-free evaluation method by measuring the likelihood of system and user responses given the preceding context. Correlation functions are provided to compare any human turn evaluation with the graph and reference-free scores.

Finally, the platform provides a function to plot interaction sequences over turns mapping annotations of dialog act, likelihood, and emotions on a negative and positive scale. Dialog acts such as **negative answer**, **complaint** are considered negative whereas **positive answers**, **appreciation** are positive. A similar mapping is given for the assigned GO emotions (Demszky et al., 2020). In the case of the turn likelihood, a threshold is set below which we consider the score as negatively expected and above which as positively expected by a model. Appendix B shows two generated

28. `https://huggingface.co/docs/evaluate/en/index`

example plots: the first plot shows the interaction sequence with a Pepper robot and the second plot with an agent implemented on the AI2Thor platform (Kolve et al., 2017) to find objects.

To measure the capacity to maintain long-term memories and relationships, we have started a series of interactions over the past two years with students in one of our master courses. The students are asked to act as one of 12 fake personas, which were given some basic properties such as age, gender, profession, home town, and relationships. The students were free to talk about themselves as the persona and ask questions. Similarly, our Leolani agent could ask questions and give comments. The students interacted several times as the eKG grew and the agent became more responsive. In the second year, other students continued the conversation for the same persona. In Table 1, the overall statistics give an indication of the total volume of signals and triples the agent processed. In total, the students had 79 interactions with the chat version of the agent and 13 interactions with the same agent running with a Pepper robot. The latter interactions were performed using speech and include visual recognition of objects (1,428 perceived images) and identities of people (1,738 face images). In total, the students interacted for more than 146 hours with the agent. The chat version of the agent was distributed as a Docker image and made available as a remote server so that the students could interact independently.[29]

The left-hand side of the table shows the number of utterances under both conditions, as well as the average token length, tokens per utterance, and utterance length in characters. As can be observed, these averages are stable in both conditions.[30] The right-hand side of the table shows the volume of content that is processed in terms of detection of emotions and dialog acts, visually detectable objects mentioned and people referred to in the conversations, people identifications by their face and the number of different faces identified (171, students, staff and bystanders during multiple encounters), geopolitical entities (GPE) mentioned in the conversations. Finally, we show the total number of triples stored in the eKG, broken down in the preloaded concepts and relations in the **ontology**, the **instances** added through the conversation, the total number of interactions (including perceptions), the claims made during these interactions, and the possible perspectives of the sources of the claims, i.e., the polarity, sentiment and certainty expressed.

The question under investigation is how much information and knowledge is processed and how does that affect the performance in terms of runtime latency and response quality? There is yet little research on such long-term multi-user performance. The interactions in Table 1 are the first attempts to measure such a performance impact on an agent that interacts with different users over a longer period of time. This agent also uses a wide range of processing modules across modalities in combination with an episodic Knowledge Graph that continuously reasons over incremental changes and knowledge growth and generates potential responses.

Table 2 shows the (finetuned) Large Language Models that are loaded to run the Leolani-agent with the Pepper robot in a multimodal setting. We see that a considerable amount of memory is required for each functionality provided by separate fine-tuned models. However, the models are kept separate for experimentation. Obviously, they can be combined in a single base model with

29. The docker-image is 8.7GB in size and can be downloaded from DockerHub: `https://hub.docker.com/repository/docker/piekvossen/leolani-text-to-ekg`

30. Between 2023 and 2024, the Leolani-agent was extended with Llama3.2.1B to rephrase the agent responses to make these more fluent. This has significantly doubled the length of the utterances: 6.78 versus 13.08 tokens per utterance and 33.61 versus 65.99 utterance length for the chat versions of 2023 and 2024 respectively. The robot interactions show a similar increase. Clearly, the use of an LLM such as Llama makes the communication more verbose.

students	23	images	1,428
persona	12	emotions	2,449
chat-interactions	79	dialog acts	6,123
- duration in minutes	8619	Object mentions	303
- utterances	3,417	object perceptions	5,048
- token length	4.04	person mentions	2,983
- tokens per utterance	9.93	person identifications (171 identities)	1,738
- utterance length	49.80	GPE mention	517
robot-interactions	13	triples	2,518,594
- duration in minutes	187	ontology	72,530
- utterances	837	instances	8,737
- token length	3.89	interactions	14,098
- tokens per utterance	9.82	claims	10,859
- utterance length	48.30	perspectives	46,090

Table 1: Overview statistics obtained from 23 master students acting as 12 personas in 2023 and 2024, talking freely with the Leolani-agent. Interactions were done just with a chat interface (79 in total) or through speech with a Pepper robot (13 in total). In the latter case, 1,428 images were recorded by Pepper. Total number of annotations and cumulation of triples are showsn in the right side of the table.

Module	Pretrained Model	Size
triple extraction	2 multilingual BERT-base	1.42 GB
face emotion models	Emotic	91 MB
image detection	yolo5	15.84 GB
age-gender	MLP-IMDB-WIKI	4.96 GB
face-detection	MLP-IMDB-WIKI	2.89 GB
speech-recognition	whisper-small	3.9 GB
dialog act	XLM-RoBERTa	1.1 GB
GO emotion	BERT-base	440 MB
text generation	Llama3.2-1B-Instruct-Q4_K_M.gguf	807.7 MB

Table 2: Finetuned Large Language Models used in the multimodal Leolani-agent implementation running with a Pepper embodiment

multiple classification heads to reduce memory load. As explained in the following, each of these modules can be placed in separate servers as Dockerized containers.

6. Developing your own agent

As components are self-contained and loosely coupled through the event bus, developing an agent is reduced to composing the necessary components and defining a flow of events, which is done by configuring the input and output topics of the components appropriately.

6.1 Adding, replacing and mixing components

To enable compatibility between components, we also use EMISSOR as the preferred data format of the payload carried by the events. The signal components publish EMISSOR *signals*, eventually split into a start and stop event. Interpretation components publish EMISSOR *Mentions*, defining segments and their annotations. Like this, any component that can process e.g. EMISSOR text signals or interpretations of a certain type, can be integrated into an agent by simply configuring it to listen to a topic that provides events with a text signal or the desired type of interpretation as payload. Furthermore, data from components that provide EMISSOR-based event payloads can be directly recorded in EMISSOR by a central storage component without additional adaption.

As long as component implementations process the same type of events, one implementation can simply be swapped for another. Even multiple implementations of the same component can be included in the same application, either for comparison or to increase performance. In the current version of the platform, we have, for example, various components for extracting triples and for speech recognition that can be applied simultaneously to increase recall or precision through voting. The source of signals and annotations is included in all EMISSOR data structures, which allows downstream components to determine the origin of an event.

Our modular architecture aims at easy integration of components that were not designed for our platform. The overhead to turn a software application into a component compatible with our platform consists of converting input and output of the application from and to the EMISSOR format and connecting it to the event bus. This can usually be achieved by adding these steps to existing code, without the need for modification.

6.2 Application types and performance

By choosing different implementations of the event bus (local vs. remote), agent implementations can reach from e.g. a monolithic Python application to a containerized setup, running each component as a separate process on a single machine, for instance using *docker-compose*[31], or even running in a cluster consisting of many physical machines using e.g. *Kubernetes*[32]. These different setups do not require any code adaption of the components itself, as those communicate only through the event bus interface, which remains identical. In a containerized setup, our architecture also allows mixing components from different platforms, e.g. using Python next to Java components, or components running different Python environments in the same agent to resolve dependency conflicts, which is a common problem in software development.

As long as modules can perform their tasks in parallel, as is the case e.g. for the processing of image signals parallel to audio signals or creating various interpretations (object recognition, face recognition, etc.) of the same image signal, a containerized setup also allows for scaling an application to many modules. Parallelizing processing of events within a module, for instance, by running multiple instances of a module and load balancing events between them, allows for scaling a module to a high throughput of events. However, whether such parallelization is possible depends on the task and design of the module. The total processing time of an input signal in an application will be determined by individual event processing times along the sequences of modules, as described in Section 4.4.The latency introduced by a remote implementation of the event bus itself[33] is expected

31. `https://docs.docker.com/compose`
32. `https://kubernetes.io`
33. The local implementation comes with no latency overhead.

to be low compared to the computationally heavy modules used in our applications. Widely used implementations, such as *RabbitMQ*, report latencies in the millisecond range at throughput rates of 10k+ messages per second[34].

6.3 Out of the box

For the components listed in Section 4 we provide Python implementations, each in its own repository published in the Leolani Github organization [35]. Each component can be packaged as a Python package and provides a service module containing the EMISSOR and event bus integration, which can be run from a Python application. Packages also provide the possibility of reuse of component code across components. Some components already run in a Docker container. In the future, we plan to add Docker configurations to enable running each component in a Docker container.

Our current agent implementations are realized as monolithic Python applications, which initialize and run the service modules from each of the included component packages, and use a local implementation of the event bus. We provide a Python implementation of the back-end server that can be run on Linux/OS X/Windows platforms[36], as well as a back-end server that can be run on Pepper robots[37]. Our agents can be used as a skeleton to build more agents, and in the future we will provide *docker–compose* and *Kubernetes* setups based on component Docker[38] images to run the agents as fully containerized applications. In the containerized setting, we will be able to run components locally or remotely in the cloud.

To collect all the code needed for an agent, we created parent repositories for each agent which contain all of the agent's components as *git submodules*. We provide build tooling to package each of our Python components, share the packages between components, and set up Python environments with all dependencies needed to run the agent application and individual components. This process can be executed centrally from the parent repository using our build setup. Following this pattern, new agents can also be created outside of our organization, mixing components under different governance.

To build components, `https://github.com/leolani/cltl-combot` provides infrastructure libraries for event bus integration, configuration management, and resource management (locking). The `cltl.combot.infra.event` module provides our event bus interface with a local implementation, as well as an implementation based on the *kombu library*[39] supporting the *AMQP*[40] protocol. This allows us to connect our framework to various available messaging servers such as, for example, *RabbitMQ*[41]. Furthermore, `cltl.combot.infra.topic_worker.TopicWorker` provides a utility class to listen to a configurable set of topics and process incoming events sequentially in a single thread, so that only the actual processing function, accepting a single event as input, needs to be implemented by the library user. Our own components are structured to separate functionality, using the `cltl` namespace, from event bus integration, using the `cltl_service` namespace.

34. See e.g. `https://www.rabbitmq.com/blog/2024/08/21/amqp-benchmarks` or `https://www.confluent.io/blog/kafka-fastest-messaging-system/`
35. `https://github.com/leolani`
36. `https://github.com/leolani/cltl-backend`
37. `https://github.com/leolani/cltl-backend-naoqi`
38. `https://www.docker.com`
39. `https://github.com/celery/kombu`
40. `https://www.amqp.org`
41. `https://www.rabbitmq.com`

A simple example is presented in the appendices C, D and E. For a complete example including configuration and resource management, we provide a template component at `https://github.com/leolani/cltl-template` with more detailed code templates, makefiles for our build setup, and in the future a Dockerfile template.

7. Conclusions

We described a platform for creating interactive agents which also captures the interaction at the signal level, the interpretation level and the integrated knowledge level. Central to our platform is an event bus on which signals and interpretations events can be published in temporal sequence to particular topics. By defining components as services that take certain topics as input and publish results to other topics as output, we have the flexibility to engineer any combination of components to create agent behavior. Our platform also supports the specification of higher-level intentions that can be carried out to fulfill a task or goal. Such intentions define which low-level components should be active to achieve this.

Multimodal interaction is extremely complex and rich. The agents currently built through our platform are far from what is needed to deal with all aspects of human interaction. A lot of research and development is needed, not only on individual components but also on their interaction and integration. Our platform will specifically help develop such complex integrated agents.

In the near future, we will use the platform to create different agents that can be used in interaction experiments. The rendered multimodal data and eKGs can be analyzed and compared to assess and evaluate the agents and their components. As such, we hope that it will function as a laboratory for agent development and testing.

References

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, 2015. URL `https://aclanthology.org/P15-1034`.

Selene Báez Santamaría, Thomas Baier, Taewoon Kim, Lea Krause, Jaap Kruijt, and Piek Vossen. Emissor: A platform for capturing multimodal interactions as episodic memories and interpretations with situated scenario-based ontological references. *Proceedings of the 1st Workshop on Multimodal Semantic Representations (MMSR)*, pages 56–77, 2021. URL `https://aclanthology.org/2021.mmsr-1.6`.

Selene Báez Santamaría, Piek Vossen, and Thomas Baier. Evaluating agent interactions through episodic knowledge graphs. *Proceedings of the 1st Workshop on Customized Chat Grounding Persona and Knowledge @ COLING2022, Korea, October 17, 2022*, pages 15–28, 2022. URL `https://aclanthology.org/2022.ccgpk-1.3.pdf`.

Michael Beetz, Moritz Tenorth, and Jan Winkler. Open-ease. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1983–1990. IEEE, 2015.

Dan Bohus, Sean Andrist, and Mihai Jalobeanu. Rapid development of multimodal interactive systems: a demonstration of platform for situated intelligence. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 493–494, 2017.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. Multiwoz-a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, 2018.

Emile Chapuis, Pierre Colombo, Matteo Manica, Matthieu Labeau, and Chloé Clavel. Hierarchical pre-training for sequence labelling in spoken dialog. *arXiv preprint arXiv:2009.11152*, 2020. URL `https://arxiv.org/abs/2009.11152`.

Yuya Chiba, Koh Mitsuda, Akinobu Lee, and Ryuichiro Higashinaka. The remdis toolkit: Building advanced real-time multimodal dialogue systems with incremental processing and large language models. In *Proc. IWSDS*, pages 1–6, 2024.

Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade, and Sujith Ravi. GoEmotions: A Dataset of Fine-Grained Emotions. In *58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Joshua Eisenberg and Michael Sheriff. Automatic extraction of personal events from dialogue. In *Proceedings of the First Joint Workshop on Narrative Understanding, Storylines, and Events*, pages 63–71, 2020.

Antske Fokkens, Piek Vossen, Marco Rospocher, Rinke Hoekstra, Willem R van Hage, and Fondazione Bruno Kessler. Grasp: grounded representation and source perspective. *Proceedings of Knowledge Resources for the Socio-Economic Sciences and Humanities associated with RANLP*, 17:19–25, 2017.

Tingchen Fu, Shen Gao, Xueliang Zhao, Ji-rong Wen, and Rui Yan. Learning towards conversational AI: A survey. *AI Open*, 3:14–28, 2022. doi:10.1016/j.aiopen.2022.02.001.

Jing Gao, Peng Li, Zhikui Chen, and Jianing Zhang. A survey on deep learning for multimodal data fusion. *Neural Computation*, 32(5):829–864, 2020.

Alison Gopnik and Henry M. Wellman. Why the child's theory of mind really is a theory. *Mind & Language*, 7(1-2):145–171, 1992. doi:10.1111/j.1468-0017.1992.tb00202.x.

Minlie Huang, Xiaoyan Zhu, and Jianfeng Gao. Challenges in building intelligent open-domain dialog systems. *ACM Transactions on Information Systems (TOIS)*, 38(3):1–32, 2020. URL `https://dl.acm.org/doi/abs/10.1145/3383123`.

Casey Kennington, Ting Han, and David Schlangen. Temporal alignment using the incremental unit framework. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 297–301, 2017.

Casey Kennington, Daniele Moro, Lucas Marchand, Jake Carns, and David McNeill. rrSDS: Towards a robot-ready spoken dialogue system. In Olivier Pietquin, Smaranda Muresan, Vivian Chen,

Casey Kennington, David Vandyke, Nina Dethlefs, Koji Inoue, Erik Ekstedt, and Stefan Ultes, editors, *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 132–135, 1st virtual meeting, July 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.sigdial-1.17. URL https://aclanthology.org/2020.sigdial-1.17/.

Taewoon Kim and Piek Vossen. Emoberta: Speaker-aware emotion recognition in conversation with roberta, 2021. URL https://arxiv.org/abs/2108.12009.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Kumar Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *ArXiv*, abs/1712.05474, 2017. URL https://api.semanticscholar.org/CorpusID:28328610.

Ronak Kosti, Jose M Alvarez, Adria Recasens, and Agata Lapedriza. Context based emotion recognition using emotic dataset. *IEEE transactions on pattern analysis and machine intelligence*, 42 (11):2755–2766, 2019. URL https://ieeexplore.ieee.org/document/8713881.

Lea Krause, Selene Baez Santamaria, Lucia Donatelli, and Piek Vossen. The role of personal perspectives in open-domain dialogue: Towards enhanced data modelling and long-term memory. In *Proceedings of BNAIC/BeNeLearn the Joint International Scientific Conferences on AI and Machine Learning, Delft, November 8-10, 2023*, 2023.

Jaap Kruijt. *Referring Expressions in Dynamic Human-Robot Common Ground*. PhD thesis, Vrije Universiteit Amsterdam, 2025.

Jaap Kruijt, Peggy van Minkelen, Lucia Donatelli, Piek TJM Vossen, Elly Konijn, and Thomas Baier. Spotter: A framework for investigating convention formation in a visually grounded human-robot reference task. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 15202–15215, 2024.

Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi:10.1126/scirobotics.abm6074.

Shikib Mehri and Maxine Eskenazi. Usr: An unsupervised and reference free evaluation metric for dialog generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 681–707, 2020.

Thilo Michael. Retico: An incremental framework for spoken dialogue systems. In Olivier Pietquin, Smaranda Muresan, Vivian Chen, Casey Kennington, David Vandyke, Nina Dethlefs, Koji Inoue, Erik Ekstedt, and Stefan Ultes, editors, *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 49–52, 1st virtual meeting, July 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.sigdial-1.6. URL https://aclanthology.org/2020.sigdial-1.6.

Kazuki Miyazawa and Takayuki Nagai. Survey on multimodal transformers for robots. *Authorea Preprints*, 2023. doi:10.36227/techrxiv.21993317.v1.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 28492–28518. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/radford23a.html.

Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Kurt Shuster, Eric M Smith, et al. Recipes for building an open-domain chatbot. *arXiv preprint arXiv:2004.13637*, 2020. URL https://arxiv.org/abs/2004.13637.

David Schlangen and Gabriel Skantze. A general, abstract model of incremental dialogue processing. *Dialogue & Discourse*, 2(1):83–111, 2011.

Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, et al. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. *arXiv preprint arXiv:2208.03188*, 2022. URL https://arxiv.org/abs/2208.03188.

Sonja Stange, Teena Hassan, Florian Schröder, Jacqueline Konkol, and Stefan Kopp. Self-explaining social robots: An explainable behavior generation architecture for human-robot interaction. *Frontiers in Artificial Intelligence*, 5:866920, 2022.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Chantal van Son, Tommaso Caselli, Antske Fokkens, Isa Maks, Roser Morante, Lora Aroyo, and Piek Vossen. Grasp: A multilayered annotation scheme for perspectives. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1177–1184, 2016.

Yuli Vasiliev. *Natural Language Processing with Python and SpaCy: A Practical Introduction*. No Starch Press, 2020.

Piek Vossen and Antske Fokkens. Grasp: A model for the perspective web. *Creating a More Transparent Internet: The Perspective Web*, page 260, 2022.

Piek Vossen, Lenka Bajčetić, Selene Baez, Suzana Bašić, and Bram Kraaijeveld. Modelling context awareness for a situated semantic agent. In *Modeling and Using Context: 11th International and Interdisciplinary Conference, CONTEXT 2019*, pages 238–252, Trento, Italy, 2019.

Piek Vossen, Selene Báez Santamaría, and Thomas Baier. A conversational agent for structured diary construction enabling monitoring of functioning & well-being. In *HHAI 2024: Hybrid Human AI Systems for the Social Good*, pages 315–324. IOS Press, 2024.

Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966. URL https://dl.acm.org/doi/10.1145/365153.365168.

Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023.

Dian Yu and Zhou Yu. Midas: A dialog act annotation scheme for open domain human machine spoken conversations. *arXiv preprint arXiv:1908.10023*, 2019. URL https://arxiv.org/abs/1908.10023.

Appendix A. EMISSOR multimodal data representation

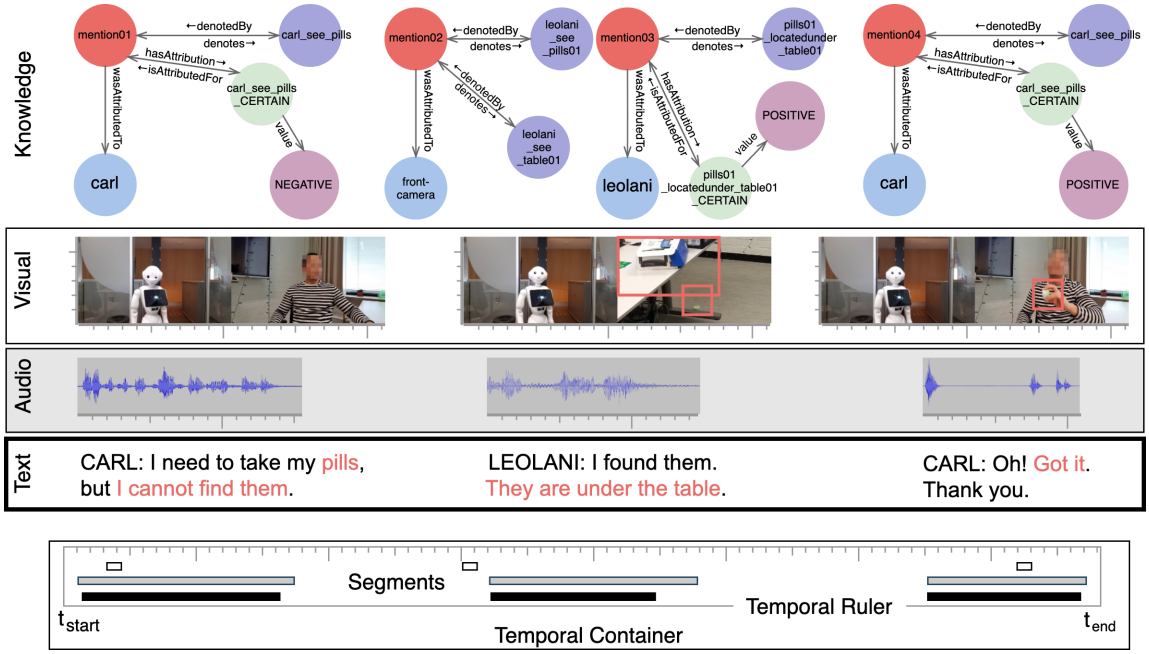

Figure 3: Visualization of an interaction in the EMISSOR data format.

Figure 3, taken from (Báez Santamaría et al., 2021), shows a visualization of four modalities (text, audio, visual, and knowledge). Signals are grounded in a temporal container on the horizontal axis, with bars marking alignments through the temporal ruler. Red boxes mark *segments* annotated as *mentions* of objects (pills and the table). Text *segments* highlighted in red are annotated as *mentions* of triples. The upper graphs represent the corresponding triples of the eKG generated from the annotated source modalities along the temporal sequence. The visual modality shows two different camera viewpoints (left is what Carl sees and right is what Leolani sees) concatenated side by side.

Appendix B. Interaction plots

Figures 4 and 5 show plot representations of interactions where each interaction point is scored on the y-axis as a negative or positive interaction "move". Interaction points are text signals (possibly derived from audio signals), perceptions, or actions. Text signals are annotated for dialog acts, emotions and likelihood using the USR model. The following mapping is used for dialog acts and emotions:

- Dialog acts:

  negative : neg_answer, complaint, abandon, apology,non-sense, hold.

  positive : pos_answer, back-channeling, appreciation, thanking, respond_to_apology.

- Emotions:

  negative : remorse, nervousness, fear, sadness, embarassment, disappointment, grief, disgust, anger, annoyance, disapproval, confusion.

  positive : joy, amusement,excitement, love, desire, optimism, caring, pride, admiration, gratitude, belief, approval, curiosity.

A threshold is set to accept only emotions and dialog acts scoring above.5. Dialog acts such as statement and opinion, which are the most frequent, are ignored. The same holds for a neutral emotion. Likelihood scores between 0 and 1 are deducted with 0.3 so that a score below this threshold adds negatively to the interaction score and a score above adds positively. Every positive dialog act or emotion adds 1 point to the score, and every negative act and emotion -1. The cumulative scores are plotted on the y-axis. In the case of interaction with the embodied Leolani-agent emotions are also extracted from human faces.

The labels in the plot start with the source of the signal, followed by the content of the signal (expressed text or annotation label), and any annotation with a score. Note that the responses from Leolani are based on the episodic Knowledge Graph. The triple results are first converted to text using templates. Next, Llama3.2.1B is prompted to paraphrase the input in simple English using a maximum of 100 tokens and a temperature of 0.2.

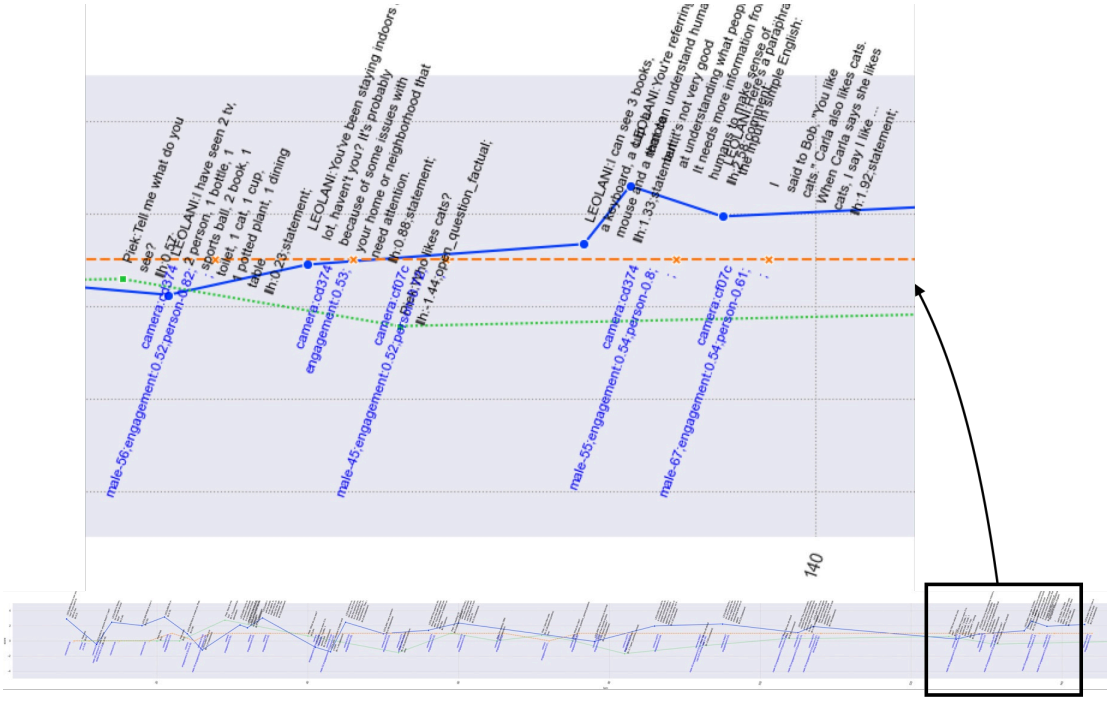

Figure 4: Multimodal interaction plot showing text and image signals picked up and created by Leolani. Signal annotations are shown per source, speakers, and camera, where the camera shows ids for people, types of object detected with a confidence score, and gender and age predictions for human faces. Dotted green lines represent the human interlocutor, striped red lines represent utterances from the agent, and solid blue lines the perceptions by the agent. The text signals are annotated with a USR likelihood score, and if above a threshold of 0.5 confidence, the dialog act and emotion annotation are shown. The annotations are mapped on the y-axis ranging from -5 to 5 to the extent that they indicate negative or positive turns in the conversation. Face expressions are also mapped to the negative-positive scale.

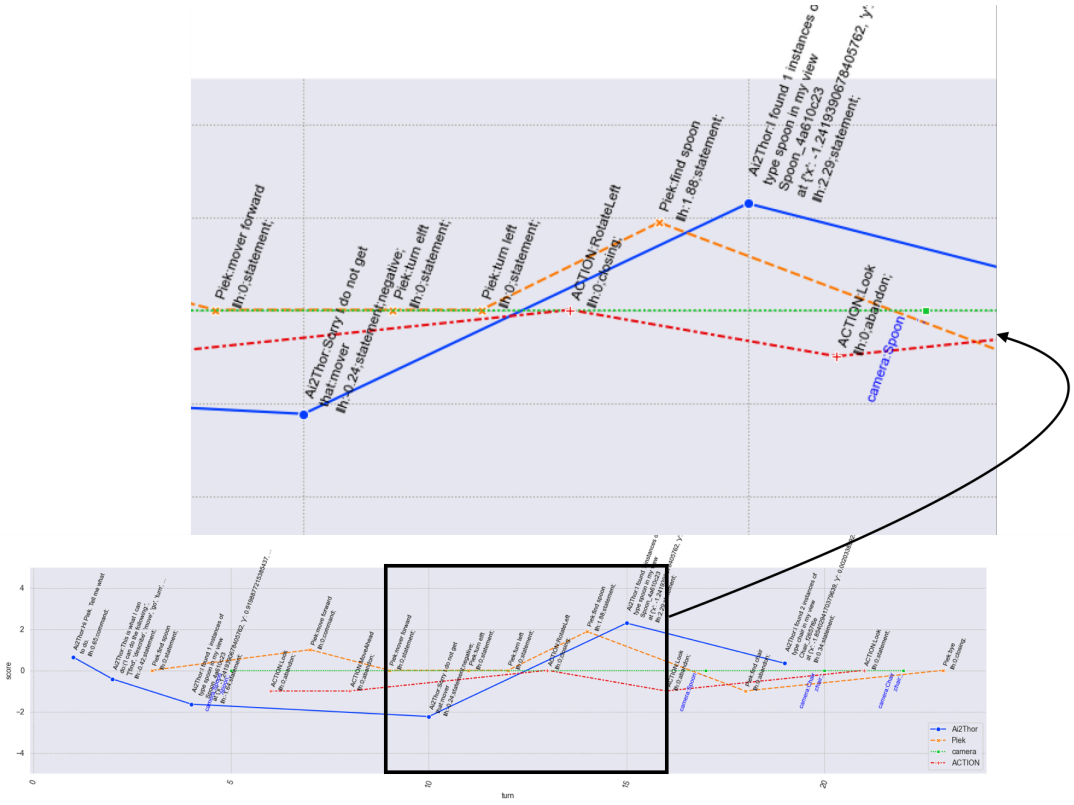

Figure 5: Multimodal interaction plot showing text, image and action signals picked up and created by an AI2Thor agent in a virtual world. Striped orange lines represent the utterances from the human interlocutor, striped-dotted red lines represent the actions by the AI2Thor agent and solid blue lines the utterances rom the AI2Thor agent. The dotted green line represents the perceptions of the agent. The text signals (utterances and actions) are annotated with a USR likelihood score, and if above a threshold of 0.5 confidence, the dialog act and emotion and sentiment labels. The annotations are mapped on a range from -5 to 5 to the extent that they indicate negative or positive turns in the conversation.

Appendix C. Example component structure

```
cltl-mycomponent
├── config/
├── requirements.txt
├── setup.py
└── src/
    └── app.py
        ├── cltl/
        │   └── mycomponent/
        │       ├── api.py
        │       └── implementation.py
        └── cltl_service/
            └── mycomponent/
                └── service.py
```

Appendix D. Example `service.py` module

```python
from cltl.combot.infra.event import Event, EventBus
from cltl.combot.infra.time_util import timestamp_now
from cltl.combot.infra.topic_worker import TopicWorker
from cltl.combot.event.emissor import TextSignalEvent
from emissor.representation.scenario import TextSignal

class SimpleService:
    def __init__(self, input_topic: str, output_topic: str, event_bus: EventBus):
        self._input_topic = input_topic
        self._output_topic = output_topic
        self._event_bus = event_bus
        self._topic_worker = None

    def start(self):
        self._topic_worker = TopicWorker([self._input_topic], self._event_bus,
                                         provides=[self._output_topic],
                                         processor=self._process)
        self._topic_worker.start().wait()

    def stop(self):
        if not self._topic_worker:
            pass

        self._topic_worker.stop()
        self._topic_worker.await_stop()
        self._topic_worker = None

    def _process(self, event: Event[TextSignalEvent]):
        input = event.payload.signal.text)
        response = "" ### PROCESS THE INPUT
        if response:
            output_event = self._create_payload(response)
            self._event_bus.publish(self._output_topic, Event.for_payload(output_event))

    def _create_payload(self, response):
        signal = TextSignal.for_scenario(None, timestamp_now(),
        timestamp_now(), None, response)
        return TextSignalEvent.create(signal)
```

Appendix E. Example `app.py` application

```
from cltl.combot.infra.event.memory import SynchronousEventBus
from cltl_service.mycomponent import SimpleService

event_bus = SynchronousEventBus()
service = SimpleService("text_in", "text_out", event_bus)
try:
    service.start()
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    self.service.stop()
```

Appendix F. Dialog flow implementations

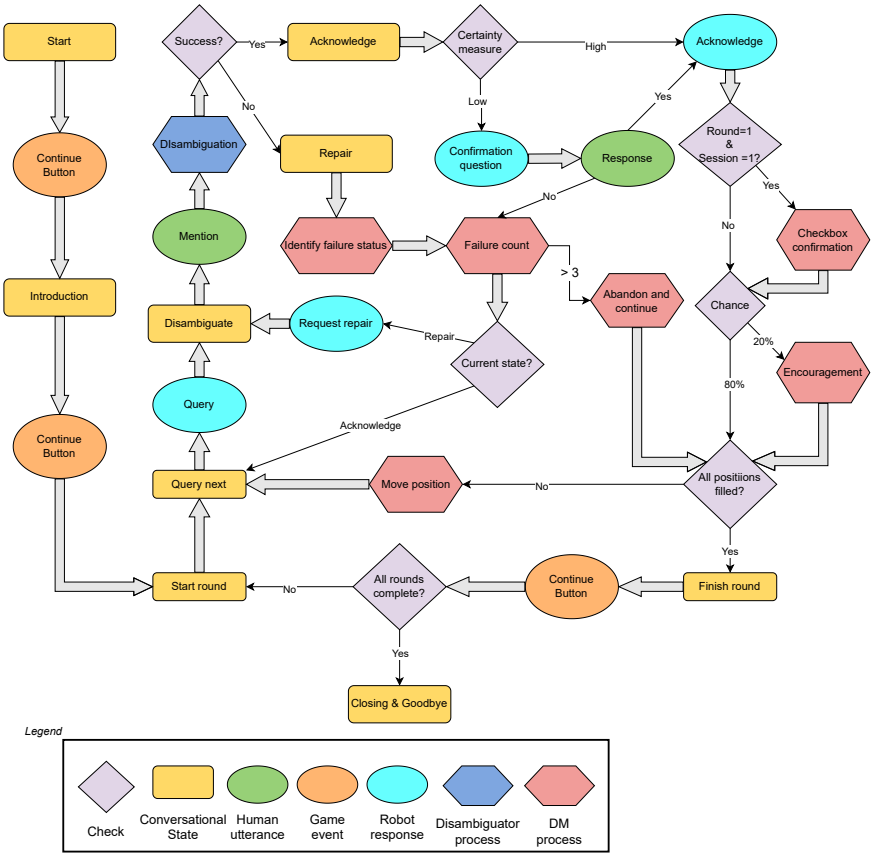

Figure 6: Dialog flow of the SPOTTER application taken from Kruijt (2025), showing Conversational States and Dialog Management processes that are modeled as high-order intents in the Leolani platform to control the dialog for the purpose of the game.

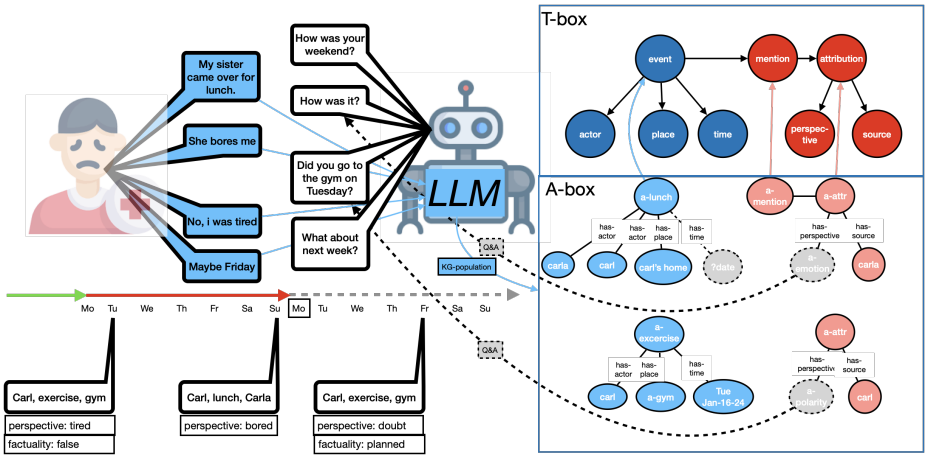

Figure 7: Dialog example of an agent that uses conversations to reconstruct a person's timeline of dialy activities of life, taken from Vossen et al. (2024)