

# Ravestate: Distributed Composition of a Causal-Specificity-Guided Interaction Policy

Birkner, Joseph  
joseph.birkner@tum.de

Dolp, Andreas  
andreas.dolp@tum.de

Karimi, Negin  
negin.karimi@tum.de

Basargin, Nikita  
n.basargin@tum.de

Kharchenko, Alona  
alona.kharchenko@tum.de

Hostettler, Rafael  
rh@robey.org

September 20, 2023

## Abstract

*In human-robot interaction policy design, a rule-based method is efficient, explainable, expressive and intuitive. In this paper, we present the Signal-Rule-Slot framework, which refines prior work on rule-based symbol system design and introduces a new, Bayesian notion of interaction rule utility called Causal Pathway Self-information. We offer a rigorous theoretical foundation as well as a rich open-source reference implementation Ravestate, with which we conduct user studies in text-, speech-, and vision-based scenarios. The experiments show robust contextual behaviour of our probabilistically informed rule-based system, paving the way for more effective human-machine interaction.*

## 1 Introduction

Non-physical, especially verbal and emotional interaction between humans and robots is envisioned to be fluent, context-sensitive and entertaining. To this end, a robot persona is designed with the agenda to encourage users to conduct certain specific activities with it, such as playing games, taking a selfie or buying ice cream. However, given the unbounded input space of natural language, it is hard to "insert" such an agenda into an interaction policy to facilitate a continuous flow of conversation. Ultimately, conversational disfluency events where the intent of a user's input is disrespected must be minimised while optimising for a series of diverse interactions that showcase specific distinct capabilities.

Understanding the intent of an interlocutor must take a mixture of the current and previous input signals into account, the result of which may again be a higher-level input signal which adds contextual information [1, 2]. Also, in a multi-modal system, different input sources (such as speech and vision) must be considered simultaneously.

One way to solve this problem is through end-to-end application of data-driven neural systems [3]. However, these are very hard to "steer" from the

perspective of a human interaction policy designer, and usually require them to express any necessary context for an action as a set of examples, which can neither be proven to be sufficient nor forced to be adequately generalised [4, 5].

Another approach are symbolic grammatical rule-based systems. For example, the rule  $\{ \text{🦖} \} \rightarrow \{ \text{🦖} \text{ 😬} \}$  describes how an agent develops anxiety at the sight of a Tyrannosaurus. Such rules exhibit a very large control surface to the interaction policy designer. For this reason, symbol production systems have been highly popular as mechanisms for the design of artificial agents in video games [6]. More importantly, they have been explored as architectures for cognitive modeling and inference [7].

In this paper, we introduce Roboy's cognitive system *Ravestate*<sup>1</sup>, an open-source architecture for natural-language and vision-based interactive state machines which builds on prior work described in Section 2. Its architecture is based on the concept of a Signal-Rule-Slot automaton, which is described in Section 3. The Signal-Rule-Slot automaton introduces a new probabilistic notion of action rule utility, which solves the problem of contextual action rule precedence and allows for dynamic combination of contextually overlapping actions. Interaction rules in this framework are productions which act as contextual modifiers of context, firing once a particular set of constraints in the current context is satisfied. This paper reveals the Signal-Rule-Slot algorithm, the conflict resolution strategies between rules it employs (Section 3.5), its reference implementation *Ravestate* (Section 4) and system evaluations through user studies (Section 5).

The reference implementation comes with a set of task- and modality-specific modular rule sets for verbal, visual and emotional interaction (see Figure 10).

<sup>1</sup> <https://github.com/robey/ravestate>

## 2 Prior Work

The problem of mapping perception to action is fundamental to the field of artificial intelligence [8]. Through the Subsumption Architecture [9] it was discovered that by overlaying simple actions in response to some stimuli, complex life-like behavior of an artificial agent could be simulated.

SOAR (State, Operator And Result) [10] is a symbolic cognitive architecture where symbolic stimuli are modeled as so-called "*working memory elements*". Problem-solving occurs by matching rule left-hand sides against working memory elements in the current state. All matching rules are executed in parallel. The same applies to the TRINDI natural language dialogue toolkit [11].

Another well-known symbolic cognitive architecture is ACT-R (adaptive control of thought-rational) [12, 13]. In ACT-R, rules ("*processes*") are executed in response to patterns of symbols ("*chunks*") which are placed in memory buffers. In contrast to SOAR, at each point in time, chunk patterns activate only a single rule, even if multiple left-hand rule patterns show a match. Only the rule with the highest utility estimated via a Bayesian framework gets activated.

It should not go unmentioned that grammatical cognitive symbol systems also go by the name of "Blackboard architectures" [14], highlighting how a distributed rule-based cognitive architecture is based on independent rules operating over a shared working memory space (the blackboard). A truly multi-modal blackboard-based interaction architecture was realised through the IrisTk [15] framework, which models state transitions in a hierarchical graph.

What we are missing from these existing architectures is a combination of the following properties:

1. Distributed definition of rules.
2. Parallel execution of matching rules.
3. Intrinsic recognition of conflicting rules.
4. Estimation of rule utility from causal structure.

Perhaps closest to our needs, the OpenDial [16] system allows for distributed specification of independently operating rules with utility derived from causal pathway probabilities through training, however no attempt is made to execute multiple matching rules in parallel or to derive intrinsic rule utility.

In the following we present a new architecture for rule-based automata covering the criteria above, which we implemented in the *Ravestate* Python 3 library.

## 3 System Design

### 3.1 Building Blocks

The architecture of *Ravestate* borrows concepts from both grammatical symbol systems and traditional Spoken-Language Understanding (SLU). SLU operates in terms of intents and slots [17]. Slots are variables which are filled throughout a dialogue session to accomplish a goal which is guided by a detected intent. The combination of slot values and intent guides the action of the dialogue system. The following components make up *Ravestate*'s Signal-Rule-Slot architecture:

1. **Slots** are specific variables which accommodate the need to parameterise interaction state. For example, such variables could include the most recent user utterance and the system's verbal response in a dialogue application setting, or a wish expressed by the user, say strawberry ice cream, in a specific application.
2. Symbols are adopted under the term **Signals**, highlighting their atomic and transient information storage capacity. Signals are events emitted deliberately through rules, or implicitly as "changed-events" when the value of a slot is changed. They are also outfitted with a payload value, which allows information transfer between a signal emitter and a consuming rule. Signals form the atoms of the left-hand rule condition parts.
3. **Rules** are adopted as procedural memory elements which have the capacity to manipulate the signal pool and read/write slot values in response to a Boolean combination of signals, by executing a user-defined routine. In practice, a rule is simply an annotated function which is automatically executed by a rule-pattern matching loop.

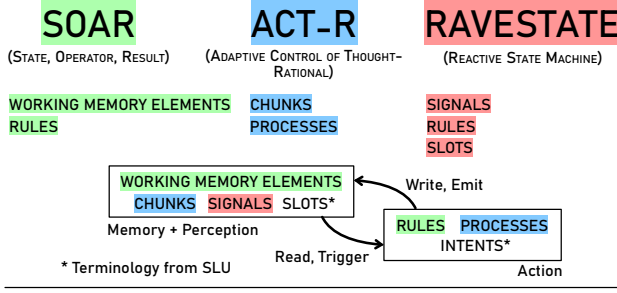
In the *Ravestate* framework, a collection of symbols (working memory), slots (long-term memory) and rules (procedural memory) is called a *context*. Note that *Ravestate* does not yet support the formulation of arbitrary rule condition predicates over signal payload or slot values, although this would be a straight-forward addition.

### 3.2 Signal-Rule-Slot Diagrams

Signal-Rule-Slot diagrams allow visualising relationships between slots, signals and rules (see Fig. 2).

### 3.3 Parallel Rule Execution

Much like in *SOAR*, rules in *Ravestate* are executed in parallel if they do not interfere with each other.



**Figure 1:** Terminology comparison regarding Perception-Action loops in SOAR, ACT-R and Ravestate.

	slot $p_i$
	signal $s_i$
	rule $\lambda_i$
	$\lambda_y$ reads $p_x$
	$\lambda_x$ writes to $p_y$
	$p_x$ emits $s_y$
	$\lambda_x$ emits $s_y$
	$\lambda_x$ emits detached $s_y$
	$s_x$ is condition of $\lambda_y$
	conjunction of conditions
	disjunction of conditions

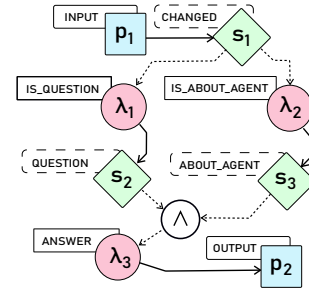
**Figure 2:** Signal-Rule-Slot Diagrams

The potential for colliding rules is explained in more detail in Section 3.5. Running rules in parallel has a number of practical advantages:

1. **Long-running rules:** Other rules are not blocked by a long-running rule function executing background tasks.
2. **Fillers/Disfluencies:** In cases where response generation takes a few seconds, a filler rule voicing an utterance such as "Uhm..." or "Let me think..." is able to fill in the gap.
3. **Heterogeneous output:** In a multi-modal interactive system, some rules might complement each other positively when executed simultaneously, for example, by changing the facial expression and replying with a spoken remark, even without knowledge of mutual action.
4. **High event frequency:** If specific signals are generated at a high frequency, it might be necessary to execute a rule again before the prior execution instance finishes.

### 3.4 Example

In combination, sets of rules  $\Lambda$ , signals  $S$ , and slots  $P$  describe an agents response to an input. We provide an example for this in Figure 3: The slot **INPUT** contains a textual representation of a user utterance (e.g. "How are you?"). As the value is placed into the slot, the **INPUT:CHANGED**-Signal is triggered, which satisfies the condition for the **IS\_QUESTION** and **IS\_ABOUT\_AGENT** rules, which fire their respective **QUESTION** and **ABOUT\_AGENT** signals depending on the value of **INPUT**. The conjunction of these two signals triggers the personal question **ANSWER** rule, placing respective responses in the **OUTPUT** slot.



**Figure 3:** Signal-Rule-Slot diagram for simple conditioned question answering.

*Note:* The exact elements of Signal-Rule-Slot-graphs as used in Figure 3 are described in Appendix 3.2.

### 3.5 Rule exclusion

In the previous example, all rules coexist and cooperate meaningfully simply by running as soon as their condition signals are available. However, it is easy to come up with a scenario where two rules would not cooperate in this way. Specifically, there may be two rules that overlap in their purpose to generate a behavior in response to a given stimulus, resulting in a bad combination of two response variants. The toy Signal-Rule-Slot automaton in Figure 4 showcases such a situation. In this example there are two rules which may write to the **OUTPUT** slot; the **WILDTALK** rule and the **QUESTIONANSWERING (QA)** rule. The desired behavior is such that the **QA** rule should apply when the input is a question, while **WILDTALK** (perhaps encapsulating a generative model) applies otherwise. As such, the two rules are designed to be mutually exclusive for each **INPUT:CHANGED** signal, since executing both would entail a race condition. On the other hand, the **EMOTION** rule which writes to **FACIALEXPR** can always run independently from **QA** or **WILDTALK**.

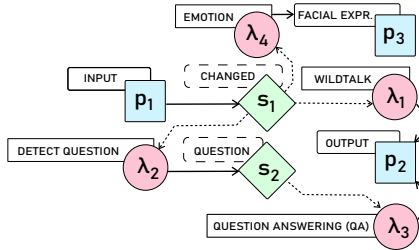
The example therefore illustrates how slots are devices which reduce the number of possible behaviors

of a Signal-Rule-Slot automaton. Specifically, all rules  $\lambda_x, \lambda_y \in \Lambda$  like

$$S_x \rightarrow \lambda_x \rightarrow P_x$$

which write to a set of slots  $P_x$  in reaction to a conjunction of signals  $S_x$  are mutually exclusive, if they share both written slots and condition signals ( $(P_x \cap P_y \neq \emptyset) \wedge (S_x \cap S_y \neq \emptyset)$ ).

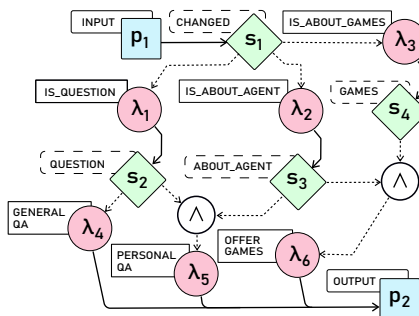
Note that this also applies to the **WILDTALK** and **QA** rules in Figure 4. Even though their immediate condition signal-sets  $\{\text{INPUT:CHANGED}\}$  and  $\{\text{QUESTION}\}$  are exclusive, the **QUESTION** signal can actually only occur as an effect of **INPUT:CHANGED**. Therefore, the complete condition of **QA** is  $\{\text{INPUT:CHANGED} \wedge \text{QUESTION}\}$ .



**Figure 4:** Example for exclusive and non-exclusive rules: The rule  $\lambda_3$  is racing with  $\lambda_1$  to reply to signal  $s_1$ , while  $\lambda_4$  is independent.

### 3.6 Causal Pathway Utility

Given the example above, one might simply solve the rule precedence problem by counting the condition signals of a rule, and prioritising a rule with more conditions over a rule with less. This metric is called *specificity*, and it is a traditional component of rule-based systems [18]: The higher the specificity, the higher the situational utility and therefore the priority of a rule. Unfortunately, the basic counting metric easily breaks down in certain scenarios.



**Figure 5:** Breakdown scenario of count-based specificity:  $\lambda_5$  and  $\lambda_6$  each have three condition signals.

The basic flaw of count-based specificity becomes apparent in the example presented by Figure 5: The **OFFERGAMES** and **PERSONALQA** rules each have

three signals in their condition conjunctions. Therefore, an input like "Do you like games?" can not be decisively routed to either rule under count-based specificity. Revisiting the intuition of  $\text{SPECIFICITY} \propto \text{UTILITY}$ , the proportional relationship may be explained by the lower cross-entropy between a given situation and a rule which is modeled *specifically* for this situation vs. a rule that is modeled to cover a broader spectrum of situations. Therefore, what we might actually want to model is simply the prior probability of a rule condition. This probability's negative logarithm is its self-information which correlates with specificity, giving us Equation 1.

**Equation 1** Given the self-information  $H$  of a random event  $x$  with probability  $p(x)$  as  $H(x) = -\log(p(x))$ , for any rule  $\lambda_i$

$$\text{UTILITY}(\lambda_i) \propto \text{SPECIFICITY}(\lambda_i) \propto -\log(p(\lambda_i))$$

We therefore need to find a way to calculate the probability estimate  $p(\lambda_i)$  for a rule  $\lambda_i$ , without any data from which a Maximum Likelihood Estimate may be derived. Using a single assumption, we can derive a probability estimate just from the structure of the automaton itself: The amount of the information carried by a signal is inversely proportional to the number of rules depending on it. This assumption implies that

1. A signal which occurs in every rule condition does not carry any information.
2. A signal which occurs in only a single rule condition carries the highest possible amount of information.

Since a logical conjunction corresponds to a product of probabilities which corresponds to a sum of self-information values, we can calculate the self-information of a rule as the sum of the self-information values of its conjunct condition signals. This is expressed in Equation 2.

**Equation 2** Given a Signal-Rule-Slot automaton  $(S, \Lambda, P)$  consisting of slots  $P$ , signals  $S$  and rules  $\Lambda$ , the self-information of any single rule  $\lambda_i \in \Lambda$  with its conjunct condition clause  $S_i \subseteq S$  is calculated as

$$\begin{aligned} \text{UTILITY}(\lambda_i) &= \sum_{s \in S_i} -\log\left(\frac{|\{s \in S_k | \lambda_k \in \Lambda\}|}{|\Lambda|}\right) \\ &= H(\lambda_i) \end{aligned}$$

Note, that this definition of utility introduces an optimization requirement on the interaction policy designer (a human for now) to model rule conditions such that there is minimal mutual information between a rule's condition's signals.



### 3.7 Loops And Causal Groups

Figure 4 in Section 3.5 shows that the *causal history* of a signal must contribute to its *exclusion calculation* with other signals. From the example, the **QUESTION** signal is overlapping with **INPUT:CHANGED** because it is *caused by* **INPUT:CHANGED** through the **QUESTION** rule. We denote signals which overlap in their sets of recursive causes as belonging to the same *causal group*. The clause  $S_x \cap S_y = \emptyset$  from Section 3.5 therefore expresses that in order to be non-conflicting, two rules must have conditions which belong to separate causal groups.

This requirement, together with the requirement that any implementation of the Signal-Rule-Slot system automatically performs cause-based recursive completion of condition signals, would greatly restrict the number of possible behaviors that can be expressed with a Signal-Rule-Slot automaton: The impossible behaviors would be all those which causally originate from a previous behavior of the agent herself. Since this is a particularly interesting set of behaviors, the concept of *detached emission* is introduced.

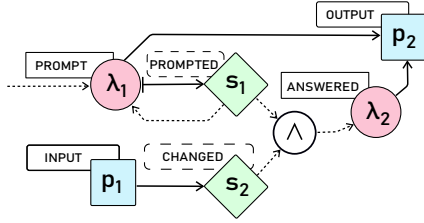


Figure 6: Example for a “detached” signal emission scenario.

An example for detached emission is given in Figure 6: Through some previous cause, the agent prompts the user with a question. This state is saved by the **PROMPTED** signal. Through detached emission, **PROMPTED** is not completed with the condition signals of the **PROMPT** rule. Thereby, **PROMPTED** can both trigger a loop of asking the question again after a certain time-out, or trigger the **ANSWERED** rule in conjunction with **INPUT:CHANGED**.

### 3.8 Integration of Visual Stimuli

As far as specific mechanics are concerned, examples in Section 3 only showcase how Signal-Rule-Slot automata (and by extension our *Ravestate* reference implementation) function in scenarios based on verbal interaction. As a proof-of-concept, this Section contains an example of how vision-based interlocutor tracking and recognition (using face recognition in particular) is integrated into *Ravestate* using the Signal-Rule-Slot system.

Figure 7 shows how the visionio module interacts with the interloc and persqa modules to store

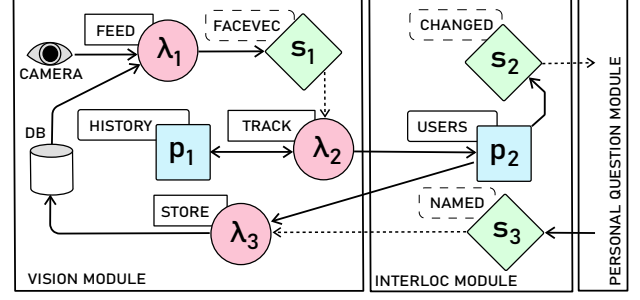


Figure 7: The VisionIO module architecture.

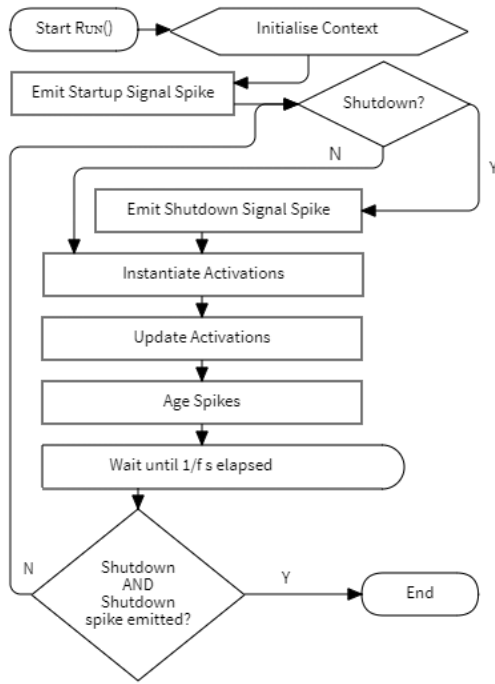
and recall the present interlocutor based on her facial feature vector: The **FEED** rule continuously emits **FACEVEC** signals for facial feature vectors which are extracted from a generic camera input stream, and associated with primary keys from persistent graph memory if a close feature match based on cosine similarity is found. The **FACEVEC** feature signals are received by the **TRACK** rule, which tracks a history of such signals in a ring-buffer placed in the **HISTORY** slot. The ringbuffer debounces the **FACEVEC** signal, and allows the **TRACK** rule to decide when confidence is high enough to update the interloc module’s central **USERS** slot with a new graph database node that represents the new interlocutor. If the new user is unknown, their database node will be transient. Only once the name of the user is inferred by the persqa module, the node will be persisted, which is expressed in the **NAMED** signal. This allows the **STORE** rule to update the facial feature vector database.

### 3.9 Spikes and Activations

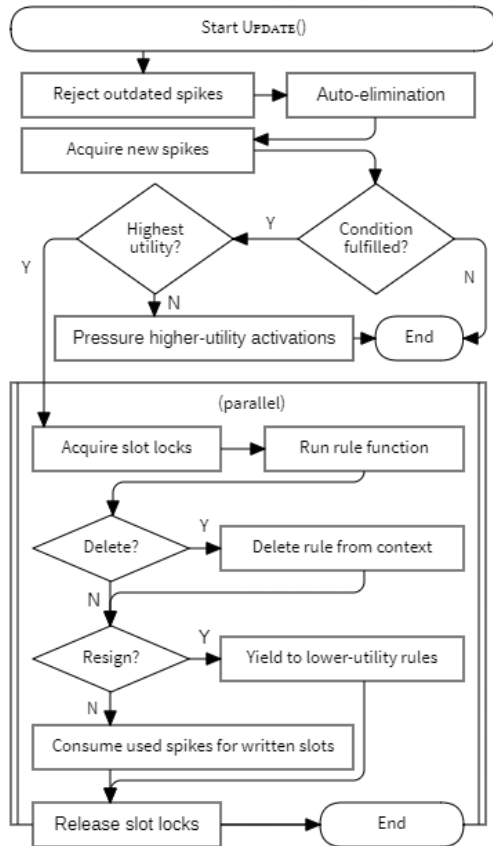
We have described the declaration of Signal-Rule-Slot automata, and how they can be used to model behavioral policies for interactive agents. We have however not specified the procedures by which the automaton’s state is transitioned at runtime: Most importantly, we have not introduced *spikes* and *activations*, and the role they play in the system: *Spikes*, consisting of a signal reference and an age, encapsulate instances of signals that are *emitted* during the run of a rule function. *Activations*, consisting of a rule reference and a set of acquired spikes, encapsulate all state that is associated with the execution of a rule.

A combination of rules  $\Lambda$ , activations  $A$ , signals  $S$ , spikes  $\Delta$  and slots  $P$  is aggregated into a *context* structure  $C = \langle \Lambda, S, P, \Delta, A \rangle$ . Such a context fully describes a single Signal-Rule-Slot system at any given point in time. Its temporal transition follows the **RUN** routine (see Fig. 8).

The **RUN** routine creates activations during the step “*Instantiate Activations*”. For its activation collection  $A$ , the context guarantees that for every signal  $s_i$  in a condition of one of its rules  $\lambda_i$ , there exists a



**Figure 8:** Context RUN routine. The routine references a fixed update frequency  $f$ .



**Figure 9:** Activation UPDATE routine. It is repeatedly called by a context's RUN routine for each activation.

"standby" activation which does *not* hold a spike which fulfills said signal.

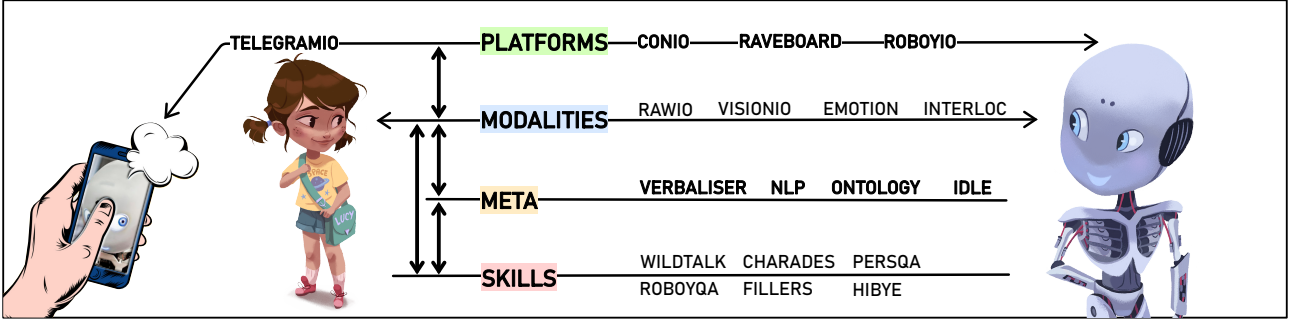
Each activation that is held in a context is continuously updated using the UPDATE function (see Fig. 9) to facilitate the Activation's lifecycle: Activations live until they have acquired a sufficient spike set fulfilling their rule's condition. Once an activation has acquired the necessary spikes, it may execute its rule function and remove itself from its context's activation set.

The UPDATE algorithm also includes a mechanic which ensures that only the activation that promises the highest utility for its combination of held signal spikes and write-slots is executed: Once ready to run, the activation determines whether to go forward through a highest-utility check. If the activation's rule does not promise the highest available utility, unfulfilled higher-utility activations are *pressured*.

A *pressured activation* is subject to a so-called *auto-elimination* process: It must either gather sufficient signal spikes to run within a certain timeout window, or reject its previously acquired spikes. This way, either a high-utility rule activation gains precedence over a lower-utility one, or it eventually yields when the specific signals spikes it requires are not becoming available.

Yielding to lower-utility activations may also be achieved explicitly by returning RESIGN from a rule function, indicating that the rule could not be applied for any reason. If the executed activation does not resign, it forces competing lower-utility activations to give up on the used spikes by *consuming* them for the written slots.

The UPDATE routine also enables time-based rule signal constraints: Signal constraints for rules may be annotated with a minimum and a maximum age for matching spikes. While an acquired spike is too young, the condition-fulfilled check will fail. If the spike becomes too old, it is rejected at the beginning of the UPDATE routine.



**Figure 10:** Our open-source reference implementation of a causal-specificity-aware automaton comes with modular sets of interaction rules which allow composing a behavior policy from independent building blocks. These building blocks are divided into categories based on their role in the system. The categories are **Platform Modules**, **Modality Modules**, **Meta Modules** and **Skill Modules**.

## 4 Implementation

We open-source our reference implementation of the Signal-Rule-Slot automaton in the *Ravestate* Python 3 library, together with modular sets of rules, slots and signals which allow composing an interactive agent from separate independent building blocks. These building blocks are divided into categories based on their role in the system, as shown in Figure 10. The categories are **Platform Modules**, **Modality Modules**, **Meta Modules** and **Skill Modules**. Each module is generally structured as a separate Python package.

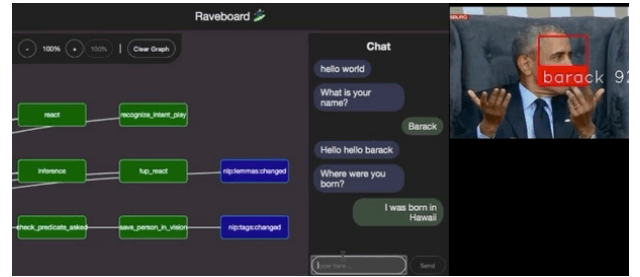
### 4.1 Platform Modules

Platform modules provide bindings to specific platforms for certain modalities. Implementations of speech, emotion expression or vision input all require specialised middleware for specific platforms like instant messaging, the physical robot and debug tools:

- The web platform *webio* allows interaction with a *Ravestate* agent through a chat interface, while inspecting signal spikes and activations as they occur in real-time. A more basic implementation for text-based interaction is offered through the *conio* module, which implements chatting on the command line.
- For instant-messaging, the *telegramio* module supports textual interaction with *Ravestate* agents on the Telegram platform.
- The *roboyio* module provides platform bindings to physical instances of the *Roboy* robot. The robotic platform supports conversational interaction through speech-recognition and synthesis, facial expressions, head movement and face recognition.
- Adding more platforms is simplified through

the *ros1* and *ros2*<sup>2</sup> modules, which provide specialised slot types that can bind to publish-subscribe network communication topics.

Importantly, the nature of the Signal-Rule-Slot architecture allows for these modules to run in parallel without interfering with each other. For example, *webio* and *roboyio* can naturally co-exist at runtime.



**Figure 11:** *Raveboard* UI with *visionio*

### 4.2 Modality Modules

Modality modules provide platform-independent I/O interfaces for specific modalities to upstream platforms and downstream meta processors or skills:

- The *rawio* module provides the `RAWIO:IN` and `RAWIO:OUT` slots for bidirectional raw textual utterances.
- For visual interaction, the *visionio* module provides face recognition and interlocutor tracking based on a generic stream of input images (see Section 3.8).
- Emotional I/O is provided through the *emotion* module, which offers emotion-specific signals that can be used by downstream applications.
- Perhaps an odd-one-out, the *interloc* module provides a standardised way for platforms and applications of accessing present interlocutors through the `USERS` slot.

<sup>2</sup> <https://www.ros.org/>

### 4.3 Meta Modules

Meta modules provide middleware components that execute generic application-independent I/O refinements or expose auxiliary tools to downstream applications:

- The verbaliser module offers an abstract way for applications to produce verbal responses, by writing an intent key to the `VERBALISER:INTENT` slot which is then translated into a phrase.
- For advanced textual input processing the `nlp` module provides various signals and slots that are updated with output from the *spacy* library [19] for subject-predicate-object triple extraction, sentence type detection, named entity recognition and other tasks.
- The `idle` module provides the `IDLE:BORED` and `IDLE:IMPATIENT` signals that simplify the detection of "awkward pauses" in the interaction, due to processing lag or lack of user engagement.
- Long-term persistent memory is provided by the ontology module through a slot type that automatically binds values to nodes in a graph database.

### 4.4 Skill Modules

Skill modules provide implementations for specific interactive capabilities that can be cherry-picked to assemble the behavioral components of an artificial agent:

- Neural generative response models are provided through the `wildtalk` module with *ConvAI* and *GPT-2* [20], and the `genqa` module for general knowledge question answering based on *DrQA* [21].
- The `charades` module implements the charades game which involves guessing a user activity from visual cues, voicing the guess and reacting to the user's judgement.
- Greeting and farewell triggered by changes in the `INTERLOC:USERS` slot are covered by the `hibye` module.
- The `persqa` module implements personalized smalltalk to store and recall trivia like name, hobby or occupation of an interlocutor.
- Through `agentqa`, an application can provide the interlocutors with answers to basic personal questions about the dialogue agent.
- The `fillers` module activates on `IDLE:IMPATIENT` to utter conversational filler words when processing an utterance is taking longer.

## 5 User trials

We conducted two user studies in the scope of Bachelor's theses, focusing on various aspects of human-machine interaction policy design, using the robot *Roboy* [22]. Both were implemented using the *Ravestate* framework. Experimental results aided in the evaluation of the proposed system's features, such as modularity and the ability to facilitate multi-modal communication.

### 5.1 Study A

In the first study (*Study A*), 20 users were offered to conduct an *open-domain dialog* with a chatbot in textual form and with a humanoid robot as a spoken conversation. In the first part of the experiment only *Ravestate's* language-related input and output modalities were enabled: text messages for the chatbot and speech recognition and synthesis for physical robot. In the second part of the experiment, with the goal to assess the participants' perception of the dialogue with additional modalities, we extended *Ravestate's* outputs to include facial expressions (e.g. surprised, shy, affectionate) and head movements for the physical robot, and emojis for the chatbot. Every person spent 5 minutes conversing with the chatbot and 5 minutes talking to *Roboy* in person. Note that the utterance-response examples from Appendix ?? originate from this study.

Participants answered 14 questions on a five-point Likert scale from "1 - Not at all accurate" to "5 - Extremely accurate" for both conversations. The keywords "robot" and "chatbot" have been replaced by system. The resulting means and standard deviations of the answers to the five-point scale questions are shown in Table 1.

### 5.2 Study B

The second study (*Study B*) was designed as a *goal-oriented interaction* between the physical robot and a user. Here, we observed 18 users playing charades - a word guessing game - with the physical robot. We evaluated user responses to a new modality: visual feedback. A real-time human activity recognition system was integrated with *Ravestate* which provides information about the current action of the interlocutor. In addition to visual feedback, speech recognition, speech synthesis and facial expression I/O modules were enabled. On average participants spent 8 minutes 36 seconds playing with the robot. Since the interaction with *Ravestate* in this experiment was exclusively embodied, we employed metrics from human-robot interaction research and evaluated the agent along the dimensions of perceived intelligence,



Question	$\mu$ All	$\sigma$ All
The system understood what I said.	2.65	0.80
I understood what the system said. <sup>3</sup>	3.15	0.67
I was anxious during the conversation.	1.57	0.87
The system's answers were out of context.	3.25	0.95
The utterances of the system were unvaried .	2.55	0.88
The system was repeating itself.	2.77	1.23
The system showed some kind of emotion.	2.90	1.15
The system was friendly.	3.98	0.80
The conversation felt slow.	3.27	1.43
The conversation was tedious.	2.90	0.96
I was able to gather new information in the conversation.	2.58	1.28
It felt similar to a dialog with another human.	1.85	0.92
I enjoyed the conversation.	3.60	1.01
My overall impression of the conversation was good.	3.15	0.92

**Table 1:** Study A survey results

likability, sociability and animacy among other characteristics.

The *Study B* survey uses a semantic differential scale, which provides a scale from 1 to 5 with so-called anchors on each end (words with the opposite meaning). The questionnaire measures five key concepts of HRI: anthropomorphism (ATPR), animacy (ANIM), likability (LIK B), perceived intelligence (PINT), and perceived safety (PSAF). Each of the variables is represented by a group of related questions, which form a single scale.

### 5.3 Sample Conversations

Figure 12 shows hand-picked examples from interaction logs of users with *Ravestate*, showcasing specific mechanisms of the Signal-Rule-Slot automaton as they play out with modules from our reference implementation.

While *wildtalk* answers the first two questions of *Example 1* using generated responses, *genqa* responds to the last one with a general knowledge answer.

*Example 2* demonstrates the usage of active engagement. At first, *genqa* answers the question from the participant. After a pause in the conversation, proactive engagement (see Appendix ??) leads to *persqa* asking about personal information of the conversation partner. *persqa* then stores the answer in a

Question	Concept	$\mu$ All	$\sigma$ All
Fake/Natural	ATPR	3.28	0.96
Machinelike/Humanlike	ATPR	2.83	0.86
Unconscious/Conscious	ATPR	3.11	0.90
Artificial/Lifelike	ATPR	2.94	0.80
Dead/Alive	ANIM	3.72	0.89
Stagnant/Lively	ANIM	3.06	1.00
Mechanical/Organic	ANIM	2.56	1.04
Artificial/Lifelike	ANIM	2.78	0.81
Inert/Interactive	ANIM	3.22	1.22
Apathetic/Responsive	ANIM	3.61	0.70
Dislike/Like	LIK B	4.28	0.75
Unfriendly/Friendly	LIK B	4.50	0.71
Unkind/Kind	LIK B	4.28	0.67
Unpleasant/Pleasant	LIK B	4.22	0.81
Awful/Nice	LIK B	4.39	0.70
Incompetent/Competent	PINT	3.39	0.78
Ignorant/Knowledgeable	PINT	3.17	0.79
Irresponsible/Responsible	PINT	3.17	0.99
Unintelligent/Intelligent	PINT	3.06	0.64
Foolish/Sensible	PINT	3.50	0.51
Anxious/Relaxed	PSAF	4.33	0.69
Agitated/Calm	PSAF	4.22	0.88
Quiescent/Surprised	PSAF	3.28	0.67


**Table 2:** Study B survey questions and results

database and generates a positive response to show that the answer was understood.

*Example 3* includes multiple modules and active engagement. After *hibye* reacts to the greeting, *agentqa* answers the question about the dialogue agent. Triggered by the mention of the dialogue agent in the input, *emotion* generates a signal for shyness which *telegramio* converts into an emoji. In addition to *emotion*, *wildtalk* reacts to the input and generates a response. *agentqa* prompts the user to ask about the dialogue agent as proactive engagement. The corresponding question by the participant is then answered by *agentqa*.

### 5.4 Discussion of Results

Both studies demonstrated the ability of the proposed framework to cope with a variety of modalities: Written language and spoken language, as well as bidirectional visual feedback in the form of facial expressions display and vision-based human activity recognition. In *Study A*, participants ranked various aspects of the interaction on a Likert scale from 1 (not at all accurate) to 5 (extremely accurate). The conversation with the chatbot ( $\mu = 2.10$ ,  $\sigma = 0.88$ ) was deemed to be more similar to another human than the conversation with the robot ( $\mu = 1.50$ ,  $\sigma = 0.97$ ). Similarly, the rankings for "The system understood what

Input	Output
<i>Example 1</i>	
will it rain today	i don't like rain
know if it's gonna rain today	yeah it is going to rain
when did was start to rain	And the answer is 1985
<i>Example 2</i>	
what is your favorite animal	i know for sure, it must be monkey
	which town do you call your home?
hannover	oh, i should visit hannover.
<i>Example 3</i>	
Hello royboy	greetings
how are you	my lawyer says i don't have to answer that question.
What are you up to today	 nothing much
	is there anything you want to know about me?
How old are you?	being 1 years old means i am still a child.

**Figure 12:** Our system in a conversation. Color coding: *wildtalk*, *genqa*, *persqa*, *hibye*, *agentqa*, *emotion*

*I said*" statement are lower for the robot ( $\mu = 2.10$ ,  $\sigma = 0.74$ ) vs. chatbot ( $\mu = 2.90$ ,  $\sigma = 0.74$ ) - which can be partially attributed to unreliable automatic speech recognition. *Study A* respondents found *Roboy* rather friendly in both parts of the experiment ( $\mu = 3.98$ ,  $\sigma = 0.8$ ). According to the survey results, *Study B* participants clearly noticed the newly added modalities and, based on our observations, spent more time looking at the robot's face. Moreover, *Study B* showed that integrating visual feedback into the agent's responses increases perceived friendliness even more ( $\mu = 4.5$ ,  $\sigma = 0.71$ ). Similar to *Study A*, in the second experiment all of participants also noticed the changes in *Roboy*'s facial expressions and were able to provide explanations for them. Furthermore, 94% of the respondents reported that they experienced empathy (35.29%), joy (29.41%) or slight frustration (23.53%) while playing charades with the robot. Both studies were conducted with *Ravestate*, indicating the framework's utility as a design tool for in-depth HRI research experiments.

## 6 Conclusion

The main contributions of this paper are

1. The Signal-Rule-Slot automaton, a conceptual combination of grammatical symbol systems with slots for intrinsic rule exclusion.
2. The causal pathway self-information metric which derives an estimate for the prior probability of a signal from the structure of a Signal-Rule-Slot automaton.
3. The *Ravestate* software framework for design and deployment of multimodal interactive behavioural policies based on the Signal-Rule-Slot system, including a structured set of diverse rule modules.

Experimental evidence suggests that the integration of signals, rules and slots adds to the foundation of rule-based interactive systems, improving on its predecessors by simplifying descriptions of concurrent multimodal behaviors. We have shown that *Ravestate*, implementing the Signal-Rule-Slot system, can effectively employ a contextual selection of task-oriented rules in parallel to specific generative models, enabling effective human-robot interaction policy design. As neural models are reaching human performance on narrow tasks, symbol systems provide an efficient method to implement a switch between them.

## 7 Future Work

In its current form, the Signal-Rule-Slot system is suitable for modular declarative interaction design. We recognize potential for future work in the system, such as a lack of autonomous long-term goal selection, which prevents the Signal-Rule-Slot architecture from more direct comparison with mature cognitive architectures such as SOAR. Also, additional experiments could be performed to prove the framework's robustness towards user-initiated context changes. Findings from both user studies also indicate an acute need to improve reliability of human-agent interfaces, such as automatic speech recognition and vision. However, we are confident that further research with the presented system would strengthen the role of hybrid probabilistic symbol systems in interaction policy design, especially when rule formation would be aided by learning [23].

## References

- [1] H. Bunt. "Context and Dialogue Control". In: *THINK Quarterly* 3 (1994), pp. 19–31.

- [2] Sheizf Rafaeli. "From new media to communication". In: *Sage annual review of communication research: Advancing communication science* 16 (1988), pp. 110–134.
- [3] J. Harms et al. "Approaches for Dialog Management in Conversational Agents". In: *IEEE Internet Computing*. Vol. 23. 2. 2019, pp. 13–22.
- [4] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *CoRR abs/1412.6572* (2015).
- [5] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. ISSN: 0027-8424. DOI: 10.1073/pnas.1611835114. eprint: <https://www.pnas.org/content/114/13/3521.full.pdf>. URL: <https://www.pnas.org/content/114/13/3521>.
- [6] Elan Ruskin. "Rule Databases for Contextual Dialog and Game Logic". In: *Game Developers Conference*. 2012.
- [7] N. Webb. *Rule-based Dialogue Management Systems*. 2000.
- [8] R. Callan. "Artificial Intelligence". In: Red Globe Press, 2003. Chap. 1.5.1.
- [9] Rodney A Brooks. "Elephants don't play chess". In: *Robotics and autonomous systems* 6.1-2 (1990), pp. 3–15.
- [10] J. Laird et al. "An Analysis of Soar As an Integrated Architecture". In: *SIGART Bull* 2.4 (1991), pp. 98–103.
- [11] Staffan Larsson and David R Traum. "Information state and dialogue management in the TRINDI dialogue move engine toolkit". In: *Natural language engineering* 6.3 & 4 (2000), pp. 323–340.
- [12] J. R. Anderson, M. Matessa, and C. Lebiere. "ACT-R: A Theory of Higher Level Cognition and Its Relation to Visual Attention". In: *Hum.-Comput. Interact.* 12.4 (Dec. 1997), pp. 439–462. ISSN: 0737-0024. DOI: 10.1207/s15327051hci1204\_5. URL: [https://doi.org/10.1207/s15327051hci1204\\_5](https://doi.org/10.1207/s15327051hci1204_5).
- [13] J. Anderson et al. "An Integrated Theory of the Mind". In: *Psychological review* 111 (Nov. 2004), pp. 1036–60.
- [14] Frank Van der Velde and Marc De Kamps. "Neural blackboard architectures of combinatorial structures in cognition". In: *Behavioral and Brain Sciences* 29.1 (2006), pp. 37–70.
- [15] Gabriel Skantze and Samer Al Moubayed. "IrisTK: a statechart-based toolkit for multi-party face-to-face interaction". In: *Proceedings of the 14th ACM international conference on Multimodal interaction*. 2012, pp. 69–76.
- [16] Pierre Lison and Casey Kennington. "Opendial: A toolkit for developing spoken dialogue systems with probabilistic rules". In: *Proceedings of ACL-2016 system demonstrations*. 2016, pp. 67–72.
- [17] Gokhan Tur and Renato De Mori. *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons, 2011.
- [18] John McDermott and Charles Forgy. "Production system conflict resolution strategies". In: *Pattern-directed inference systems*. Elsevier, 1978, pp. 177–199.
- [19] M. Honnibal and I. Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing". To appear. 2017.
- [20] T. Wolf et al. "TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents". In: *CoRR abs/1901.08149* (2019). arXiv: 1901.08149. URL: <http://arxiv.org/abs/1901.08149>.
- [21] D. Chen et al. "Reading Wikipedia to Answer Open-Domain Questions". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 1870–1879.
- [22] S. Trendel et al. "CARDSFlow: An End-to-End Open-Source Physics Environment for the Design, Simulation and Control of Musculoskeletal Robots". In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2018, pp. 245–250.
- [23] Ronald Römer et al. "Reinforcement learning of minimalist grammars". In: *arXiv preprint arXiv:2005.00359* (2020).