Veracity Technology Spearhead

Enabling end-to-end veracity within value exchange ecosystems

Using the Event Calculus to explore Veracity Logic on a timeline

Josh Carter University of Otago
Stephen Cranefield University of Otago
David Eyers University of Otago

Motivation

The *veracity logic* allows formal reasoning about claims (statements made by people with associated evidence) and assertions of graded trust relationships between people. It defines rules for constructing more complex claims from smaller claims, while collecting the combined evidence for them and the degree of trust in the claim. However, it does not consider how claims and trust relationships change over time. In this work we allow reasoning over changing claims and trust relationships by combining the veracity logic with the *event calculus*, a formal language for reasoning about how information changes over time, given a narrative of events.

Common-sense reasoning over time: Discrete Event Calculus

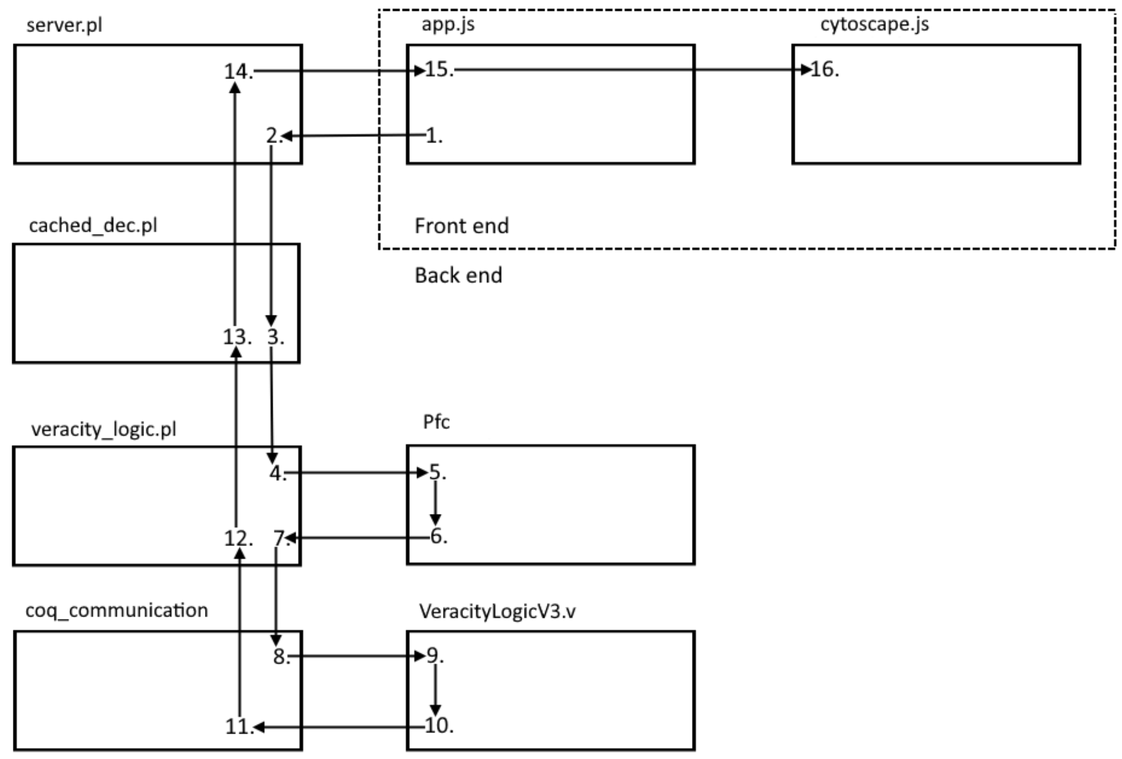
The Discrete Event Calculus provides a straightforward mechanism to perform reasoning about how *events* that occur at points in time (discretised) initiate and terminate various states of affairs—*fluents*. Practical implementations can be achieved using small amounts of logic programming (Prolog in this case), which is ideal for rapid prototyping.

Coq proof assistant

Coq is an automated proof assistant that can programmatically and formally verify the logic derivations encountered by our application.

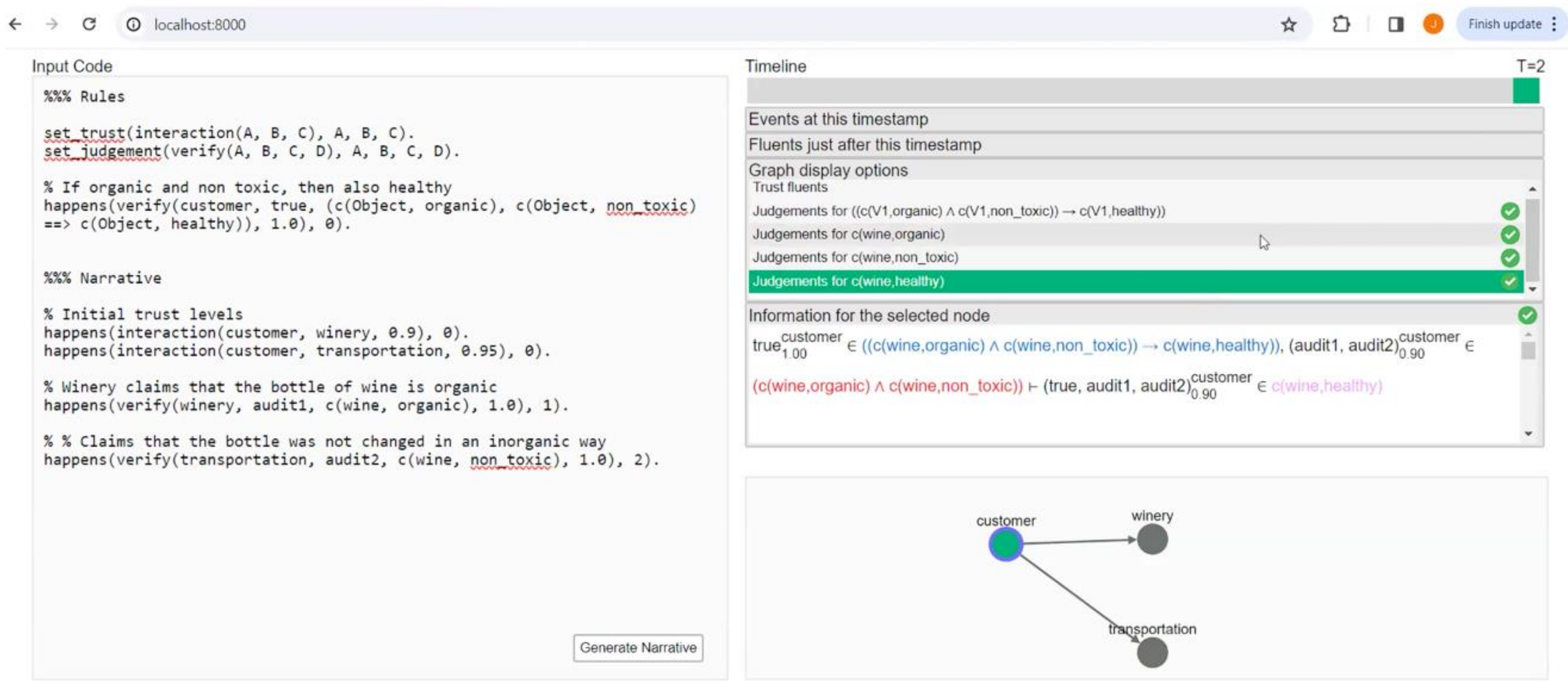
Software architecture and back-end interaction sequence

We include a depiction of the software architecture of our demonstrator in the top-right of this poster. In step 2, a web server component uses the Event Calculus (cached_dec.pl—step 3) to generate the complete narrative. This may involve updates related to the Veracity Logic, which are handled by veracity_logic.pl—step 4.



Step 5 uses the Prolog forward chaining library (PFC) to determine what new Veracity Logic derivations can be made. This involves addition of trust and implication fluents, and removing fluents that are no longer supported. When completed (step 6) control passes back to the Veracity Logic module, which grounds proofs that have multiple sources of verification (step 7).

Coq is passed the representation for each judgement in step 8 (using a pipeline to the Coq tool—steps 9–10) with the overall success of the judgement's proof passed back to the Veracity Logic module (steps 11–13). Finally, the server translates the results into JSON for visualisation within the web front-end components.



Demonstrator: a web application that facilitates evaluation and visualisation of Veracity Logic conclusions over time

The above screen capture is from the web application that was developed in this project. On the left pane, the input to the simulation and visualisation framework are shown: the 'happens' clauses encode Event Calculus events that occur at points in time, indicating the interactions between parties and the Veracity Logic effects of those interactions. On the right-top pane, the timeline control allows moving to different points in time, which updates the display of the events and fluents from the Event Calculus. The "Graph display options" panel determines what is visualised in the graph display on the bottom-right pane along with additional details about any selected node. In this screen capture, information is being shown regarding why a given customer believes that the wine that they have taken delivery of is correctly certified as organic.

University of Otago
ŌTĀKOU WHAKAIHU WAKA

National Science Challenges
SCIENCE FOR TECHNOLOGICAL INNOVATION
Kia kotahi mai – Te Ao Pūtaiao me Te Ao Hangarau

Victoria University of Wellington
University of Auckland
University of Otago
University of Canterbury
University of Waikato