cognitive control of digital twins in swarm systems

Dave Raggett <dsr@w3.org> ERCIM

8 July 2025

Grateful acknowledgement for support from the Nephele and SmartEdge projects funded by the European Union's Horizon Europe research and innovation programme under grant agreements 101070487 and 101092908

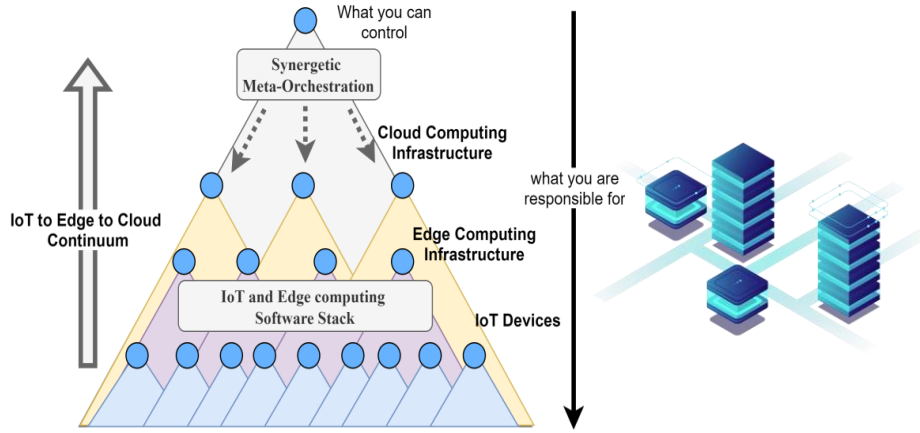
Overview

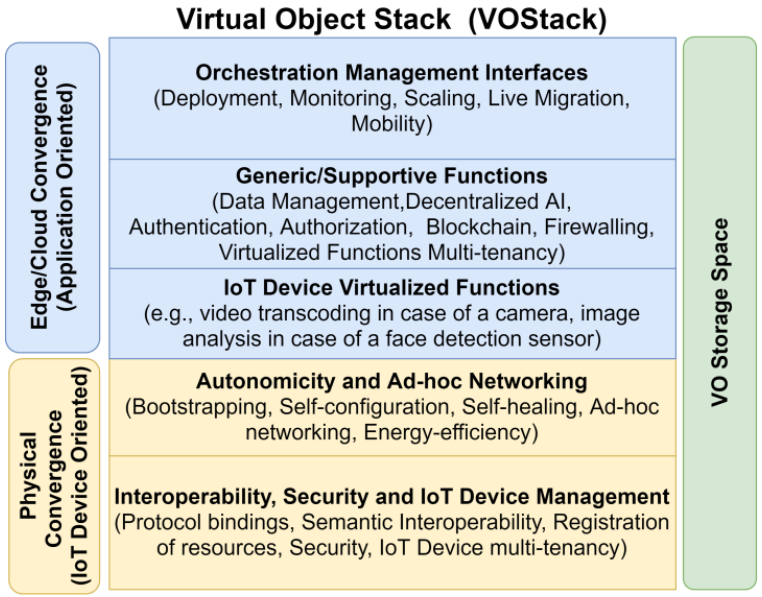
- Acknowledgements
- Digital Twins
- Web of Things
- RDF 1.2 and related standards
- NGSI-LD and RDF
- Web-based monitoring, orchestration and simulation
- Chunks & Rules for low-code adaptive control
- Extension to swarm computing
- Where next with AI?

Nephele Project

https://nephele-project.eu

- Part of the MetaOS cluster of projects
  - and EUCloudEdgeIoT.eu
- IoT and edge computing software stack
- Focus on orchestration across the cloud and edge using a system of systems approach
  - Goal-based meta-orchestrator with machine learning
  - Hyper-distributed applications
  - Federated resource manager
  - Compute continuum network manager
- Use cases in disaster recovery, container port logistics, building energy management and remote healthcare services





Grant agreement ID: 101070487, September 2022 – September 2025

SmartEdge Project

https://www.smart-edge.eu

- Part of the Swarm computing cluster of projects
- Focus on semantic low-code programming tools for edge intelligence
  - Continuous semantic integration enabling device interaction through standardised semantic interface
  - Dynamic swarm network with automatic discovery and real-time swarm formation
  - Low-code tool chain featuring multimodal data fusion
- Use cases in manufacturing, automotive and healthcare

Grant agreement ID: 101092908, January 2023 – December 2025

Digital Twins

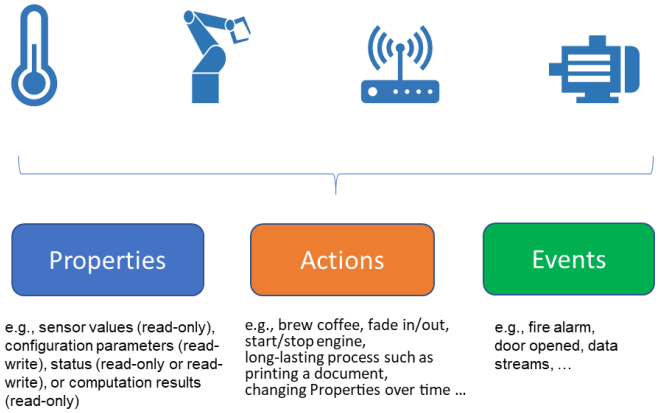
- I will use “Digital Twins” as a term for computer models of physical systems and processes
- Digital Twins can be used for:
  - monitoring current state of their physical twins
  - controlling their physical twins
  - diagnosing and fixing faults using causal models
  - planning and simulation
  - optimisation based upon applying machine learning to recorded data, e.g. in respect to zero-defect manufacturing

A robot arm with its digital twin

Web of Things

Protocol Agnostic

https://www.w3.org/WoT/



- W3C Web of Things (WoT) simplifies application development by decoupling applications from the details of the underlying protocols
- Digital Twins are modelled as objects with properties, actions and events
- RDF (JSON-LD) is used for declarative descriptions of the object model, the semantics, communication protocols and security settings
- Applicable to modelling locally connected devices, devices at the edge, and digital twins in the cloud

Semantic Interoperability = shared understanding

- W3C Web of Things activity
  - WoT architecture
  - WoT Thing Descriptions
  - WoT Scripting API
  - WoT Discovery
  - WoT Bindings Templates
  - WoT Security and Privacy
  - WoT Working Group, Interest Group and Community Group

Working Group Chairs

Sebastian Käbisch
Siemens

Michael Koster
Invited Expert

Michael McCool
Intel

RDF 1.2 and related standards

- RDF is W3C's suite of standards for the Semantic Web as envisioned in the following article
  - Scientific American, May 2001
- Knowledge Graphs modelled in terms of triples: *<Subject, Predicate, Object>**
  - Triples as labelled directed edges between vertices
- Global identifiers with IRIs as the basis for standardised vocabulary terms
  - IRI = internationalized resource identifier
- RDF Semantics are independent of the serialization, e.g.
  - RDF/XML, Turtle, N-Triples, JSON-LD, N3
- Queries with SPARQL
- Ontologies with RDF schemas and OWL
- Shape constraints with SHACL

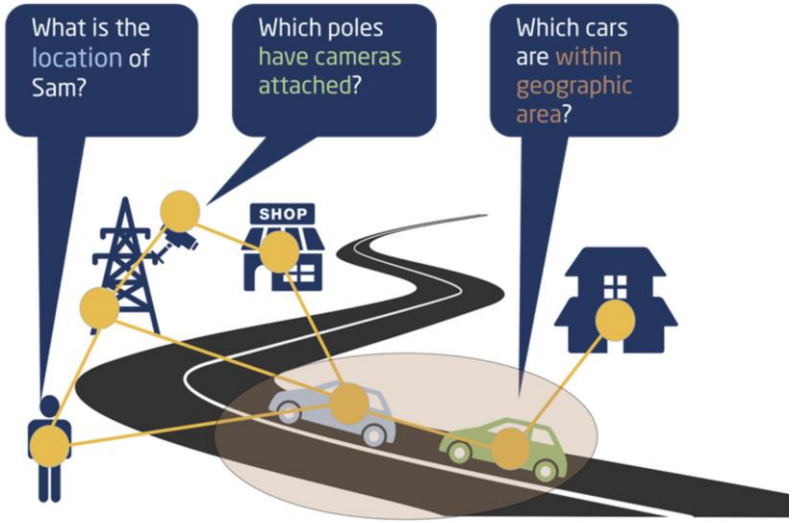
- RDF first draft released in 1997
- RDF 1.0 released in 1999
- RDF 1.1 released in 2014
- W3C is now finalising RDF 1.2 which is designed to simplify expressing annotations to triples
  - Use of RDF triples as a triple term in the object position of another triple

    ```
    _:x rdf:reifies <<(:Alice :hasName "Alice")>>
    _:x :accordingTo :Bob
    ```
  - Backwards compatible with earlier versions of RDF
- RDF 1.2 is motivated in part by the success of labelled property graphs
  - Properties for graph Vertices *and* Edges
- Basis for semantic interoperability for information held in a variety of different formats
  - RDBMS, LPG, CSV, PDF, Spreadsheets, …

* There is also a four-place version with an extra parameter for named graphs

NGSI-LD and RDF

- NGSI-LD is a standard for publishing, querying and subscribing to (IoT) context information via REST protocol

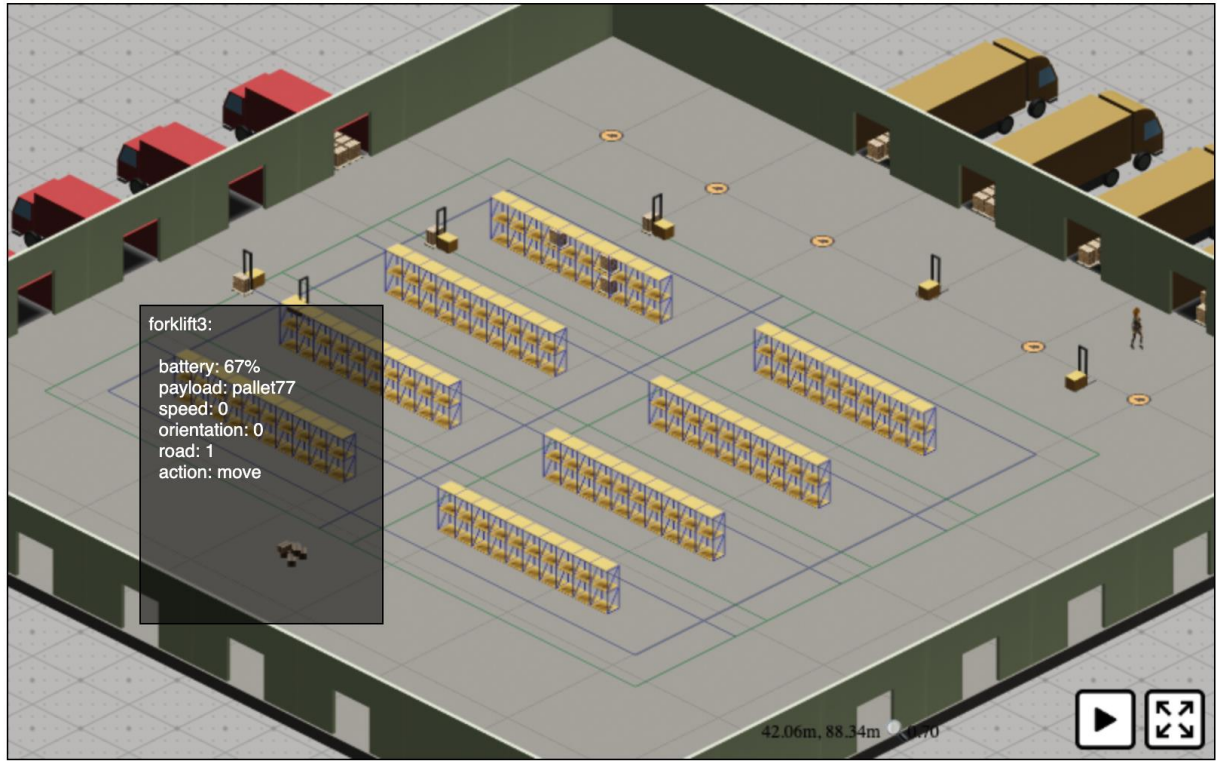


Courtesy of ngsild.org

- Standardised by ETSI through ISG CIM, inspired by OMA's NGSI specifications and the FIWARE community
- Semantic interoperability using JSON-LD, but originated using Labelled Property Graphs (LPG)
- W3C and ETSI holding regular liaison meetings to drive alignment with W3C's Web of Things
- Extending WoT Thing Descriptions to support NGSI-LD information servers

Web-based monitoring, orchestration and simulation

- W3C standards for web browsers enable web-based monitoring, orchestration and simulation
  - HTML5, WebGPU, WebXR, WebNN, …
- Use web sockets to stream state updates to the web page
  - Local prediction for smooth animation
- 2.5D isometric rendering akin to SimCity with offscreen 3D rendering for robot arms, etc.
- Situation Reports describing what's happening, departures from the plan, and where attention is needed



SimSwarm: a Web-based simulation of a smart warehouse

WebGPU is a higher performance replacement for WebGL

Simple Commands

- When fixed repetitive behaviour is sufficient for robots and other machinery, commands can be declared as JavaScript objects with properties
- The example on this page refers to robot forklifts and pallets in a smart warehouse
- Implemented under NodeJS with an integrated web server supporting HTTP and WebSockets
- Command interpreter can be used to
  - either map commands to robot control protocol, e.g. ROS
  - or to run a swarm simulation
- WebSockets used to stream the state to connected web pages that visualise the swarm
- By default, the *next* action causes control to jump back to first command
  - You can also jump to a given sequence of commands, or even to make a random choice between several such sequences

See: https://github.com/w3c/cogai/tree/master/demos/Swarms/visualise

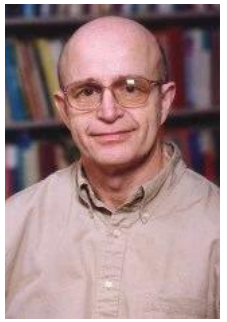
```
// declare the scene components
let pallet1 = new Pallet({
   name:"pallet1",
   x:20,
   y:30,
   orientation:0,
   loaded:true
});

let forklift1 = new Forklift({
   name:"forklift1",
   x:10,
   y:30,
   z:0,
   orientation:0,
   speed:1,
   forkspeed:0.5,
});

// declare sequence of commands
forklift1.setTask([
   {act:"wait", time:4},
   {act:"grab", pallet:"pallet1"},
   {act:"move", x:10, y:30},
   {act:"release"},
   {act:"move", x:20, y:30},
   {act:"grab", pallet:"pallet1"},
   {act:"move", x:20, y:30},
   {act:"release"},
   {act:"move", x:10, y:30},
   {act:"next"}
]);
```

Cognitive Approaches to Low-Code control

Robots are currently programmed to repeat simple actions over and over again. For greater resilience we need more flexible behaviour that dynamically adapts to changes in the context.

Inspired by decades of work in the cognitive sciences, we can use facts and rules to model agents in terms of event-driven concurrent threads of behaviour. Reasoning is decoupled from real-time control, mimicking the human brain where the conscious mind delegates actions to the cortico-cerebellar circuit that manages large numbers of muscle activations.

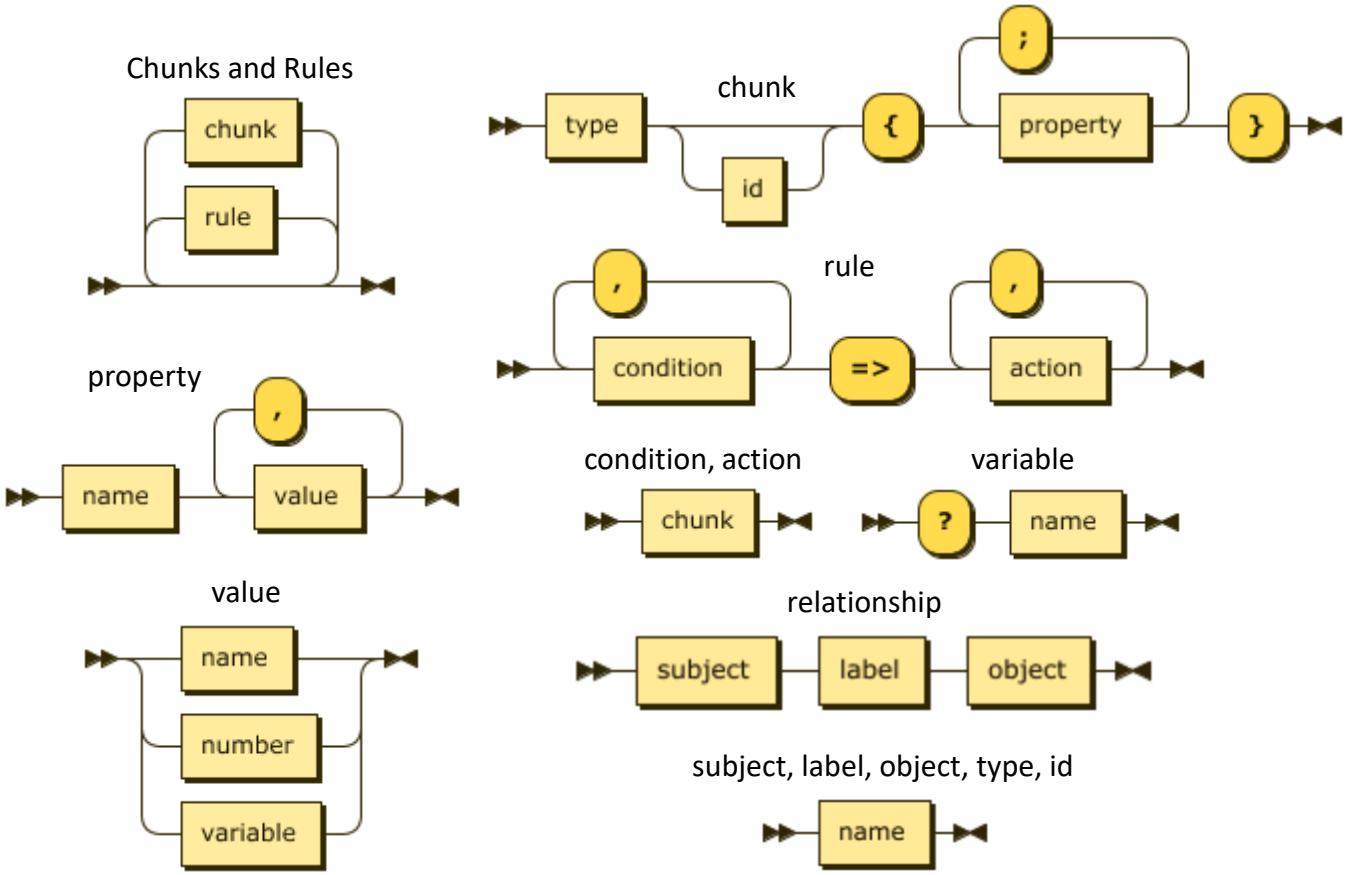
I'd like to acknowledge John R. Anderson, who is a pioneer in the cognitive sciences, and well known for his work at CMU on ACT-R, a computational theory about the operation of the mind for use in practical tests with human subjects.



Chunks and Rules*

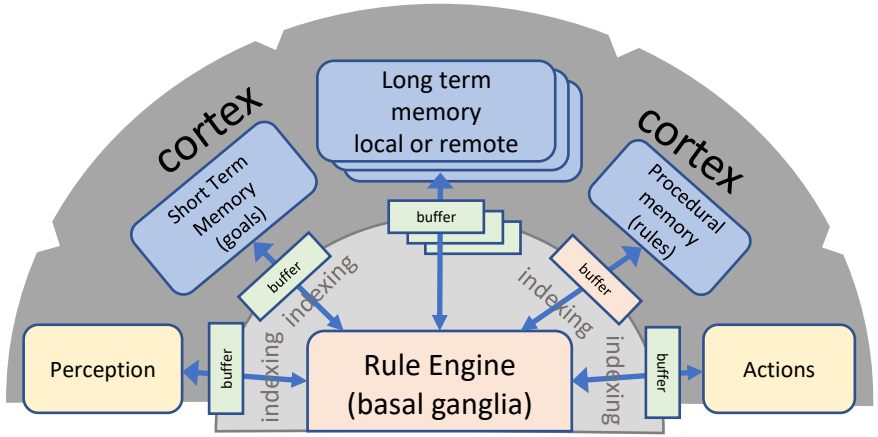
higher level than RDF for facts and rules

Simple syntax as railroad diagrams



names beginning with "@" are reserved, e.g. @do for actions

Cognition – Sequential Rule Engine



Cognitive Buffers hold single chunks
Analogy with HTTP request-response model

- Inspired by John Anderson's ACT-R and decades of cognitive science research at CMU and elsewhere
- Mimics characteristics of human cognition and memory, including stochastic recall, spreading activation and the forgetting curve
- Rule conditions and actions specify which cognitive module they apply to with goal module as default
- Variables are scoped to the rule they appear in
- Actions either directly update the buffer or invoke operations on the buffer's cortical module, which asynchronously updates the buffer
- Predefined suite of built-in cortical operations
- Reasoning decoupled from real-time control over external actions, e.g. a robot arm

* From the W3C Cognitive AI Community Group. See the chunks & rules specification.

Tutorial Example: Let's count

- Count up from one digit to another, e.g. from 2 to 5 yielding the sequence *2, 3, 4, 5*
- Uses three modules: goals, facts, rules

  facts and *rules* initialised from files, see right:
- Start by using scripting API to queue chunk to *goal* module buffer to trigger rule matching

  count {start 2; end 5; state start}
- First rule updates *goal* module's buffer to

  count {start 2; end 5; state *counting*}

  The default @do action (@do update) synchronously updates the buffer by merging the properties, in this case leaving *start* and *end* unchanged
- The first rule initiates a query on the *facts* module's chunk graph using @do get to find a matching chunk and @do show to show the value

  Note use of @module to select *facts* module, where the default is the goal module
- Second rule iterates through the *increment* chunks displaying the number

  Note use of two condition chunks that match the goal and facts module buffers, and use of the ! operator to ensure that start and end are different
- Third rule halts the iteration on final digit, setting goal buffer *state* property to *stop*

  This works because the goal module buffer is count {start 5; end 5; state counting}

facts.chk

```
# Declarative knowledge for counting demo
increment {number 1; successor 2}
increment {number 2; successor 3}
increment {number 3; successor 4}
increment {number 4; successor 5}
increment {number 5; successor 6}
increment {number 6; successor 7}
increment {number 7; successor 8}
increment {number 8; successor 9}
increment {number 9; successor 10}
```

rules.chk

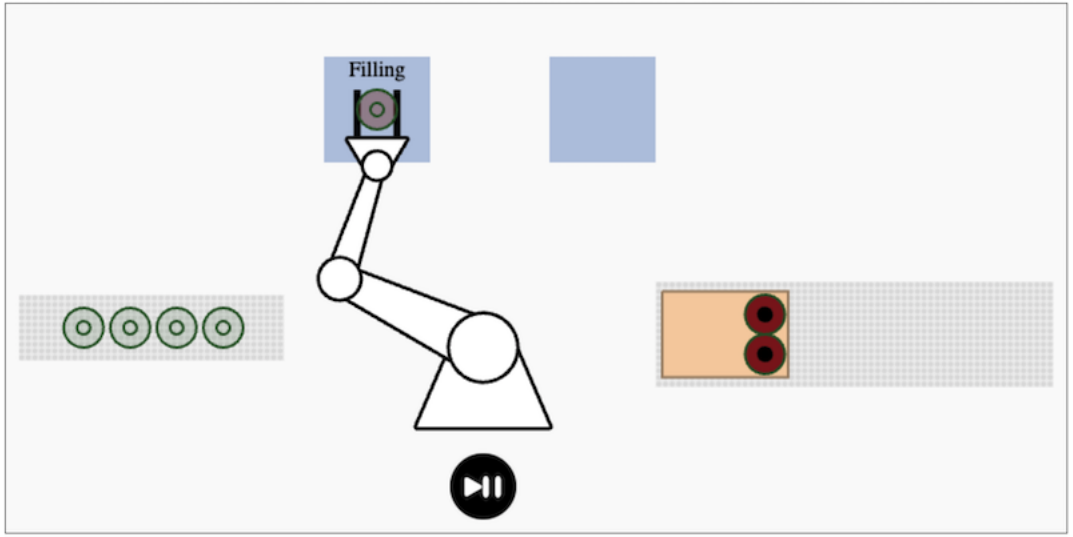
```
# prepare to count up
count {start ?num; state start} =>
    count {state counting},
    increment {@module facts; @do get; number ?num},
    console {@do show; value ?num}
# count up one at a time
count {state counting; start ?num1; end !?num1},
  increment {@module facts; number ?num1; successor ?num3} =>
    count {start ?num3},
    increment {@module facts; @do get; number ?num3},
    console {@do show; value ?num3}
# stop after last one
count {start ?num; end ?num; state counting} =>
    count {state stop}
```

See: https://github.com/w3c/cogai/tree/master/demos/counting

Chunks and Rules

- Mature JavaScript library for use in webpages or with NodeJS
  - Along with specification and test suite
  - Fast execution as actions are handled asynchronously
- Application script can declare custom operations, e.g. for robot control, layered above ROS operations
  - Implemented in JavaScript using real-time clock as well as networking for external messaging
  - Can be used with the ROS JavaScript library for NodeJS
- Iterative refinement as new requirements come to light, e.g. in unusual situations like faults, or where humans have intervened in an unexpected way
  - Ongoing need for curation of use cases
- Stochastic behaviour where needed (video)
  - If several rules match the chunk buffers, then one of them will be randomly selected for execution
  - Rules with same conditions and different actions

Note: rules can use variables for dynamic parameters
e.g. where an object location is determined at run-time.



Factory demo: filling, capping and packing bottles of wine with real-time control over conveyor belts, filling and capping machines, and a robot arm

```
# move robot arm into position to grasp empty bottle
# when move completes queues chunk: do {step 2}
do {step 1} =>
   robot {@do move; x -170; y -75; angle -180; gap 30; step 2}

# grasp bottle and move it to the filling station
do {step 2} =>
   goal {@do clear},
   robot {@do grasp},
   robot {@do move; x -80; y -240; angle -90; gap 30; step 3}
```

See also smart homes demo

Chunks and Rules

https://w3c.github.io/cogai/chunks-and-rules.html

- Test suite and suite of demos
- Whenever any of the module buffers are updated, the rule engine checks to see which rules match and randomly picks one of them for execution
- Rule actions are asynchronous except for *@do update* and *@do clear* which synchronously update the buffers
  - *@do clear* clears module buffer
- A large suite of built-in operations on chunk graphs using *@do* with
  - *clear, delete, get, next, patch, properties, put, queue* and *update**

* *get, put, patch* and *delete* borrowed from HTTP method names

- Use chunk IDs as property values to reference chunks
  - Linked chunks as knowledge graphs
- Built-in support for iterating through properties, lists and matching chunks
- Chunks can be scoped to contexts using *@context* property
  - Support for statements about statements, beliefs, stories, reported speech, abductive reasoning, etc.
  - Application to episodic reasoning
- Convenient mapping to RDF
  - @rdfmap, @base, @prefix

Extension to Swarm Computing

- Each cognitive agent can control multiple devices
  - In the bottling demo, the agent controls two conveyor belts, the robot arm, as well as the filling and capping machines
- But on a larger scale, we need multiple cognitive agents working together to control many devices
- Agents can send each other asynchronous messages as chunks
  - Either treat as events to trigger behaviour
    - Queue chunk to cognitive module buffer
  - Or use to update the agent's model of the world
    - Update agent's cognitive module chunk graphs
  - Named agents or groups of agents
  - Named topics for interested agents
- Layered on top of existing protocols that offer reliable, timely, in-sequence message delivery
  - e.g. Zenoh, MQTT, DDS, WebRTC, Web Sockets

- Agents send arbitrary chunks with *@message* to identify the recipient, e.g. as in the following rule action property*

```
# tell agent4 to start
start {@message agent4}
```

  - Messages can be sent with *@topic* to subscribers of named topics, e.g.

```
# send stop message on topic12
stop {@topic topic12}
```

- Agents can subscribe to a topic with *@subscribe*, e.g.

```
# subscribe to topic12
listen {@subscribe topic12}
```

* Recipient sees chunk: *start{@from agent1}*
You are free to set the chunk type and additional chunk properties as needed

See: https://github.com/w3c/cogai/blob/master/demos/Swarms/tasks/README.md

Tasks and Synchronisation

- Actions are akin to JavaScript promises
  - Actions queue chunks to buffers some time in the future
  - Chunk property *@task* identifies a thread of behaviour
- A subtask is initiated with *@do task*, e.g.

  a {@do task; foo bar} *

  Queues chunk *a {@task ID; foo bar}* to trigger first rule
  The chain of rules need to preserve buffer's *@task* property
- Tasks finish when rules indicate success or failure, e.g.

  a {@do done} or *a {@do fail}*

  akin to *resolve* and *reject* in Promises
- Set additional properties to trigger follow on rules, e.g.

  a {@do fail; status unsupported}

  Queues chunk: a {status unsupported} to the module buffer
- Each agent can be executing multiple tasks concurrently
- Synchronise across multiple tasks and agents
  - When all listed tasks have succeeded: *@all*
  - When any of the listed tasks succeed: *@any*
  - When any of the listed tasks have failed: *@failed*

Local task demo, swarm task demo

```
# rule to initiate several subtasks
# queues process2 {} when all these tasks have completed
# queues recover1 {} if any of the tasks have failed

process1 { } =>
   a {@do task; @task ?task1},
   b {@do task; @task ?task2},
   c {@do task; @task ?task3},
   process2 {@all ?task1, ?task2, ?task3},
   recover1 {@failed ?task1, ?task2, ?task3}

# same thing, but this time on different agents
# initiating agent notified when tasks complete

process1 { } =>
   a {@do task; @on agent1; @task ?task1},
   b {@do task; @on agent2; @task ?task2},
   c {@do task; @on agent3; @task ?task3},
   process2 {@all ?task1, ?task2, ?task3},
   recover1 {@failed ?task1, ?task2, ?task3}
```

* *@do task* automatically generates the unique task identifier, which can be accessed via *@task* in rule conditions and actions. Choose the chunk type and additional properties as needed for the task.

Further Features for Exploration

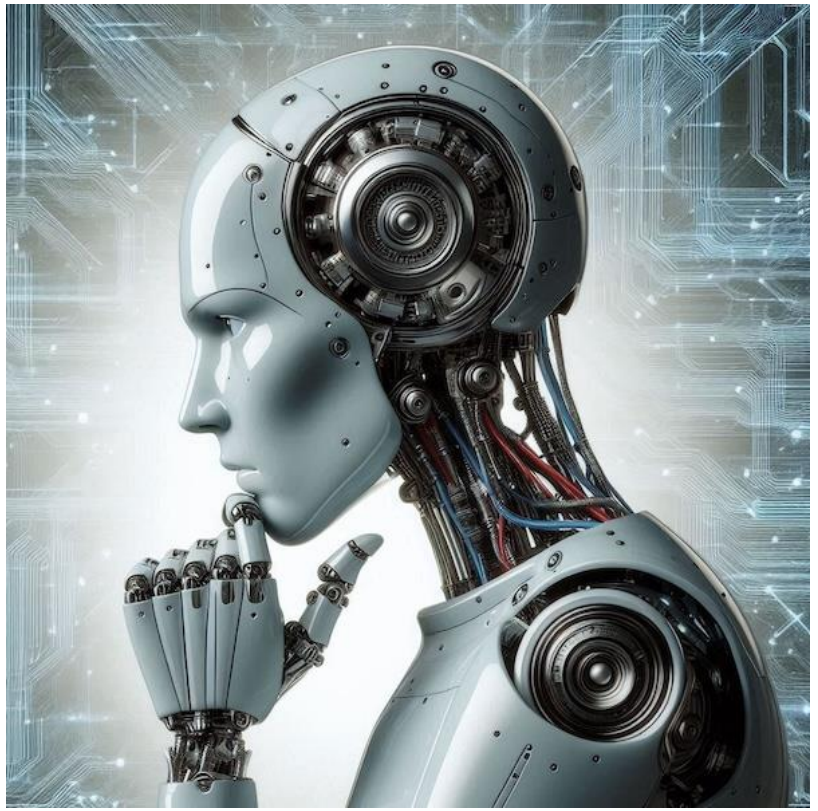
- Further abstractions for swarm intelligence
  - Support for notification and name management when agents enter and leave swarms, including discovery, assignment and transfer of agent roles
  - Techniques for consensus, auctions, automated negotiation and distributed storage, e.g. based upon DHTs and blockchain
- Agents may need to suspend tasks to attend to interruptions
  - @suspend task42
  - @resume task42
- What was I doing and what should I do next?
  - Store details in chunk graph with *@do put*
  - Retrieve details when needed with *@do get*
- Direct authoring of chunks and rules is time consuming
  - Curation of test suites for regression testing
  - Opportunities for applying machine learning
- When it is safe, learning in the real world, or …
  - Safe learning in simulated worlds, and
  - Opportunities for exploiting synthetic data
- Reinforcement learning
  - Adjust rule strengths to reflect their observed utility
  - Stochastic heuristics for generating rules and goals to try out
    - Which rules might be more effective in current context?
    - Use of neural networks to guide learning
    - Learning to learn and deep reinforcement learning
- Task hierarchies and reflective reasoning
  - Learning is more effective if challenges can be decomposed into simpler ones that the agent may already know how to do
  - Reflection (thinking about thinking) can help propose sub-tasks and decide when to abandon the current approach and look for a potentially better one
  - Parameters that influence how single minded the agent is versus a readiness to search for better approaches
- Single agent reinforcement learning generalises to multi-agent reinforcement learning (MARL)
  - Agents collaborate to explore the training space with the possibility of beneficial emergent behaviours
  - Agents can be fully cooperative, or more realistically, can have their own agenda with potentially conflicting goals
  - Agents have imperfect knowledge

Relationship to Previous Work

- Fuzzy Logic with fuzzification and defuzzification, e.g. temperature as 30% cold, 60% warm and 10% hot with separate rules for cold, warm and hot
- IEEE's FIPA (foundation for intelligent physical agents), and based upon speech act theory with intents, requests, and beliefs, subject to a shared ontology
- IEC 61131-3 suite of graphical and text-based languages for programmable logic controllers
- CAYENNE control logic generation for industrial automation, e.g. process control piping and instrumentation diagrams, layered on IEC 61131-3
- Node-RED is a NodeJS framework using web-based flow diagrams and JavaScript for code implementing the nodes
- Mendix as a low-code development platform

- Chunks & Rules is typically deterministic except where non-deterministic behaviour is appropriate, given the application requirements
- Chunks & Rules uses a simple text notation rather than structured diagrams
- Actions are asynchronous, proceeding concurrently with reasoning, mimicking human cognition
- Human learning is initially heavily dependent on cognition, but with repetition, is compiled into fast and subjectively effortless skills, e.g. riding a bicycle or playing a musical instrument
  - Compiled skills equivalent to fuzzy rules
- How we can mimic skill acquisition for multi-agent neurosymbolic systems?
  - Combination of Type 1 and 2 cognition, including reflection, use of analogies and qualitative models

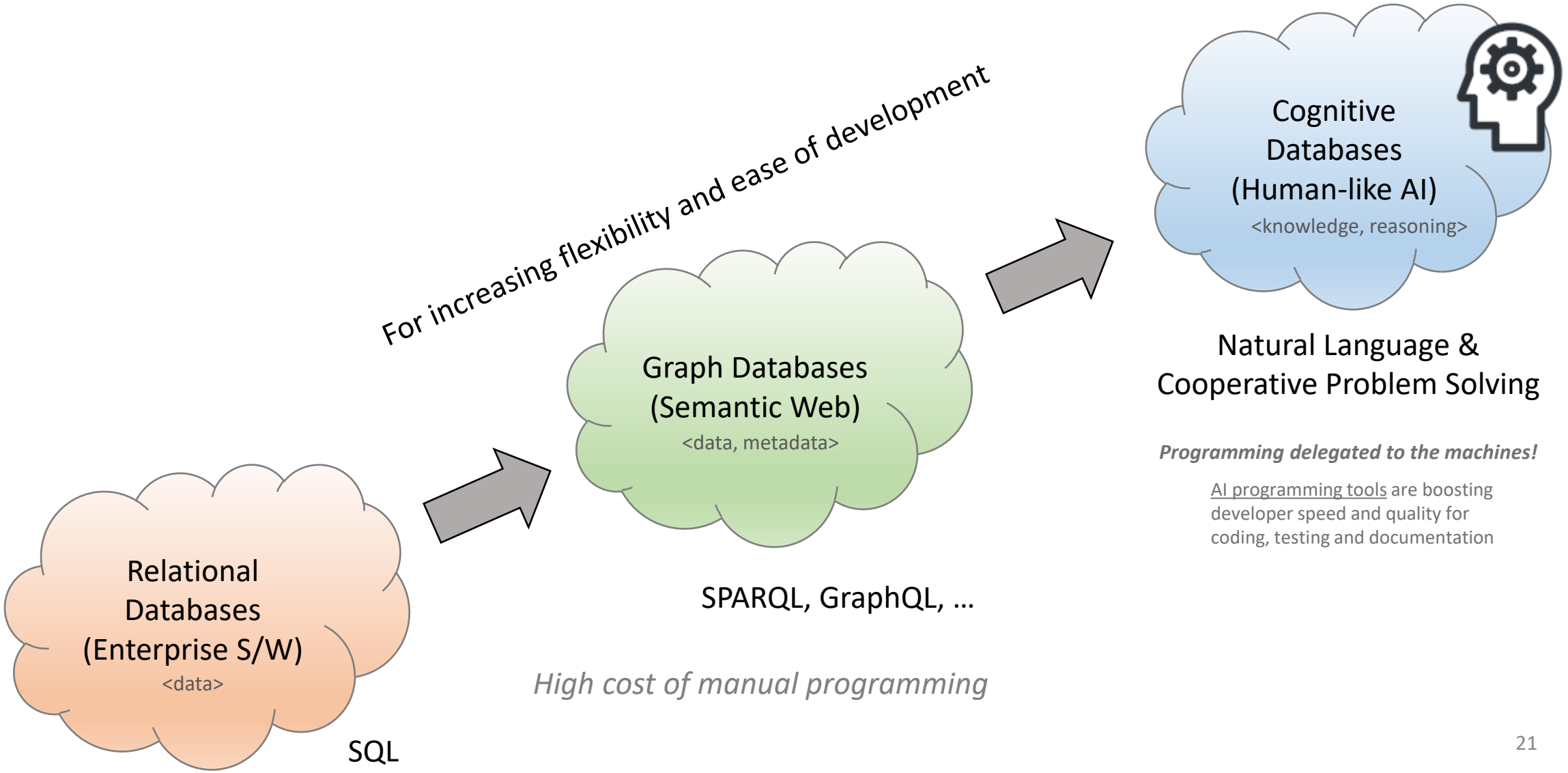
Where next with AI?



Neural networks have shown the huge power of statistical approaches to AI

Symbols as a crude approximation to a more profound statistical model of knowledge

Evolution in ICT Systems



Neural Networks with Fuzzy Rules

- Can we replace manual programming with high level instructions plus a few examples?
- AI agents as skilled collaborators
  - Recent progress in applying Generative AI to software development
  - Learning to learn to enable learning from just a few examples
- Explainability in terms of rules and explicit models of context
  - Car drivers use maps, road signs, and rules of the road plus conventions for negotiating with other drivers in respect to giving way
  - We can verbalise why we took a decision, e.g. I started moving as the traffic lights turned green

- How to bridge the vast gulf between neural networks and rule-based systems in respect to machine learning?
- Fuzzy rules using neural networks
  - Inspired by fuzzy logic*
- Means to verbalise rules in context
  - If it is warm set fan speed to medium
- Two systems for cognition and skills, mimicking the human brain
  1. Conscious deliberate cognition
  2. Unconscious execution of skills
- In a factory, we could have a powerful computer for (1) controlling many cheaper computers for (2)

* Scalar values modelled as fuzzy sets, e.g. {30% cold, 60% warm, 10% hot}
Rules on each element in the set yield fuzzy sets for recommendations

Quantum Leap Forward from Generative AI to *Sentient* AI

- Generative AI is now being applied to implement agents
  - e.g. using langchain framework together with PyTorch
- Enables high level control, e.g. with spoken or written directions in natural language, plus cameras for showing what you mean by examples
- Can be used for high level instructions for supervising simpler robot control systems
  - Overcoming high cost of manually programming robot behaviour
  - Automated situation reports

- **Generative AI** is powerful but has known limitations, e.g.
  - Lacks ability to acquire new skills once deployed, is weak on generalisation and prone to making things up (hallucination)
- **Sentient AI** adds *episodic memory*
  - Continual learning through continual prediction and continual reasoning
  - Agents with awareness of their environment, goals and performance, along with the means to remember and reflect on their past experiences
  - Research is now needed on the technical challenges to realise this breakthrough

For industrial systems we could use high-level agents based on Sentient AI as a means to manage lower-level agents using chunks & rules or neural equivalent.

Comments and Questions?

Will Neural Networks replace Rules?

Rule-Based Systems

Advantages

- Provide a clear and understandable way to express logical relationships, enhancing transparency in decision-making
- Explicit nature of rules enables users to trace the decision-making process, creating transparency in system actions
- Rule-based systems facilitate easy maintenance and debugging in the process
- Scalable and adaptable to changing requirements

Disadvantages

- Lack the ability to learn from experience, restricting their capacity to adapt and improve over time
- May struggle with uncertain or ambiguous information, leading to potential inaccuracies in decision-making
- Managing a large number of rules can become complex, posing challenges in organization

Machine-Learning Systems

Advantages

- Can be designed to adapt to changing data patterns, automatically improving their performance as they learn from new information
- Excel at automating complex tasks, reducing the need for explicit programming and enabling the handling of intricate problems.
- Can be designed to continuously learn and optimize their performance over time, enhancing their ability to make accurate predictions or classifications

Disadvantages

- Machine learning models, particularly complex ones, operate as black boxes, making it challenging to interpret their decision-making processes.
- Effectiveness of machine learning heavily relies on the quality and quantity of training data, and inadequate or biased data can lead to inaccurate predictions.
- ML models may struggle to generalize well to new, unseen scenarios if the training data does not sufficiently represent the diversity of potential situations, leading to poor performance in real-world applications.