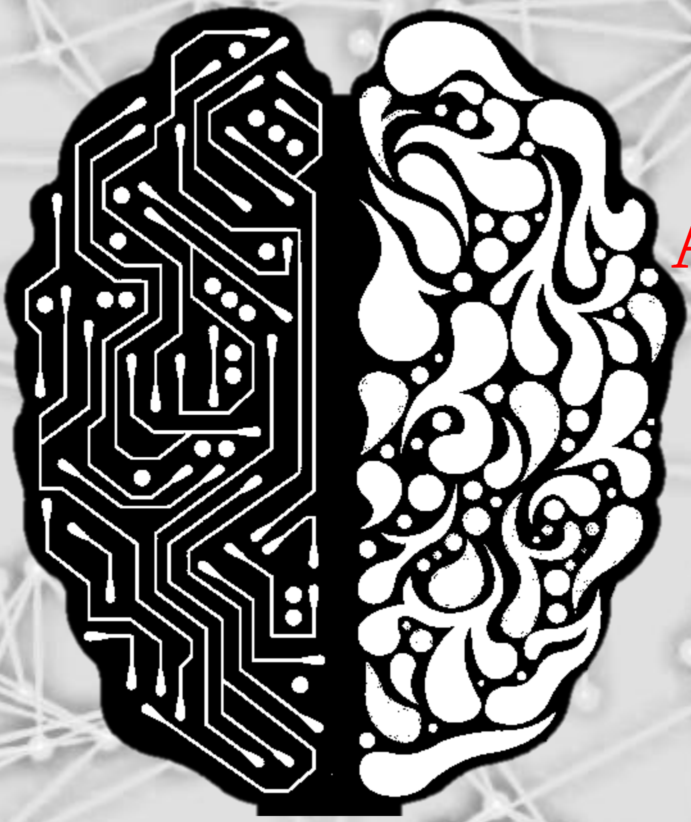


Multi-Agent Systems

Lecture V

- Dr. Nestor Velasco Bermeo,

- Researcher CONSUS (Crop OptimisatioN through Sensing, Understanding & viSualisation),
- School of Computer Science
- University College Dublin (UCD)

Lecture V Learning Objectives

- ❑ Review the characteristics and elements of Agent Oriented Programming and Object Oriented Programming

- ❑ Review the differences between An Agent and an Object

- ❑ Understand the elements and characteristics of an Agent Programming Language

- ❑Understand how Belief Management occurs on a MAS and the temporality of Beliefs
- ❑ Understand and identify the different Commitment States

Agent Oriented Programming

Introduced in 1993 by Yoav Shoham (Stanford).

Based on the idea of programming agents as mental entities.

A complete AOP System includes three primary components:

- a **restricted formal language** with clear syntax and semantics for describing mental states.
- an **interpreted programming language** in which to define and program agents, with primitive commands (such as request and inform).
- an **”agentifier”** (method), converting neutral devices into programmable agents.

Shoham illustrated this through a prototype AOP language, Agent-0.

Agents Vs Objects

- Silva defines an agent as “an extension of an object with additional features”
- Extends the definition of state and behaviour
- Agents have the “freedom” to control and change their behaviors.
- Agents are autonomous.
- Methods are made available for invocation as and when desired;
- Agents do not invoke methods but make “*requests*”
- Objects have nothing to say about differing deductive models like reactive or exhibit social abilities
- Agents are each considered to have their “*own thread of control*”.
- In standard object systems there is merely one thread

Active vs Passive Objects

- Objects do not require external stimuli to carry out their jobs.

- Agents active elements and objects passive ones.

- Active Objects blur the distinction.

- Active objects have their own thread of control and can in some senses be considered autonomous.
- They exhibit some behaviours without actually being operated upon.

OOP and AOP, a comparison

OOP

1. abstract class
2. Class
3. member variable
4. Method
5. collaboration (uses)
6. composition (has)
7. inheritance (is)
8. instantiation
9. polymorphism

AOP

1. generic role
2. domain specific role
3. Knowledge, belief
4. Capability
5. Negotiation
6. holonic agents
7. role multiplicity
8. domain specific role + individual knowledge
9. service matchmaking

OOP and AOP (Shoham,1993)

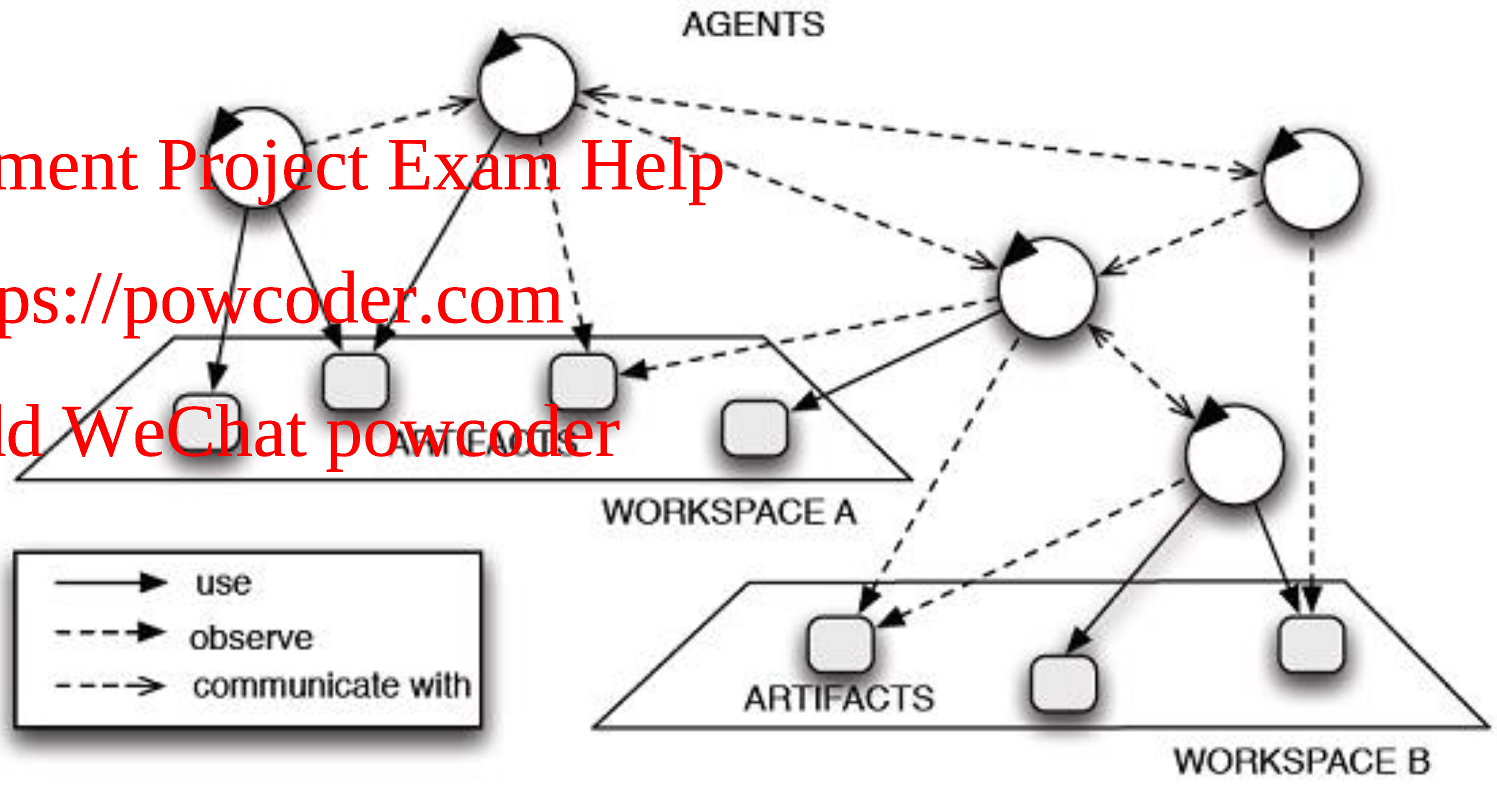
Framework	OOP	AOP
Basic unit	object	agent
Parameters defining state of basic unit	unconstrained	beliefs, commitments, capabilities, choices, ...
Process of computation	message passing and response methods	message passing and response methods
Types of message	unconstrained	inform, request, offer, promise, decline, ...
Constraints on methods	none	honesty, consistency, ...

Typical applications of agent programming

- Mobile computing
- Mobility
- Concurrent problem solving
- Proxy Handling
- Communication traffic routing
- Information scouts

Non-Exhaustive List of APLs

1993	Agent-0
1995	PLACA / AgentSp
1998	JACK / 3APL
2002	GOAL / AF-APL
2004	Jason
2008	2APL
2010	AF-AgentSpeak
2011	simpAL
2012	ASTRA
2014	Blueprint

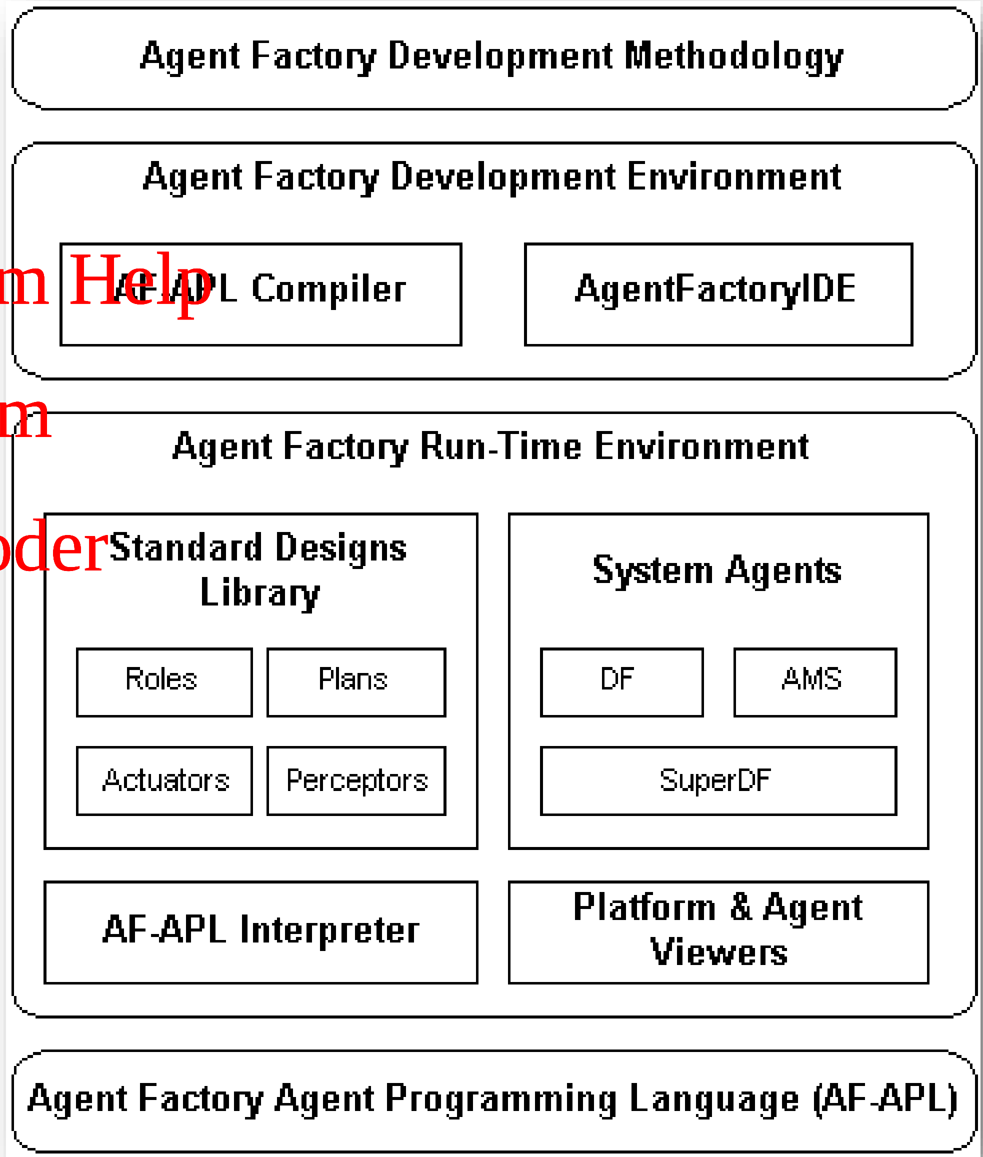


Agent Factory Layers

"A cohesive framework that supports a structured approach to the development and deployment of multi-agent systems"

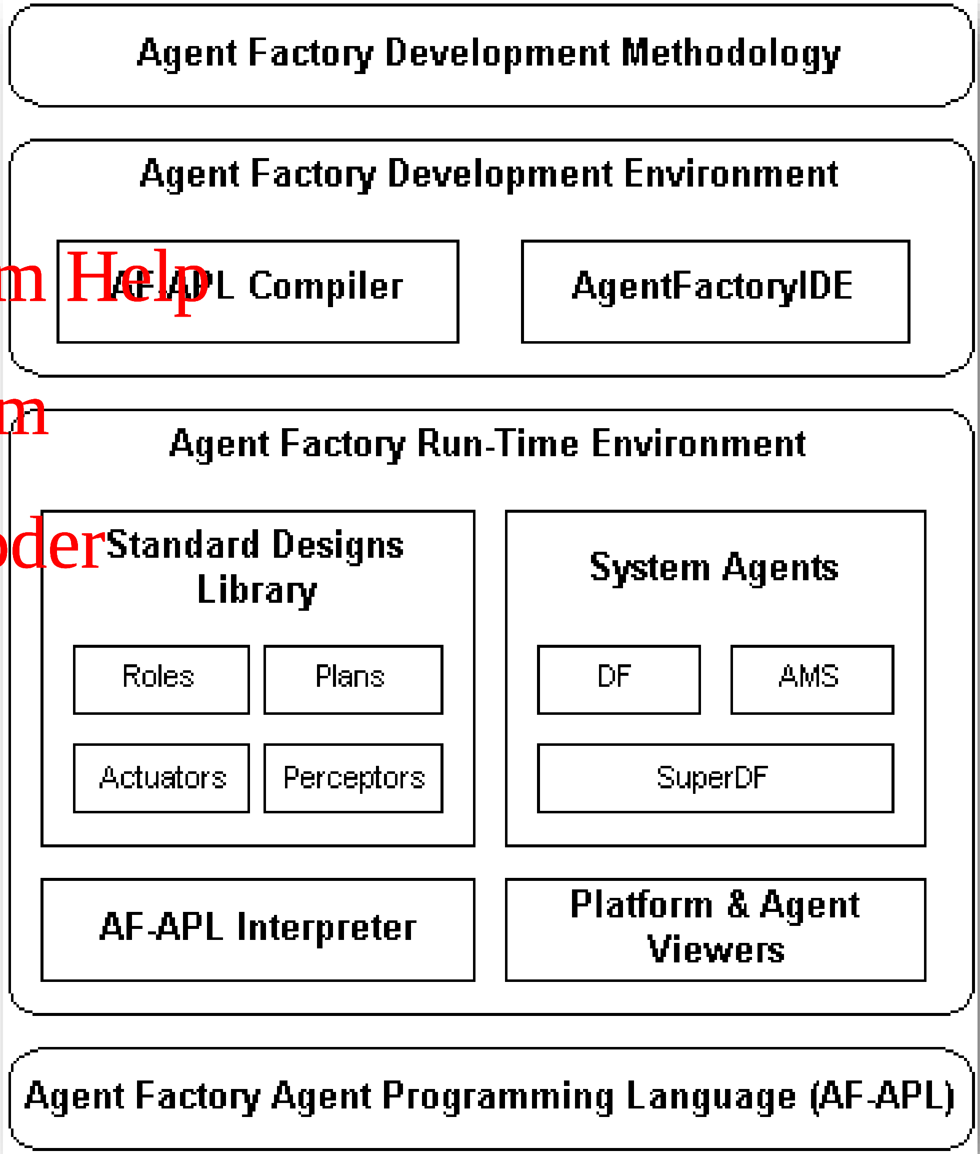
Agent Factory Layers

- **Organised over four layers:**
  1. Programming Language
  2. Run-Time Environment
  3. Development Environment
  4. Software Engineering Methodology



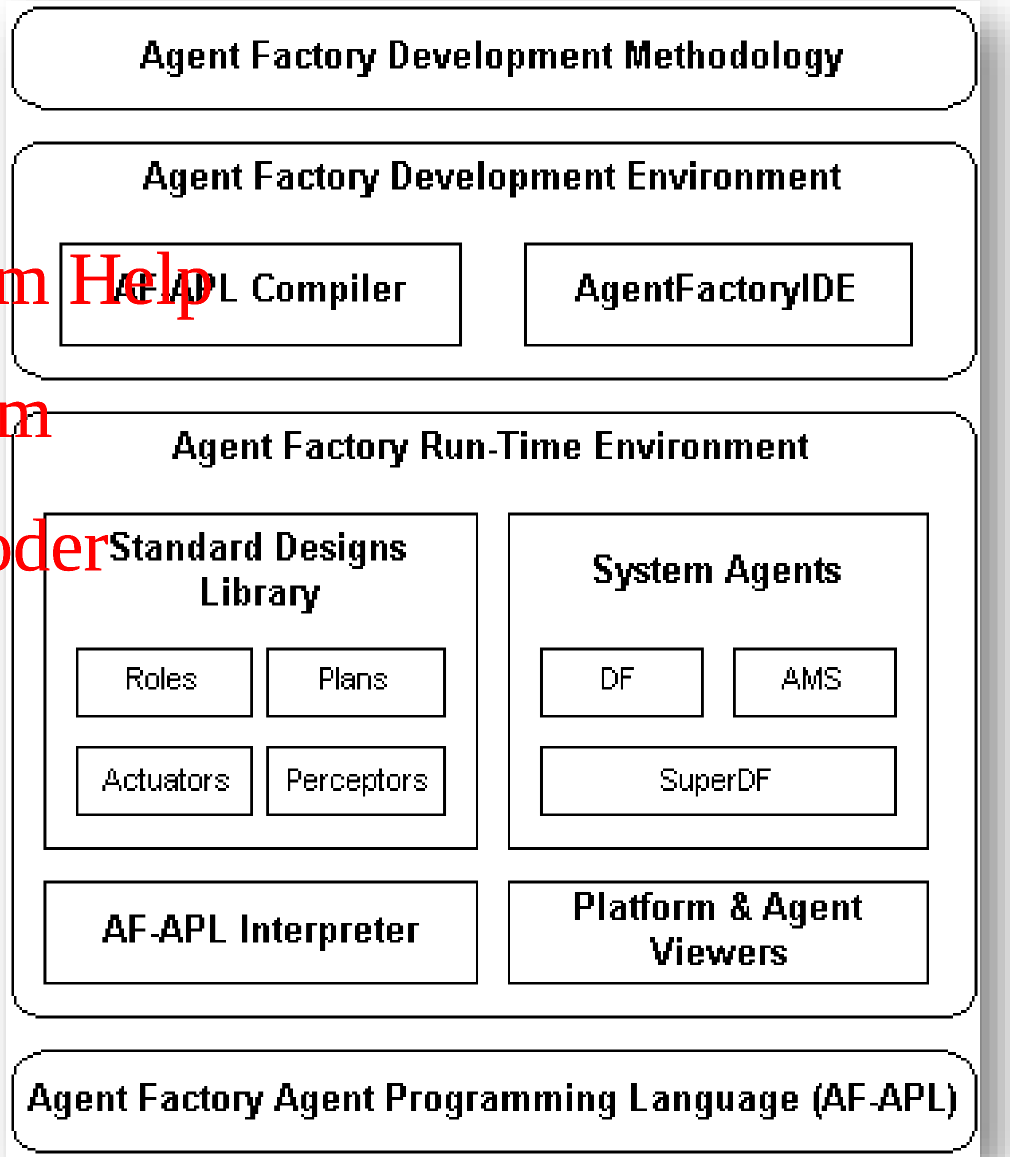
Agent Factory Layers

- **Organised over four layers:**
  1. **Programming Language**
     - Declarative
     - Formalised through a Multi-modal logic
     - Agent-specific Constructs
  2. Run-Time Environment
  3. Development Environment
  4. Software Engineering Methodology



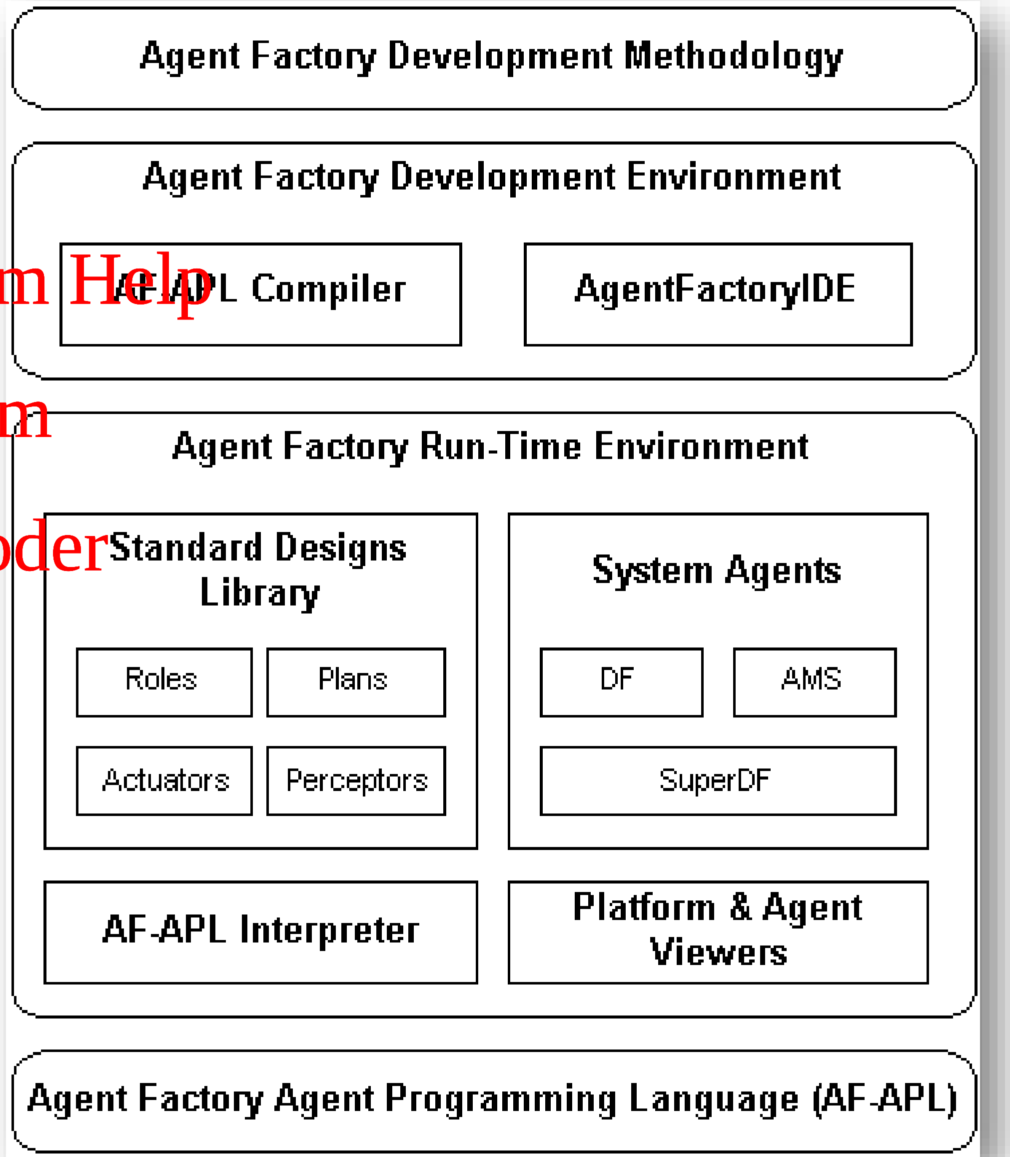
Agent Factory Layers

- **Organised over four layers:**
  1. Programming Language
  2. **Run-Time Environment**
     - Distributed
     - FIPA Compliant
     - Agent Platforms + Infrastructure
     - System Agents: AMS + DF
  3. Development Environment
  4. Software Engineering Methodology



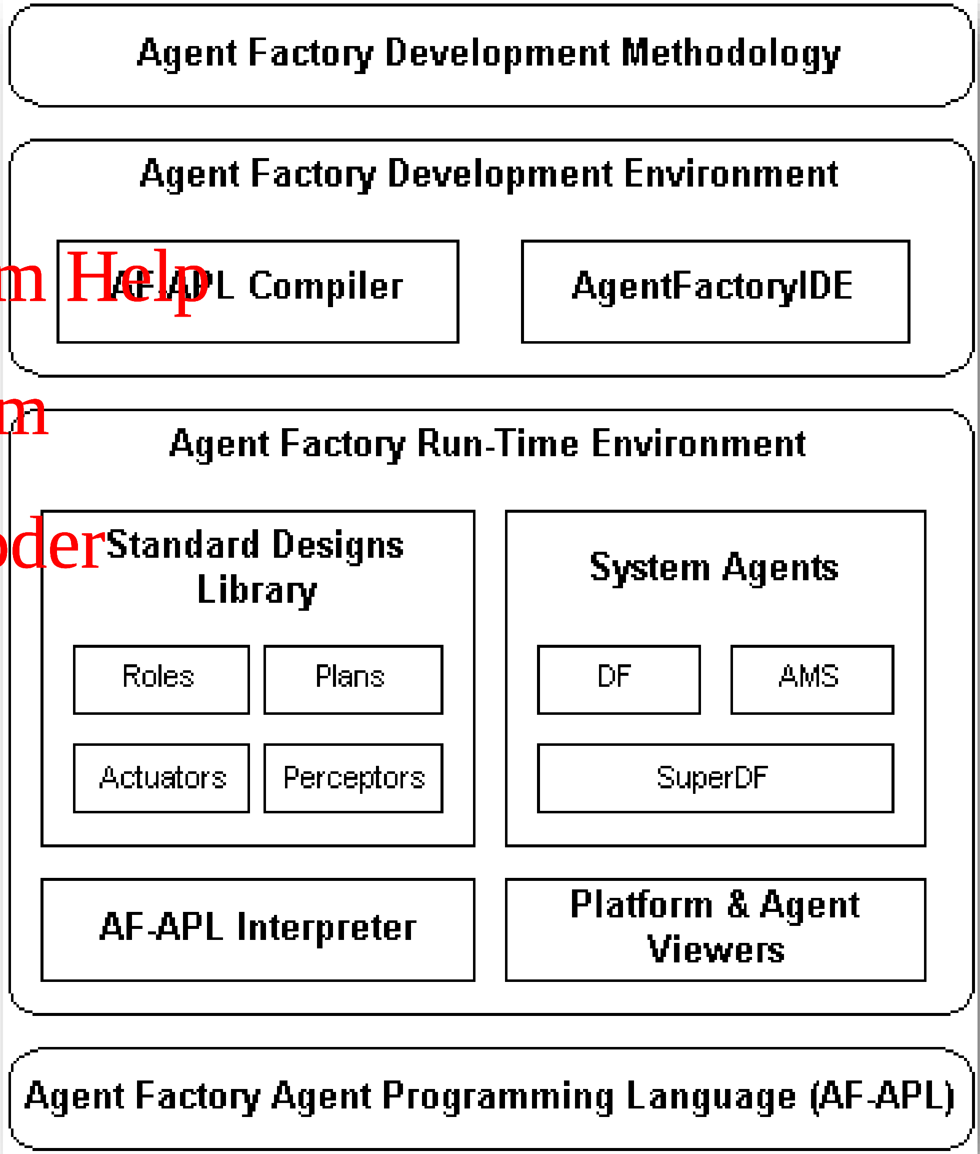
Agent Factory Layers

- **Organised over four layers:**
  1. Programming Language
  2. Run-Time Environment
  3. **Development Environment**
     - AF-APL Compiler
     - Netbeans & Eclipse Plugins
     - VIPER – Protocol Editor
  4. Software Engineering Methodology



Agent Factory Layers

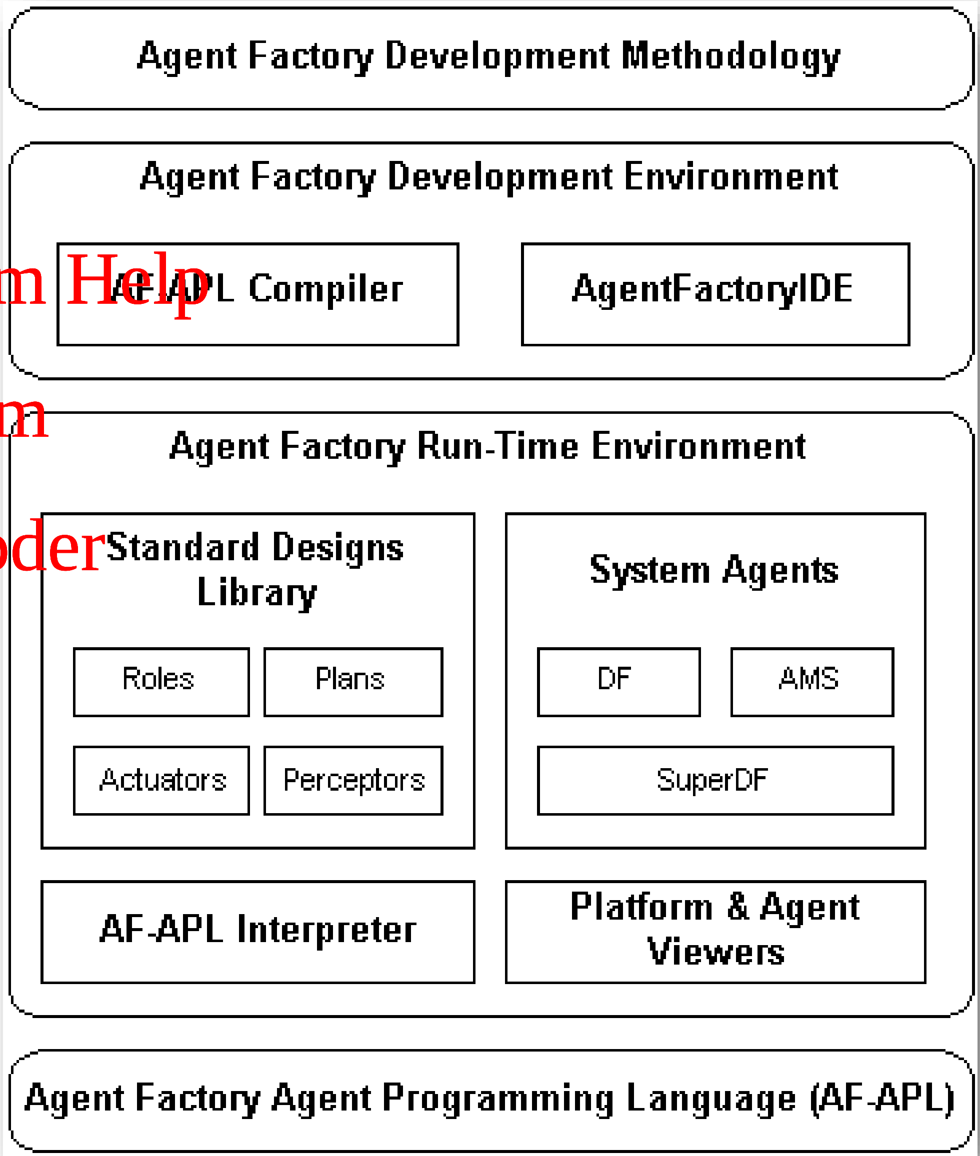
- **Organised over four layers:**
  1. Programming Language
  2. Run-Time Environment
  3. Development Environment
  4. **Software Engineering Methodology**



What is Agent Factory?

- **Organised over four layers:**
  1. Agent Programming Language
  2. Run-Time Environment
  3. Development Environment
  4. Software Engineering Methodology

- **Implemented in Java:**
- **Open Source**



AF-APL

- AF-APL Programs define:
  - Actuators
  - Perceptors
  - Modules
  - Commitment Rules
  - Initial Mental State

Source Editor

TestQueue.rle | QueueAgent.rle

```
/*
 * Queue Agent Role
 *
 * Author: Rem Collier
 * Date: 23-05-2003
 *
 * Abstract role that should be used by agents wishing to make use of internal queues.
 */

// Roles
USE_ROLE ie.ucd.core.fipa.role.FIPARole;

LOAD_MODULE queueManager ie.ucd.queue.module.QueueManagerModule;
PERCEPTOR ie.ucd.queue.perceptor.QueuePerceptor;
ACTUATOR ie.ucd.queue.actuator.CreateQueueActuator;
ACTUATOR ie.ucd.queue.actuator.AddToQueueActuator;
ACTUATOR ie.ucd.queue.actuator.RemoveFromQueueActuator;

// Rule 1: If the agent needs a queue, then create one
BELIEF(needQueue(?name)) & !BELIEF(queue(?name)) =>
COMMIT(Self, Now, BELIEF(true), createQueue(?name));

// Rule 2: If the agent needs a queue and doesn't have one, then
// believe that you are creating a queue
BELIEF(needQueue(?name)) & !BELIEF(queue(?name)) =>
COMMIT(Self, Now, BELIEF(true), adoptBelief(ALWAYS(BELIEF(creatingQueue(?name)))));

// Rule 3: If the agent believes that it is creating a queue, and has now
// created one, then stop believing that you are creating the queue.
BELIEF(creatingQueue(?name)) & BELIEF(queue(?name)) =>
COMMIT(Self, Now, BELIEF(true), retractBelief(ALWAYS(BELIEF(creatingQueue(?name)))));

// Test code
BELIEF(needQueue(testQueue));
```

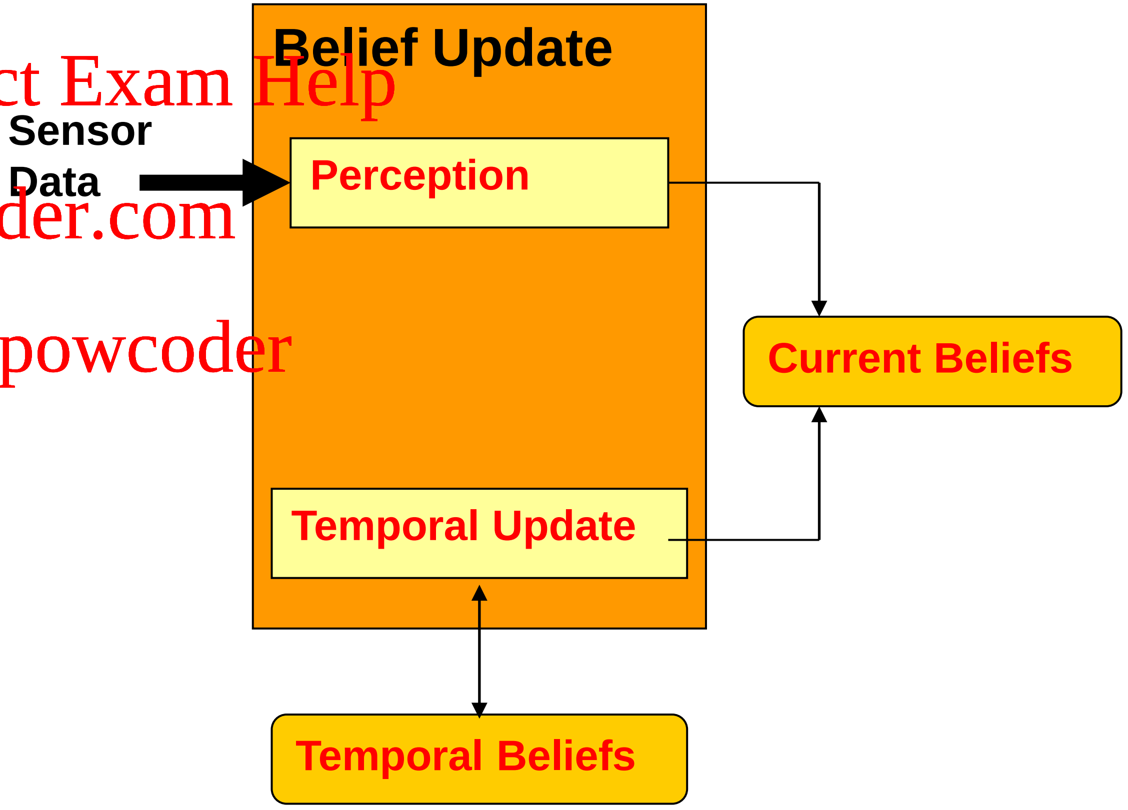
Executing AF-APL

- AF-APL is executed on a purpose-built agent interpreter.
  - **The agent class is loaded into the interpreter when the agent is created.**
  - **Control functions can be used to suspend, resume, and terminate the operation of the agent.**
- The interpreter processes the agent program by analysing the model of the environment (beliefs) and making decisions about how to act (commitments).
- Two problems arise from this:
  - **How to ensure that the model of the environment is up-to-date?**
  - **How to make the decision about how and when to act?**
- These problems are known as the belief management and commitment management problems, respectively.

Belief Management

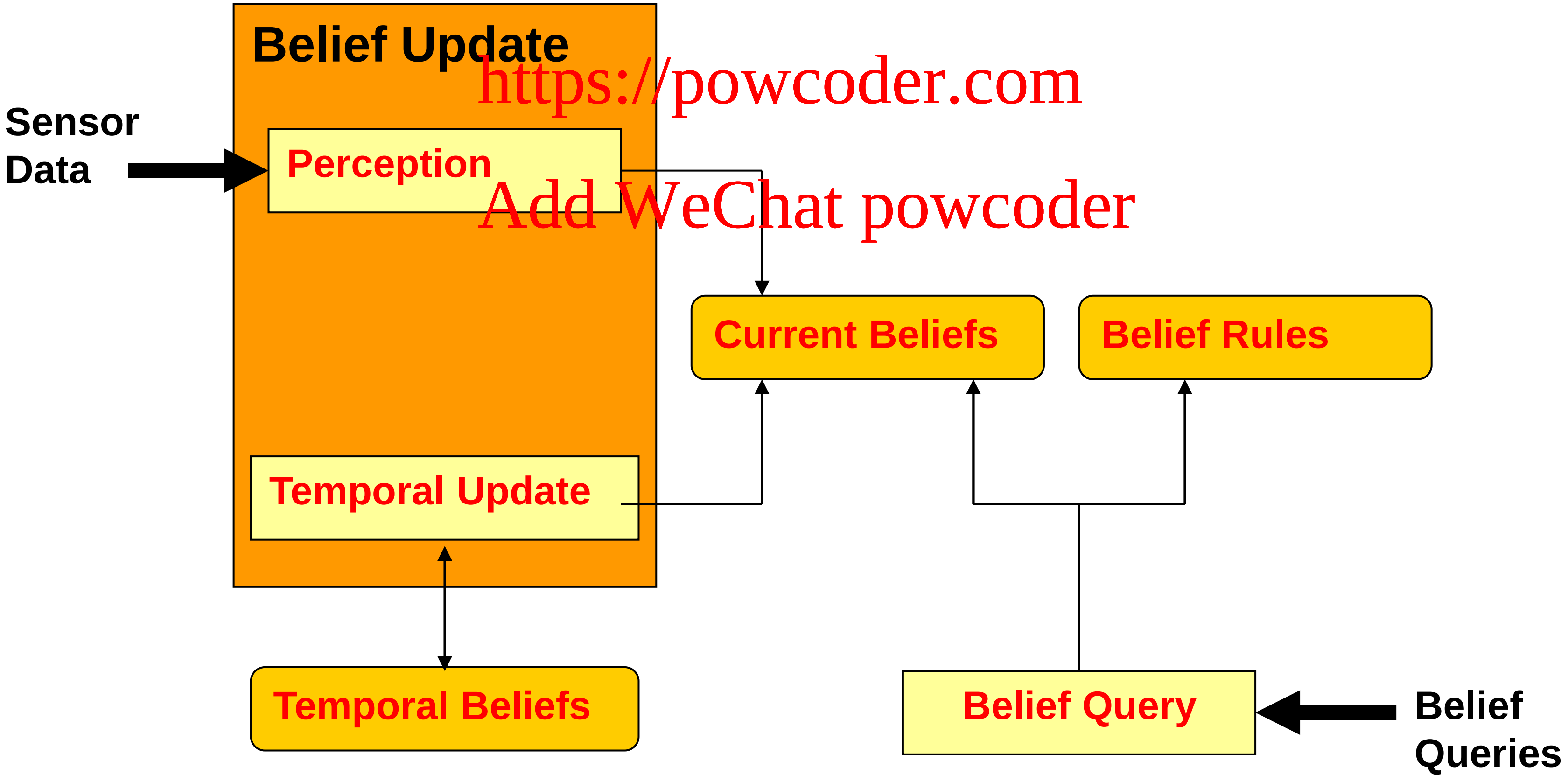
Belief Management = Belief Update + Belief Query

- Belief Update.
  - Dynamic Environment -> Transitory beliefs
  - Persistence can be supported through temporal operators
    (e.g. ALWAYS, NEXT)
  - Belief update = gathering perceptions + updating the temporal beliefs.



Belief Management

- Belief Query.
  - Beliefs = Facts + Implications (Belief Rules).
  - Resolution-based reasoning on current beliefs



Representing Beliefs in AF-APL

- AF-APL supports three forms of belief:
  - **Current Beliefs**. Beliefs that are true at the current time point.
  - **Temporal Beliefs**. Beliefs that persist over more than one time point.
  - **Belief Rules**. Rules that define inferences that can be made on the current beliefs.

- In AF-APL a belief is represented as a **first-order structure** enclosed within a BELIEF operator:
  - **BELIEF(happy(rem))** – a belief that rem is happy
  - **BELIEF(likes(?person, beer))** – a belief that some person likes beer
  - **BELIEF(bid(fred, 50))** – a belief that fred has bid 50
- These beliefs are current beliefs and apply only at the current time point. As a consequence, they are wiped at the start of each iteration of the AF-APL interpreter.

Temporal Beliefs

- **ALWAYS** – the belief is a current belief and will persist until the temporal belief is dropped.

  ALWAYS(BELIEF(happy(greg))) – always believe that greg is happy

- **UNTIL** – the belief is a current belief and will persist until either the temporal belief is dropped or the associated condition is satisfied.

  UNTIL(BELIEF(drinking(wine,greg)),!BELIEF(available(wine)))

  – believe that greg is drinking wine until do not believe that there is wine available.

- **NEXT** – the belief will be a current belief at the next time point.

  NEXT(BELIEF(finished(wine))) – at the next time point belief that the wine is finished.

- These beliefs are maintained until they are *explicity dropped*.

Belief Rules

- Belief Rules define inferences that can be made over the current beliefs of the agent.

- They take the form of logical implications:

BELIEF(likes(?food)) & BELIEF(has(?food)) => BELIEF(want(?food))

BELIEF(has(rem, icecream)) => BELIEF(happy(rem))