AGENT-ORIENTED PROGRAMMING: FROM THEORY TO PRACTICE

Costin Bădică

University of Craiova
Department of Computers and Information Technology

UNIVERSITY OF CRAIOVA

IDS Research Group

- *Intelligent Distributed Systems Research Group*
- http://ids.software.ucv.ro/
- Synergies between:
  - Intelligent computing
  - Distributed computing

Overview

I. Basics of AOP

BDI

Logic-based

II. Examples

Modeling and Enactment of Business Agents

Programming Reinforcement Learning

Freight Transportation Exchanges

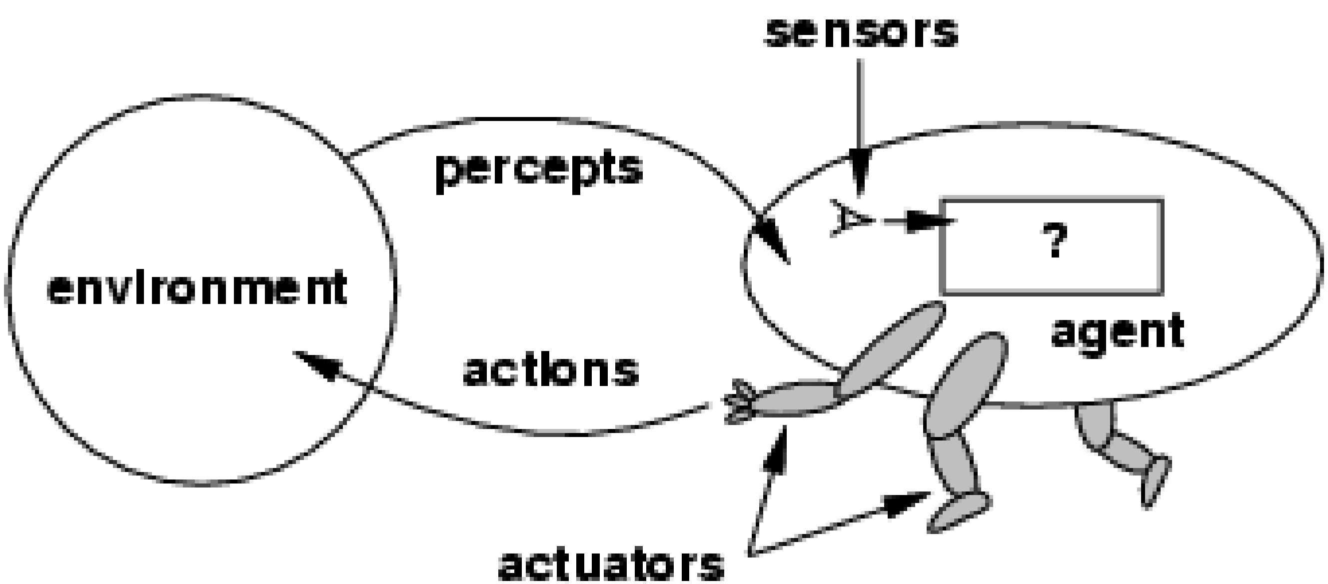
ORIGIN OF WORD "AGENT"

- The word "*Agent*" comes from the Latin word "*Agere*".
- It literally means "*to do*" with the sense of "*to act*" or "*to take action*".

AGENTS AND ENVIRONMENTS

- (Weakest) Agent = anything that can be viewed as:
  - *perceiving* its environment through *sensors* and
  - *acting* upon that environment through *actuators*.

  [*Russel & Norvig, 2010*]



ENVIRONMENTS

- (Fully / Partially / Not) Observable:
  - Noisy & inaccurate sensors
  - Immeasurable / inaccessible parts of the environment
- (Single / Multi) Agent
  - *Competitive*: multiple agents with disjoint goals
  - *Cooperative:* multiple agents sharing (parts of) goals
- Deterministic / Stochastic
  - *Deterministic:* environment state completely determined by agent action
  - *Nondeterministic:* probabilities of next states are missing
  - *Uncertain:* partially observable and stochastic

MAS TECHNOLOGIES

- Methodologies
- Standards
- Frameworks and platforms
- *Programming languages*
- Other technologies

AGENT ORIENTED PROGRAMMING

- AOP was firstly proposed 25 years ago as:

  A new programming paradigm, one based on cognitive and societal view of computation [Shoham, 1993]

- Many models and implementations of AOP ! One representative class is based on:

  Belief-Desire-Intention architecture – BDI.

AgentSpeak(L) and Jason

- AgentSpeak(L) = *abstract AOP language* (Rao, 1996)
- Jason = implementation and extension of AgentSpeak(L), based on Java.
  - *Agent program* is written in Jason.
  - *Environment* is written in Java.
  - *Agent architecture* can be customized in Java.
- AgentSpeak(L) combines *BDI* and *Logic*

Source of Inspiration

- Philosophy: Daniel Dennett's "*intentional stance*":
  - Distinguish between mental and physical phenomena by means of "*intentionality*".
  - Explain behavior of an (artificial) entity in terms of its "*mental properties*".
- Dennett introduced BDI concepts:
  - belief, desire, goal, practical reasoning, …

BDI CONCEPTS

- BDI follows practical reasoning model = *reasoning towards actions*.
- BDI agents are endowed with:
  - *Beliefs*
  - *Desires* or *goals*
  - *Intentions* or *plans*
- Agent behavior is *event-driven*.

Agent Program

- Initial belief base: *set of facts and rules*:

$$predicate(term_1, \dots, term_n)$$

- Initial goal(s): *achievement goals*

$$!predicate(term_1, \dots, term_n)$$

- Plan base: *set of plans*:

$$event\ :\ context <- plan\ body$$

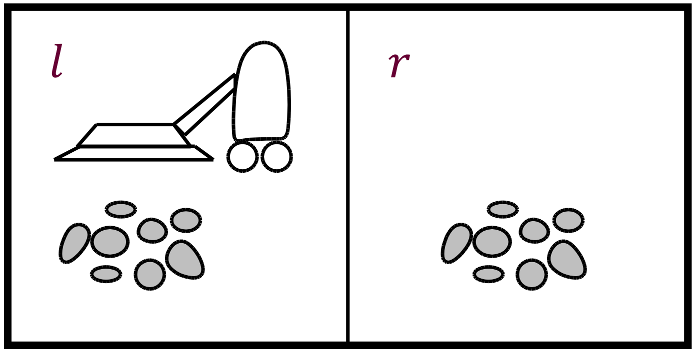
- Event: $+!goal \quad -!goal \quad +belief \quad -belief$
- Context: Conjunctive (&) / disjunctive (|) *condition*
- Plan body: *sequence of actions* separated by ;

BDI Reasoning Steps

- Update
  - perceive
  - communicate
- Select
  - event
  - applicable plan
  - intention = course of action
- Act
  - internal / external
  - mental note
  - adopt / drop goal:
    - test / achievement

Deliberation cycle

VACUUM CLEANER WORLD



- Environment:
  - 2 locations l and r
  - Status of the current location: $clean$ or $dirty$.
- Actions:
  - $left$, $right$, $suck$, no_op, for “move left”, “move right”, “suck dirt” and “do nothing”.
- Percepts:
  - Pair $[pos(Location), Status]$, for example $[pos(l), dirty]$

Reactive / Proactive

```
+pos(l) : clean <-
   .print("l clean");
   right.
+pos(l) : dirty <-
   .print("l dirty");
   suck;
   right.
+pos(r) : clean <-
   .print("r clean");
   left.
+pos(r) : dirty <-
   .print("r dirty");
   suck;
   left.
```

```
!keep_clean.
+!keep_clean : dirty &
   pos(L) <-
   .print("suck in ",L);
   suck;
   !move.
+!keep_clean : clean &
   pos(L) <-
   .print("no suck in ",L);
   !move.
+!move : pos(l) <-
   right;
   !keep_clean.
+!move : pos(r) <-
   left;
   !keep_clean.
```

Logic Programming

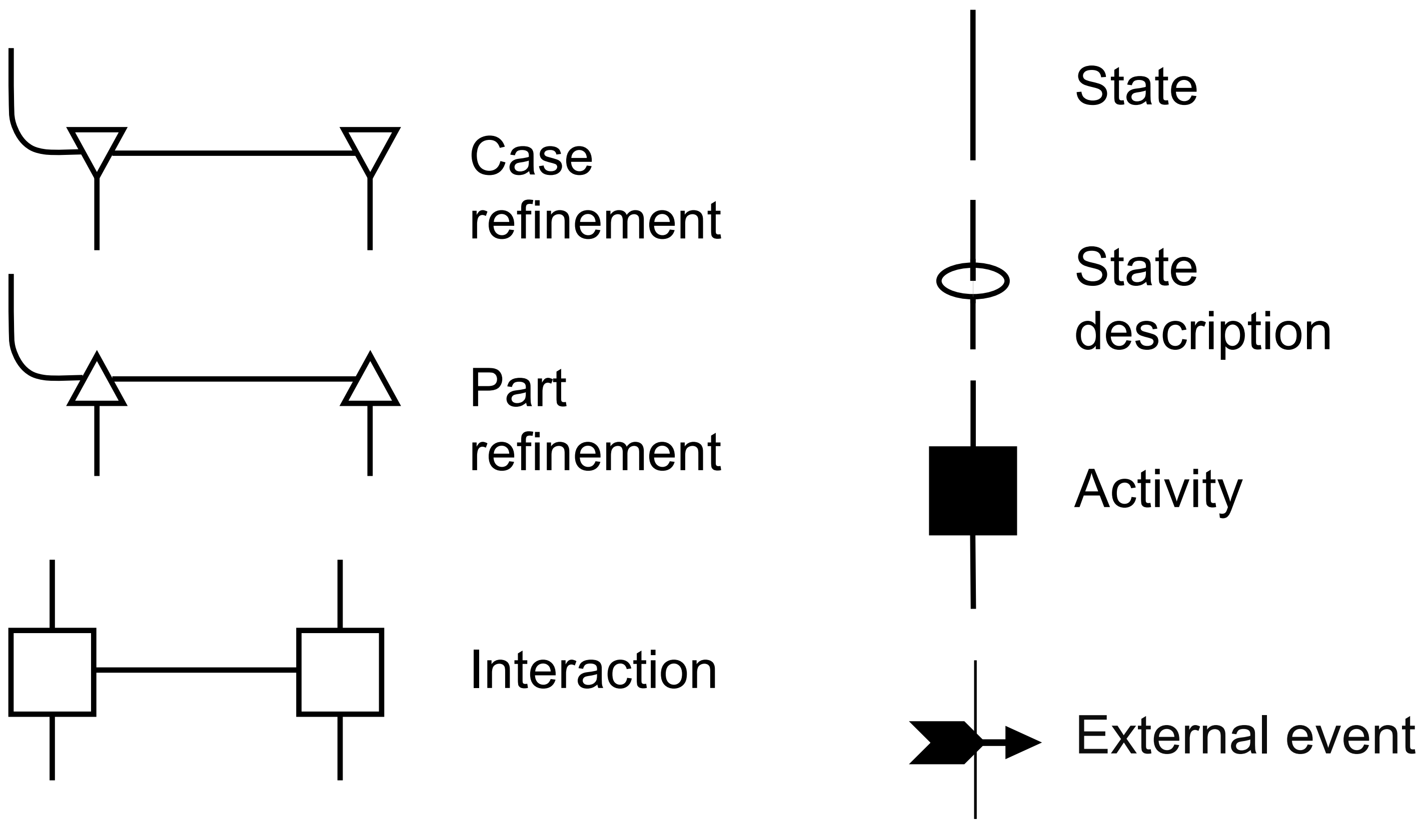
Logic Program = Facts + Rules + Queries

Facts + Rules → Belief Base

Queries → Test goals

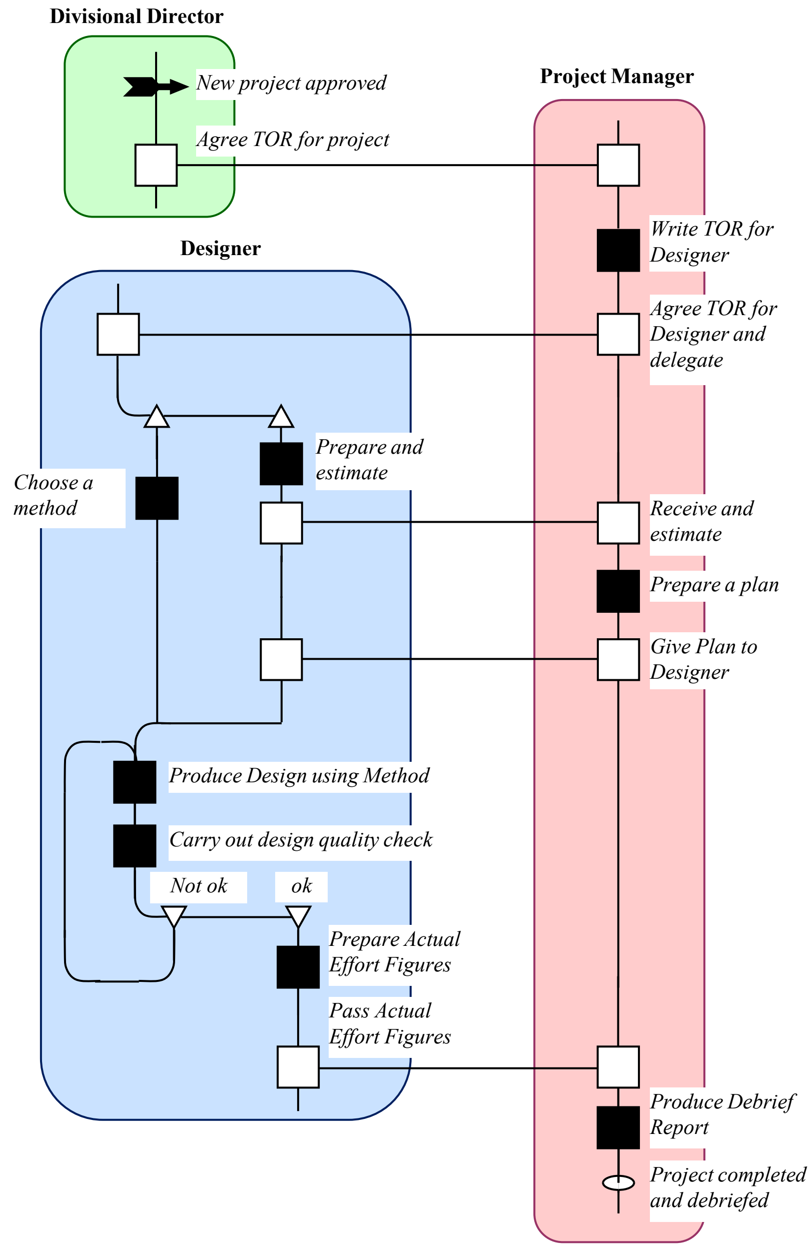
Business Agents

- **AIM:** *apply state-of-the-art AOP languages for modeling and enactment of business processes.*



ROLE ACTIVITY DIAGRAM

- Three roles:
  - Divisional Director
  - Project Manager
  - Designer



Mapping Outline

- RAD role => Jason agent. Eg.: *d*, *dd*, *pm* agents.
- State => belief base. E.g.: *dd0*, *dd1*, *dd2*, *d0*, *d1*, …
- Action => agent plan:

+!advance **:** *current state* **<-**
remove tokens
do activity
append tokens
!advance.

state update

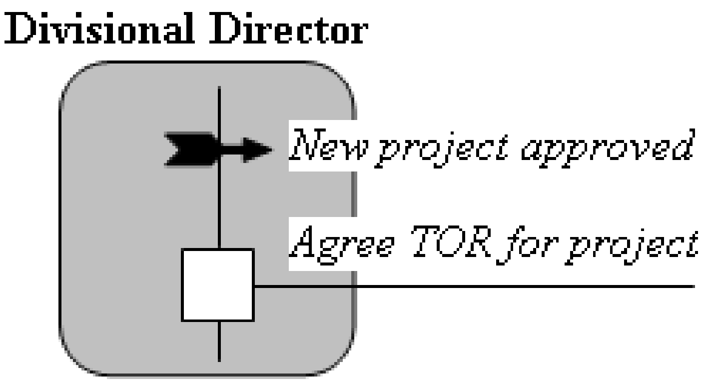
- RAD process => multi-agent program

MAPPING STATES AND ACTIVITIES

```
+!advance : dd0 <-
    -dd0;
    ?task("New project approved");
    +dd1;
    !advance.
```

```
+!advance : start <-
    -start;
    ?task("Starting ...");
    +dd0;
    !advance.
```

```
+!advance : dd2 <-
    -dd2;
    ?task("Stop").
```

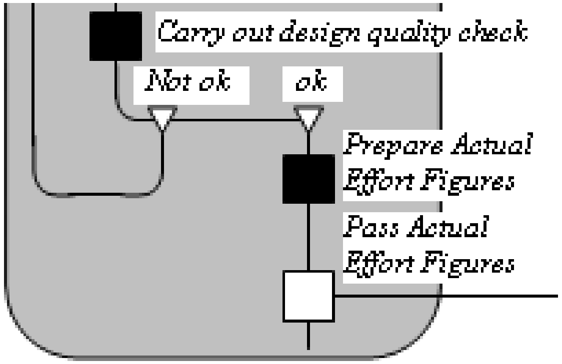


Start state

End state

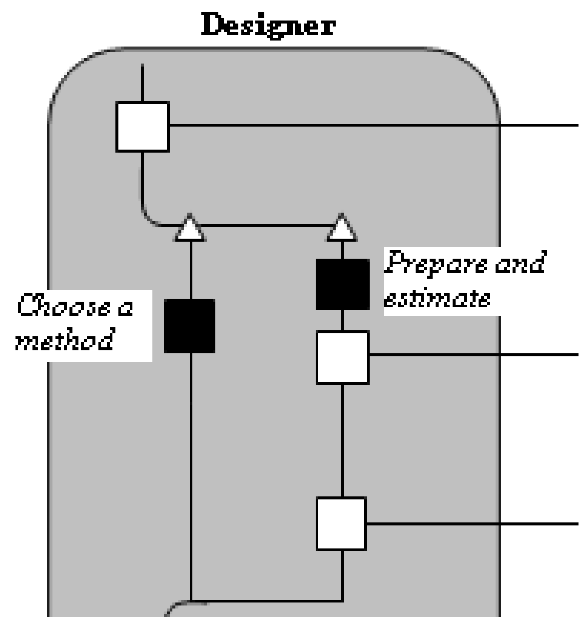
MAPPING CASE REFINEMENTS

```
+!advance : d9 <-
   -d9;
   ?task("Carry out design quality check");
   rad.choice([ok,nok],Result)
   +d10(Result);
   !advance.
+!advance : d10(nok) <-
   -d10(nok);
   ?task("Design quality not ok");
   +d8;
   !advance.
```



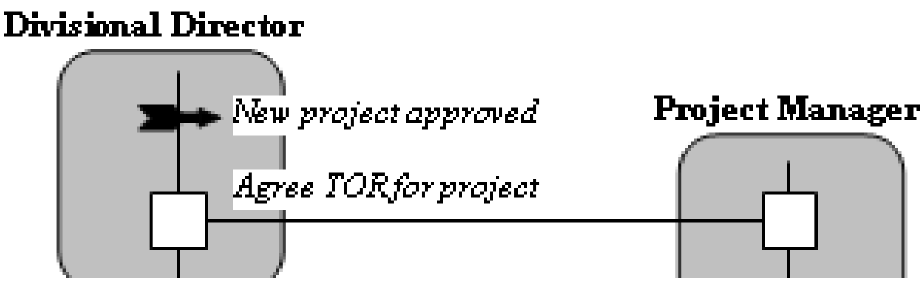
Mapping Part Refinements

```
+!advance : d1 <-
    -d1;
    ?task("Fork parallel threads");
    +d2;
    +d3;
    !advance.

+!advance : d6 & d7 <-
    -d6;
    -d7;
    ?task("Join parallel threads");
    +d8;
    !advance.
```



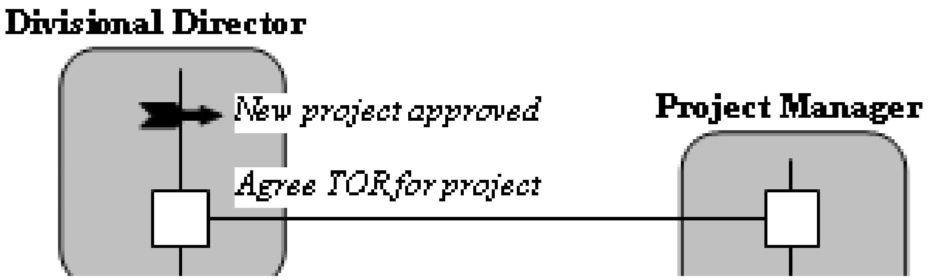
MAPPING INTERACTIONS

```
+!advance : start <-
   -start;
   ?task("Starting ...");
   +pm0;
   .send(dd,tell,pm0);
   !advance.
+!advance : dd1 & pm0 <-
   -dd1;
   -pm0[source(pm)];
   ?task("Agree TOR for project");
   +dd2;
   !advance.
```



Contingency Plan

```
+!advance : dd1 & pm0 <-
    -dd1;
    -pm0[source(pm)];
    ?task("Agree TOR for project");
    +dd2;
    !advance.

-!advance : true.

+pm0 : true <- !advance.
```



Mapping Summary

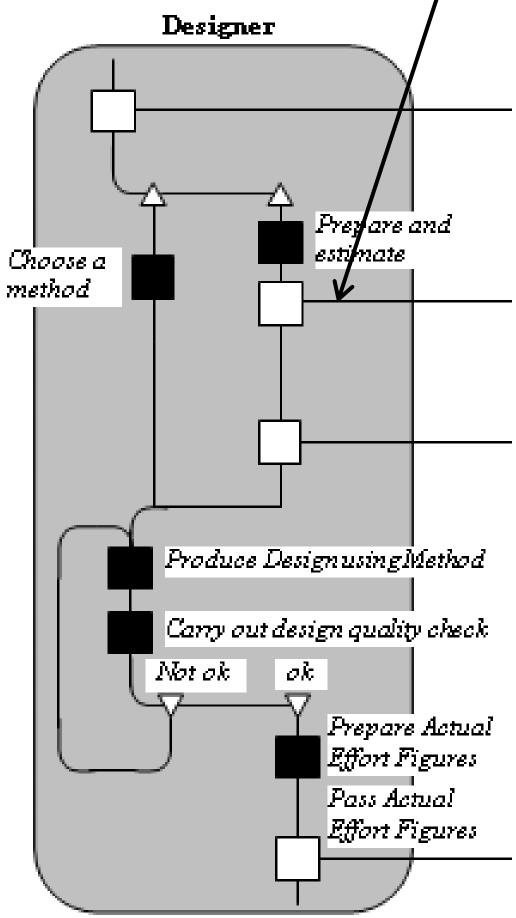
- *One proactive plan* for agent starting and one proactive plan for agent stopping.
- *A proactive plan* for each action node.
- *One contingency plan* to deal with shared beliefs that have not yet arrived from peer agents.
- *A reactive plan* for handling the arrival of each shared belief from peer agents.

Knowledge-Based Business Agents

- Generic *knowledge-based business agent architecture* – $\mathcal{KB}^2\mathcal{A}^2$.
  - A *knowledge base* that captures the operational knowledge of the agent according to a given business process.
  - A *set of template plans* that capture the generic behavioral patterns of business agents.

Knowledge Base

“Receive and estimate”

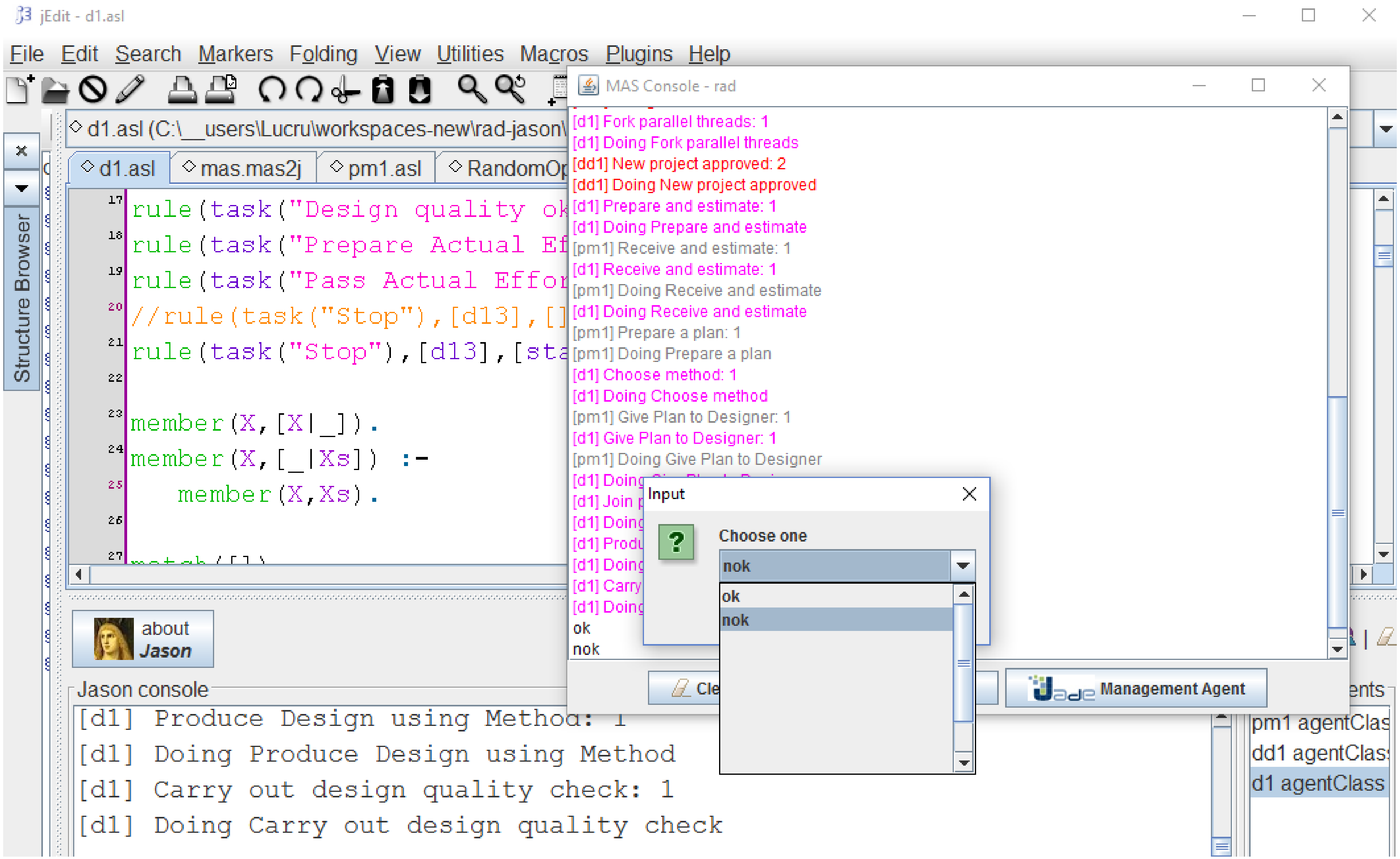


Send shared belief

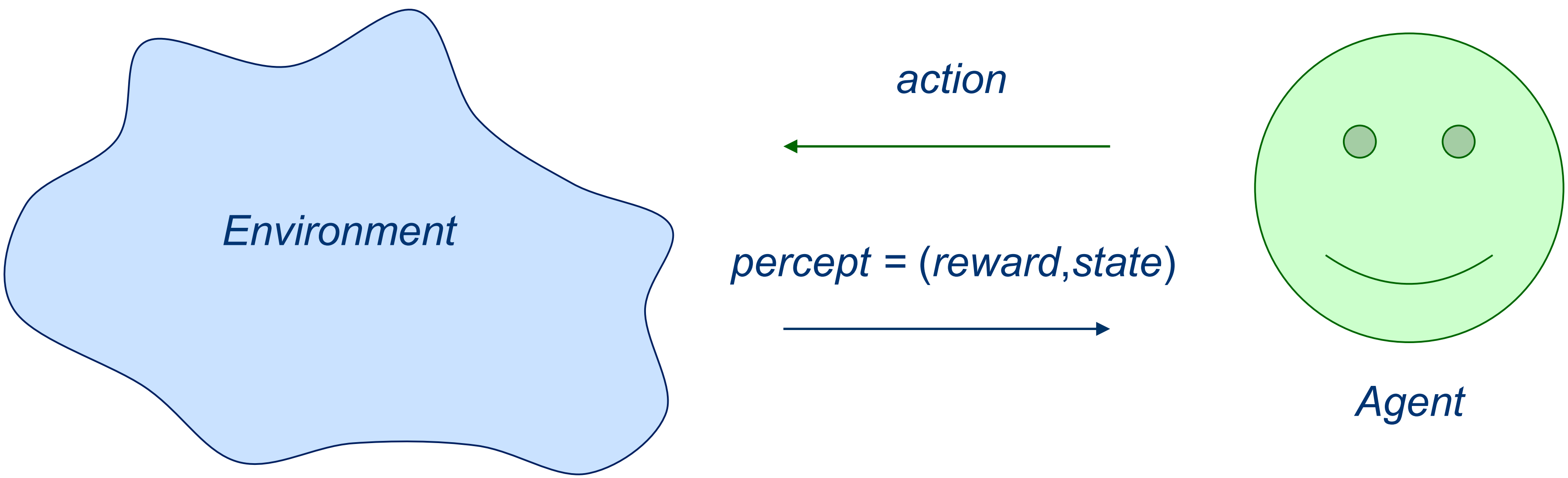
```
% rule(Action, StateIn, StateOut).

rule(task("Fork parallel threads"), [d1],
   [d2,d3]).
rule(task("Prepare and estimate"), [d3],
   [d4,s(pm,d4)]).
rule(task("Receive and estimate"),
   [d4,r(pm,pm3)],[d5,s(pm,d5)]).

...

rule(choice("Carry out design quality check",
   [ok,nok]), [d9], [[ok,d10(ok)],
   [nok, d10(nok)]]).
```

Receive shared belief

$\mathcal{KB}^2\mathcal{A}^2$ IN ACTION



PROGRAMMING REINFORCEMENT LEARNING



- Passive RL: agents act according to a fixed policy and their learning goal is to compute the utility function.
- Active RL: agents learn an optimal policy that maximizes their utility, while they are acting in their environment.

Reinforcement Learning

- Markovian stochastic environment *E*.
- For each state e agent receives *reward* $R(e)$
- Agent behavior defined by *policy* $\pi: E \to A$.
  $a = \pi(e)$ is agent *action* in environment state e.
- Agent starting in state e generates a *history*:

$$H(e) = [e_0 = e, e_1, \dots,]$$

- Each history awards agent with *utility*:

$$U_h(H(e)) = \sum_{i \geq 0} \gamma^i R(e_i)$$

- *Utility of policy* π is:

$$U^\pi(e) = \mathbb{E}\left[U_h\left(H(e)\right)\right]$$

Passive Reinforcement Learning

- **Goal**: For given policy π the agent learns U^{π}.

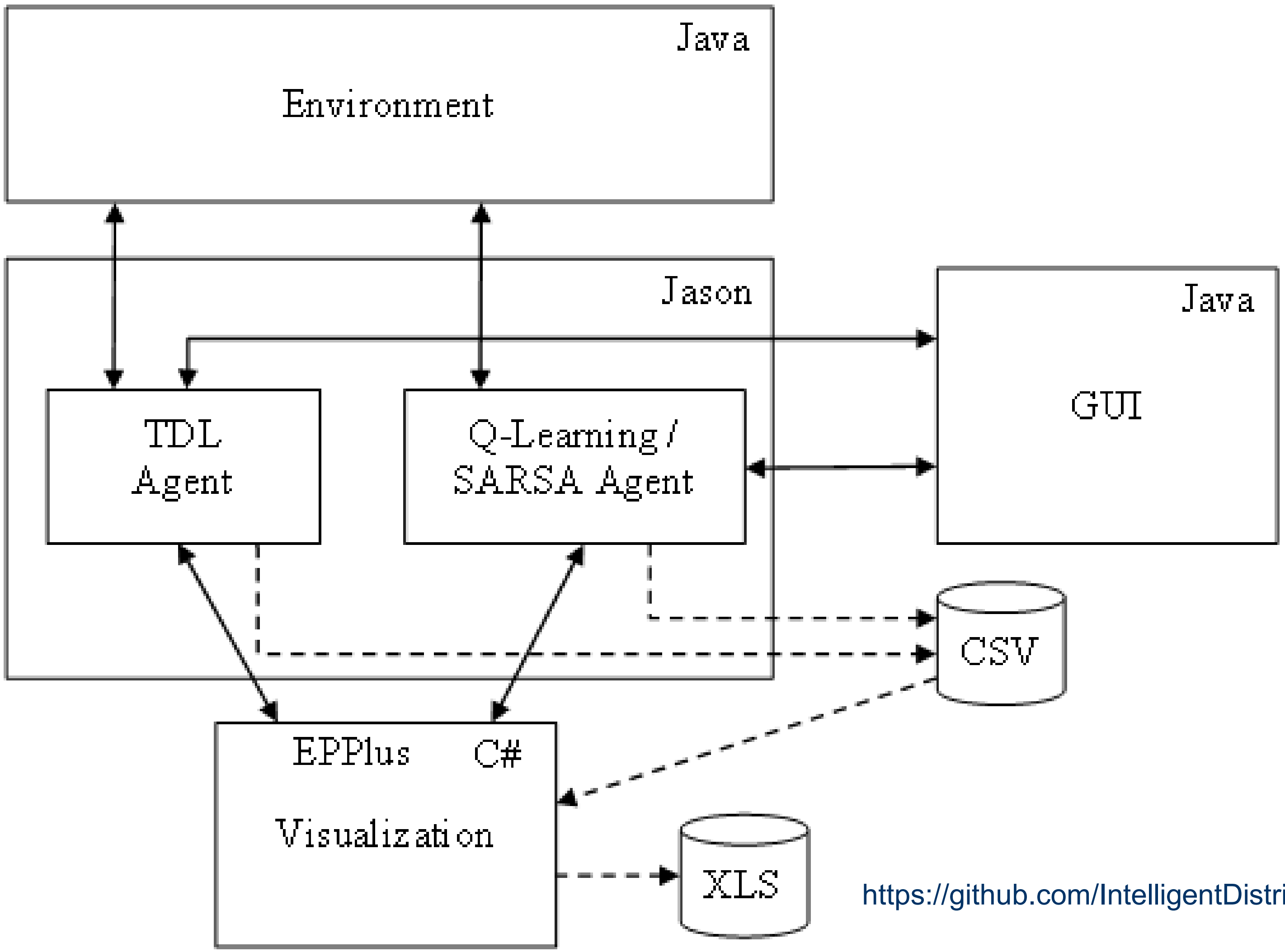
- **Temporal Difference Learning**:
  for each observed state transition $e \rightarrow e'$
  the agent computes an updated $U^{\pi}(e)$ as follows:

$$U^{\pi}(e) + \alpha\big(R(e) + \gamma U^{\pi}(e') - U^{\pi}(e)\big)$$

Active Reinforcement Learning

- **Goal:** Compute the optimal policy π^* that maximizes U^π. Let $U(e) = U^{\pi^*}(e)$.
- $Q(e, a) =$ utility of taking action a in state e so $U(e) = \max_{a \in Ac} Q(e, a)$
- **Q-learning:** update $Q(e, a)$ for each transition $e \to e'$
  $$Q(e,a) + \alpha(R(e) + \gamma \max_{\mathrm{a}' \in Ac} Q(e', a') - Q(e,a))$$
- **SARSA:** a' is the actual action taken in state e'
  $$Q(e,a) + \alpha(R(e) + \gamma Q(e', a') - Q(e,a))$$

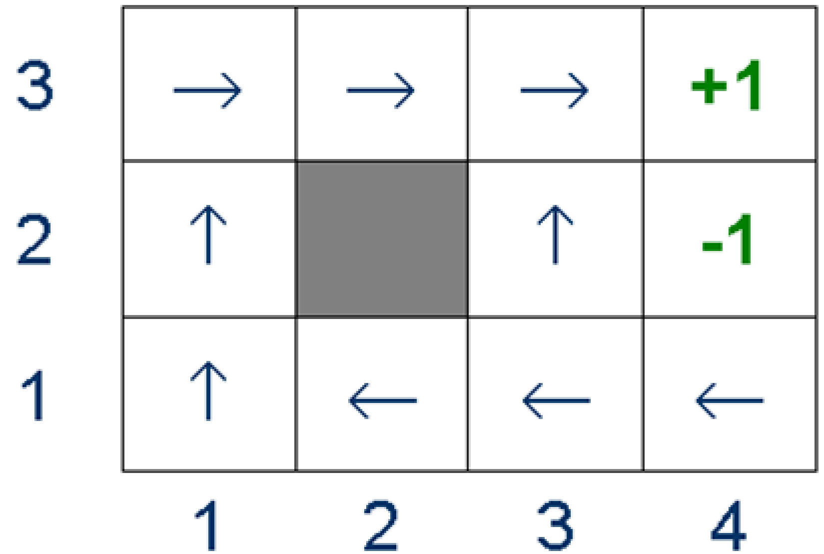
SYSTEM ARCHITECTURE

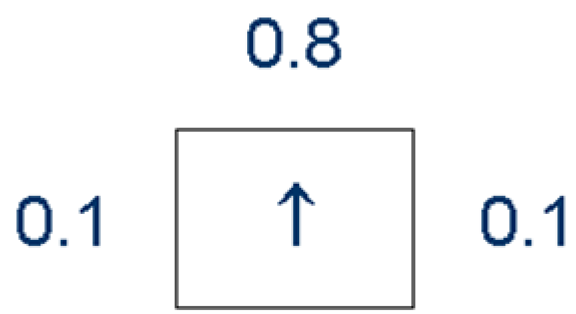


https://github.com/IntelligentDistributedSystems/SamuelFelton

Environment

- Class *MDPModel* to store & update environment state
- Class *MDPEnv* to interface with Jason interpreter
- Class *MDPView* – GUI & visualization





ACTIONS & PERCEPTS

- Agent actions:
  - up, $right$, $down$, $left$ for the agent movement
  - $null$, for restarting a new trial in a random initial position.

- Agent percepts $pos(Row, Column, R, T)$ such that:
  - Row and $Column$ give the agent position on the grid
  - R is the reward.
  - T is n for non-terminal state and t for terminal state

ENVIRONMENT GUI

STATE SUMMARY

Stats: pos(1,3,n)

Action	Q-Value	Number of trials
DOWN	-0.2534336287468129	50
RIGHT	-0.27342072330776646	21
UP	-0.256	6
LEFT	0.7198818753530539	315
NULL	0.0	0

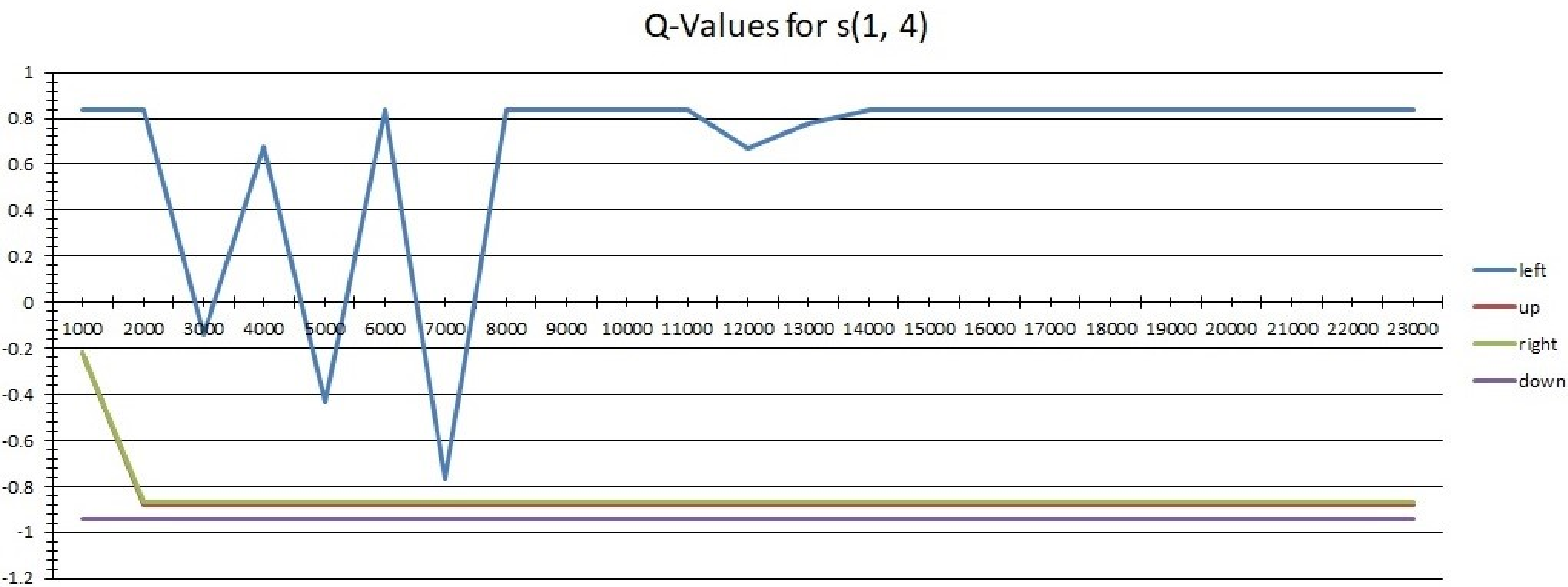
AGENT CODE

- Belief Base:
  - Counters of states and trials
  - Maximum number of trials
  - Last and next action
  - Discount factor
  - Minimum number of state visits to encourage exploration
  - Utility and number of visits for each state
- Goals
  - Achievement goal: learning to act, i.e. determine Q-values
  - Acting = sequence of trials
  - Trial = sequence of moves
  - Update Q-values following each move

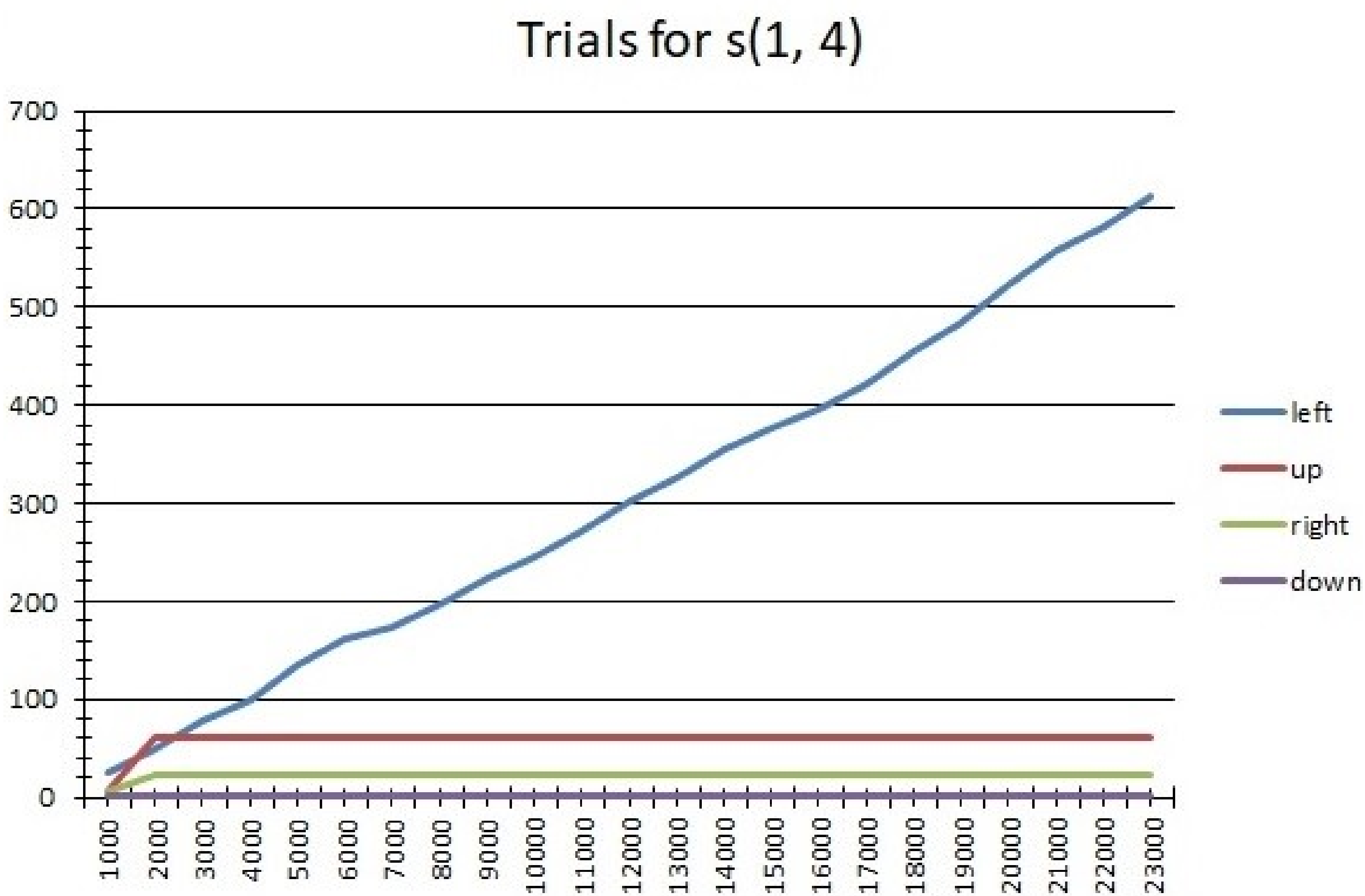
Q-Learning Agent

```
+!update_qvalue(St1,_,St,R,Q_St,A,M1) :
    non_terminal_state(St) <-
    .findall(Q,qvalue(St1,A1,Q,_),Qs);
    ?maxim_list(Qs,Q_St1);
    ?gamma(Discount);
    ?alpha(Alpha);
    Q1 = Q_St + Alpha*(R+Discount*Q_St1-Q_St);
    -qvalue(St,A,_,_);
    +qvalue(St,A,Q1,M1).
+!update_qvalue(_,R1,St,_,_,null,M1) :
    terminal_state(St) <-
    -qvalue(St,null,_,_);
    +qvalue(St,null,R1,M1).
```

Examining Q-values

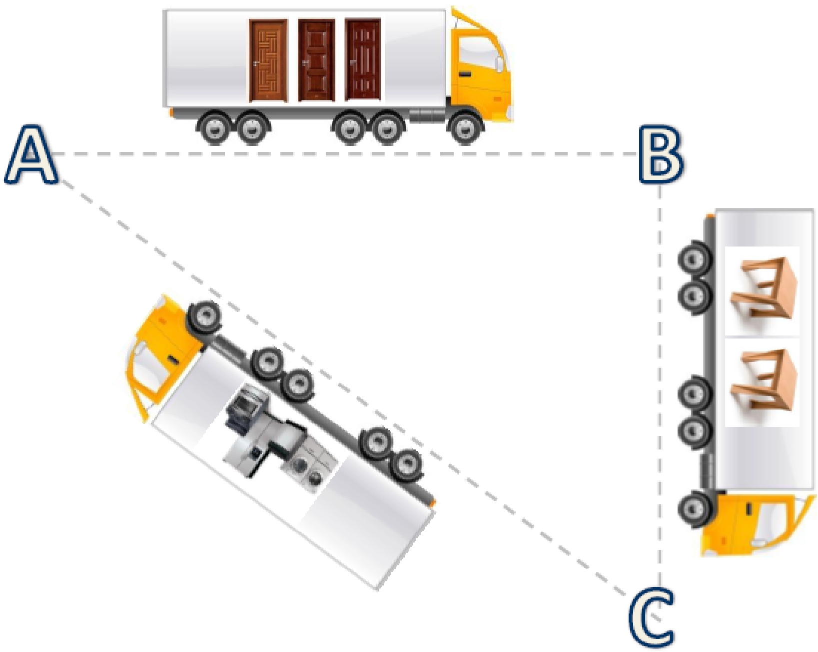


NUMBER OF TRIALS



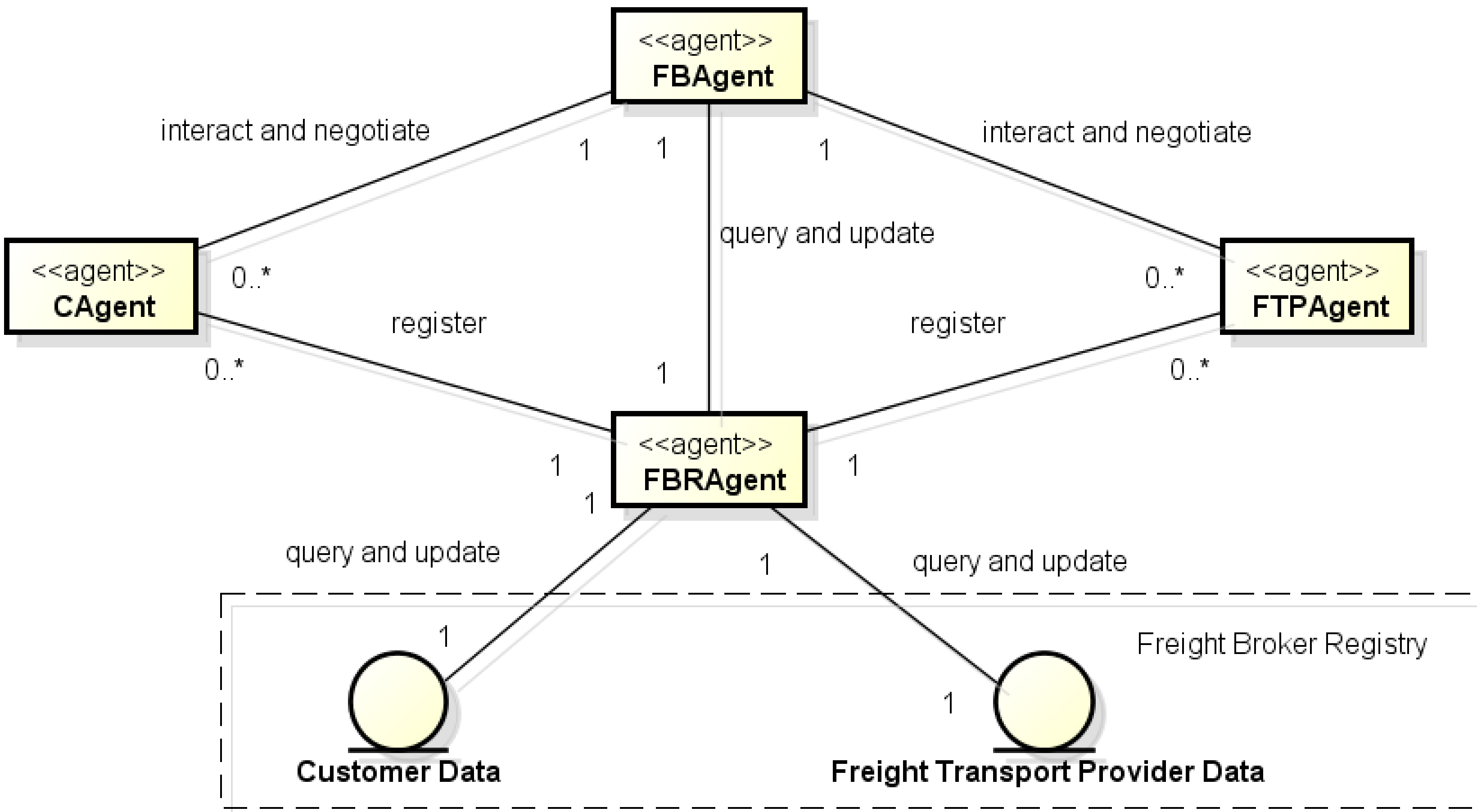
Freight Transportation Exchanges

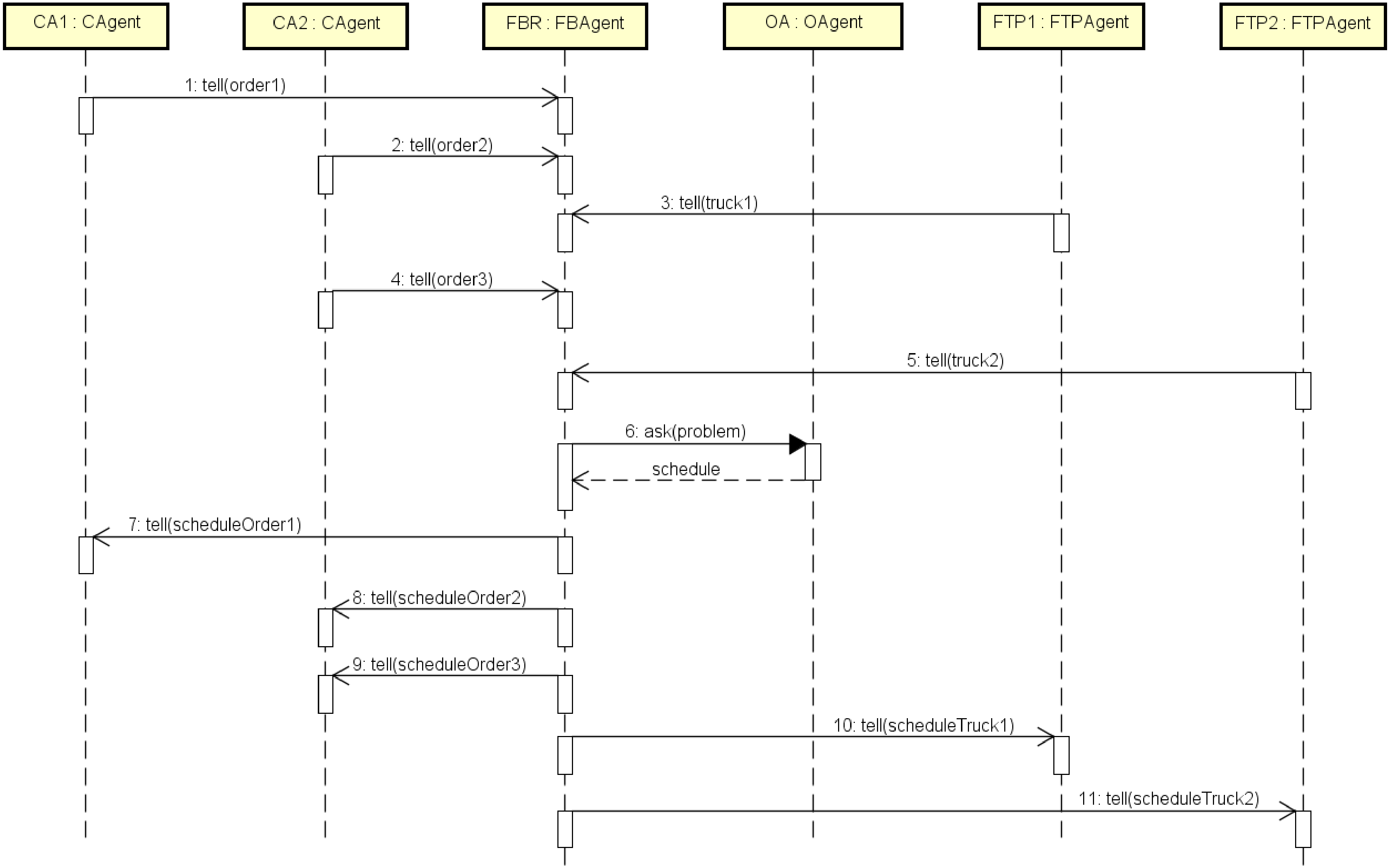
- Opportunity:
  - need of transporting goods
  - availability of free vehicles
- Goal:
  - capturing transportation opportunities
  - matchmaking of owners of goods with freight transportation providers



- New business model: *virtual logistics platforms*

MAS FOR FREIGHT BROKERING




CA1 : CAgent
CA2 : CAgent
FBR : FBAgent
OA : OAgent
FTP1 : FTPAgent
FTP2 : FTPAgent
1: tell(order1)
2: tell(order2)
3: tell(truck1)
4: tell(order3)
5: tell(truck2)
6: ask(problem)
schedule
7: tell(scheduleOrder1)
8: tell(scheduleOrder2)
9: tell(scheduleOrder3)
10: tell(scheduleTruck1)
11: tell(scheduleTruck2)

Freight Broker Agent

- Declarative optimization model of the *FBAgent*:
  - Defined as type of *Vehicle Routing with Pickup and Delivery Problem* – VR-PDP
  - Computing optimal schedules using *Constraint Logic Programming* – CLP.

VR-PDP

- Well-known problem in operation research:

Given:
– A set of customers
– A vehicle pool to service deliveries

Determine:
– A minimum cost set of vehicle routes that *services* all customers

- Service:
  – Pickup = freight loading point
  – Delivery = freight unload point

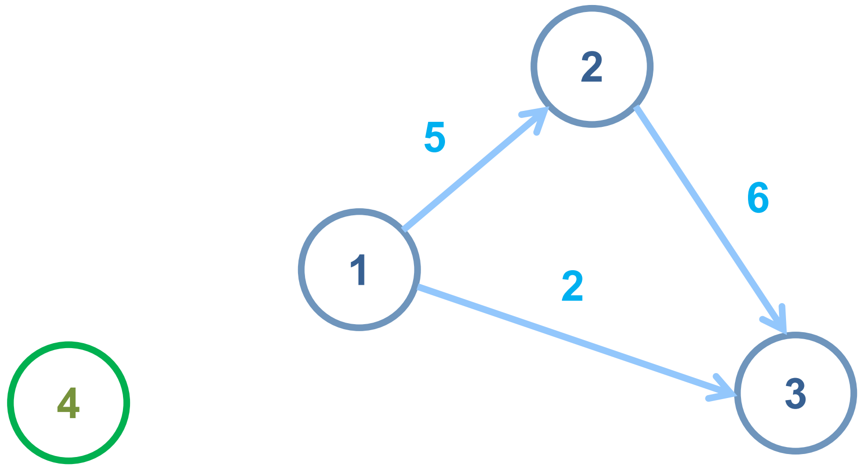
PROBLEM

Tuple $\langle \mathcal{L}, \mathcal{O}, \mathcal{T}, \Delta \rangle$ s.t.:

1. $\mathcal{L} =$ locations of interest $= \mathcal{P} \cup \mathcal{H}$
   - $\mathcal{P} = \{1,2, \dots, k\} =$ pickup and delivery points, $k > 0$
   - $\mathcal{H} = \{k+1, \dots, k+h\} =$ truck home locations, $h \geq 0$
2. $\mathcal{O} =$ customer orders, $|\mathcal{O}| = n$. $O_i = (OS_i, OD_i, C_i)$ s.t. :
   - $OS_i, OD_i \in \mathcal{P}$, $OS_i \neq OD_i$ are pickup and delivery points
   - $C_i > 0 =$ requested capacity of order i. Obs: $2 \leq k \leq 2n$.
3. $\mathcal{T} =$ set of trucks, $|\mathcal{T}| = t$, $T_i = (H_i, \Gamma_i)$ s.t.
   - $H_i \in \mathcal{H}$, $\Gamma_i > 0$ are home & provided capacity of truck i.
4. $\Delta = (k+h) \times (k+h)$ real matrix s.t. $\Delta_{ij} > 0$ are distances between $1 \leq i \neq j \leq k+h$.

Example Problem

- $k = 3$ pickup and delivery points
- $n = 3$ orders: $\mathcal{O} = \{(1 \rightarrow 2,5), (1 \rightarrow 3,2), (2 \rightarrow 3,6)\}$
- $t = 2$ trucks: $\mathcal{T} = \{(4,7), (4,5)\}$. Trucks home is 4, so there are $h = 1$ home locations



SCHEDULE

Tuple $\langle X, M, S, D \rangle$ s.t.:

1. $X \in \{1,2,\dots,k\}^m$ = hops sequence of truck routes. Each location is visited, each order needs two hops, $k \leq m \leq 2n$.
2. $M \in \{0,1,\dots,m\}^t$ s.t. M_l is the number of hops of each truck l. Total number of hops: $m = \sum_{l=1}^{t} M_l$.
   - $M_l \geq 0$ allows solutions of "*at most*" t trucks.
   - $M_l \geq 1$ constraints solutions to use "*exactly*" t trucks.
3. $S, D \in \{0,1\}^{m \times n}$ are Boolean matrices s.t.:
   - $S_{ij} = 1$ iff X_i is pickup point of order j, else $S_{ij} = 0$.
   - $D_{ij} = 1$ iff X_i is delivery point of order j, else $D_{ij} = 0$.

EXAMPLE SCHEDULE

- $k = 3$ pickup and delivery points
- $n = 3$ orders: $\mathcal{O} = \{(1 \rightarrow 2,5), (1 \rightarrow 3,2), (2 \rightarrow 3,6)\}$
- $t = 2$ trucks: $\mathcal{T} = \{(4,7), (4,5)\}$. Trucks home is 4, so there are $h = 1$ home locations

Truck t	1			2	
Γ	Maximum capacity 7			Maximum capacity 5	
T	Capacity 5	Capacity 6	Capacity 0	Capacity 2	Capacity 0
S, D	Load 1: 1,5	Unload 1: 2,5 Load 3: 2,6	Unload 3:3,6	Load 2: 1,2	Unload 2: 3,2
X	1	2	3	1	3
Hop i	1	2	3	4	5

State Space Size

- $|X| = m,\ X_i \in \{1,2,\dots,k\},\ k \leq m \leq 2n$
- $|S| = 2^{mn},\ |D| = 2^{mn}$
- $Size = \sum_{m=k}^{2n} 2^{mn} 2^{mn} m^k$

- $n = 5,\ k = 6$
- $Size = 2^{60}6^6 + 2^{70}7^6 + 2^{80}8^6 + 2^{90}9^6 + 2^{100}10^6 > \mathbf{10^{36}}$!!

CONSTRAINTS

- Pickup & Delivery Definition
- Non-Redundant Hops
- Pickup Precedes Delivery
- Service Completeness
- Truck Assignments
- Capacity Constraints
- Optimization Cost

Pickup Precedes Delivery

- For all hops i, k if there exists an order j such that i is the pickup point of j and k is the delivery point of j then i precedes k, so:

$$(\forall i, k)\ ((\exists j)((S_{ij} = 1) \wedge (D_{ij} = 1)) \Rightarrow (i < k))$$

- Using $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$ we obtain a simpler form:

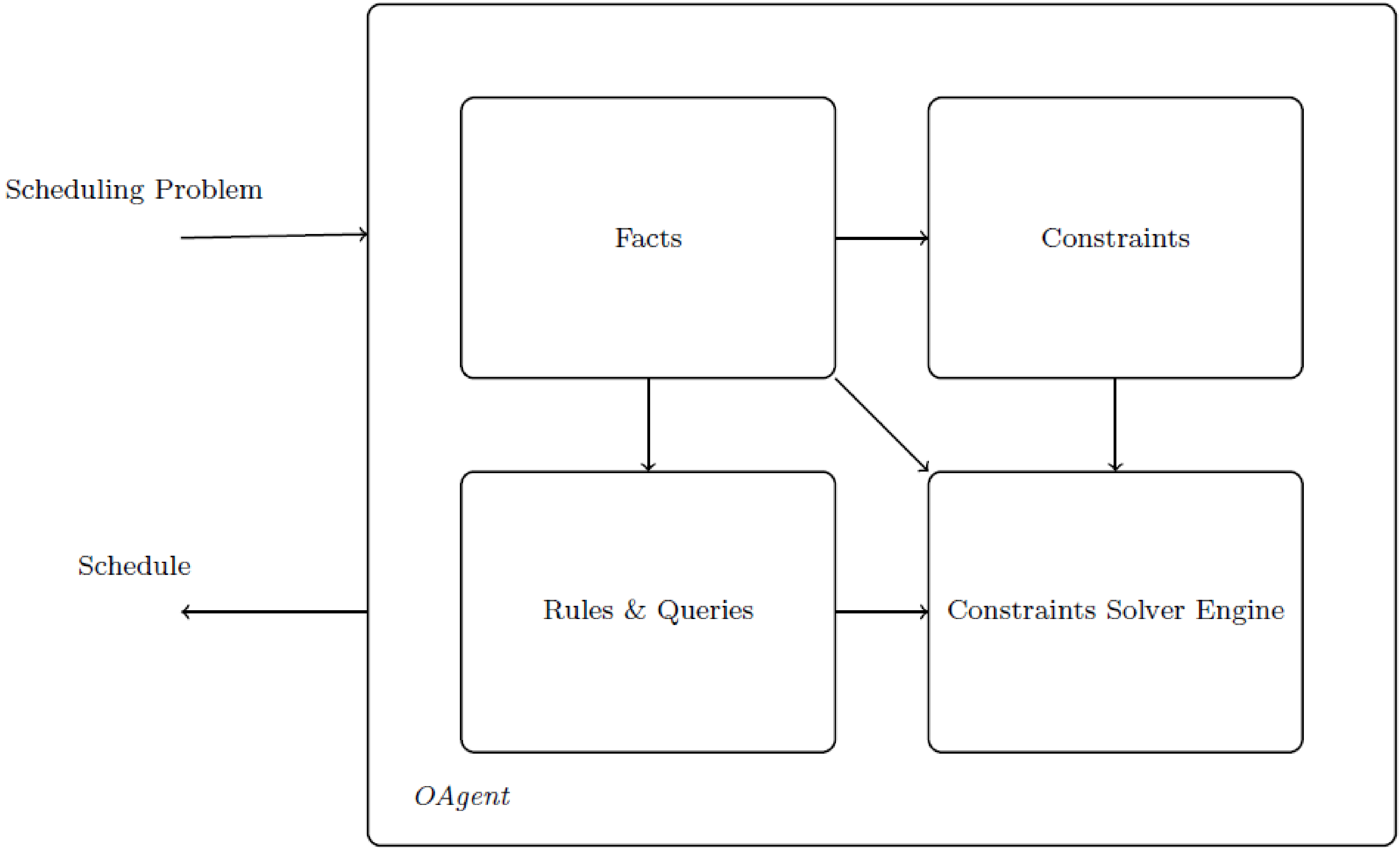
$$(\forall i, j, k)((i \geq k) \Rightarrow ((S_{ij} = 0) \vee (D_{ij} = 0))$$

Constraint Logic programming

- CLP program = facts and rules built using predicates:
  i. Normal Prolog predicates handled by Prolog engine
  ii. Constraints handles by special constraint solvers

- CLP program structure:
  i. Definition of variables and domains
  ii. Definition of constraints
  iii. Definition of cost variable
  iv. Search for optimal solution

OPTIMIZATION AGENT – *OAGENT*



SCHEDULING PROBLEM AS PROLOG FACTS

```
number_of_orders(3).
number_of_cities(3).
number_of_trucks(3).
number_of_homes(1).
% order(OrderIdx, LoadIdx, UnloadIdx, Capacity)
order(1,1,2,5).
order(2,1,3,2).
order(3,2,3,6).
% truck(TruckIdx, HomeIdx, MaxCapacity)
truck(1,4,7).
truck(2,4,4).
truck(3,4,5).
% distance(LocationI, LocationJ, DistanceIJ)
distance(1,2,10).
distance(2,1,10).
% ...
distance(4,3,10).
```

SOLUTION PREDICATE

```
solution(_m,_n,_t,M,S,D,X,Ds) :-
    domains_and_variables(_m,_n,_t,M,S,D,X,Delta),
    constraints(_m,_n,_t,M,S,D,X),
    search_query(X,S,D),
    compute_distances(_m,_t,M,Delta,X,Ds).
```

Implementation of Constraints

- Logic:

$$(\forall i,j,k)((i \geq k) \Rightarrow ((S_{ij} = 0) \vee (D_{ij} = 0))$$

- ECLiPSe-CLP:

```
constraint_2(M,N,S,D) :-
    ( for(J,1,N), param(M,S,D) do
        ( for(I,1,M), param(J,S,D) do
            ( for(K,1,I), param(I,J,S,D) do
                S[I,J] #= 0 or D[K,J] #= 0
            )
        )
    ).
```

SEARCH PROCESS

- Top level search
  - Instantiates number of hops m and vector M of route lengths of each truck such that:

$$\sum_{l=1}^{t} M_l = m$$

- Main search
  - Searches for problem solutions X, S, D for fixed values of m and M.

SEARCH QUERY

```
search_query(X,S,D) :-
    search([](X,S,D),0,
        most_constrained,
        indomain_random,
        lds(2),
        [nodes(28000)]
    ).
```

Stochastic search → indomain_random

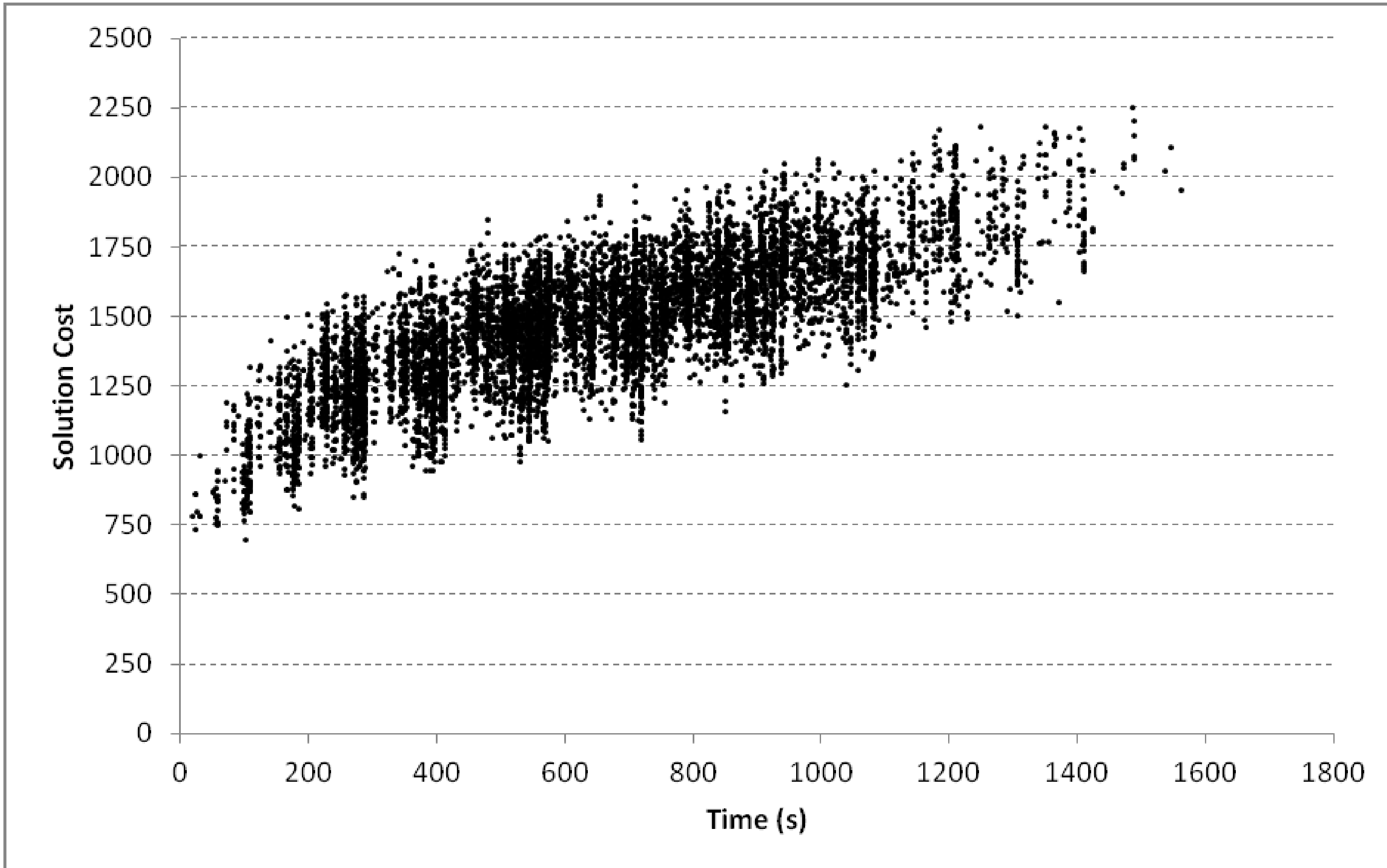
Limited discrepancy search → lds(2)

DATA SET & SEARCH CONFIGURATION

Data set	*n*	*t*	*k*	Top search	Main search	Timeout/run [s]	# Runs	# nodes/ main search	Limit reached?
10-4	10	4	6	Complete	lds(1)	60	20	10,000	No
10-8	10	8	6	1ds(3)	lds(1)	90	20	10,000	No
10-12	10	12	6	lds(3)	lds(2)	150	20	12,000	Yes
15-4	15	4	6	lds(3)	lds(1)	90	20	10,000	No
15-8	15	8	6	lds(3)	lds(1)	150	20	10,000	No
15-12	15	12	6	lds(3)	lds(2)	210	20	14,000	Yes
20-4	20	4	6	lds(3)	lds(1)	120	20	12,000	Yes
20-8	20	8	6	lds(2)	lds(2)	180	20	18,000	Yes
20-12	20	12	6	lds(2)	lds(2)	240	20	16,000	Yes
25-4	25	4	6	lds(2)	lds(2)	150	20	12,000	Yes
25-8	25	8	6	lds(2)	lds(2)	210	20	12,000	Yes
40-4	40	4	6	lds(3)	lds(2)	300	12	38,000	Yes

n is the number of orders, *t* is the number of trucks, and *k* is the number of locations

SOLUTIONS OF 10-4 PROBLEM



References

1. Amelia Bădică, Costin Bădică, Florin Leon, Ionuţ Buligiu: *Modeling and Enactment of Business Agents Using Jason*, In: Proc. 9th Hellenic Conference on Artificial Intelligence, SETN '16, ACM, (2016) doi: 10.1145/2903220.2903253
2. Costin Bădică, Alex Becheru and Samuel Felton. *Integration of Jason Reinforcement Learning Agents into an Interactive Application*. In: Proc. 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, (SYNASC 2017), 361-368, IEEE, (2017) doi: 10.1109/SYNASC.2017.00065
3. Costin Bădică, Florin Leon, Amelia Bădică. *Freight transportation broker agent based on constraint logic programming*. Evolving Systems. Springer (2018) doi: 10.1007/s12530-018-9230-3


Thank You