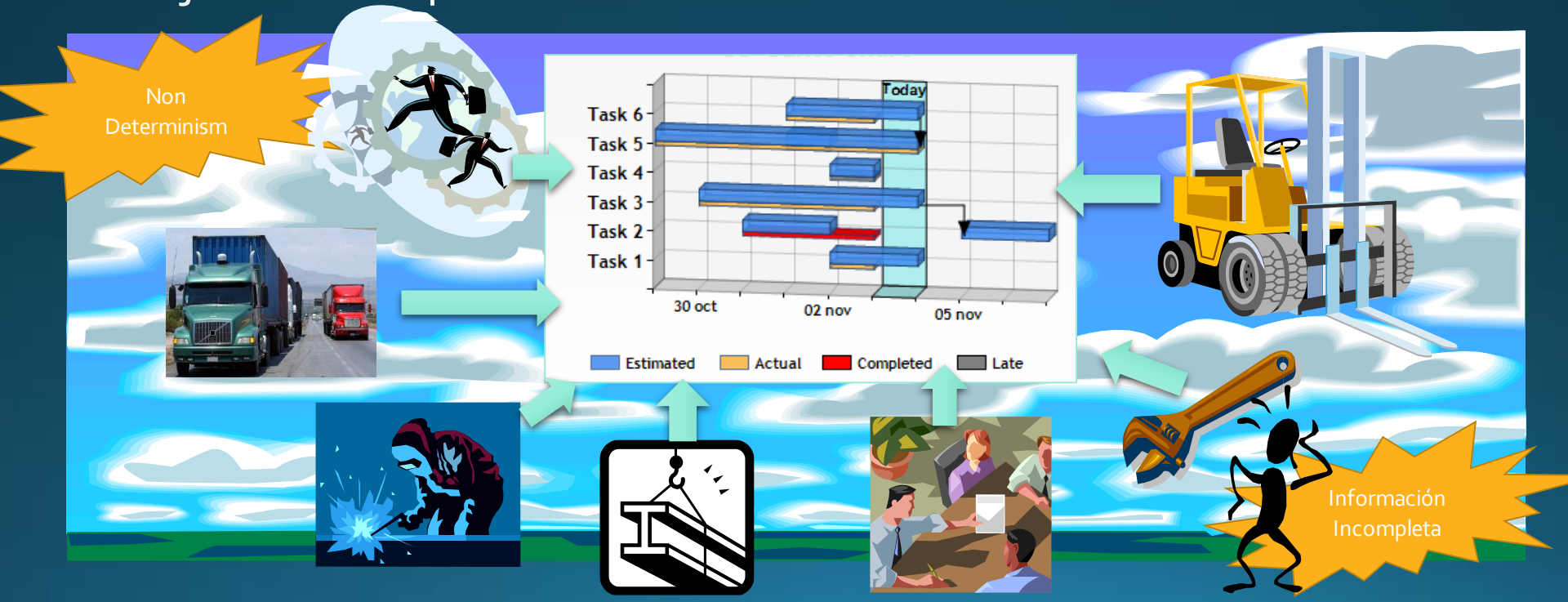
A Cognitive Approach to Real-time Rescheduling using SOAR-RL

Juan C. Barsce
Dto. De Sistemas – UTN – Fac. Reg. V. María, jbarsce@frvm.utn.edu.ar
Jorge A. Palombarini
GISIQ – Dto. De Sistemas - UTN - Fac. Reg. V. María, jpalombarini@frvm.utn.edu.ar
Ernesto C. Martínez
INGAR (CONICET-UTN), ecmarti@santafe-conicet.gob.ar

Problem statement - What is scheduling?

- The scheduling task can be described as assigning a set of activities to a limited number of resources in a consistent manner over time, so as to avoid violations of the restrictions associated with the problem, while a determined set of objectives is optimized.



Problem statement - Real-time rescheduling

- A schedule is typically subject to the intrinsic variability of a process environment where difficult-to-predict events occur as soon as it is released for execution.
- If repair decisions need to be made in real-time, then **fast rescheduling is mandatory to account for unplanned and abnormal events by generating satisfying schedules rather than optimal ones** (Vieira et al, 2003).
- Therefore, the capability of generating and representing knowledge about heuristics for repair-based scheduling is a key issue in any rescheduling strategy.

State Operator And Result (SOAR)

Developed by John E. Laird in the University of Michigan.

- Mimics human cognition.
- Allows to model domains through operators, elaborations and states.
- Supports Reinforcement Learning and Chunking to help with the decision of what operator to choose when they all have the same preference.

Reinforcement Learning

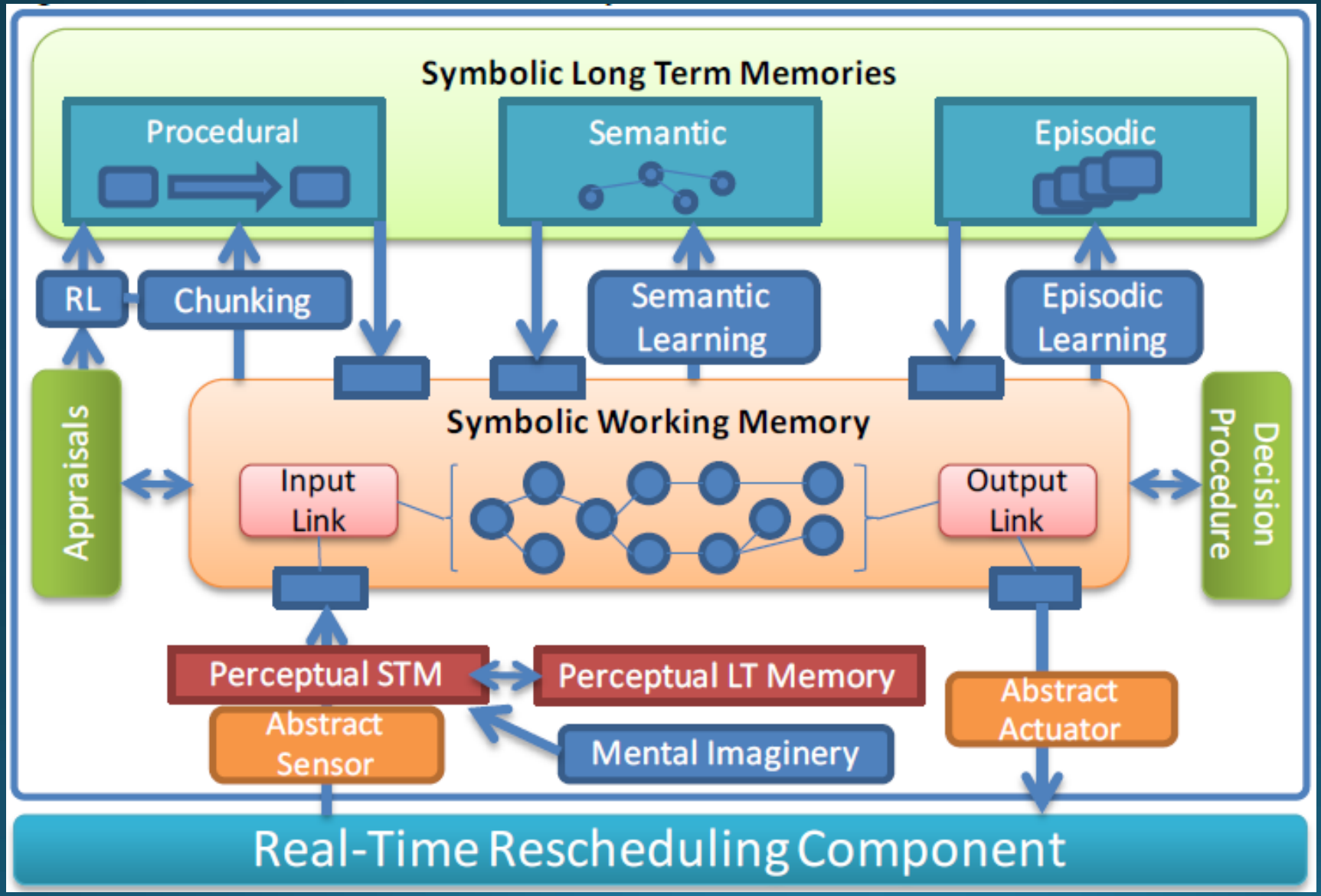
- Reward function ***r(s)***.
- Action-value function ***Q(s,a)*** that estimates the value or utility of resorting to the chosen repair operator *a* in a given schedule state *s*.

$$Q(s,a) \leftarrow r + \gamma \max_b Q(s',b)$$

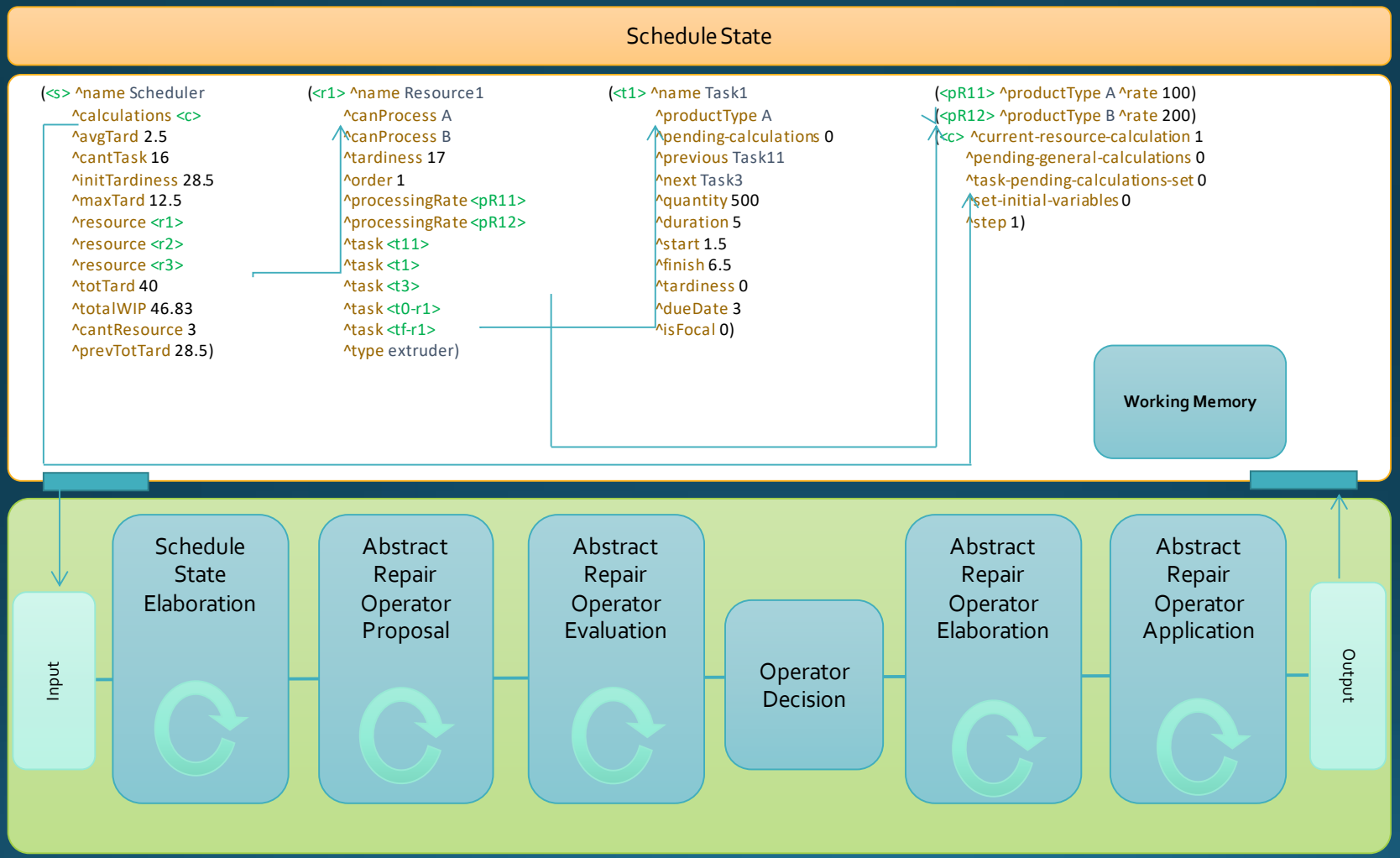
Chunking

- First SOAR mechanism to generate rules in execution time.
- Allows the cognitive system to show a reactive behavior through automatic generation of new rules in execution time, based on the facts stored in the working memory.
- It is mainly used to resolve impasses and produce new reinforcement learning rules.

SOAR – Architecture Overview



SOAR – Schedule state and decision process



SOAR – LHS and RHS of an operator

```
sp {Scheduler*apply*down-right-jump
  (state <s> ^name Scheduler ^operator <op>
         ^calculations <c>)
  (<c> ^pending-general-calculations <pgc>)
  (<op> ^name down-right-jump
     ^focalTask <nametFocal>
     ^auxTask <nametAux>)
  (<s> ^resource <r1> {<> <r1> <r2>})
  (<r1> ^task <tFocal> {<> <tFocal> <tPrevFocal>}
       {<> <tPrevFocal> <> <tFocal> <tPosFocal>})
  (<r2> ^task <tAux> {<> <tAux> <tPosAux>}
     ^processingRate <procRate>)
  (<tFocal> ^name <nametFocal>
     ^quantity <quantity> ^duration <duration>
     ^productType  <prodType>
     ^previous <prevTFocal>
     ^next <nexttFocal>)
  (<procRate> ^rate <rater2>
     ^productType  <prodType>)
  (<tAux> ^name <nametAux>
     ^previous <prevtAux> ^next <nexttAux>)
  (<tPrevFocal> ^name <nametPrevFocal>
     ^next <nametFocal>)
  (<tPosFocal>  ^name <nametPosFocal>
     ^previous <nametFocal>)
  (<tPosAux> ^name <nametPosAux>
     ^previous <nametAux>)}
```

```
{
(<tFocal> ^previous <nametAux>
          ^previous  <prevTFocal> -
     ^next <nametPosAux>
     ^next <nexttFocal> -)
(<tAux> ^next <nametFocal>
        ^next <nexttAux> -)
(<tPosAux> ^previous  <nametFocal>
           ^previous <nametAux> -)
(<tPosFocal> ^previous <nametPrevFocal>
             ^previous <nametFocal> -)
(<tPrevFocal> ^next <nametPosFocal>
              ^next <nametFocal> -)
(<tFocal> ^duration (/ <quantity> <rater2>)
          ^duration  <duration> -)
(<r2> ^task <tFocal>)
(<r1> ^task <tFocal> -)
(<c> ^pending-general-calculations 1
     ^pending-general-calculations  <pgc> -)
}
```

Industrial case study

- The plant is made up of 3 semi-continuous extruders that process customer orders for four products.
- Each extruder can process some but not all products.
- Processing rates depend on both the resource and the product.
- Each task has a due date, a required quantity (kg) and a product type.
- Disruptive event: focal task insertion.
- Goal: minimize total tardiness.

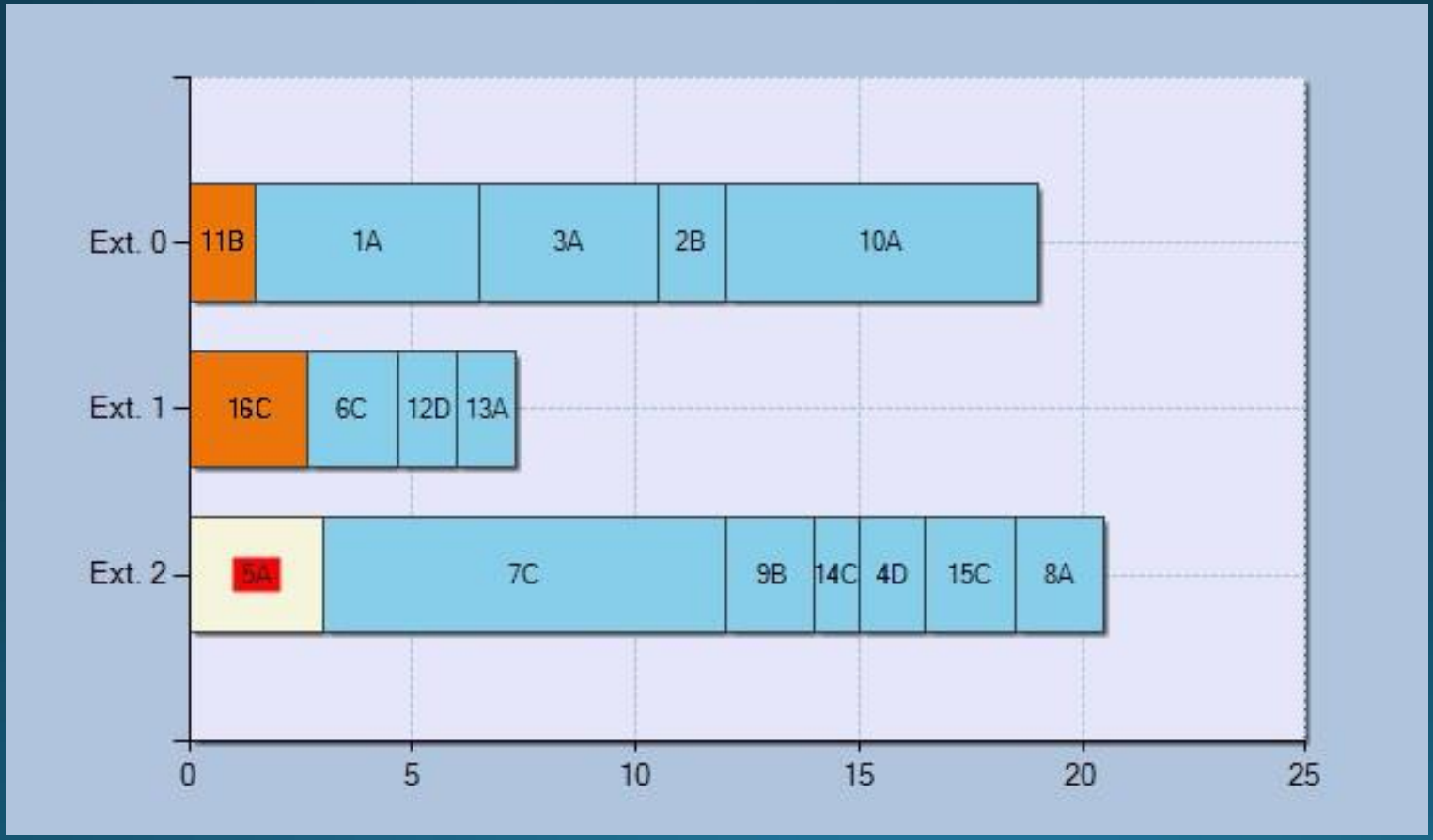
Industrial case study

Three applications have been used to implement and test the case study:

- VisualSOAR v4.61 for developing the structure of the system.
- SoarDebugger 9.2.3 for running and testing the system.
- Visual Studio 2010 for developing the Smart Gantt Prototype.

Industrial case study – Initial schedule

Initial tardiness: 28.5 h | Post-insertion T5 tardiness: 40 h



Industrial case study

Ten repair operators were proposed for any state in each repair step, divided in two classes:

- Movement operators.
- Swap operators.

For each repair operator application, a reward is given to the agent based on how close the current schedule's tardiness is from the initial tardiness.

There also were used calculation operators, designed to trigger after the application of each reparation operator.

Industrial case study – Task details

Order #	Product	Size [Kg]	DD [h]
1	A	500	3
2	B	300	7
3	A	400	12
4	D	300	4
5	A	300	4
6	C	300	10
7	C	900	7
8	A	200	10
9	B	300	14
10	A	700	18
11	B	300	19
12	D	400	15
13	A	200	13
14	C	100	15
15	C	200	16
16	C	400	20

Runtime example

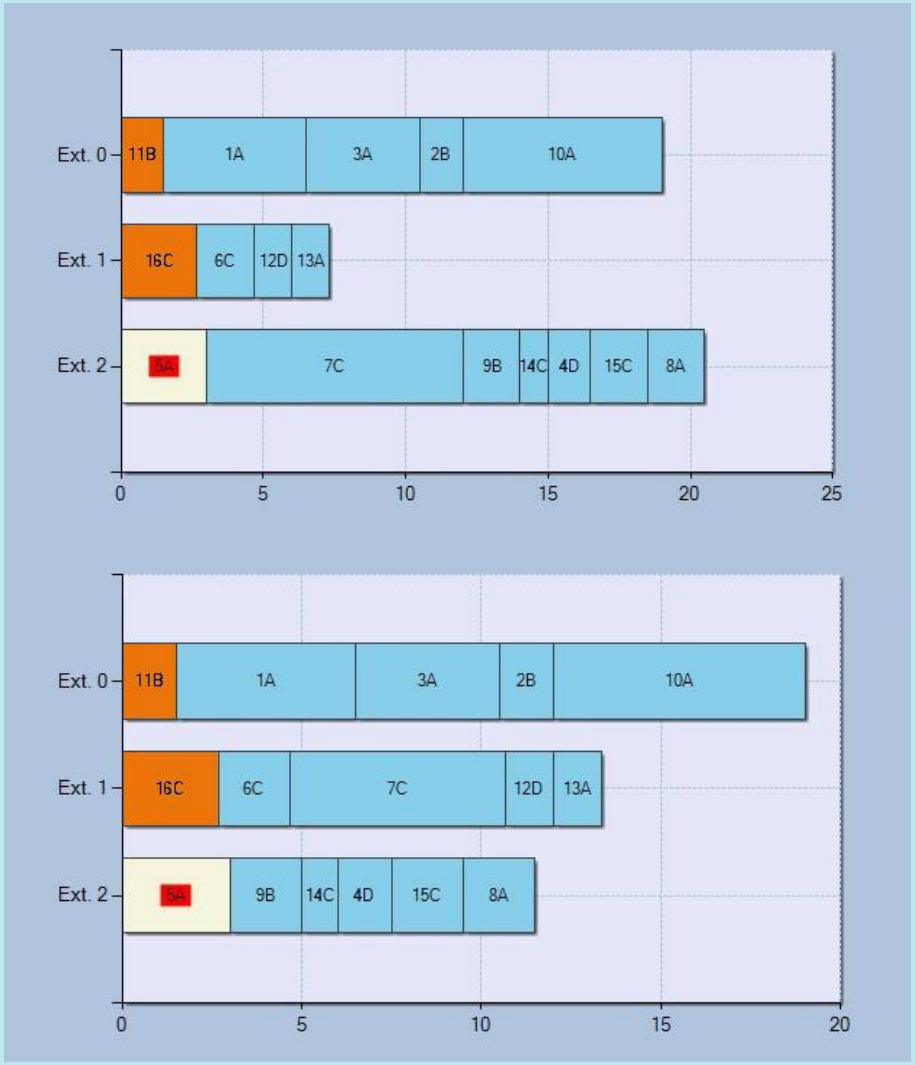
```
    30: O: O58 (move-left)
gap started (S1)
    31: O: O70 (monitor-task-pending-calculations)
    32: O: O71 (monitor-set-initial-variables)
    33: O: O73 (tardiness-calculation)
    34: O: O88 (negative-tardiness-correction)
    35: O: O78 (tardiness-calculation)
    36: O: O89 (negative-tardiness-correction)
    37: O: O81 (tardiness-calculation)
    38: O: O90 (total-tardiness-calculation)
    39: O: O83 (tardiness-calculation)
    40: O: O91 (negative-tardiness-correction)
    41: O: O82 (tardiness-calculation)
    42: O: O92 (negative-tardiness-correction)
    43: O: O83 (tardiness-calculation)
    44: O: O93 (negative-tardiness-correction)
    45: O: O83 (tardiness-calculation)
    46: O: O95 (negative-tardiness-correction)
    47: O: O94 (total-tardiness-calculation)
    48: O: O86 (tardiness-calculation)
    49: O: O96 (negative-tardiness-correction)
    50: O: O87 (tardiness-calculation)
    51: O: O97 (negative-tardiness-correction)
    52: O: O85 (tardiness-calculation)
    53: O: O87 (tardiness-calculation)
    54: O: O98 (negative-tardiness-correction)
    55: O: O87 (tardiness-calculation)
    56: O: O99 (total-tardiness-calculation)
    57: O: O101 (monitor-calculations-complete)
Tardiness: 32.5
Step: 2
gap ended (S1)
RL update rl*Scheduler*rl*rules*25 -0.0154969 0 -0.0154969 -> -0.0287751 0 -0.0287751
```

Example of reparation rule

```
{rl*Scheduler*rl*rules*157
(state <s1>
        ^totalWIP 46.83 ^taskNumber 16
        ^maxTard 15 ^avgTard 2.5  ^totTard 40
^initTardiness 28.5     ^name Scheduler
        ^operator <o1> +
        ^focalTask <t1>)
(<o1>         ^auxTask Task10
              ^focalTask Task5
                    ^name up-right-jump)
-->
(<s1> ^operator <o1> = -0.1498)}
```

Results

- Initial tardiness: 28,5 h.
- Tardiness after insertion of focal task: 40 h.
- Total training episodes: 20.
- Max. tardiness: 86 h.
- Allowable amount of steps per episode: 400.
- Best amount of steps to achieve goal: 6.
- Best tardiness: 18.5 h.



Concluding remarks

- SOAR is a cognitive architecture that provides a cognitive approach to solve problems **balancing optimization v. opportunity**.
- The innate support for Reinforcement Learning and Chunking allows the system to **learn through execution** and **generate automatically new rules**.
- This makes the approach a good solution to real time rescheduling problems without the need to recalculate all the schedule from the beginning when a disruption occurs, and use the learning done for the future states.

Current Work

We are currently working in:

1. A completely new case study, using now a complex schedule of a five-stage paint plant.

2. A general improving of the reinforcement learning rules.

3. A general cleaning of the source code, making operators and elaborations easier to maintain.

Future Work

Our plains for the next 6 – 12 months are the following:

1. Add another disruptive events.
2. Add support for the semantic and episodic memories.
3. Release the next version of the Smart Gantt Prototype, with full integration with input and output links.

Thank you!