DING+

The CABA use tutorial

责任承若 . 品质结果

2018/12/05

Catalog

- 1. CABA overview
- 2. Learn to use CABA (run, navigation, etc)
- 3. Knowledgebase
- 4. Process manager + task manager

1. CABA Overview

- Objectives:
  - to know about basic CABA information
  - to able to identify the main CABA features
- The outline:
  - CABA history
  - The architecture
  - Built-in component: request whiteboard
  - Overviews of main components
  - CABA applications
  - Next generation

1.1.1 CABA History

- CABA is partly extended or inspired from the following systems, theories, or frameworks:
  - JARE/Jess
  - CAST, Soar, and taskable agent
  - RPD (Recognition Primed Decision-making)
  - Information Supply Chain
- Initiated in 2003
- First implementation finished in late 2004
- Current version is v2.0

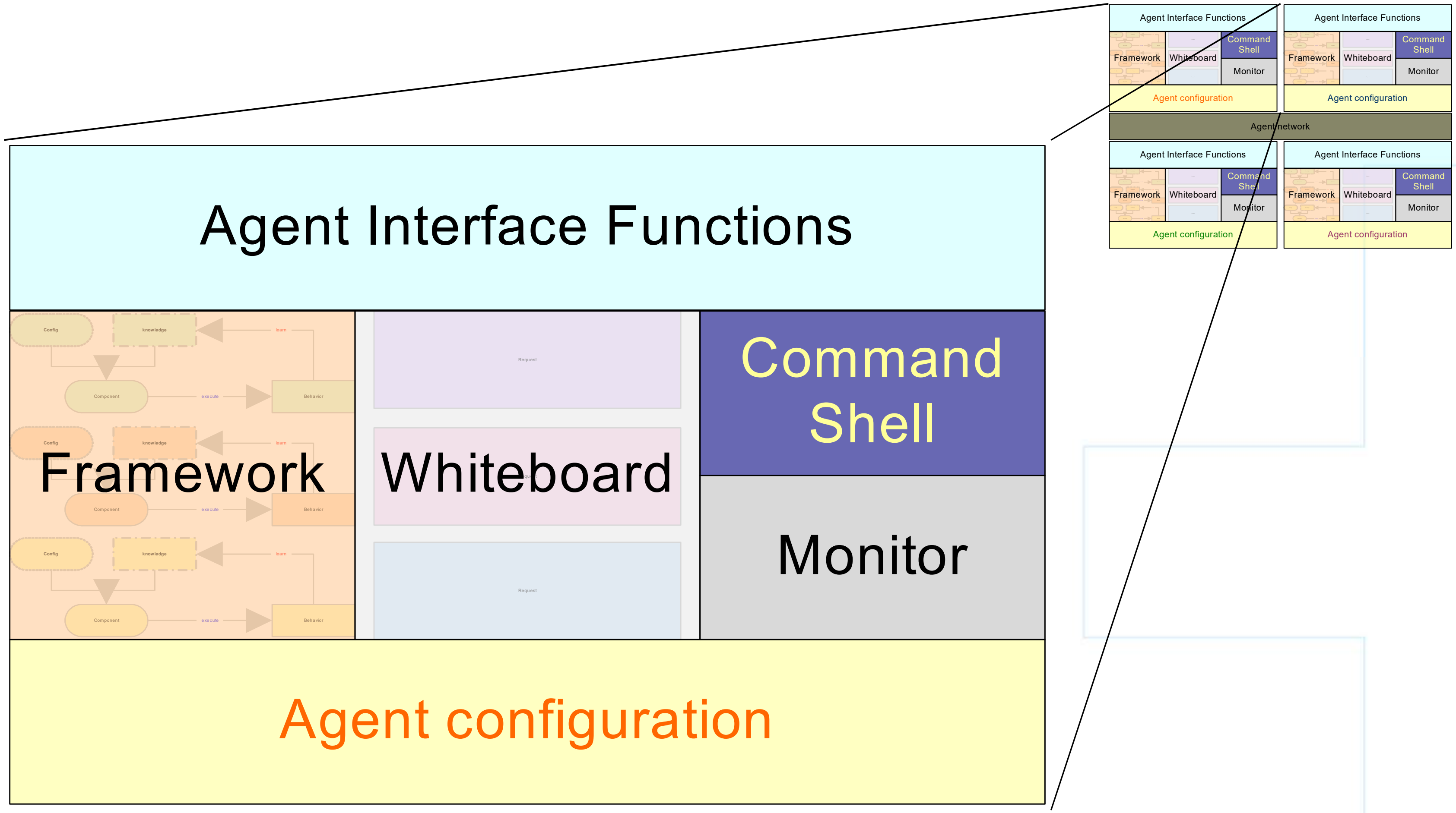
1.1.2 What is CABA

- It is
  - an intelligent agent architecture,
  - a naturalistic decision modeling tool,
  - an information sharing agent architecture.
- It can be used for building
  - intelligent systems,
  - cognitive models with three types of knowledge
    - Declarative memory as knowledge
    - Procedure memory as processes
    - Episodic memory as experiences
  - decision models or decision support systems,
  - collaboration models that can anticipate and manage information requirements.
  - practice for agent oriented software engineering

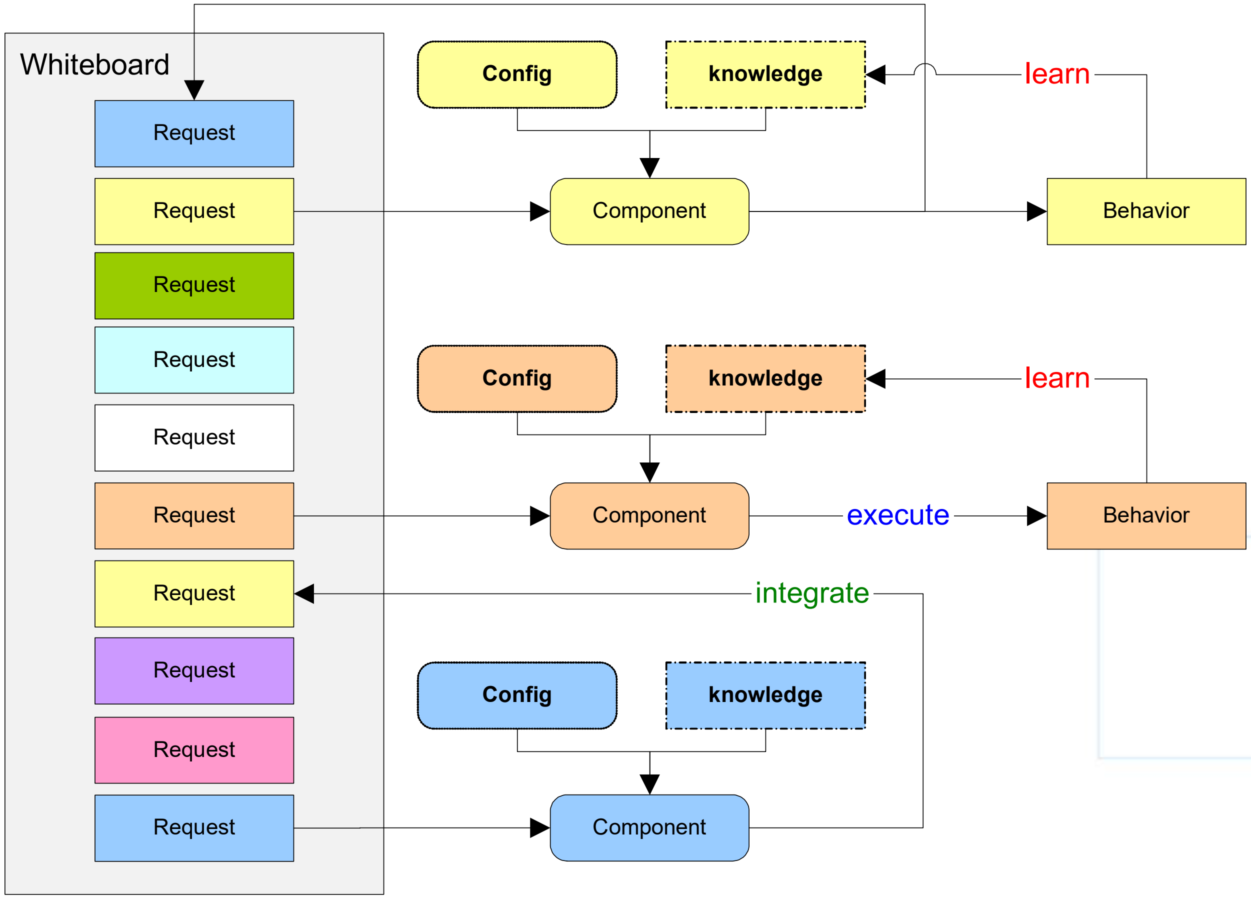
1.2 The CABA Architecture (Overview)

- Is component based, therefore CABA is configurable
  - KB+PM: Basic agent
  - KB+PM + TM: Collaborative agent
  - KB+PM+RPD: Decision modeling
  - KB+PM+IM: Extended CAST
  - KB+IM: Information agent
  - currently configurations of the components cannot be changed at run time.
- Has two perspectives:
  - Cognition:
    - Declarative knowledge
    - Procedure knowledge
    - Experience knowledge
  - Information management:
    - Demand manager
    - Supply manager
    - Information requirement planning

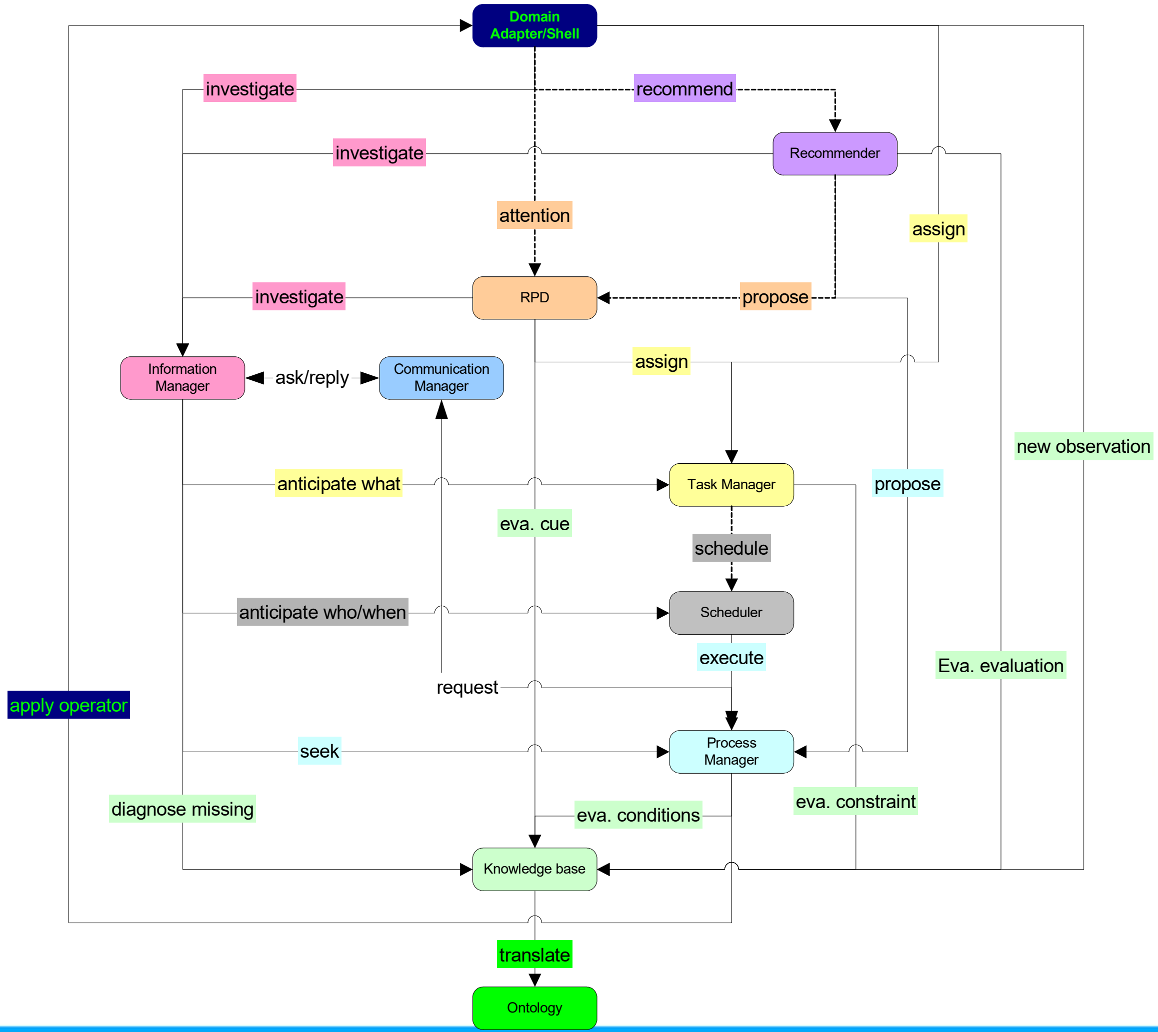
1.2.1. The CABA Architecture (Overview)



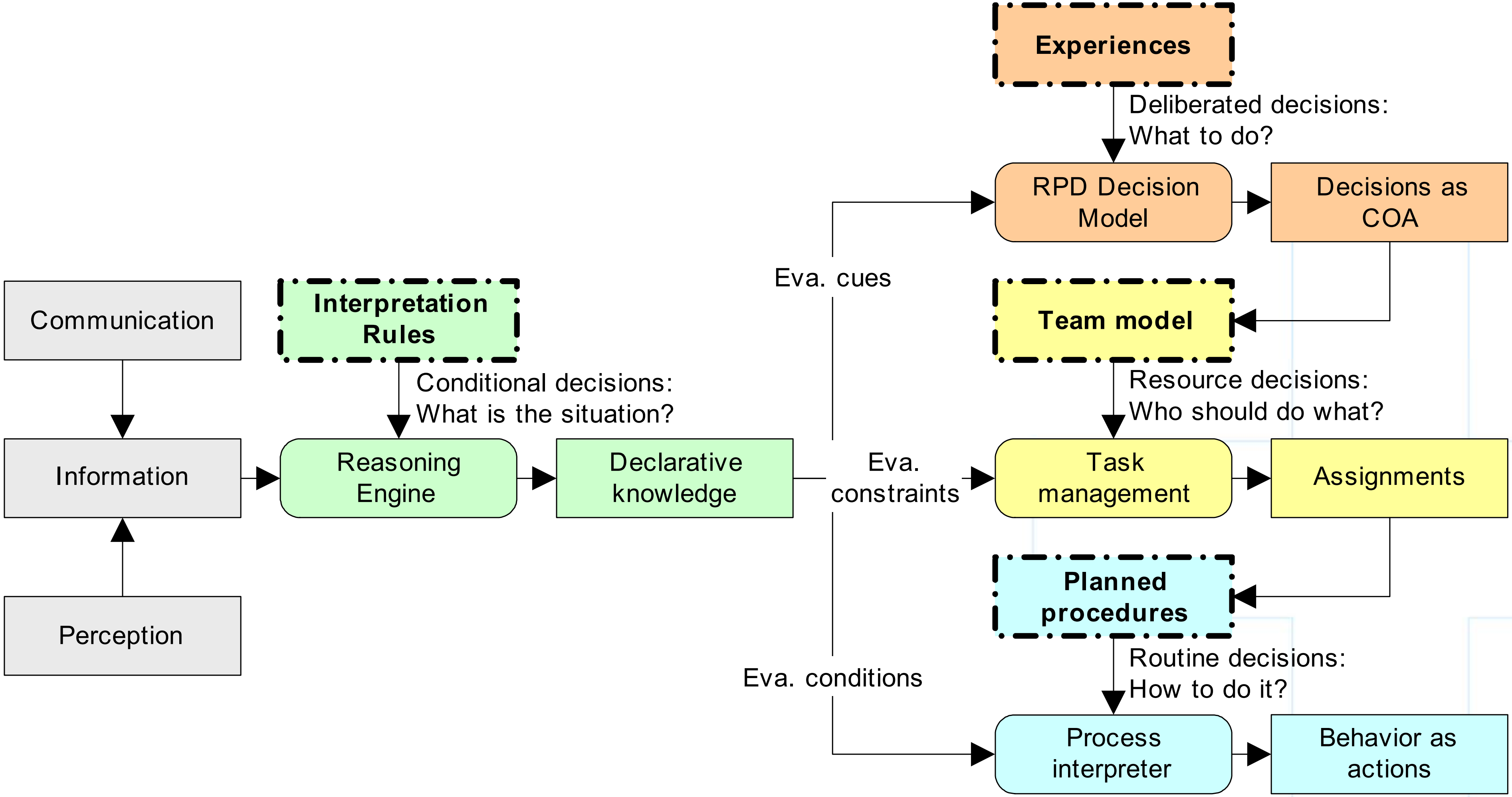
1.2.2 The CABA Architecture (Framework Perspective)



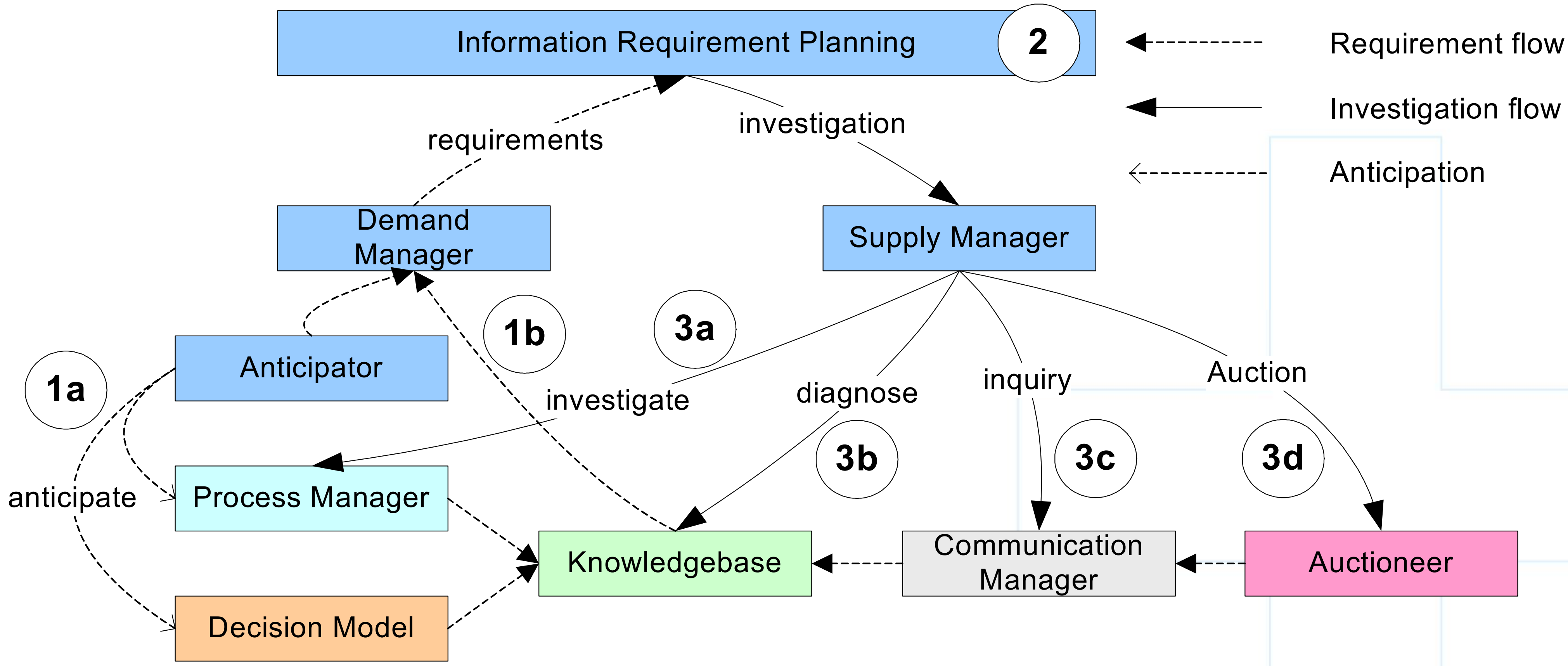
1.2.3 The CABA Architecture (System Perspective)



1.2.4 The CABA Architecture (Cognition Perspective)



1.2.5 The CABA Architecture (IM Perspective)



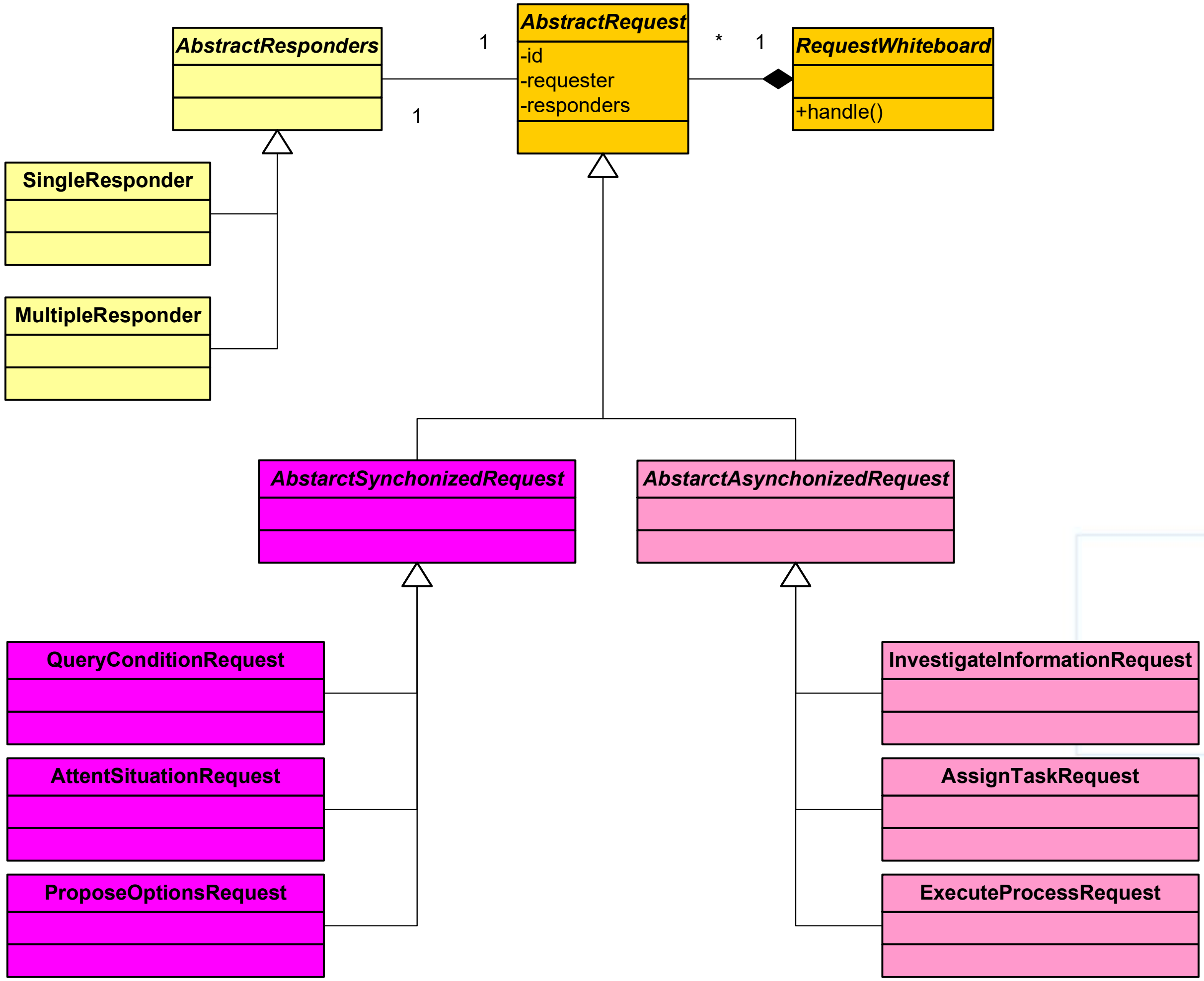
1.3 Request Whiteboard

- Objectives:
  - To understand the functions of the request whiteboard
  - To be able to add new request types
- The outline:
  - About Request Whiteboard
  - The Request Whiteboard system design
  - The whiteboard panel, configurations
  - An implementation guide
  - Practice

1.3.1 About Request Whiteboard

- The goal of Request Whiteboard is to provide a built-in integration method between two components.
- An Request Whiteboard explicitly represent functional requests among components, e.g.
  - Communication request to the communication manager
  - Investigation request to the information manager
  - Execution request to the process manager
- Basic request process:
  - Synchronized request is for requests that can be accomplished in one step
  - Asynchronized request is in two steps (request and notify when done)
- Responding
  - For single responder requests, the first component will respond. E.g. send a message.
  - For multi responder requests, any components can respond. E.g. propose options.

1.3.2 Request Whiteboard (Structure Design)



1.3.4 Request Whiteboard Configurations

- # The thread cycle time for the whiteboard, in 1/1000 sec
- whiteboardClock = 1000

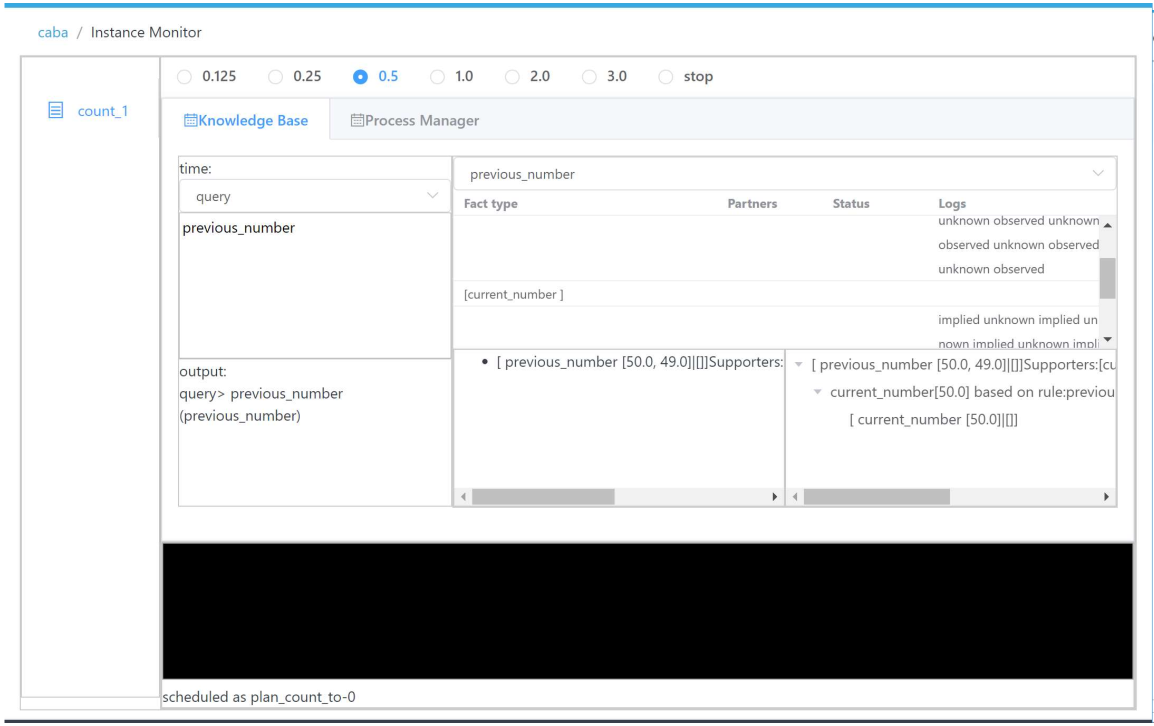
- # use whiteboard gui or not,
- whiteboardGUI = true

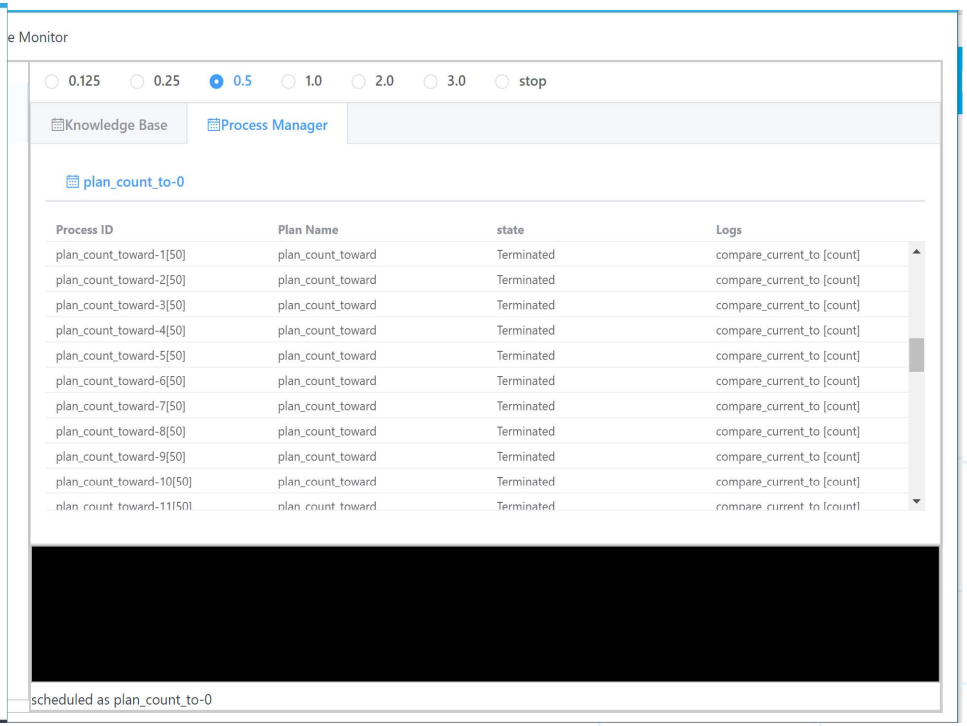
1.4 The Main Components

- Knowledge base
- Process manager
- Recognition primed decision-making
- Information manager
- Task manager
- Resource manager*
- Scheduler*
- Ontology translator*
- Auctioneer*
- Recommender
- Communication manager
- Configuration manager

- * indicates components under development

1.4.1 Some Monitors of the Main Components





1.4.2 Active Knowledge Base (AKB)

- Is a forward chaining rule base system (declarative memory only).
- Has three classes of information
- Constant fact
- Volatile fact
- Implied fact
- Implied facts are linked to their evidences (supporting facts).
- Each fact associates with a type called fact type.
- A fact type defines
- A template that is used for natural language translation.
- A source list that is used to identify information providers.
- A default expiration time for volatile facts.

1.4.2 AKB Features

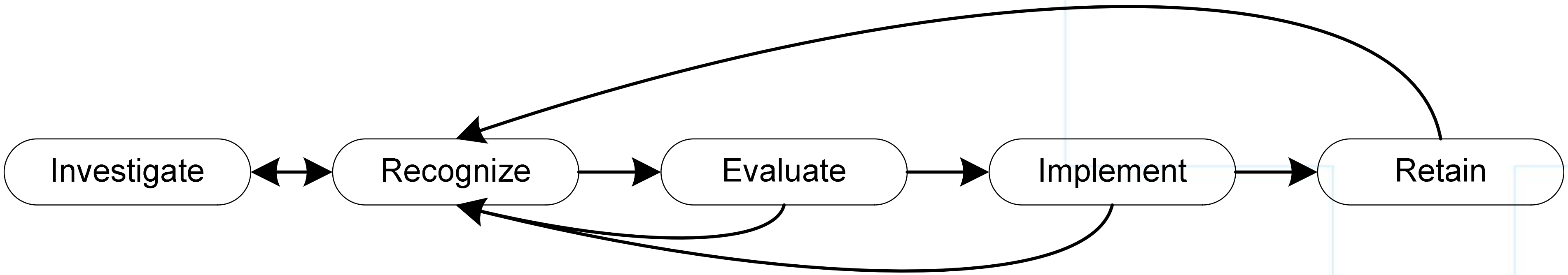
- Conventional KB features: truth maintenance, query, functions (+-*/rand, eq, dis)
- Explicitly represents information dependency relations (IDR)
  - at fact type level (schema/class)
  - at fact level (instance)
  - with visualization
- (trying) explicitly represents “Unknown” status for fact types or queries.
- is able to diagnose a condition (set) for missing information according to IDR and what is known
- can forget expired facts.
- can understand structured natural language for assertion and reply a query in structured natural languages as text or as speech.

1.4.4 Process

- Contains
  - preconditions, (conjunctive) and effects
  - termination conditions, (disjunctive/conjunctive)
  - fail conditions, (disjunctive) and contingency plan
  - a process body that including steps, each of which can be a
    - Operator including two built-in operators (print and speak)
    - Plan
    - Choice
- When being executed, a process has five states
  - active, suspended, wait,
  - failed, or terminated state

1.4.5 Recognition Primed Decision (RPD)

- Is an experience based (episodic knowledge) decision making model. (invented by Klein)
- Is similar to case based reasoning but with
  - a semantic experience (case) representation and
  - an investigation step for missing information.
- Is a computerized model.
- Is visualized.



1.4.6 RPD Features

- Experiences are organized in experience spaces, which form a tree structure.
- Recognition as a interactive navigation/search process in the experience spaces.
- Missing information is passed to the information manager (IM).
- Evaluation of a plan is a mental simulation by the process manager (PM).
- COA is implemented by the process manager (PM).
- New experiences are learned in the retain state.

1.4.7 Information Manager (IM)

- Is responsible for manage *demand* and *supply* of information.
- Is based on the information supply chain (ISC) framework.
- Is based on the CAST framework: anticipation of information needs.
- Can be viewed as a motivation for collaboration.
- Information requirements are conditions:
  - Process: precondition, termination condition, fail condition, preference condition
  - Decision: cues, anomalies, expectancies
  - Recommendation: evaluations

1.4.8 IM Functions

- Demand manger:
  - Explicit information requirements
  - Anticipation of information requirements
- Information requirement planning:
  - Consolidate open requirements
  - Systematic investigations
- Supply manager:
  - Configurable investigation strategies
    - Inquiry -> Diagnose -> Investigate
    - Investigate -> Diagnose -> Inquiry
    - Auction -> Diagnose -> Investigate

1.4.9 Communication Manager (CM)

- Is responsible to
  - maintain an address book,
  - handle message exchange, and
  - manage conversations.
- A message is read by the component that can understand it.
  - Information sharing (Inquiry/ Inform)
  - Auction (RFQ/Offer/Order)
  - Conversational (Agree, NotUnderstand)
- Designed to handle heterogeneous communication channels: RMI, JMS, web services (SOAP), etc.
- Internal *ping* function for testing.

1.4.10 Task Manager (TM)

- Is responsible to
  - maintain an team model,
  - assign tasks based on capabilities
  - assign tasks
  - monitor assignments
- Collaborative resource allocation (may be split-off into a independent resource manager

1.4.11 Auctioneer

- Auctionable items:
  - Task
  - Resource
  - Information
- The goal is to identify the provider of information, task, resources when
  - Lack of knowledge about providers
  - Lack of information to determine the most efficient providers
    - With lowest switching cost from current task
    - With a easy way direct observe, known, or infer

1.4.13 The Main Components- Review

- Active knowledge base: forward chaining
- Process manager: process execution and simulation
- RPD: a naturalistic decision model
- Information manager: demand and supply of information requirement
- Communication manager: a message based heterogeneous communication channel
- Auctioneer: a market based method for finding venders of information, resources, and tasks
- Task manager: a capability based task assignment
- Configuration manager: for making the architecture adaptable to diversified needs.

1.4.12 Configuration Manager

- Is responsible for making the agent architecture flexible.
- With over 50 designed configurations (from agent components, clock speed, to icon images), you can build an agent that fits your needs (hopefully).
- Is designed by motivation to make models free from violations of cognitive constrains.
- Configurations are interpreted by each individual components
- Configurations can be only adjusted by human users, offline or online (for some items).
- A shell is designed to take online commands: over 20 (could be more) commands are available (from adjusting agent configurations to manipulating agent's behaviors).

1.4.13 The Main Components- Review

- Active knowledge base: forward chaining
- Process manager: process execution and simulation
- RPD: a naturalistic decision model
- Information manager: demand and supply of information requirement
- Communication manager: a message based heterogeneous communication channel
- Auctioneer: a market based method for finding venders of information, resources, and tasks
- Task manager: a capability based task assignment
- Configuration manager: for making the architecture adaptable to diversified needs.

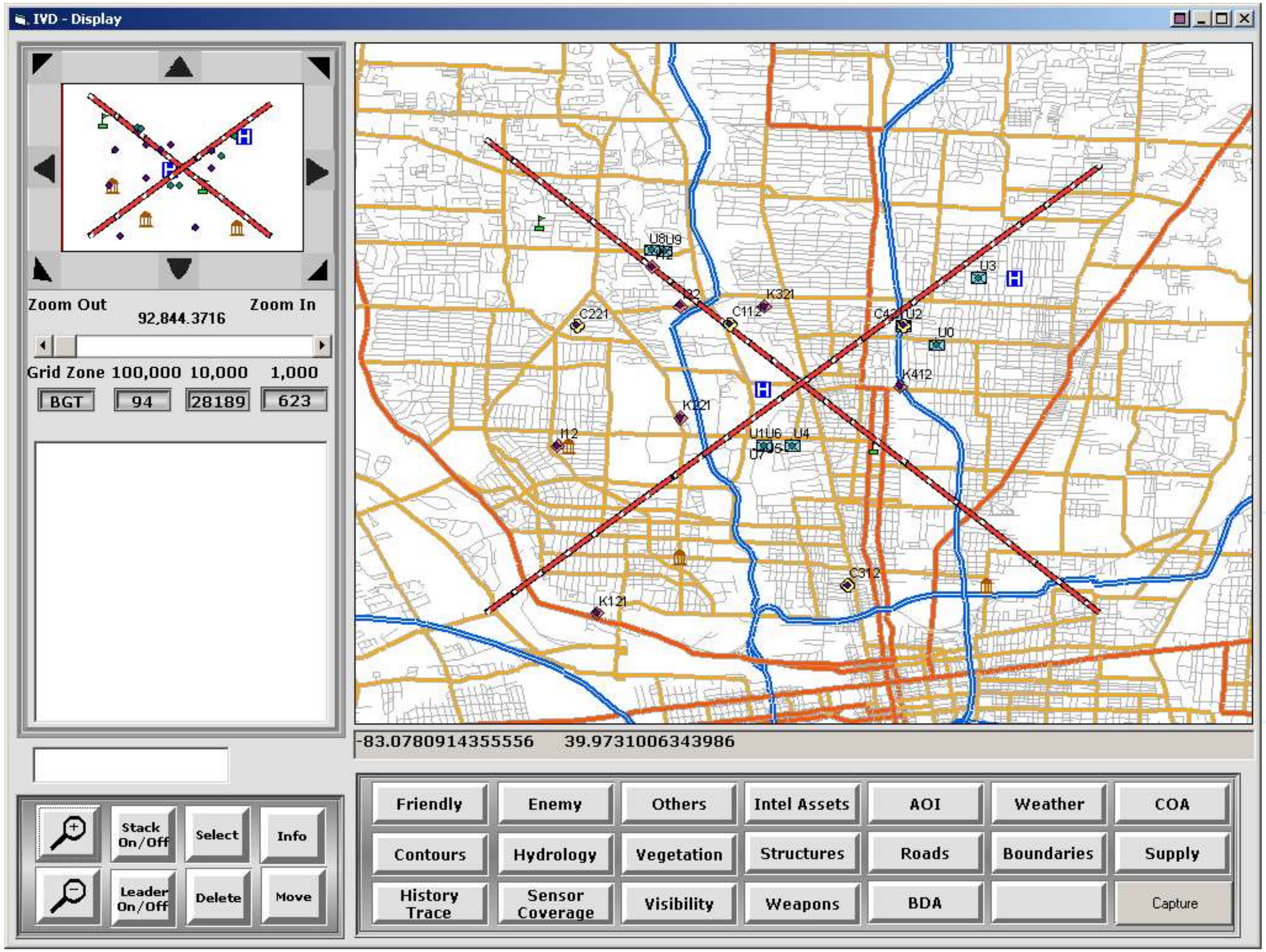
1.5 CABA Applications

- Cognitive modeling:
  - Model human behaviors
  - Naturalistic decision making
  - Computer game opponents
- Intelligent systems:
  - Optimize collaboration
  - Resource/task allocation
  - Automate control and processes
- Information management:
  - Better search engines
  - Time-critical intelligence sharing
  - Efficient information sharing
  - Trust of information

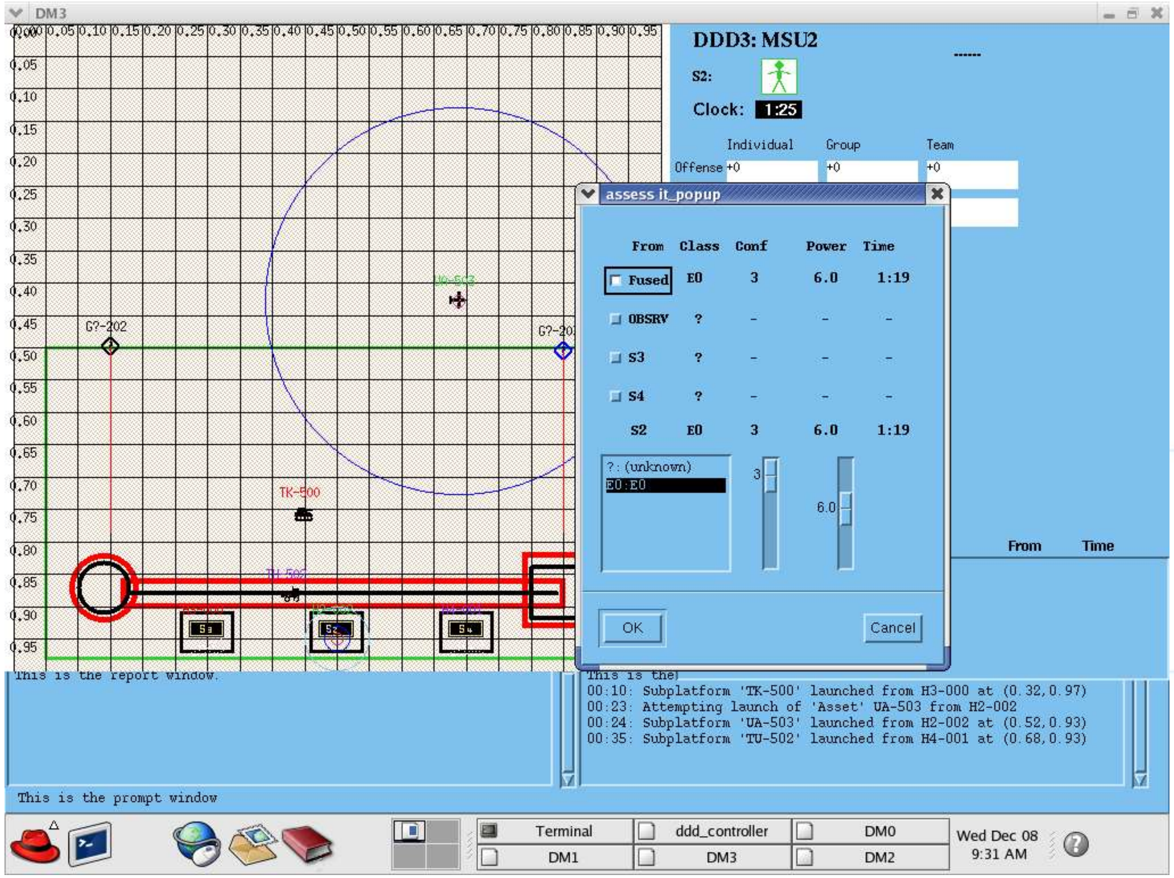
1.5.1 CABA Supported Research Projects

- Founded research projects
  - 3-block challenges, to study effective information exchange under multiple types of operations (with Army Research Lab)
  - Agent based information fusion and decision support (with Solers and Wagner in an ONR program)
- Other research that involves CABA
  - Study secure and credible information sharing (with Dr. Peng Liu)
  - Study bias-aware agents that can detect and overcome human cognitive biases (with Dr. Tracy Mullen)
  - Study information sharing among emergency response teams (with Dr. Michael McNeese)
  - Study error responding in cognitive models (with Dr. Frank Ritter)
  - Study information sharing in a simulated color block game (Sun)

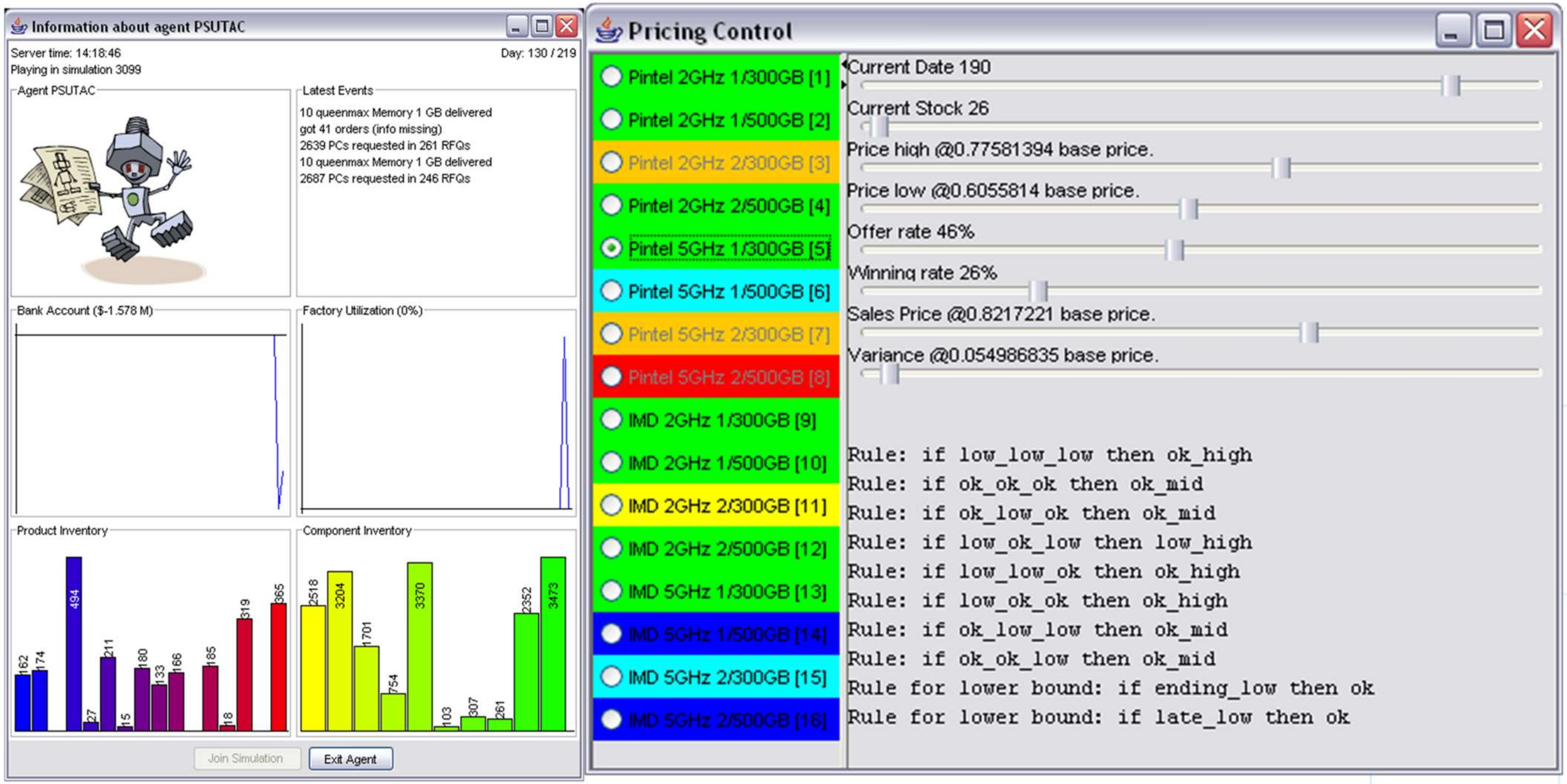
1.5.2 Three-Block Challenges: Multiple Decision Types



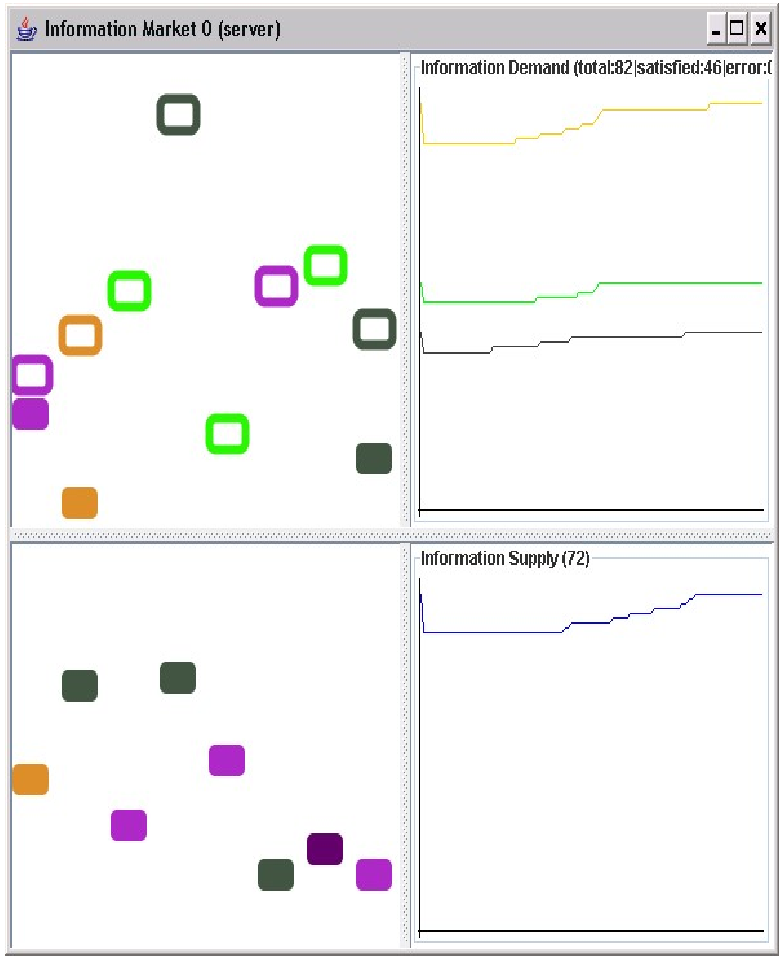
1.5.3 A Combat Scenario Based Experiment: Human Agent Interaction

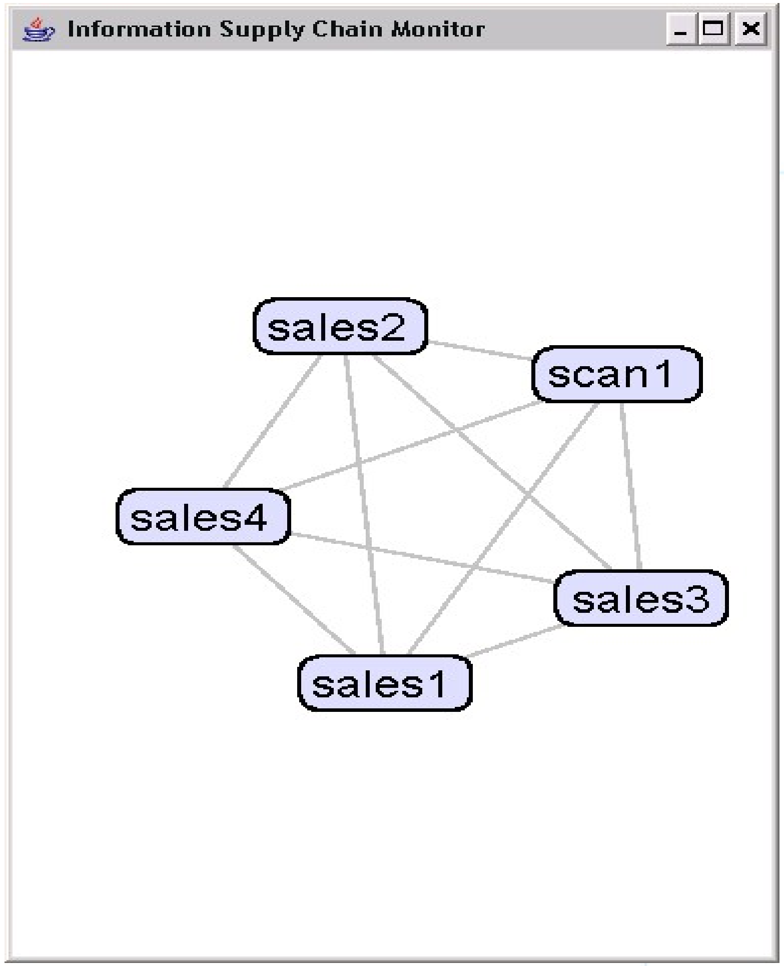


1.5.4 PSUTAC for Trading Agent Competition: Human Biases



1.5.5 Information Color Block Game: Balance Demand and Supply





DING+

1.6 CABA Next Generation

- Broader
  - Cognition
    - Meta cognition: adaptable cognition structure
    - Other decision making models: Bayesian, Fuzzy, CBR, etc.
  - Information Management
    - Credibility
    - Security
- Deeper
  - Cognition
    - Plan adaptation: adapt plan to better address the needs and to generate new experience for RPD
  - Information management
    - Performance evaluation
    - Macro/Collective view

1.7 CABA Overview - Review

- CABA history
- The architecture
- Main components
- CABA application
- Next generation CABA

2. Learn to use CABA

- Objectives:
  - Be able to identify system requirements,
  - Be able to run the agent
  - Be able to monitor and control agents by using GUI and shell
- The outline:
  - System requirements
  - Getting started

2.1 CABA System Requirements

- CABA is compiled in Java
- System requirements
  - JRE 1.8 or newer version
  - prefuse alpha release for experience display http://prefuse.sourceforge.net
  - FreeTTS 1.2.1 - A speech synthesizer http://freetts.sourceforge.net
- I guess any machine that supports Java 1.8 can run CABA agents
  - You can reduce run time system requirements by running the agents without GUI display and with configuration that contains minimum components that is required

2.2 CABA Commands

- Get: get cycleSpeed
- Set: set cycleSpeed 1.0
- List/print: list all configurations
- Start: start the agent (all components)
- Stop: stop the agent (all components)
- Step: execute one step for the agent (all components)
- Hide: hide the gui
- Show: show the gui (what are set to show)
- Help: display all the commands

2.3 CABA Configuration

- agentName = test

- agentComponents = kbImpl processImpl domainImpl decisionImpl imImpl comImpl

- useGUI = true
- cycleSpeed = 1.0
- isStoped = false

- domainImpl = edu.psu.agentcomponent.ExampleDomainAdapter
- domainGUI = false

2.4 How to Run Agents

- Main class is edu.psu.agents.Agent
- java -jar CABA.jar [agent_config_files]
  - java -jar CABA.jar agent1.conf agent2.conf
  - java –cp r-casr.jar edu.psu.agents.Agent agent1.conf agent2.conf
  - java –cp r-casr.jar edu.psu.agents.AgentNet agent1.conf agent2.conf
- Steps:
  1. Check agent specifications
     - KB,
     - process
     - experiences
     - directory
  2. Check configrations
  3. Run the agents

2.5 Navigation Practice

1. Installation
2. Test run
3. Test GUI
   - Show different components
4. Test shell
   - Start/Stop/Step
   - Adjust speed

Active Knowledgebase

- Objectives:
  - To understand AKB functions
  - To learn AKB syntax
  - To be able to write simple AKB files
- The outline:
  - About AKB
  - The AKB system design
  - The AKB syntax
  - The AKB panel, commands, configurations
  - An implementation guide
  - Practice

3.1 About AKB

- The goal of AKB is explicitly represents information dependency relations (IDR).
  - At fact type level (schema/class)
  - At fact level (instance)
  - With visualization
- Inherited some idea, functions, syntax from
  - JARE
  - JESS
- Additional functions:
  - Volatile facts (degree of stickiness)
  - Information type sources for investigation planning
  - Information source for credibility evaluation (not implemented)
  - Simple natural language translation

3.2.1 AKB Syntax (Overview)

- Variables: ?position, ?color
- Symbols: car, “state college”
- Comments: Any line that starts with “;” or “#”
- Functions: +, -, *, /, rand, dis, =, eq, <, <=, >, >=
- Facts: any assertions
- Fact Types: about a group of facts
- Rules: follow first order predicate logic
- Queries: can only query what is defined as fact types or functions

3.2.2 AKB Syntax (Example)

```
(FactType rich (?person)
      (template "?person has a lot of money")
)

(FactType money (?person ?money)
      (template "?person has ?money dollars")
      (source (test plan_count))
)

(Rule "rich"
    (money ?person ?money)
    (>= ?money 100)
    ->
    (rich ?person)
)

(Fact money (Alice 201.0)
    (source (test plan_count))
)
```

Fact type

Rule

Fact

3.2.3 AKB Syntax (Fact Type)

- must be defined before you assert facts or fire rules that use it
- Not "case sensitive"
- Names for any two types should be different.
- The template is used for
  - parsing facts from natural language, or
  - converting facts into natural languages.
  - E.g. "Alice has a lot of money" can be asserted as (rich Alice).
  - One template per type.
- A time value represents the default time and can be changed for specific facts.
- Sources are used for an agent for planning investigation.

```
(FactType rich (?person)
  (template "?person has a lot of money")
  (source
    (agent1 plan_count_money)
    (agent2 plan_observe)
  )
  (time 20)
)
```

3.2.4 AKB Syntax (Fact)

- Facts are instances of Fact Types.
- Fact type must be defined before assertion.
- Should not contains variables.
- Simple version or natural language syntax can not assert facts with sources or time information.
- Expired (timeout since assertion) facts will be retracted.

1. Formal syntax

```
(Fact rich (Alice)
  (source
    (agent1 plan_count_money)
    (agent2 informed)
  )
  (time 50)
)
```

2. Simple syntax

```
Assert ((rich Alice))
```

3. Natural language

```
naturalAssert Alice has a lot of money
```

3.2.5 AKB Syntax (Rules)

- All predicate names must be defined as fact types except functions sun as +,-,*,/,=, etc.
- Rules and facts generates implied facts.
  - E.g. ((money Alice 1000)) -> ((rich Alice))

```
(Rule "rich"
; Each rule has a unique name.
  (money ?person ?money)
; Predicate names must be predefined
  (>= ?money 100)
->
; "->" separate antecedents and
;consequence part
  (rich ?person)
)
```

3.2.6 AKB Syntax (Implied Facts)

- As implied facts are implied, you can not assert an implied fact.
- You can assert a primitive fact (constant or volatile) with the same fact type as an implied one e.g. ((rich Bob))
- Asserting facts will trigger/fire rules for asserting implied facts.
- Retracting facts may trigger retracting of depended implied facts.
  - Therefore, even implied facts are not explicitly volatile, they will be retraced when the volatile evidences are expired
- Limitations:
  - Currently, asserting facts CANNOT trigger rules for retracting facts.
  - Currently, retracting facts CANNOT trigger rules for asserting facts.

3.2.7 AKB Syntax (Queries)

- Queries cannot be stored.
- Must be able to unify with predefined fact types.
- Return either as variable bindings or as unified facts.
- “Natural queries” can also have responses in natural languages.

1. Normal query
query ((rich ?person))
Response:
((rich Alice))
((rich bob))
2. Natural query
naturalQuery ((rich ?person))
Response:
Alice has a lot of money;
Bob has a lot of money;

3.2.8 AKB Syntax (Negation)

- Can be used in predicates in rules, or in queries.
- Negations are currently not allowed in the head of a rule. E.g.
  - ((not (threat low)) (enemy_found ture) is not allowed.
- May have bugs. (please report bugs about negation)

```
((attack_pattern xxx)
    (task_class ?task1 ?class1)
    (task_class ?task2 ?class2)
    (not (eq ?task1 ?task2))
)
```


DING+

3.2.9 AKB Syntax (Retract)

- Must be able to unify with defined fact types.
- Allows retraction of queries result.
- Retraction of a fact may result in retraction of depended facts.
- Volatile facts will be retracted when they expired.

1. Retract fact
Retract ((rich Alice))
2. Retract query results
Retract ((rich ?person))

3.3 AKB Commands

- query: query ((rich ?person))
- naturalQuery: naturalQuery ((rich ?person))
- assert: assert ((rich Alice))
- naturalAssert: naturalAssert Alice has a lot of money
- diagnose: diagnose ((rich Alice))
- printKB: printKB will list all fact types, rules, facts
- parseKB: parseKB will parse any text in AKB syntax
- retract: retract ((rich Alice)) or retract ((rich ?person))

3.4 AKB Configurations

- kbImpl = com.dingjust.caba.activeknowledgebase.AKB
- kbFile = test.kb
- kbClock = 1000
- kbGUI = true
- kbSpeakNatrualReply = true

3.5 How to Implement an AKB

1. Define fact types
2. Define rules
3. Assert primitive facts
4. Test your rules by asserting the facts, query implied facts
5. Avoid using recursive rules

4. Process Manager

- Objectives:
  - To understand functions of the process manager
  - To learn process specification syntax
  - To be able to write simple process files
- The outline:
  - About the process manager
  - The system design
  - The syntax
  - The PM panel, commands, configurations
  - An implementation guide
  - Practice

4.1 About the Process Manager

- Reduced complexities: agent bind, parallelism
- Increased robustness: multiple process instances
- Increased expressiveness:
  - termcondition, failcondition, contingency plan
  - Mental simulation
- Explicit representation and maintenance of process state

4.2.1 PM Syntax (Overview)

- A process contains
  - Header (conditions, effects, etc.)
  - Body (a sequence of steps)
- Each step can be either one of the followings:
  - Operator: will be execute
  - Plan: will be decomposed
  - Choice: will make a selection

4.2.2 PM Syntax (Example)

```
(operator count (?number)
   (precondition (next_number ?number ?next))
   (effect (not (current_number ?number))(current_number ?next)))
(plan plan_count_current
    (precondition (next_number ?number ?next))
    (termcondition (current_number ?next))
    (process
   (choice big_small
       ((prefcondition (> ?number 5))
                (print big))
       ((default) (print small)))
   (print ?number)
   (count ?number)
    ))
(plan plan_count_from_to(?from ?to)
    (termcondition (current_number ?to))
    (process
   (plan_count_current)))
```

Operator

Plan

Choice

4.2.3 PM Syntax (Operator)

- Preconditions are conjunctive.
- Process will “wait” if preconditions are not satisfied.
- Parameters will be bound before testing, but testing will not keep variable bindings.
- Must be defined before use in process specification.
- Needs to be defined in the domain interface:
  - public void action(String command, Vector args);
  - Print, speak are built-in operators for print/speak outs (values).

```
(operator count (?number)
  (precondition
        (next_number ?number ?next))
  (effect
        (not (current_number ?number))
        (current_number ?next))
)
```

4.2.4.1 PM Syntax (Plan)

- Preconditions and effects are similar to those of the operators.
- Testing of precondition in plans will keep variable bindings.
- Termination and fail conditions are disjunctive.
  - Termcondition: success or irrelevant
  - Failcondition: failure
- Process are all in sequential.
- No if condition, while loop, agent bind, or parallel process is available.
- A plan can be recursive.
- All steps in process must be defined either as (sub)plans or as operators.

```
(plan plan_count_from_to
        (?from ?to)
  (failcondition (< ?to ?from))
  (contingency (plan_countback))
  (termcondition    (current_number ?to))
  (process
        (plan_count_current)
  )
)
```


DING+

4.2.5 PM Syntax (Choice)

- Unlike plans or operators, choices are specified inside a process.
- Prefconditions are conjunctive.
- Prefconditions are not tested in the order that is specified.
- Default will be executed when all the prefconditions in the choices are not satisfied.
- A default action must be specified for a choice.
- Choices are not prioritized based on the order, but not random either.

```
(process
  (choice big_small
    ((prefcondition (> ?number 7))
          (print big))
    ((prefcondition (< ?number 3))
          (print small))
    ((default) (print normal))
  )
)
```

4.2.4.1 PM Syntax (Plan)

- Preconditions and effects are similar to those of the operators.
- Testing of precondition in plans will keep variable bindings.
- Termination and fail conditions are disjunctive.
  - Termcondition: success or irrelevant
  - Failcondition: failure
- Process are all in sequential.
- No if condition, while loop, agent bind, or parallel process is available.
- A plan can be recursive.
- All steps in process must be defined either as (sub)plans or as operators.

```
(plan plan_count_from_to
        (?from ?to)
   (failcondition (< ?to ?from))
   (contingency (plan_countback))
   (termcondition     (current_number ?to))
   (process
        (plan_count_current)
   )
)
```


DING+

4.3 PM Commands

- schedule:
  - schedule plan_name
  - schedule plan_name arguments…
- terminate
  - terminate processID (not plan_name)
- listProcesses
  - Will list all available processes.
- simulate
  - simulate plan_name arguments…

4.4 PM Configurations

- processImpl = com.dingjust.caba.process.ProcessManager
- processFile = test.process
- processInitialProcess = null
- processClock = 2000
- processGUI = true

- processTerminateIfEnd = false;
- processTerminateWithoutTermCond = true;
- processRemoveIfInactive = false;

- simulationNewKB = true;
- simulationRelaxPrecond = true;
- simulationTestPrecond = true;
- simulationDepth = 1;

4.5 How to Design Processes

1. Identify the operators.
2. Try to group plans in a tree structure.
3. Identify the preconditions for the operators and plans.
4. Identify the terminations of the plans (the most important and difficult step).
5. Identify the failconditions and contingency plan
6. Plan the variable bindings.
7. Test your plans from the basic ones to the complex ones.

推动全球C2B变革

Enable global C2B transition