Agent Language Analysis: 3-APL

Sergio Alvarez Napagao, Benjamin Auffarth, Norman Salazar Ramirez

3-APL (Triple APL) is an agent programming language and platform, created as a way of developing intelligent agents. The intelligence in the agents is provided by a complex mental state, which consists of different mental attitudes, and by a deliberation process that provides a transition mechanism between the mental states. Also the agents are capable of interacting with each other, directly by communication, or indirectly through the shared environment. The platform part of 3-APL provides the mechanism to deploy multiple agents at the same time as well as managing their communication.

3-APL stands for *Abstract Agent Programming Language* or *Artificial Autonomous Agents Programming Language* and was created in the Utrecht University as an academic tool.

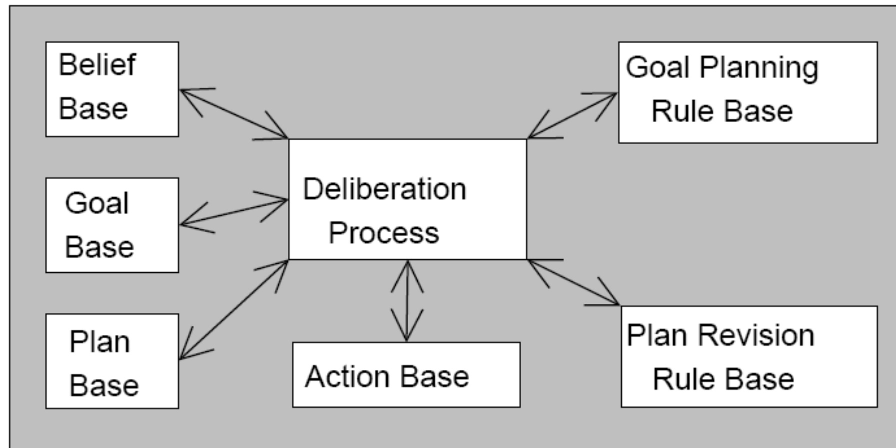
The Agent

The first component of an agent in 3-APL are its mental attitudes which are data structures representing beliefs, goals, plans, actions and practical reasoning rules. Thus we can say that an agent has a purpose (goals) which prompts it to act or interact in the environment; in order to achieve its purpose it has plans outlining the necessary actions required; it has rules that allow it to modify its plans in case its necessary; and it has beliefs about itself and its environment which are necessary in deciding the plan of action and to evaluate if its purpose has been accomplished. The relationship between the mental attitudes and the BDI logic can be seen in Table 1.

BDI Theory	3-APL
Beliefs	Beliefs
Desires	Goals(declarative)
Intentions	Plans (procedural)

Figure 1. BDI relationship

The second component is a deliberation process which uses the beliefs and the goals to decide which plan of action to follow, as well as deciding when the current plan needs to revised or dropped. Figure 1 shows the basic structure of a 3-APL agent.



Fifure 1. 3-APL agent architecture

Beliefs

An agent needs to have beliefs about itself and about its environment. A belief is represented by a prolog like formula, i.e. a subset of first-order predicate language. The set of all beliefs an agent has at a given point in time is called the *Belief Base*. The belief base is based on a closed world model, which means that if a belief is not on the base then it is false.

In the 3-APL language the initial belief base if preceded by the keyword **BeliefBase:**, an example of this can be seen in figure 2.

```
BELIEFBASE:
pos(room4),box(room2),delpos(room1),gain(5),cost(0),
forbid(room4, room2),door(room1, room2),door(room1, room3),
door(room2, room4),door(room3, room4),
door(R1,R2):-(R2,R1),forbid(R1, R2):-forbid(R2,R1)
```

Figure 2. Belief Base

The beliefs *pos(room4)* and *box(room2)* indicate to the agent that currently he believes he is on room 4 and that there is a box in room 2. After moving to room 2 and picking up the box, which could be part of a plan, these beliefs are removed and new beliefs are added, such as pos(room2) and box(self). Note that the belief base can also contain rules like door(*R*1,*R*2):-(*R*2,*R*1) which can be helpful in reducing the number of beliefs explicitly stated.

Basic Actions

A basic action in an agent represents the capabilities it has to modify its mental state or its environment. Basic actions are the only 3-APL construct capable of modifying

the belief base. There three types of basic actions: mental actions, communication actions and external actions.

The main purpose of a mental action is to update the belief base. These actions are specified in the terms of pre-conditions and post-conditions, which are defined as belief base queries. If an agent calls for the execution of a mental action, the agent will first check if the pre-condition holds, and in case it does it will update the belief base in such way that the post condition holds. If we take the mental action *{pos(R1),cost(X),not forbid(R1, R2)} Go(R1, R2) {not pos(R1),pos(R2),cost(X+1)}* from figure 3 based on the belief base seen on figure 2, we call for the action Go(room4,room3) then the pre-condition will check the belief base to see if the current position is room4 and if it is forbidden to move between rooms 4 and 3; since this conditions hold then the belief base will be updated with the post conditions making the new position room 3. It should be noted that the pre-condition *Cost(X)* has the purpose of instancing the variable *X* to the current cost in the belief base, and this value is still available in the post condition.

In the 3-APL language the set of mental actions is preceded by the keyword **Capabilities:.**

```
CAPABILITIES:
{pos(R1),cost(X),not forbid(R1, R2)}
Go(R1, R2)
{not pos(R1),pos(R2),cost(X+1)},
{pos(R1),cost(X),forbid(R1,R2)}
Go(R1, R2)
{not pos(R1),pos(R2),cost(X+5)}
```

Figure 3. Agent Capabilities

A communication action sends a message to another agent or to the platform. A message contains the name of the receiver of the message, the speech act or performative (e.g. inform, request, etc.) of the message, and the content, where the content comes in the form of a predicate. The 3-APL language instruction to achieve this is the *Send(receiver,performative,content)* instruction.

The last type of action is called external action, and as its name says, it has the purpose of executing actions external to the agent, i.e. the environment. The exact cause the action will have in the environment may not be known to the agent and the only way to know its effects may be through the execution of another external action which could be appropriately called *sense action*. We can also always assume that we know the exact result an external action and update the belief base accordingly.

In 3-APL external actions are the methods of the Java class that represents the environment (i.e., the methods specify the effect of those actions in that environment). In the language external actions are called by the *Java("Classname",method,List)* instruction, where Classname is the name of the class representing the environment, method is the name of a method of the class, and List is a variable well the return values of the method will be saved. The method can be implemented to return the result of the

action in the list, or the list could for example be empty. In that case, an explicit sense action would have to be executed to obtain the result of the action.

Goals

Goals represent the state of affaires desired by the agents; they are agent's motivation and purpose. Seen from the BDI point of view goals represent the desires. In 3-APL goals are represented by a conjunction of facts, these facts represent the beliefs we would like to achieve. A goal driven agent will continue working until his desired belief forms part of its belief base. The set of goals an agent has is called *goal base*. The initial goal base of agent is preceded by the keyword **GoalBase:**, as seen in figure 4.

```
GOALBASE:
transportBox()
```

Figure 4. Goal Base

Plans

In order to achieve its goals an agent has a mental attitude called plan. In BDI logic plans represent the intentions. Plans can be of three types: basic, abstract and composite.

Basic plans can have the form of a basic action (mental, communication or external); a belief base query to determine if a belief is true or false, which also has the function of binding values to the variables in the plan; a *AdoptGoal* instructions which can adds a new goal to the goal base; a *DropGoal* instruction which drops a goal; and a *SKIP* instruction which basically does not do anything.

An *abstract plan* is an abstract representation of a plan which can be instantiated with a (more concrete) plan during execution.

A c*omposite plan* is a plan that is formed by other plans (basic, abstract or composite). They can be of the *sequential* type, which is a set of plans executed in sequential order; or of the *conditional* type which present and if-then-else choice, where the condition is a belief query and it chooses between to plans. The last type is the *iterative* type, in which a plan will be executed iteratively while a belief holds true; this is done using a while-do construct.

The specification of the initial plan base in 3-APL is preceded by the keyword **PlanBase:**.

```
PLANBASE:
start(),
while not pos(0,0) do
      pos(X,Y)?;
      if X=0 then
            Goto(X,Y-1)
      else
```

```
                if Y=0 then
                      Goto(X-1,Y)
                else
                      Goto(X-1,Y-1)
od
```

Figure 5. Plan Base

The plan base in figure 5 contains two plans: an abstract plan called start() and an iterative plan formed from sequential and conditional sub-plans.

Reasoning Rules

Reasoning rules provide the means-end reasoning component of the agent. They give the agent the capability of constructing or revising plans. From the belief that a plan is sufficient to achieve a desired goal the agent concludes it should adopt the goal.

Reasoning rules are divided into four classes: reactive rules, which are used not only to respond to the current situation but also to create new goals; plan-rules, which are used to find plans to achieve goals; failure-rules, which are used to re-plan when plans fail; and optimization-rules, which can replace less effective plans with more optimal plans.

The basic structure of a reasoning rule consists of a head which can be either a goal or a plan, a body which is a plan and a guard which is a belief query.

The 3-APL implementation divides the reasoning rules in two types depending on their purpose, these types are *goal planning rules* and *plan revision rules*.

Goal Planning Rules

As the name says the purpose of these rules is to create a plan capable of satisfying a desired goal. The structure of a goal planning rule is the same as that of a practical reasoning one, except the head of the rule is always a goal (which can be empty). Informally a goal planning rule states that if we want to achieve the goal in the head and we find ourselves in a situation matching the guard then we should implement the plan in the body.

There can also be rules with an empty goal, which represent rules that will generate a plan the moment the agent finds itself in a situation matching the guard, in other words they are rules that react to situations, thus they are rules of the reactive.

In 3-APL language the rules are preceded by the **Planning-Rules:** keyword. Figure 6 shows an example of a rule, which will be fired if the goal transportBox() is in our goal base and if the belief base contains belief matching the queries in the guard. It should be noted that the variables in the guard will take the values of existing beliefs in the belief base, so whenever a variable appears in the body of the rule it will be replaced by these values.

```
PLANNING-RULES
transportBox()<- pos(R1),box(R2),delpos(R3) |
goxy(R1,R2);GetBox();goxy(R2, R3);PutBox()
```

Figure 6. Goal Planning rules

Plan Revision Rules

These rules are used to adopt, revise or drop plans and so in terms of the classes specified before they can be of the failure (re-plan) class or of the optimization class. They have the structure of a practical reasoning rule but the head is always a rule. Informally for the failure case the rule means that if we are executing the plan in the head but we cannot continue because the condition in the guard holds true then we should drop the current plan and adopt the plan in the body of the rule. For the optimization case we have that if we are currently using the plan in the head but this plan is not so efficient in the current situation (specified by the guard) we should consider replacing it with the plan in the body.

In the implantation of the 3-APL the rules are preceded by the keyword **PR-Rules:**. A simple example can be seen in the second rule in figure 7, where if we are executing the plan of going from room r1 to room r2 but both room are the same then we should instead do nothing and skip to the next plan in the plan base. It should be noted that the variables of the head will take values of a matching plan in the plan base and that this values will hold for the guard and the body.

```
PR-RULES:
goxy(R1,R2) <- pos(R1),door(R1,R3),not R1 = R2 |
Go(R1, R3);goxy(R3,R2),
goxy(R1,R2) <- R1 = R2 | SKIP.
```

Figure 7. Plan revision rules

Deliberation Process

The deliberation process of the 3-APL agents is formed by a series of deliberation operations, such as executing a rule, selecting a plan, etc. In other words the purpose of this process is to modify the mental attitudes of the agent until it reaches its goals or completes its plans. This process or program can also be seen as an interpreter that determines the order in which the operations are performed. For example it can be programmed to drop unachievable goals; or it can check whether a goal still exists during a plan execution, to avoid continuing plan which goals has already been achieved or dropped; it can also work as a garbage collectors to remove leftovers of plans no longer existent. The use of two parallel plans could constitute a more complicated use. The interpreter should decide if two plans for the same goal can be maintained at the same time or if two goal with is own individual plans can be executed concurrently and at the same time resolve any conflict of interests that may come with the parallelism. More information about different uses can be found in [1].

The creators of 3-APL have propose the implementation of the interpreter as a meta-level program that can be customized for the needs of the user. They proposed a set of deliberations operations to work as basic operations (*SelectPGrule*, *SelectPlanRevisionrule*, *SelectPlan*, and *ExecutePlan*) that can be combined with sequential composition, conditional choices, tests (of beliefs, plans and goals) and iterative loops to create more complex programs. The implementation of this meta-level is briefly discussed in [1], and a more formal proposal is discussed in [3].

Even though the idea of a meta-level deliberation programming appears to be a very good idea and is discussed at great length by the 3-APL creators, at the moment this does not form part of the current implementation. In its place a static deliberation process is provided that implements a cyclic order of the deliberation operations. An illustration of the cycle can be seen in figure 8. Since this cycle is static it is by no mean applicable to every situation, and so the agent behavior may give unexpected results. A way to overcome this or at least to some extent, is modeling the mental attitudes of our agents based on this cycle in such a way that we can steer them to our desired result.

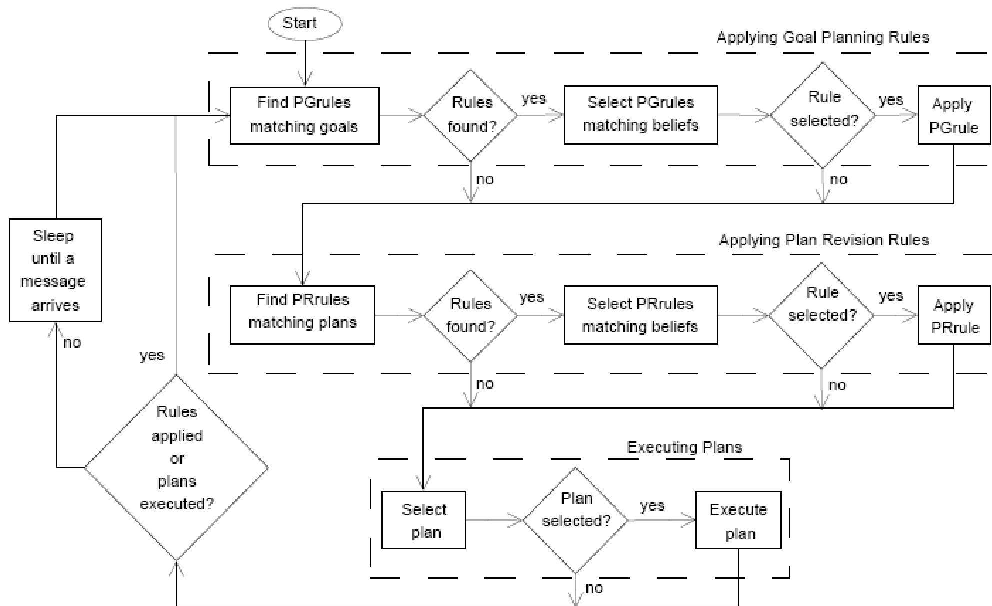


Figure 8. 3-APL deliberation cycle

The steps of the cycle are the following:

1. Find Plan Generation Rules that Match Goals
2. Remove Plan Generation Rules with atoms in head that exist in Belief Base
3. Find Plan Generation (PG) Rules that Match Beliefs
4. Select a Plan Generation (PG) Rule to Apply
5. Apply the Plan Generation (PG) Rule, thus adding a plan to the planbase
6. Find Plan Revision (PR) Rules that Match Plans

7. Find Plan Revision (PR) Rules that Match Beliefs
8. Select a Plan Revision (PR) Rule to Apply to a Plan
9. Apply the Plan Revision (PR) Rule to the Plan
10. Find Plans To Execute
11. Select a Plan To Execute
12. Execute the (first basic action of the) Plan

It can be seen clearly that this loop consists of three parts. The first part is the deliberation about the goals (steps 1-5). The second part consists on the revision of the current plans and thus provides the adaptation component (rules 6-9). Finally the last part consists on the execution of the selected goals (steps 11-12). This cycle has the advantage that it provides a somewhat reactive element, since it executes one plan action at the time; however it has the disadvantage that no "real" long term plan is created. The advantage of a real planning stage is that plans can be developed and evaluated and subsequent backtracking over the plans can take place when the plan was not satisfactory. The 3-APL cycle does not provide means for backtracking, once a rule has been applied the current plan changes and the agent has to move from the newly created plan. One way to provide some level of backtracking is to increase the complexity and the number of the agent rules but this can make things quite messy.

Communication

As we saw before, agents can talk with each other using communication actions, and this action is executed using the instruction *Send(receiver,performative,content).* If an agent sends a message to another agent the belief base of both agents is updated, the sender is updated by the formula *sent(Receiver, Performative, Content)*, while the receiver gets the *received(Sender,Performative, Content)*.

The first thing we can notice about the communication level in 3-APL is that it does not provide means for processing the message content, so the user has to create action and rules to be able to extract the content. Also it does not provide any method to determine the trust level of the agents in the environment so again is up to the user to define rules for this. Another interesting point is that even though the message contains a performative element, 3-APL does not have a definition for performatives so as before the user has to define rules for this (there is one exception in which the environment can inform the platform of its capabilities and other agents can ask the platform about this). This lack of a communication protocol makes 3-APL more suited for applications where agent cooperation and negotiation is not necessary. It should be noted that ongoing research is concerned with communication, e.g. the one described in [8].

With all these shortcomings and apparent lack of complexity the communication structure is still FIPA compliant and so 3-APL agent can communicate with agents in other FIPA compliant platforms such as JADE(X), however, as 3-APL does not support ontologies, messages between platforms have to be kept as simple as possible in order to prevent misunderstandings.

Formal Semantics

Formal semantics for 3-APL have been defined in Plotkin's operation semantics [1][6] and in Z [7]. These semantics specify the transitions between the agent's configurations by means of transition rules.

When comparing the formal semantics with its implementation we can see that the implementation follows heavily the formal semantics, that the agent configuration corresponds to its mental state and that the transitions correspond to applying the reasoning rules.

MAS Developing Methodologies

Some work has been done in using multi-agent systems developing methodologies with 3-APL. Table 2 shows the relationship between the elements of the GAIA methodology and 3-APL. As you can see, it does not seem to be an ideal match, especially in the Role Models part. On the other hand table 3 shows the relationship with the Prometheus method and in this case the match looks more convincing..

Gaia	3APL
The Environment Model	Java Class
The Roles Model	beliefs, goals, plans and actions
– Responsibility	– goals and plans
– Permission	– beliefs and actions
The Interaction Model	communication and external actions
Organizational structure	all components
– Topology structure	– communication structure, beliefs, goals, plans and actions
– Control structure	– algorithms and reasoning rules

Table 2. GAIA relationship

Prometheus	3APL
Plan Descriptions:	
• Triggering event	Guards of reactive rules
• Plan steps	• Plan expressions
• Context of performing plans	• Guards of rules, test actions, action pre-conditions
• Data used/produced	• Beliefs, Java data, action post-condition, communication
Event Descriptions:	
• Event purpose	• Reasoning rules
• Data carried by event	• Substitutions in reasoning rules
Data Descriptions:	
• Data structures	• Terms, atoms, rules, Java data
• Methods manipulating data	• Actions

Table 3. Prometheus relationship

3-APL Platform

3-APL is not only the language but the platform that allows programming, deployment and execution of the agents. The 3-APL platform is also in charge of the transportation of the communication messages; and also provides information about the existing agents to other agents through the agent management system (AMS). Moreover, once the agents are running it provides monitoring tools, such as a sniffer, for the message exchange and specific windows to monitor the mental states of the agents. It also provides means for step by step monitoring. A more detailed description can be found in [4]. Figure 9 illustrates the agent platform architecture.

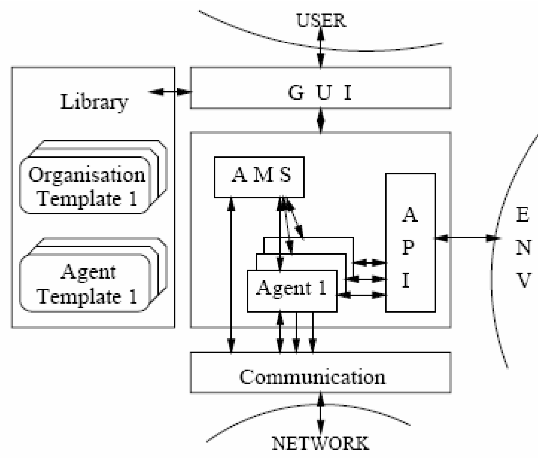


Figure 9. 3-APL Platform

The agent environment forms part of the 3-APL platform and it comes in form of a programmable Java class, which as we stated provides the available external actions of the agents. In particular, the environment is modeled as plugin to the platform. This is a systematic way to interface between the 3APL platform and Java classes. The plugin facilitates the interaction between individual agents running on the platform and the

instantiation of the Java classes. For the user interaction 3-APL provides a series of classes with graphical I/O purposes. Figure 10 shows an example of the graphical output interface.

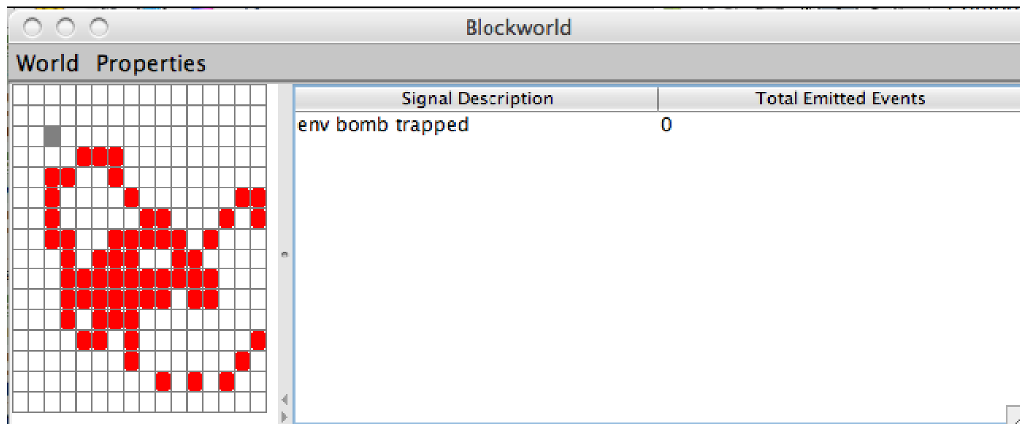


Figure 10. 3-APL graphical I/O example

On the specification side 3-APL requires Java 1.5 to run, and it is it is easy to install by downloading and executing an 850kb .jar file. During deployment time it can be set as it can be set as a server to host multiple agents or as a client.

3-APL Tools

In the last section we saw that the 3-APL platform is the main existing tool that implements 3-APL (figure 11 shows a screenshot of it). However another important version exists, called 3-APL-M, which is for use in mobile equipments and can run in any Java enabled mobile device. Since some devices have limited space capabilities it is possible to divide the processing in a server module that runs in J2SE and in a mobile module that runs in J2ME. 3-APL has found use in robots to some success [9].

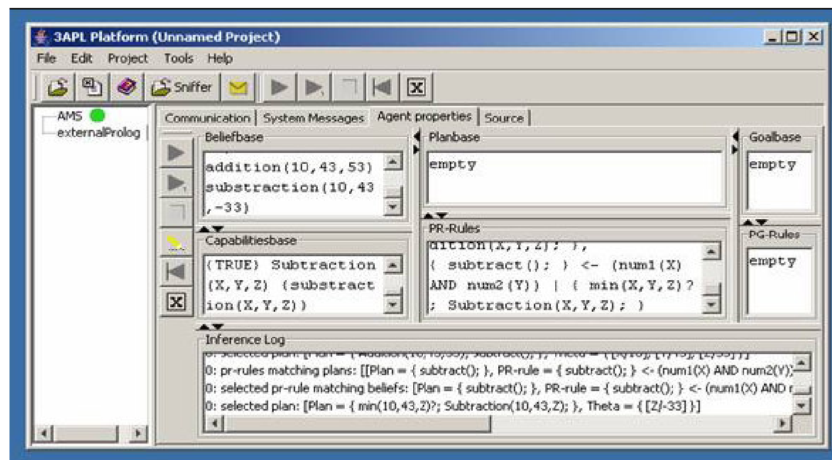


Figure 11. 3-APL platform GUI

Comparisons

Table 4 shows a comparison between different agent languages. The first column shows the programming languages in which the agent languages have been implemented. The second column shows whether the language has a model of formal semantics. The third column refers to industrial applications. It can be seen that 3-APL has not found any industrial strength application, yet.

	Implementations	Formal Semantics	Industrial-strength applic.
PRS	UMPRS, PRS-CL, others	No	Yes
dMARS	In 1995, AAII implemented a C++ platform running on Unix; in 1997 dMARS was ported to Windows/NT	Operational	Yes
JACK	Java	No	Unmanned vehicle
JAM	Java	No	No
Jadex	Java	Operational	Yes
AS(L)	SIM Speak, AgentTalk, Jason	Operational	Virtual environments
3APL	Java and Prolog	Operational; meta-level	No
Dribble	No	Operational, dynamic logic-based	No
Coo-BDI	Coo-AgentSpeak	Operational	No

Table 4. Comparison between agent languages 1.

Table 5 extends the above comparison to basic components, the operation cycle, ontologies, and to whether the agent languages realize dynamical resolution. 3-APL is the only language that has practical reasoning rules which can be a very powerful tool.

	Basic components	Operation cycle	Ont	Dyn
PRS	Standard	Standard	No	No
dMARS	Standard	Standard	No	No
JACK	Standard + capabilities (that aggregate functional components) + views (to easily model data)	Standard	No	No
JAM	Standard + observer (user-specified declarative procedure that the agent interleaves between plan steps) + utility of plans	Utility-based	No	Yes
Jadex	Beliefs + goals + plans + capabilities (that aggregate functional components)	Standard	Yes	No
AS(L)	Standard	Standard; efficient	Yes	Yes
3APL	Beliefs, plans, practical reasoning rules, basic action specifications	Think-act	No	Yes

Dribble	Beliefs, plans, declarative goals, practical reasoning rules, goal rules, basic action specifications	Think-act	No	Yes
Coo-BDI	Standard + cooperation strategy (trusted agents + plan retrieval and acquisition policies) + plans' access specifiers	Perceive-cooperate-act	No	Yes

Table 5. Comparison between agent languages 2.

Conclusions

We think it became apparent that 3-APL is a powerful tool for the creation of intelligent agents, especially its practical reasoning rules provide ways of creating very complex mechanisms. However the use of a static deliberation cycles limits its functionality and forces the user to explode the number of rules to balance this out, which complicates the task of agent implementation.

The lack of a strong communication protocol also is a further point which lacks in the language, complicating the task of creating coordination and cooperation between agents.

We also find lacking the absence of agent actions dedicated to computer processing; right now if an agent requires an action not related with changing its mental state it has to be done through the environment. For example it would be nice to have agents in which part of its executing plan will consist on running some classification algorithm or some numerical method which is only owned by the agent and so it could be encapsulated in its program.

3-APL is a comparatively new model (first publications date back to 1998). It has been noted that there are no industrial-strength applications yet, which could boost its development.

References

[1] Dastani, M.M., Riemsdijk, M.B. van, & Meyer, J-J.Ch. (2005). Programming *Multi-Agents Systems in 3APL*. In R. H. Bordini, M. Dastani, J. Dix, & A El Fallah Seghrouchni (Eds.), *Multi-Agent Programming (Languages, Platforms and Applications)* (pp. 39-67). New York: Springer Science.

[2] Mehdi Dastani, 3APL: A Programming Language for Multi-agent Systems (Syntax), http://www.cs.uu.nl/docs/vakken/map/slides/3apl-Syntax.pdf, accessed November 18, 2006

[3] M. Dastani, F. de Boer, F. Dignum, J.J. Meyer, *Programming Agent Deliberation: An Approach Illustrated Using the 3APL Language*. Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), Melbourne, July 2003, ACM Press, 2003.

[4] Mehdi Dastani, 3APL Platform: User Guide, 19th January 2006, www.cs.uu.nl/3apl/download/java/userguide.pdf; accessed November 18, 2006. BNF specification of 3APL programming language.

.

[5] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J-J. Ch. Meyer. *Formal Semantics for an Abstract Agent Programming Language*. In Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages, Lecture Notes in Artificial Intelligence 1365, pages 215–229. Springer-Verlag, 1998

[6] J.J. Meyer, (Cognitive) Agents Agent metaphor, http://www.siks.nl/act/jj-siksday-2005.pdf, accessed

November 18, 2006.
[7] M. d'Inverno, K. V. Hindriks, and M. Luck, *A formal architecture for the 3APL agent programming language*, in Proc. of ZB'00, 2000, pp. 168–187.
[8] J. van der Ham, *Extending 3APL with Communication.* Master Thesis Cognitive Artificial Intelligence, Utrecht University.
[9] Verbeek Marko, *3APL as Programming Language for Cognitive Robots*. Master Thesis, Utrecht University.