

User Guide of NorJADE Framework

NorJADE is an open source framework to develop normative multi-agent systems using the very popular **JADE** platform and well known technologies (like **Aspect-Oriented Programming** and **Ontologies**). Hence, we will present the ontology used to create **NorJADE** framework, then we will present the aspects used to enhance **JADE** platform with the normative aspects.

1. NorJADE ontology

Obviously, using norms requires representing them. Hence, we choose ontology to represent the norms. For this reason, we developed an ontology that includes all the important concepts to represent different types of norms. Figure 1 represents the concepts used to specify the norms in NorJADE framework (like: agent, behaviours, deontic operators, regulation mechanism...etc.). Naturally, these concepts are interconnected using several relations (Figure 2). For example, a law is **specified by** a deontic operator (***Obligation***, ***Recommendation*** or ***Prohibition***), **controls** behaviour, **has validity**, **applied by** an enforcer, **implemented by** a regulation mechanism (Regimentation or Enforcement) and **has a consequence** (Reward or Punishment).

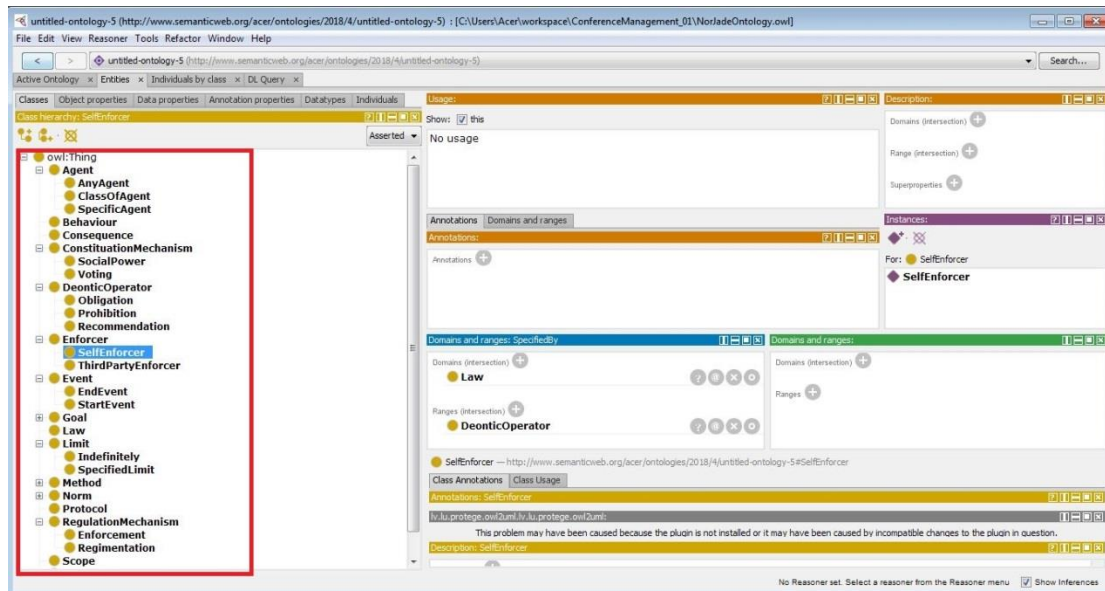


Figure 01: Concepts used to specify norms in NorJADE.

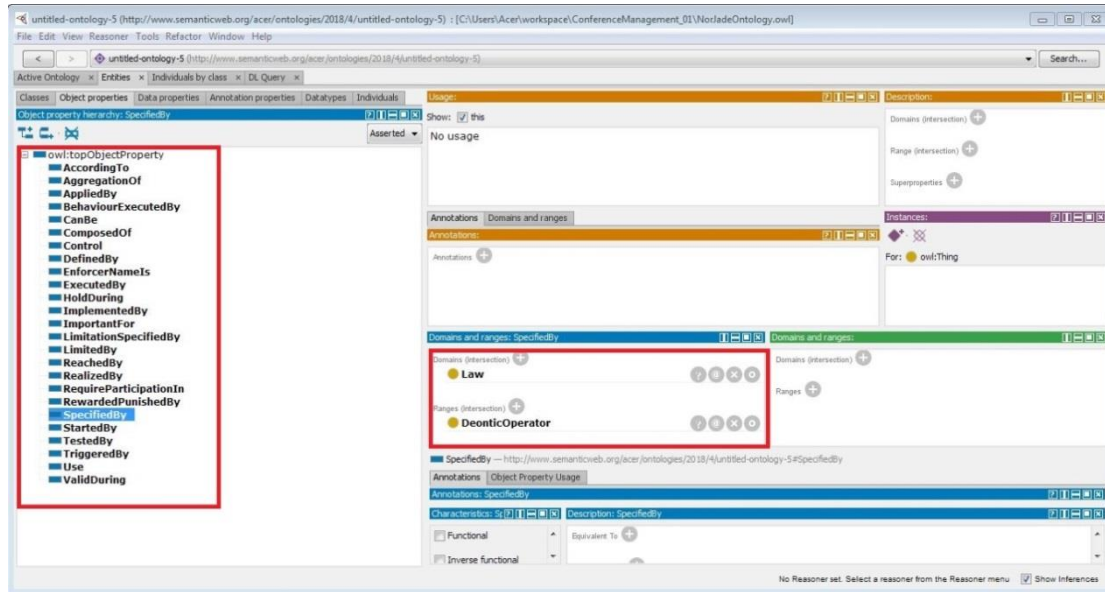


Figure 02: Relations between concepts in NorJADE ontology.

In order to use our framework, you have to create individuals from the existence classes to represent the norms that control your system. For example, to develop a conference management system (code available online), you must create laws that control agents in such system like *Dispatching Papers' law*, *Notification Law*, *Redaction Paper Law* and *Submitting Paper Law*.

Figure 03 presents *DispatchingPapersLaw*. This law used the *obligation deontic* operator to control the *DispatchingPaper* behaviour (That means it is mandatory to the conference chairman to execute the *DispatchPaper* behaviour during the validity time of this law). The *validity* of this law is specified by the *DispatchingValidity* which *started by* the deadline and limited by a *scope* (10 time unit). If the agent did not execute this behaviour during the validity period of the law, a *punishment* will be executed (it is consisted in this case of decreasing the credibility of the conference). This punishment is executed by the *enforcement mechanism*. This law is applied by *self-enforcement* because the conference chairman will decrease its credibility by himself.

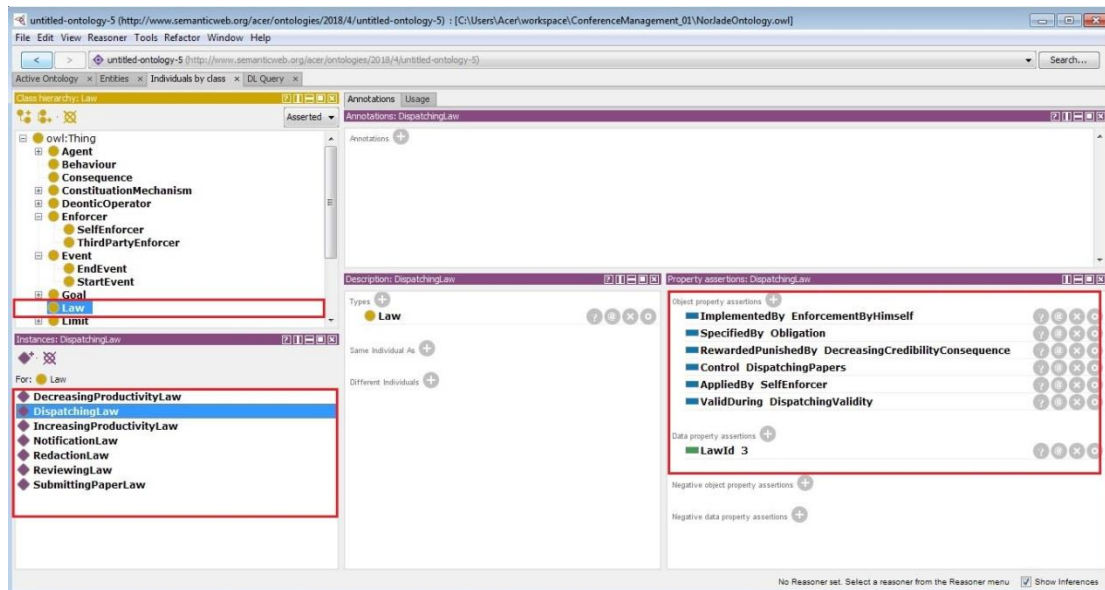


Figure 03: The dispatching paper law represented in NorJADE ontology.

Figure 04 represents another law (the `SubmittingPaperLaw`). According to this law, `SubmittingPaper` behaviour () is *prohibited* (specified by **Prohibition**) during the validity time (`ValidDuring SubmittingValidity`) which *started by* dispatching paper and it is *indefinitely* (Naturally, it is not allowed sending papers after deadline and dispatching papers). However, in this case the regulation mechanism is *regimentation*. Consequently, our framework does not allow the execution of submitting paper behaviour even if the agent attempted this action.

2. NorJADE Aspects

In order to use our framework, the users should use some aspects developed to enhance the **JADE** platform with the normative manipulation capabilities. These aspects are regrouped mainly in two packages: `NorJADEAspects` and `NorJADEontology`. Because **NorJADE** is developed as a framework, the users can reuse all the source code with just negligible changes. Exactly, users should only update `InterceptEvents` aspect to specify the pertinent events according to their application.

`NorJADEontology` package contains only one class (called: `NorJADEontologyBase`) used to manipulate the ontology in order to extract the different information about norms. So, this class is composed of some **SPARQL** requests that are used to manipulate the **NorJADE Ontology** in order to collect information about the specified norms (for example the methods: `getLaw`, `getLawProperties`, `getEnforcer`, `getEventproperties`...etc.). These methods are used later by the different aspects to check the compliance of the executed behaviours with norms and to execute the consequences of these norms.

`NorJADEAspects` package contains several aspects to manipulate norms. First of all, the aspect `ExtendingAgent` is used to enhance the agent with the different functions and structures required to manipulate norms thanks to introduction mechanism. As example of structures used to enhance the standard JADE agent, we specify tow vectors to save all the executed behaviours and intercepted events. Saving the executed behaviours allows checking if an obligate behaviour is executed or not. The intercepted events allow checking the validity of a law. Moreover, we enhanced the standard **JADE** agent with two methods that will execute the consequence of a given law. The first method is applied when the enforcer is self-enforcer (`SelfEnforcementApplication`) and the second method is applied in the case of the third party enforcer (`ThirdPartyEnforcement`).

The second aspect is `ExtendingBehaviour`. It is used to extend **JADE** behaviours with some required extensions to abort behaviours according to some specific norms (case of regimentation mechanism where an agent must be prevented to execute a behaviour).

The third aspect (called `InterceptBehaviour`) is used to intercept the execution of a behaviour (the method `action()`) and process this behaviour according to the specified norms. Hence, when we intercept the execution of a behaviour, we should send a request to the ontology with the name of the behaviour to verify if this behaviour is regulated by a norm or not. If the behaviour is regulated by a norm, then we should execute the adequate process (for example, if the behaviour is prohibited with regimentation mechanism, then we should prevent its execution; but if the behaviour is prohibited with the enforcement mechanism, then we should allow its execution with the execution of a punishment).

The aspect `InterceptEvents` is used to intercept the pertinent events related to the norms. Events are used mainly to specify the validity of norms. Hence, when we intercept an event we will process it according to its type. If the intercepted event is a start one, we should save it. On the other hand, if the intercepted event is an end one, we will check if the associated obligate behaviour is executed or not.

The aspect `ListofAgents` is used to save references to all agents created in the project. These references allow manipulating agents mostly when we must execute self-enforcement. In this case the agent is processed as a simple object because it will be obligate to execute the punishment (i.e the agent will lose its autonomy).

The aspect `LoadNorJADEontology` is used only to load the `NorJADEontology` when the main method is executed.

In the aspect `ProceduralNorm`, we extend the standard JADE agent with the method `ReachGoal` that allows determining the required behaviour to reach this goal.

The package `RewardPunishment` is used to implement various punishment or reward.

3. Details about packages, aspects, classes and methods

Package	Class / Aspect	Method / structure	description
NorJADEAspects	Event		Class used to define an event and it includes some methods to manipulate an event
	ExtendingAgent	ExecutedBehaviours	A vector to save the executed behaviours by an agent
		InterceptedEvent	A vector used to save the intercepted events
		SelfEnforcementApplication	This method is used to apply a self punishment/reward
		ThirdPartyEnforcement	This method is used to apply a third party punishment /reward
		IsExecutedBehaviour	This method is used to verify either a behaviour is executed or not
		IsConcernedByLaw	This method verify if the agent is concerned by a law or not
		IsValidLaw	This method verify either the law is valid or not
	ExtendingBehaviour		This aspect is used to abort a behaviour that is regulated by a regimentation mechanism
	InterceptBehaviour		This aspect is used to intercept behaviour execution and process it according to the regulation mechanism and the deontic operator
	InterceptEvents		This aspect is used to intercept events and process them according to their type (start or end type).
	ListOfAgents	Agents	A vector that is used to save references about agents in order to manipulate them (for example, when

			we need the execution of a method as punishment reward)
	LoadNorJadeOntology		An aspect used to load the NorJadeOntology when main function is executed
	ProceduralNorm		This aspect is used to define procedural norms
NorJADEOntology	NorJadeOntologyBase	getLaw	Return a law associated to a behaviour
		getLawProperties	Return the properties of a law
		getEnforcer	Return the enforcer
		getAgentClass	Return the class of an agent
		getEventProperties	Return the properties of an event
		getBehaviour	Return a behaviour associated to a goal
		getBehaviourName	Return the name of a behaviour
		getEventId	Return the identifier of an event
		getLawValidity	Return the validity of a law
		getEvent	Return an event
RewardPunishment	Behaviours to define by users		Describe reward or punishment to execute in the case of norms violation

4. How can we use the framework?

In order to use our framework, you have to :

1. Create an instance of **NorJADEOntology** that reflects the norms of the developed software. For this reason, you have to create individuals for all the classes defined in **NorJADEOntology**.
2. Put the developed ontology in your project directory.
3. Use all the aspects and classes defined in the packages **NorJADEOntology** and **NorJADEAspects** (except **InterceptEvents** aspect) without changing.
4. Update the aspect **InterceptEvents** by specifying the pertinent events for the developed system.
5. Create in the package **RewardPunishment** the behaviours that represent the punishment or reward.
6. Update (if necessary) the code of the application by procedural and constitutive norms.

5. Examples

a. Conference Management System

Conference management system is a famous example that is used to validate many aspects of multi-agent organization. We developed a simulation of this example using NorJADE in order to validate some features of this framework. In order to develop this example, we followed steps presented in the previous section.

- Creating an instance of NorJADE ontology

Conferenced management system is governed by several norms that represent academic laws and traditions. For example, the submission of papers to a conference is recommended, submission a paper to a conference after the dispatching date is prohibited (for example by closing the submission system), it is obligate to notify authors just after the notification date and the reviewers must send their evaluation before the notification date. Hence, we represented these laws using NorJADE ontology. Figure 4 shows the different laws that govern the conference management system. In this figure, we can show different laws like DispatchingLaw, NotificationLaw, RedactionLaw, ReviewingLaw and SubmittingLaw. SubmittingLaw controls SubmittingPaper behaviour by a regimentation mechanism. This means that the execution of this behaviour will be prohibited during the validity time of this law.

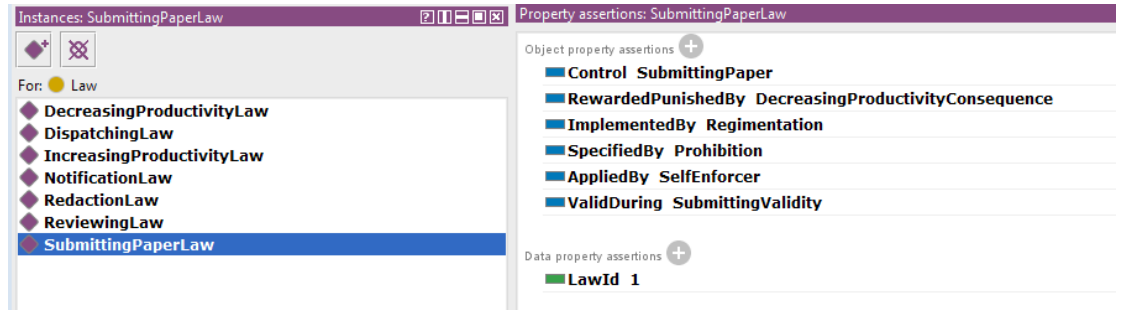


Figure 4: the different laws that govern the conference management system.

Figure 5 shows another law that control reviewing process. Hence, ReviewingLaw controls SendEvaluationResult by the enforcement mechanism. It is mandatory to execute this behaviour during the validity time of the law. Otherwise, this agent will be punished by the sanction WarningSendingConsequence. This punishment is executed by the enforcer Enforcer1.

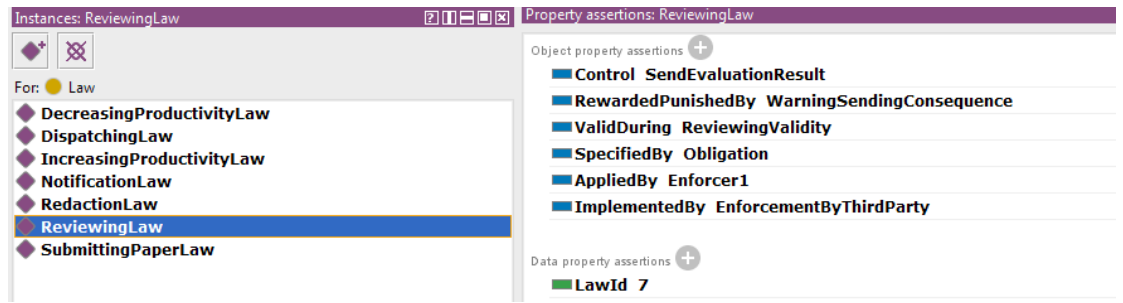


Figure 5: The law reviewingLaw.

Of course, representing this laws using NorJADE ontology, require representation of all the concepts related to these laws like behaviours, agents and consequences. For example, figure 6 presents the different behaviours executed by the agents of this system. In this figure, we can show that SendEvaluationResult is a behaviour that will be executed by Reviewer agents.

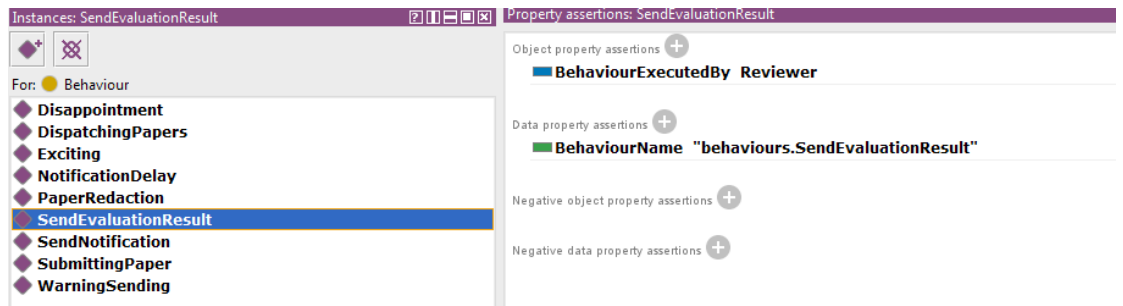


Figure 6: The behaviours of the conference management system..

- Reusing all the aspects of NorJADE aspect and updating the class `InterceptEvents`

After formulating the laws as an ontology, we have to reuse all the aspects defined in the package NorJADEaspects. Almost of these aspects should be used without any update like it is presented in figure 7. However, we need to customize the aspect `InterceptEvents` in order to intercept the relevant events of this system. These events are mainly used to control the validity of the different norms. In our system, the events represent just updating of some variable in the system. These variables represent the role of flags to show the state of the conference (deadline, dispatching, notification date...etc). Figure 8 represents this aspect.

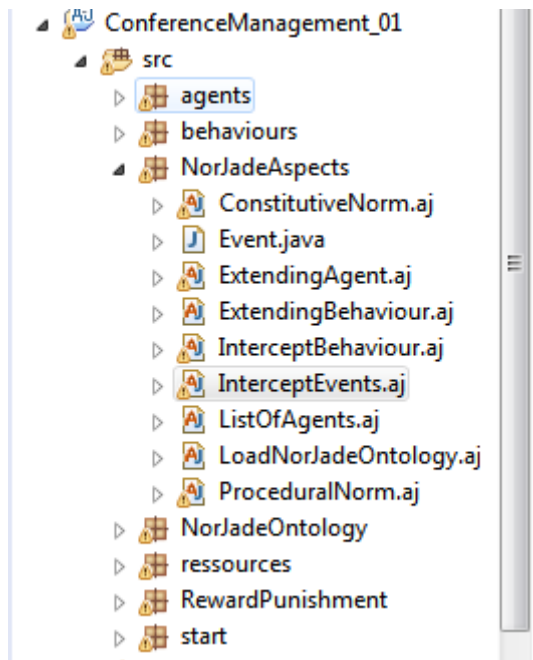


Figure 7: the aspects of NorJADEAspects.

```

8
9 public aspect InterceptEvents {
10
11     private pointcut VariableUpdating(boolean arg) : ( set(boolean ressources.Conference.DeadlineReached ) ||
12                                                         set(boolean ressources.Conference.NotificationReached) ||
13                                                         set(boolean ressources.Conference.DispatchingReached) ||
14                                                         set(boolean ressources.Conference.ReviewingReached) )
15     && args(arg);

```

Figure 8: the aspect InterceptEvents.

- Creating the punishment/reward

Normative multi-agent systems are based on mechanisms that use the punishment consequence to the undesirable behaviour instead of prevent their executions. Obviously, we need to represent the punishment and the reward of such behaviours in the developed system. For example, figure 9 presents the punishment of the notification delay which consists in displaying a sorry message. This message will be displayed by the conference chairman. However, if this delay is due to a reviewer who did not send its evaluation, it must be punished by the conference chairman and its name will be removed from the conference program committee as it is presented in the figure 10.

```

3 import jade.core.behaviours.*;
4
5 public class NotificationDelay extends OneShotBehaviour{
6
7     public void action(){
8
9         System.out.println("[NorJADE Framework - Punishment] : The agent " + this.getAgent().getLocalName() + " say: we are so
10     }
11 }
12
13

```

Figure 9: The NotificationDelay behaviour.


```

3 import agents.Reviewer;
7 |
8 public class WarningReceived extends OneShotBehaviour{
9
10 public void action(){
11     Conference C = ((Conference)((Reviewer)this.myAgent).CM.getListOfConferences().elementAt(0)) ;
12     System.out.println("[NorJADE Framework - Punishment] : The agent : " + this.myAgent.getLocalName() + " received a pun:
13     C.removeFromProgramCommittee(this.myAgent.getLocalName());
14     System.out.println("[NorJADE Framework - Punishment] : The agent : " + this.myAgent.getLocalName() + " is removed from
15     this.myAgent.addBehaviour(new Disappointment());
16
17 }
18 }
19 }
20

```

Figure 10: The WarningReceived behaviour.

- Update the code of the application by procedural norms

NorJADE framework supports also the procedural and constitutive norms. These kinds of norm provide an abstraction level which allows manipulating norms and exploiting their advantages. For example, in the behaviour SearchNotificationDateReached, we specified sending notification as a goal instead of a behaviour (Figure 11). Consequently, the agent must interrogate NorJADE ontology to get the behaviour that can be used to reach this goal. In fact, this allows to update behaviours without updating the agent code (for example, updating the notification method).

```

...
        if (C.getNbrReviewing() == 0) {
...
            this.myAgent.ReachGoal("SendNotifications");
            //(((ConferenceChair)(this.myAgent)).addBehaviour(new SendNotification());
...

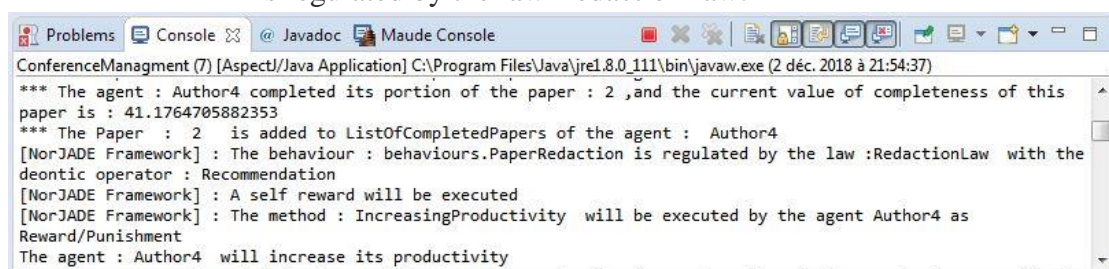
```

Figure 11: A part of the code SearchNotificationDateReached behaviour.

After developing this system, we can execute it with different scenario in order to validate different features.

- The first scenario

This scenario is configured to provide sufficient time to process all papers (CFP date = 0, deadline = 10, notification = 100). In this case, we can show in figure 12 that the author 4 has completed its portion of the paper 2. Consequently, he has rewarded but increasing its productivity because the redaction paper behaviour is regulated by the law RedactionLaw.



```

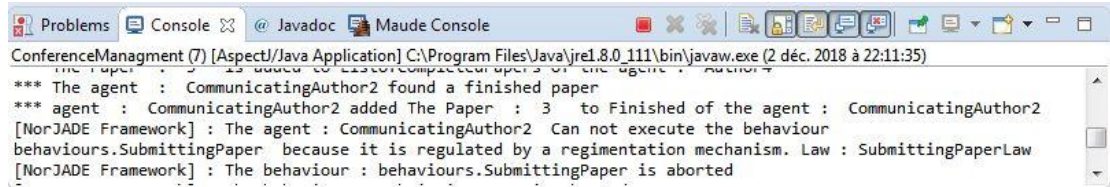
ConferenceManagement (7) [Aspect/Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (2 déc. 2018 à 21:54:37)
*** The agent : Author4 completed its portion of the paper : 2 ,and the current value of completeness of this
paper is : 41.1764705882353
*** The Paper : 2 is added to ListOfCompletedPapers of the agent : Author4
[NorJADE Framework] : The behaviour : behaviours.PaperRedaction is regulated by the law :RedactionLaw with the
deontic operator : Recommendation
[NorJADE Framework] : A self reward will be executed
[NorJADE Framework] : The method : IncreasingProductivity will be executed by the agent Author4 as
Reward/Punishment
The agent : Author4 will increase its productivity

```

Figure 12: A part of execution of the first scenario.

- The second scenario

In the second scenario, we configured the system with a deadline date that prevent agents to send their papers (deadline = 1). Consequently, as it is presented in figure 13, the submitting paper behaviour is aborted because this behaviour is regulated by the regimentation mechanism.



```

ConferenceManagement (7) [Aspect/Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (2 déc. 2018 à 22:11:35)
*** The agent : CommunicatingAuthor2 found a finished paper
*** agent : CommunicatingAuthor2 added The Paper : 3 to Finished of the agent : CommunicatingAuthor2
[NorJADE Framework] : The agent : CommunicatingAuthor2 Can not execute the behaviour
behaviours.SubmittingPaper because it is regulated by a regimentation mechanism. Law : SubmittingPaperLaw
[NorJADE Framework] : The behaviour : behaviours.SubmittingPaper is aborted

```

Figure 13: A part of execution of the second scenario.

➤ The third scenario

In the the third scenario, we configured the system with dates that prevent ensuring normal reviewing process (Deadline date = 10 and Notification date = 13). Hence, after intercepting the reviewing date, NorJADE framework will test either the obligate behaviour SendevaluationResult is executed or not (Figure 14 – A). Because this later is not executed, the conference chairman (the enforcer) will request the execution of the punishment (Figure 14 – A). In figure 14 – B, the agent reviewer 2 has received a punishment and he will be removed from program committee. Obviously, because reviewer did not send their evaluation result, the chairman will not be able to send the notification. Consequently, he will be punished by himself by the execution of the notification delay (Figure 14 – C).

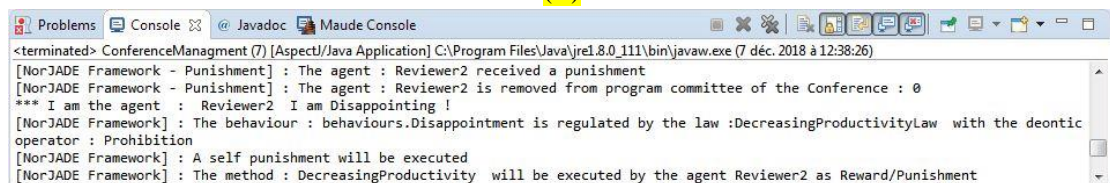


```

<terminated> ConferenceManagement (7) [Aspect/Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (7 déc. 2018 à 12:38:26)
[NorJADE Framework] : The event : ReviewingReached is intercepted
[NorJADE Framework] : The end event : ressources.Conference.ReviewingReached is intercepted. We will test either the obligate
behaviour : behaviours.SendEvaluationResult is executed or not.
[NorJADE Framework] : The agent : ConferenceChairMan has requested the execution of the behaviour RewardPunishment.WarningReceived
by the agent Reviewer0 as a Reward/Punishment
[NorJADE Framework] : The agent : ConferenceChairMan has requested the execution of the behaviour RewardPunishment.WarningReceived
by the agent Reviewer2 as a Reward/Punishment

```

(A)

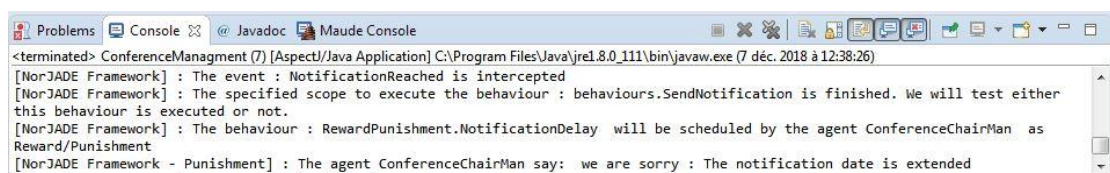


```

<terminated> ConferenceManagement (7) [Aspect/Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (7 déc. 2018 à 12:38:26)
[NorJADE Framework - Punishment] : The agent : Reviewer2 received a punishment
[NorJADE Framework - Punishment] : The agent : Reviewer2 is removed from program committee of the Conference : 0
*** I am the agent : Reviewer2 I am Disappointing !
[NorJADE Framework] : The behaviour : behaviours.Disappointment is regulated by the law :DecreasingProductivityLaw with the deontic
operator : Prohibition
[NorJADE Framework] : A self punishment will be executed
[NorJADE Framework] : The method : DecreasingProductivity will be executed by the agent Reviewer2 as Reward/Punishment

```

(B)



```

<terminated> ConferenceManagement (7) [Aspect/Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (7 déc. 2018 à 12:38:26)
[NorJADE Framework] : The event : NotificationReached is intercepted
[NorJADE Framework] : The specified scope to execute the behaviour : behaviours.SendNotification is finished. We will test either
this behaviour is executed or not.
[NorJADE Framework] : The behaviour : RewardPunishment.NotificationDelay will be scheduled by the agent ConferenceChairMan as
Reward/Punishment
[NorJADE Framework - Punishment] : The agent ConferenceChairMan say: we are sorry : The notification date is extended

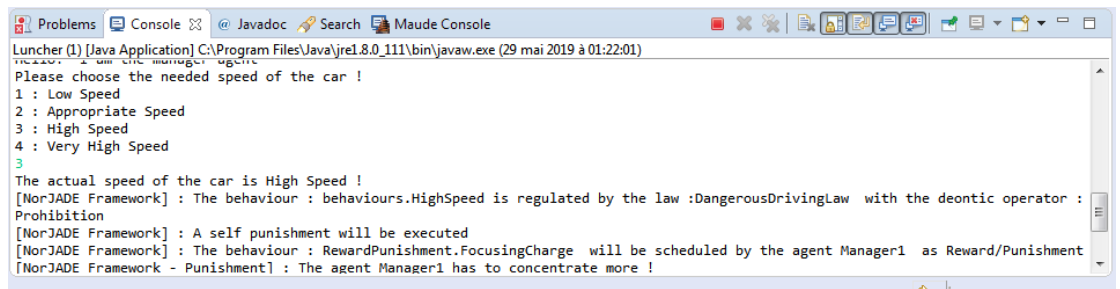
```

(C)

Figure 14: Some portions of the execution of the third scenario.

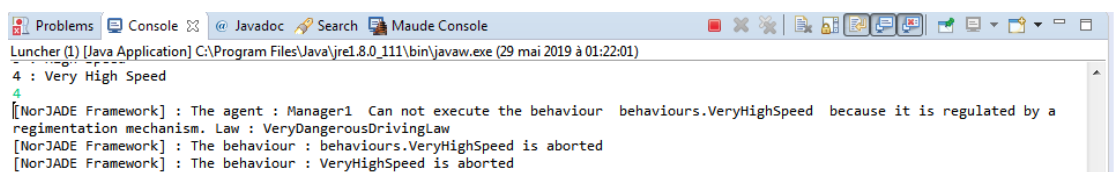
b. Speed Manager

Speed manager example is an application developed using NorJADE framework. It simulates a tool that manages a speed of a car. The user will choose the speed of the car, and the speed manager agent will react according to its category. We distinguish four speed categories: low, appropriate, high and very high speed. Each one is governed by a specific law (Low speed is allowed behaviour, appropriate one is recommended, high speed is prohibited by enforcement mechanism and high speed is prohibited by regimentation mechanism). This example is mainly used to validate the difference between the enforcement and regimentation mechanism. When a user chooses high speed level (which is governed by enforcement mechanism), the agent manager allows the execution of this behaviour but with the execution of a punishment (for example, require more attention of the driver) like it is presented in figure 15. However, executing very high speed will be allowed even the user chooses it. This behaviour will be aborted because it is regulated by regimentation mechanism (figure 16).



```
Luncher (1) [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (29 mai 2019 à 01:22:01)
Please choose the needed speed of the car !
1 : Low Speed
2 : Appropriate Speed
3 : High Speed
4 : Very High Speed
3
The actual speed of the car is High Speed !
[NorJADE Framework] : The behaviour : behaviours.HighSpeed is regulated by the law :DangerousDrivingLaw with the deontic operator :
Prohibition
[NorJADE Framework] : A self punishment will be executed
[NorJADE Framework] : The behaviour : RewardPunishment.FocusingCharge will be scheduled by the agent Manager1 as Reward/Punishment
[NorJADE Framework - Punishment] : The agent Manager1 has to concentrate more !
```

Figure 15: The execution of speed manager example with the enforcement mechanism.



```
Luncher (1) [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (29 mai 2019 à 01:22:01)
4 : Very High Speed
4
[NorJADE Framework] : The agent : Manager1 Can not execute the behaviour behaviours.VeryHighSpeed because it is regulated by a
regimentation mechanism. Law : VeryDangerousDrivingLaw
[NorJADE Framework] : The behaviour : behaviours.VeryHighSpeed is aborted
[NorJADE Framework] : The behaviour : VeryHighSpeed is aborted
```

Figure 16: The execution of speed manager example with the regimentation mechanism.

c. Cleaner World

Cleaner world example is an example we developed to validate procedural and constitutive norms. It is composed of an agent (robot) that can clean a world using two different behaviours. The first one is used when the world is known. Then, the robot will move directly to the garbage positions. The second behaviour is used when the world is unknown. Hence, the robot must explore its environment for searching garbage. Because we do not

know where the robot will be executed, we can not choose a priori the appropriate behaviour. Consequently, we defined cleaning world as a goal of the robot (Figure 17). This goal is defined in the NorJADE ontology. Hence, when an agent join an environment will just interrogate the ontology to define the appropriate behaviour to reach its goal. Figure 18 gives the execution of this example. We can remark that the robot choose the execution of the CleaningKnownWorld behaviour.

```

9  public class CleanerRobot extends Agent {
10
11  public void setup(){
12      System.out.println("Hello ! I am the cleaner robot !");
13      this.ReachGoal("CleaningWorld");
14  }
15
16 }
17

```

Figure 17: The code of the agent Robot.

```

[NorJADE Framework] : The agent : Robot will execute the behaviour : behaviours.CleaningKnownWorld to reach the goal :
CleaningWorld
This is the cleaner behaviour and I am in the position : 0 ; 0
A garbage is found in the position : 0 ; 1 , and it is removed !
This is the cleaner behaviour and I am in the position : 0 ; 1
A garbage is found in the position : 1 ; 1 , and it is removed !
This is the cleaner behaviour and I am in the position : 1 ; 1
A garbage is found in the position : 1 ; 3 , and it is removed !
This is the cleaner behaviour and I am in the position : 1 ; 3
A garbage is found in the position : 4 ; 2 , and it is removed !
This is the cleaner behaviour and I am in the position : 4 ; 2
A garbage is found in the position : 4 ; 0 , and it is removed !

```

Figure 18: The execution of cleaner world example.

In the same example, we validate the constitutive norms. In fact, a specific agent (called supervisor agent) will update the NorJADE ontology at runtime to specify the kind of the world. Figure 19 shows the initial ontology that represents the behaviour CleaningKnownWorld behaviour as a behaviour to reach the goal cleaning world. After the execution of the supervisor agent and choosing unknown world, we can remark that this ontology of norm is updated (figure 20). Exactly, the behaviour to reach this goal is became CleaningUnknownWorld behaviour.



Figure 19: The initial ontology of norm.

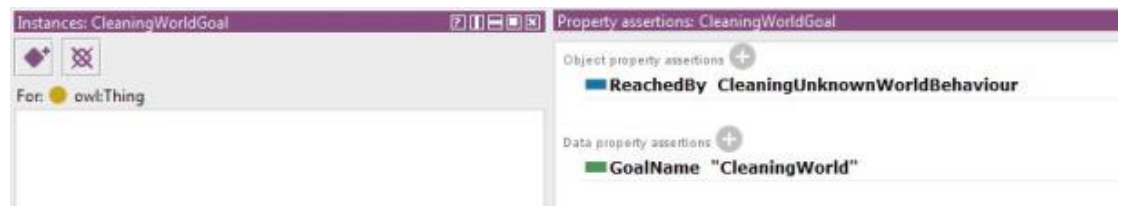


Figure 20: The updated ontology after execution.