4
Contextual Norm Enforcement

DynaCROM is an approach for implementing dynamic NMAS in which norms can be updated at system runtime and also for continuously supporting agents with precise information about their current norms. Nevertheless, a regulated NMAS should verify if a performed action is legal or illegal based on its defined norms, which might be enforced. However, it is the responsibility of the system developer to define if norms are allowed or not to be violated in his NMAS.

Norm enforcement in NMAS can be carried out *a posteriori*, by punishing infringing agents, or *a priori*, by avoiding norm violation. *A posteriori* enforcement does not guarantee norm compliance, however, the implementation of punishments inhibits infringing agents. *A priori* enforcement guarantees norm compliance while enforcing the norms of the regulated actions of a NMAS.

DynaCROM supports the norm enforcement *a posteriori* and *a priori*, as presented in the following two subsections, respectively. In subsection 4.3, an overview of the process on how DynaCROM works as an input mechanism to third-party enforcers is given. Then, in the two subsequent subsections, the norm enforcement based on the agents' external and internal behaviors are explained. The chapter is concluded with the results of DynaCROM being a combined solution for contextual norm enforcement in NMAS.

4.1.
A Posteriori Norm Enforcement

The aim of any society and its norms is to provide a common space for the realization of individual and global objectives of its participants. Sometimes, depending on both the goals of agents and their priorities, the violation of norms is better for agents (and also for the society). In this sense, norms act as a goal-oriented decision mechanism in regulated systems, being the means to achieve goals in a society. Hence, norms are allowed to be violated instead of being defined as constraints which unable any undesired behavior to happen.

For instance, in [Felicíssimo *et al.*, 2005b], an example of a NMAS, from the urban traffic domain, in which its norms might be violated is presented. In the motivating scenario of the example, an agent playing a car driver role is going from his home in the city to his summer home on the mountains (see Figure 11). Suddenly, on the way, his pregnant wife begins to go into labor. Now, the goal of the driver agent to get to a hospital, as soon as possible, emerges as the one with the highest priority. Then, he considers violating some norms. In the emergency situation, the agent prefers to pay the fine for going through a red traffic light if that will get him to the hospital faster. However, if pedestrians were crossing the street in front of the traffic light, then, the driver would not go through the red light because of the risk of killing pedestrians and because the fine he would have to pay would be too high. Both fines can be informed by DynaCROM, if the agent requests so.

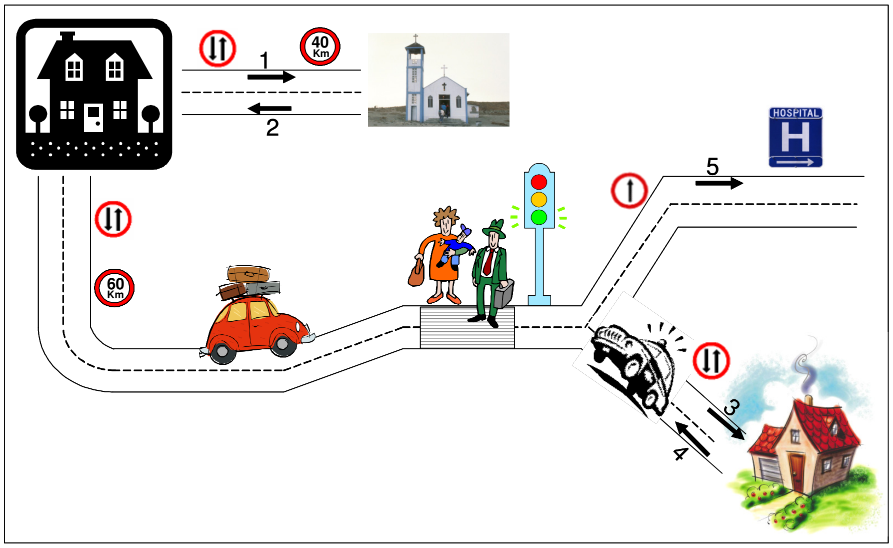

Figure 11 – An urban traffic's scenario

Another emergency situation in which norms might be violated is illustrated in Figure 12. An agent is driving down a road when he perceives that another driver agent is trying to pass his car on the left. Suddenly, a cow appears in front of his car. Now, what should the driver agent do? His decision must be based on the norms applicable to him, *i.e.*, on his contextual norms, which can be informed by DynaCROM.

If the situation of the motivating scenario occurred in an area in which the *Hinduism* religion is practiced, as in *India*, then, it would be better for the car driver to crash into the other car. This is because the fine of hurting a cow in a *Hindu* territory might be too high (*Hindus* believe that cows are sacred animals and, therefore, must be kept in safety). However, in *non-Hindu* countries, perhaps, it would be better to run over the cow instead of crashing into the other cars. In both cases, if no risk to human lives was involved, then, the decision of the driver would be based on how high a fine he would have to pay.

Figure 12 – A situation in which contextual norms must be considered

A norm violation is a situation in which an agent breaks one or more norms, entering an illegal (unsafe) state. Sanction – set of actions whose realization will remove the violation, making agents legal again (entering in a safe state) – can be used in *a posteriori* enforcement for defining the consequence of a norm violation. In order to support *a posteriori* enforcement, the DynaCROM ontology can be extended with a punishment concept for holding the sanction information of a violated norm.

Figure 13 illustrates an example that killing cows is prohibited in the *Hinduism* religion. In the example, the DynaCROM ontology was extended with the *Religion* and *Sanction* domain concepts; moreover, the DynaCROM *Environment* concept was extended with the domain object property *hasReligion*, for concretizing religions in each environment, and the DynaCROM *Norm* concept was extended with the domain object property *hasSanction*, for concretizing sanctions in each norm.

In the case illustrated in Figure 13, the *India* environment holds the *Hinduism* religion, which, in turn, holds the *PrhToKillCows* prohibition norm. This norm regulates the *KillCows* action and has *A10YearPrisonSentence* as its sanction.

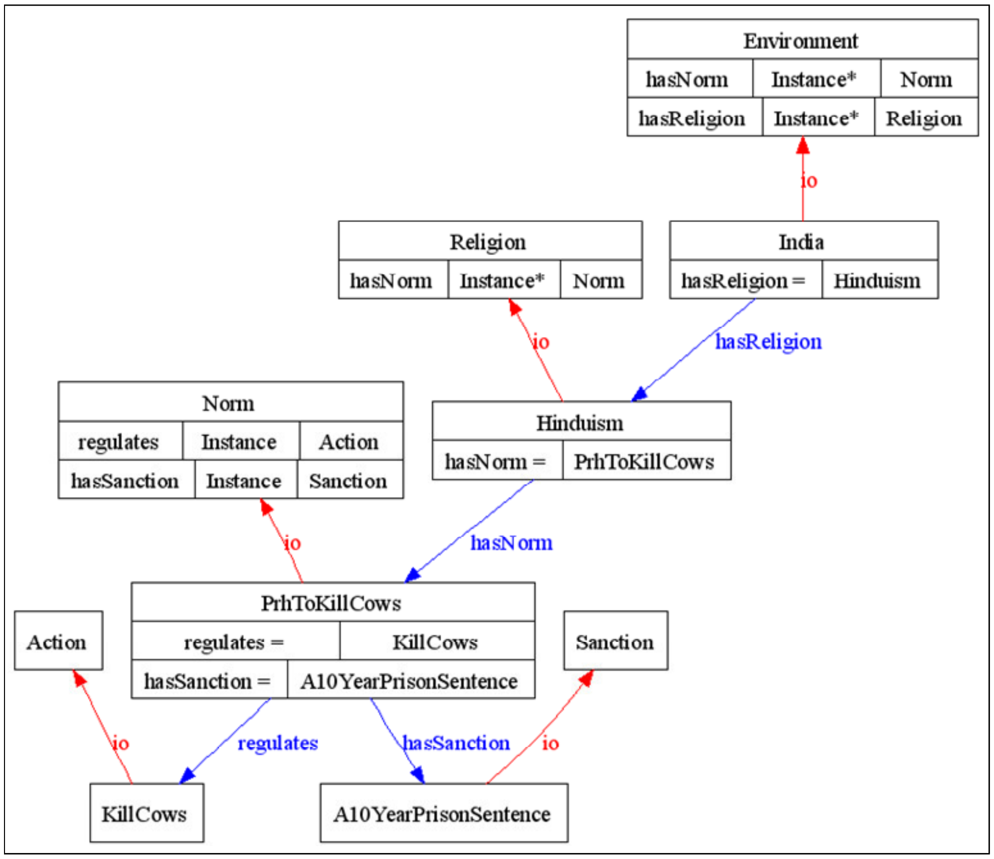

Figure 13 – A *Hinduism* religious norm concretized in the *India* environment

4.2.
A Priori Norm Enforcement

In order to support *a priori* norm enforcement, DynaCROM can be enhanced with an enforcer that will be in charge of guaranteeing norm compliance when there is an attempt to violate a norm.

Experiments were made integrating DynaCROM with SCAAR and MOSES, two solutions for norm enforcement. In SCAAR [Chopinaud *et al.*, 2006], the enforcement is done based on the internal behavior of agents; and in MOSES [Minsky_MOSES, URL], it is based on the external behavior of agents. For both solutions, DynaCROM works providing precise norm information as their input.

In the following section, an overview of how DynaCROM works as an input mechanism for norm enforcement solutions is presented. Then, in the two subsequent sections, the norm enforcement based on the agents' external and internal behaviors are explained. Because the enforcement solution is not the focus of

DynaCROM, this thesis does not deal with the problems related to that part (*e.g.*, malfunction of the enforcer).

4.3.
DynaCROM as an Input Mechanism for Norm Enforcement Solutions

DynaCROM can be used for providing information as input to norm enforcement solutions. For this, each time an agent starts the execution of a regulated action, DynaCROM retrieves, in the domain ontology, the applicable norms according to the agent's current contexts and, then, sends those norms to be enforced by the chosen norm enforcement solution.

Figure 14 illustrates an overview of the process for contextual norm enforcement. Once an agent executes a regulated action, DynaCROM verifies, in the domain ontology instance, the norms of the action, according to the agent's current contexts. Then, DynaCROM concretizes those norms in a file that is used by the chosen enforcer as its input. The enforcer reads the input file and, then, enforces the norms of the performed action.

In the case of *a posteriori* norm enforcement, the information about the violated norms is sent back to DynaCROM for the application of sanction actions. In this phase, a third-party sanction system can be used for enhancing the DynaCROM solution. However, this idea is not developed in the text of this thesis, but in [Silva, 2008] more information about this issue can be found.

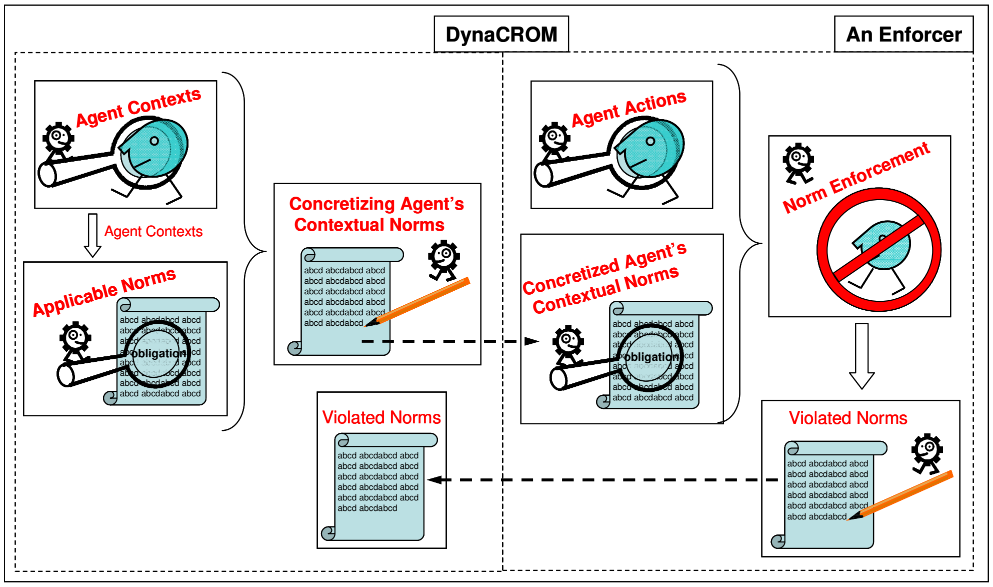

Figure 14 – DynaCROM providing precise norm information as input to enforcers

Contextual norm enforcement can be done based on the agents' external or internal behavior. In order to exemplify both situations, the following simplification of the FIPA Contract-Net interaction protocol [FIPA_Contract-Net, URL] is considered:

1. A manufacturer wants to build 100 computers;
2. He issues a call for proposal (CFP) to computer suppliers;
3. Computer suppliers answer the CFP with their proposed price;
4. The manufacturer chooses one proposal among the ones he received and informs his decision to the chosen supplier.

In the example, the action of suppliers to 'propose a price' is regulated by the norm for effecting negotiations ("*negotiations are obliged to be paid by using the national currency of the seller's country*") and by the norm for calculating prices ("*a state corporate income tax rate is obliged to be imposed on all sales, for immediate delivery or if the deliver address is in North America*"). Both norms were previously mentioned in the text of this thesis. For the enforcement of the last norm, the base price of a computer is predefined to make it possible to calculate its acceptable minimum price.

4.4. Norm Enforcement Based on the Agents' External Behavior

Figure 15 illustrates an example in which the norms for payments are enforced, by a created police agent, based on the agents' external behavior. In brief, the *MissourianManufacturer* (an agent playing the *manufacturer* role in the *Missouri* environment) sends a CFP to the *JapaneseSupplier* (an agent playing the *supplier* role in the *Japan* environment). The *JapaneseSupplier* answers the CFP message with a PROPOSE message in which the currency value is different from the one expected (*JPY* instead of *USD*, the national currency of *USA*).

When the message arrives at the *ManufacturerPolice* (the police agent created to enforce the system norms in the manufacturer agent), the *ManufacturerPolice* blocks the sending of the message to the *MissourianManufacturer* agent and sends an INFORM(*Nok*,*NationalCurrency*) message with the error occurred (*i.e.*, wrong currency) to the *JapaneseSupplier*. Then, the *JapaneseSupplier* sends a new PROPOSE message with the correct currency, however, he does not considered the state corporate income tax of *6.25* from *Missouri*. So, the *ManufacturerPolice* also enforces the other norm and sends an INFORM(*Nok*, *StateCorporateIncomeTaxOf*) message to the *JapaneseSupplier*, informing him that now the error is with the missing state corporate income tax.

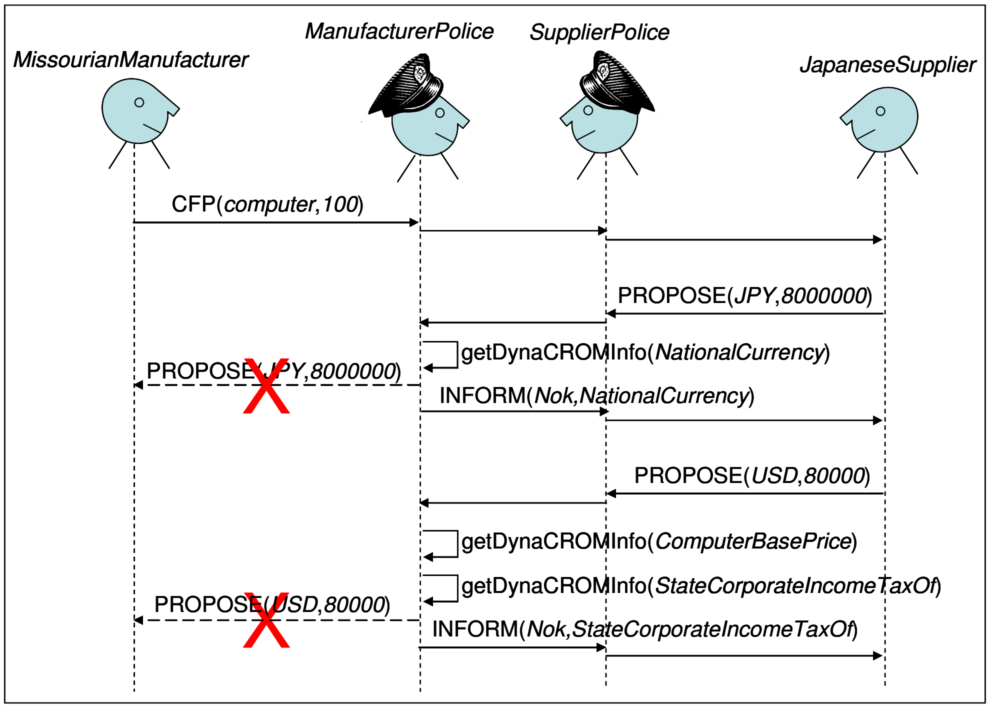
MissourianManufacturer
ManufacturerPolice
SupplierPolice
JapaneseSupplier
CFP(computer,100)
PROPOSE(JPY,8000000)
getDynaCROMInfo(NationalCurrency)
PROPOSE(JPY,8000000)
INFORM(Nok,NationalCurrency)
PROPOSE(USD,80000)
getDynaCROMInfo(ComputerBasePrice)
getDynaCROMInfo(StateCorporateIncomeTaxOf)
PROPOSE(USD,80000)
INFORM(Nok,StateCorporateIncomeTaxOf)

Figure 15 – An example of a police agent enforcing norms for payments

4.4.1. DynaCROM as an Input Mechanism for MOSES

The MOSES solution is considered in order to exemplify how the enforcement of norms based on the agents' external behavior effectively works. In MOSES, the norm enforcement is based on a written law file in which the system developer specifies all the norms that will regulate his system. In this law file, a set of simple predicative is used for checking the conformance of the norms of senders and receivers. Only messages that are in conformance with the defined system norms are delivered to its final destination. The norm enforcement is guaranteed by created police agents, which are in charge to filter all the messages changed between sender and receiver agents for checking if those messages are in conformance with the norms of the system.

Briefly presenting part of the structure and syntax of an implementation in MOSES, it provides the 'sent(...)', 'arrived(...)', 'obligationDue(...)' and 'getMessageContent(...)' methods and also the 'startsWith(...)', 'doAdd(...)', 'doImposeObligation(...)’, ‘doDeliver()’ and ‘doForward()’ predicative, all written in Code 7.

Code 7. An overview of the MOSES methods and predicative

```
(1)   public void sent(String source,
                              String message, String dest) {
(2)    if (message.startsWith(…)) {
        …
(3)     doImposeObligation(…);
(4)     return;
(5)    }
(6)   …}

(7)   public void arrived(String source,
                                String message, String dest) {
(8)    if (message.startsWith(…)) {
        …
(9)     doAdd(…);
        …
(10)    doDeliver();
(11)    return;
(12)   }
(13) …}

(14) public void obligationDue(Term obligationTerm) {
(15)  if (obligationTerm.toString().equals(…)) {
        …
(16)    doForward();
(17)   }
(18) …}
```

In the ‘sent(...)’ method (line 1 from Code 7), the possible messages that can be sent in a system (*e.g.*, the CFP, PROPOSE, ACCEPT_PROPOSAL, QUERY_REF, INFORM and CANCEL FIPA performatives) are specified by using the ‘startsWith(...)’ predicative (line 2). This restriction is important for preventing malicious agents to intentionally overload their systems with lots of undesired messages.

The obligations to be enforced in the system are specified by using the ‘doImposeObligation(...)’ predicative (line 3) and described by using the ‘obligationDue(...)’ method (line 14 to 18). The ‘doForward()’ predicative (line 16) is used to guarantee that only the messages that are in conformance with the expected values or obligations of the system will be forwarded to their respective recipients.

In the ‘arrived(...)’ method (line 7), the ‘doAdd(...)’ performative (line 9) is used to define an expected variable value, informed by the sender agent, that will be checked when a reply message is sent back to him. Then, the ‘doDeliver()’ performative (line 10) is used to guarantee that only the messages that are in conformance with the expected values or obligations of the system will be delivered to their respective recipients.

Figure 16 illustrates a simplified example of how agents inform an expected value and Code 8 presents the example codified in MOSES. The *Missourian-Manufacturer* agent sends an INFORM message to the *JapaneseSupplier* agent with the information about the national currency expected. When the message arrives at the *ManufacturerPolice* agent, he verifies the parameter value of the 'startsWith' performative. As the message is an informative message (*i.e.*, the parameter value of the 'startsWith' performative is equal to "INFORM", see line 2 of Code 8) and no obligation is defined for the message, then, the *Manufacturer-Police* just forwards the received message to its recipient (line 3).

When the message arrives at the *SupplierPolice* agent, the 'arrived(...)' method is executed. The parameter value of the 'startsWith(...)' performative is verified by the *SupplierPolice* (line 7) and, then, the '*manufacturerNationalCurrency*' variable receives the *USD* value obtained from the received message (line 8). Finally, the message is delivered to its final recipient, the *JapaneseSupplier* (line 9).

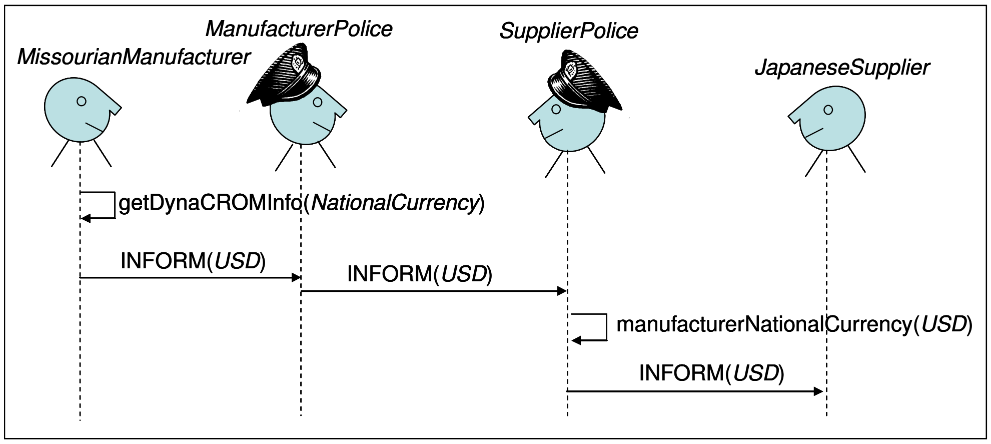

Figure 16 – Receiving and adding the information about the national currency expected by the manufacturer agent

Code 8. Part of a MOSES law for informing an expected variable value

```
(1)  public void sent(…) {
(2)   if (message.startsWith("INFORM")){ …
(3)    doForward();
(4)    return;
(5)  }

(6)  public void arrived(…) {
(7)   if (message.startsWith("INFORM")) {
(8)    doAdd("manufacturerNationalCurrency
             (" + getMessageContent(message) + ")");
(9)    doDeliver();
(10)   return;
(11) }
```

In order to give the necessary dynamics for regulation in NMAS, DynaCROM uses its output (applicable norms retrieved according to the agents' current contexts) for activating MOSES predefined norms. Only norms sent by DynaCROM are enforced by MOSES, even if those norms are already predefined in a MOSES (*static*) law. This is a dynamic solution because DynaCROM output is based on an inferred domain ontology instance, which normally evolves according to the social changes characteristic of open systems.

Figure 17 illustrates an example in which the norm for accepting payments with only the national currency is enforced and Code 9 presents the example codified in MOSES. The *JapaneseSupplier* answers a *CFP* received message with a PROPOSE message. When the message arrives at the *SupplierPolice*, he adds the value about the proposed currency of the supplier agent (see line 3 of Code 9).

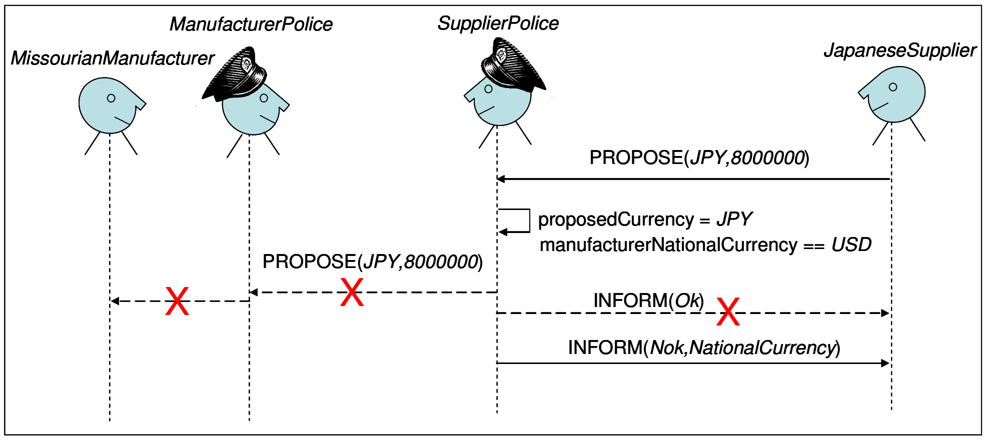

Figure 17 – Enforcement of a norm for payments with national currency

Code 9. Part of a MOSES law for enforcing payment norms

```
(1)   public void sent(…){
(2)    if (message.startsWith(“PROPOSE")){
(3)     doAdd(“proposedCurrency(“ +
              getMessageContent(message) + ")");
(4)     if “OblToPayWithNationalCurrency” isIn
         domainEnv.hasNorm{…
(5)       doImposeObligation("checkNationalCurrency",1,"sec")};
(6)     if “OblToImposeAStateCorporateIncomeTax” isIn
         domainEnv.hasNorm{…
(7)       doImposeObligation("checkStateCorporateIncomeTaxOf",
                            1,"sec");
(8)     return;
(9)   }}

(10)   public void obligationDue(Term obligationTerm){
(11)   if (obligationTerm.equals("checkNationalCurrency")){…
(12)    if (proposedCurrency.equals
            (manufacturerNationalCurrency)){
```

```
(13)    doForward();
(14)    doDeliver(CS.toString(),“INFORM(Ok)",
                   sourceAddress); …}
(15)   else{
(16)    doDeliver(CS.toString(),“INFORM(Nok,National-
                  Currency)", sourceAddress); …} …}}

(17) public void arrived(…){
(18)  if (message.startsWith("PROPOSE")){…
(19)   doDeliver();
(20)   return;
(21) }}
```

If the norms for payments are presented in the DynaCROM ontology instance, then, MOSES checks the obligation to pay with national currency (lines 4 and 5) and the obligation to add the state corporate income tax (lines 6 and 7).

For instance, the ‘doImposeObligation(...)’ predicative defined in line 5 is used in the method for checking if the currency proposed by the supplier agent is the same as the currency expected by the manufacturer agent. If both currencies are the same (line 12), then, the message is forwarded to its final destination (line 13) and an INFORM(*Ok*) message is sent to the supplier agent (line 14) in order to inform him that his message was sent. When the message arrives at the *ManufacturerPolice* (line 18), then, he only delivers it to its final destination (the manufacturer agent (line 19)).

However, if the currencies are different, as is the case illustrated in Figure 17, then, an INFORM(*Nok,NationalCurrency*) message is sent to the supplier agent (line 16) in order to inform him that his message was not sent because an error exists with his proposed value for the ‘*NationalCurrency*’ variable.

Likewise, the enforcement of the norm for the incorporation of the state corporate income tax is done.

It is important to explain here that, because MOSES does not have/provide the ‘*isIn*’ predicative (used in lines 4 and 6 of Code 9) and was not implemented to retrieve ontology values (*e.g.*, the ‘*domainEnv.hasNorm*’ one), then, these aditional resolutions are implemented by DynaCROM. DynaCROM verifies the condition for the enforcement of obligation norms (*e.g.*, lines 4 and 6 of Code 9) and, if it is the case, the MOSES law file is rewritten (by DynaCROM) with the ‘doImposeObligation(...)’ predicative (*e.g.*, lines 5 and 7). This way, the MOSES solution is enhanced with the ability and dynamics of DynaCROM to deal with domain ontology values and system conditions.

4.5. Norm Enforcement Based on the Agents' Internal Behavior

Figure 18 illustrates an example in which the norms for payments are enforced by the agents themselves (*i.e.*, agents are self-regulated) based on their internal behavior. In brief, the *MissourianManufacturer* sends a CFP to the *JapaneseSupplier*. The *JapaneseSupplier* tries to answer the CFP message with a PROPOSE message, but, because the proposed currency value is different from the one expected (*JPY* instead of *USD*, the national currency of *USA*), the message is not sent due to the (self-)enforcement of the norm for payments with the national currency.

Then, the supplier agent tries to send a new PROPOSE message (now, with the correct currency), however, he does not consider the state corporate income tax of *6.25* from *Missouri*. Then, the message is not sent due to the (self-) enforcement of the norm for payments with the state corporate income tax.

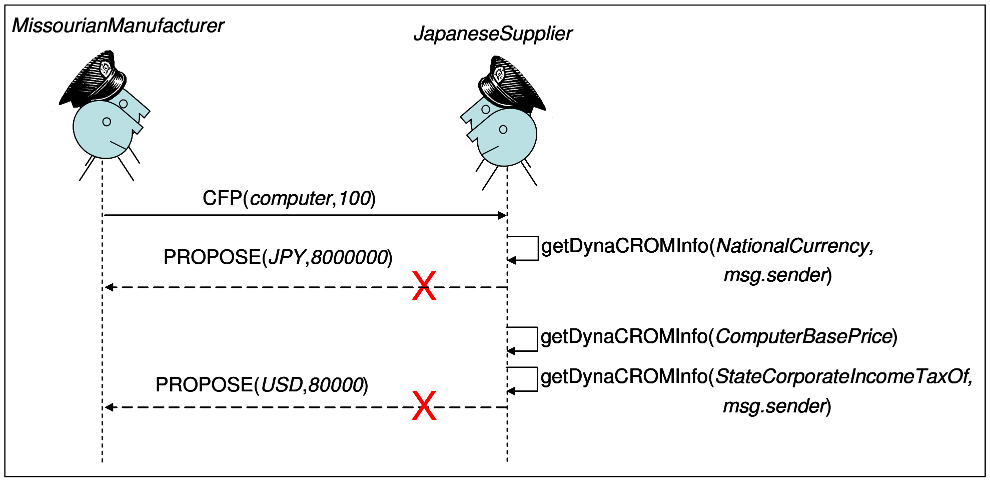
MissourianManufacturer
JapaneseSupplier
CFP(computer,100)
PROPOSE(JPY,8000000)
getDynaCROMInfo(NationalCurrency, msg.sender)
getDynaCROMInfo(ComputerBasePrice)
PROPOSE(USD,80000)
getDynaCROMInfo(StateCorporateIncomeTaxOf, msg.sender)

Figure 18 – Self-regulated agents enforcing norms for payments

4.5.1. DynaCROM as an Input Mechanism for SCAAR

The SCAAR solution is considered in order to exemplify how the enforcement of norms based on the agents' internal behavior effectively works. SCAAR is a norm enforcement mechanism that enhances agents with a self-monitoring capability for avoiding norm violation. SCAAR uses control hooks to trigger an enforcement core each time that a regulated action is executed. When agents spontaneously incorporate the DynaCROM behavior, aiming to receive updated system norms, they also incorporate SCAAR. In the incorporation process, DynaCROM automatically replaces, inside agents, the headers of the regulated methods (which have their signature predefined by DynaCROM) by the methods enhanced with the SCAAR control hooks.

Control hooks can be inserted, inside the code of an agent, before a regulated action, for preventing norm violation, or after, for detecting norm violation. This is a decision of the system developer when implementing the DynaCROM headers to be replaced inside the agents' codes.

If the system developer decides to use the SCAAR norm prevention mechanism in his regulated NMAS, then, when an attempt to violate an obligation or prohibition norm is made, the enforcement core blocks the execution of the infringing action and informs it to DynaCROM. If the system developer decides to use only the SCAAR norm detection mechanism, then, when a norm violation of an obligation or prohibition norm occurs, the enforcement core informs it to DynaCROM. For a permitted norm, no specific action is taken by SCAAR.

For DynaCROM and SCAAR work together properly, the system developer should write the abstract norms of his system according to both the SCAAR syntax and the DynaCROM domain ontology's structure. Then, he should concretize those norms, in the domain ontology, with instance values. Regulated actions and norms must be written in the same way in both SCAAR norms and DynaCROM domain ontology. For the enforcement process, DynaCROM reads the ontology and rule files to automatically instantiate the abstract norms with domain values (see Figure 10), providing concrete norms as input to SCAAR.

SCAAR considers norms written according to the following definition, in which each term represents a set of clauses.

Norm Definition. N (a DO A [AND P]) [IF (a BE in S [AND P])]

```
N := OBLIGED | FORBIDDEN
a := an agent playing a system role
A := an action expression
S := a state
P := a proposition concerning A or S
```

The system developer must write the SCAAR norms by respecting the exact identification of data represented in the DynaCROM domain ontology of the NMAS to be regulated. For instance, Code 10 presents an example of a SCAAR norm that a system developer wrote to regulate the manufacturer agents of his system when they pay a placed order.

Code 10. A SCAAR norm written to regulate manufacturers when they pay a placed order

```
(1)SCAARNorm_OblToPayWithNationalCurrency:
(2)[OBLIGED (agt DO pay(agtCurrency) AND
             (agtCurrency == domainEnv.hasCurrency))
(3) IF (agt BE in Environment AND (agtEnv == domainEnv)
(4)     AND ("OblToPayWithNationalCurrency" isIn
             (domainEnv.hasNorm)))]
```

In the example of Code 10, a manufacturer agent informs its proposed currency via the '*agtCurrency*' variable (line 2). Then, SCAAR compares this value with the value of the '*domainEnv.hasCurrency*' attribute (line 2), which is automatically instantiated by DynaCROM according to the agent's current environment (*e.g.*, with the *USD* value when the agent is in *Missouri*). The norm is enforced every time that it is presented in the analyzed environment due to the verification occurred in lines 3 and 4. Hierarchical environments have also inherited norms enforced due to the '*DynaCROMRule_EnvWithOEnvNorms*' predefined norm, which was presented in Code 1.

Code 11 presents another example of a SCAAR norm. In this example, a system developer wrote the norm to regulate the manufacturer agents of his system when they calculate prices. A manufacturer agent informs its proposed tax via the '*agtStCorpIncTax*' variable (line 2). Then, SCAAR compares this value with the value of the '*domainEnv.hasAStateCorporateIncomeTaxOf*' attribute (line 2), which is automatically instantiated by DynaCROM according to the agent's current environment (*e.g.*, with the *6.25* value when the agent is in *Missouri*).

Code 11. A SCAAR norm written to regulate manufacturers when they calculate prices

```
(1)SCAARNorm_OblToImposeAStateCorporateIncomeTax:
(2)[OBLIGED (agt DO calculatePrice(agtStCorpIncTax) AND
   agtStCorpIncTax == domainEnv.hasAStateCorporateIncomeTaxOf)
(3) IF (agt BE in Environment AND (agtEnv == domainEnv)
(4)     AND ("OblToImposeAStateCorporateIncomeTax" isIn
             (domainEnv.hasNorm)))]
```

Norms are represented in SCAAR by using Petri nets [Murata, 1989] in order to permit the verification of norm compliance, and inhibitor arcs are used to permit the norm enforcement. When a regulated action is executed, its associated control hook activates the Petri nets that represent the norms of the action. Then, all inhibitor arcs of each active Petri net are analyzed for verifying if its token stands in its previous place. If it is the case, the inhibitor arc of the infringing action forwards an exception to the agent's enforcement core because the norm was violated.

The pseudo algorithm for the SCAAR norm enforcement mechanism is presented in Code 12.

Code 12. SCAAR pseudo algorithm for norm enforcement

```
(1) Let $I$ be the information received about an agent;
(2) Let $\{t_1,…,t_n\}$ be the set of transitions associated with $I$;
(3) Let $\{P_1,…,P_m\}$ be the set of Petri nets associated with $I$;
(4) Let $\{Pact_1,…,Pact_p\}$ be the set of activated Petri nets
     associated with $I$;
(5) Let $t_{ij}$ be the transition i of a Petri net j;
(6)  **for all** $P_k \in \{P_1,…,P_m\}$ with $t_{1k} \in \{t_1,…,t_n\}$ **do**
(7)   $Pact_{(p+1)} \leftarrow$ activate $P_k$ if it is not already activated;
(8)   add $Pact_{(p+1)}$ in $\{Pact_1,…,Pact_p\}$
(9)  **end for**
(10) Let $\{Pact_1,…,Pact_m\}$ be the set of the activated petri
      nets including $t_{ij} \in \{t_1,…,t_n\}$;
(11)  **for all** $Pact_j \in \{Pact_1,…,Pact_m\}$ **do**
(12)   **for all** $t_{ij} \in \{t_1,…,t_n\}$ **do**
(13)    activates the transition $t_{ij}$ from $Pact_j$;
(14)    **if** ($t_{ij}$ is fireable) **then**
(15)     fire the transition $t_{ij}$;
(16)     remove $t_{ij}$ from $\{t_1,…,t_n\}$;
(17)    **else**
(18)     throw an exception
(19)   **end for**
(20)   **if** ($Pact_j$ is in a final state) **then**
(21)    remove $Pact_j$ from $\{Pact_1,…,Pact_m\}$;
(22)  **end for**
```

In order to exemplify the pseudo algorithm presented above, the '*SCAAR-Norm_OblToPayWithNationalCurrency*' (previously presented in Code 10) is considered. The norm is represented by the Petri net illustrated in Figure 19. The '*PetriNet_OblToPayWithNationalCurrency*' is automatically created by SCAAR and it is activated when an agent executes the action to pay an order (*i.e.*, when the agent executes the *PayWithNationalCurrency* regulated system action, which is illustrated in Figure 7).

Following Code 12, in (1), I = {Environment(*Missouri*)}, for instance; in (2), $\{t_1,...,t_n\}$ = $\{t_1\}$; in (3), $\{P_1,...,P_m\}$ = {*PetriNet_OblToPayWithNationalCurrency*}, the Petri net is created for representing the SCAAR norm; in (4), $\{Pact_1,...,Pact_p\}$ = {}; in (5-9), the created Petri net is activated; in (10), $\{Pact_1\}$ = {*PetriNet_OblToPay-WithNationalCurrency*}; in (11-13), the transition t_1 is activated; in (15-16), t_1 is fireable if, and only if, its previous place is empty (verification in line 14); in (18), t_1 is not fireable (*e.g.*, *JPY* is the currency proposed by a supplier agent), then, an exception is thrown; finally, in (20-22), the '*PetriNet_OblToPayWithNational-Currency*' is removed.

PetriNet_OblToPayWithNationalCurrency

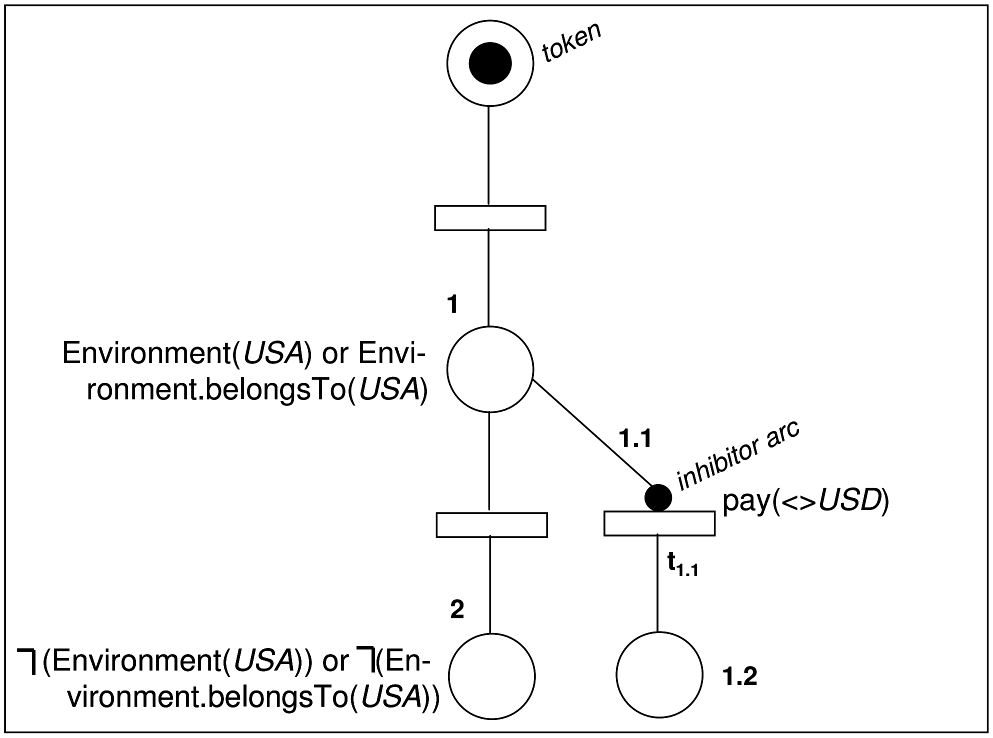
token
1
Environment(*USA*) or Environment.belongsTo(*USA*)
1.1
inhibitor arc
pay(<>*USD*)
$t_{1.1}$
2
¬(Environment(*USA*)) or ¬(Environment.belongsTo(*USA*))
1.2

Figure 19 – A Petri net created to represent a payment norm in *American* environments

4.6. Discussion

In this chapter, the subject of contextual norm enforcement is analyzed. The achievement of a norm enforcement contextualized for each application agent is due to the integration of DynaCROM with third-party enforcers. The integration of DynaCROM with MOSES exemplifies a contextual norm enforcement based on the agent's external behavior; and the integration of DynaCROM with SCAAR exemplifies a contextual norm enforcement based on the agent's internal behavior. For both cases, the *a priori* norm enforcement was the chosen one. Nevertheless, the *a posteriori* norm enforcement is presented in the beginning of the chapter, where an overview of a DynaCROM solution for it is also given.

The integration of DynaCROM with MOSES and SCAAR, benefited both norm enforcement solutions. In chapter 2 of this thesis (more specifically, in subsection 2.2.7), some research questions were proposed for a comparative study done among some solutions for norm enforcement. Those research questions are rewritten below (in the exact way that they were presented in chapter 2) for presenting, in Table 4, the results of DynaCROM combined with MOSES and SCAAR for a contextual norm enforcement in NMAS.

In summary, the advantages presented bellow point out the modular solution of DynaCROM while working as a fine-grained mechanism for third-party norm enforcers.

rq.i. Does it explicitly support the implementation of an organizational normative dimension?

rq.ii. Does it have a manager/governor/police agent for norm enforcement?

rq.iii. Does it directly enforce prohibition norms?

rq.iv. Does it make a distinction between the implementation of an organization entity and a group of roles? *I.e.*, Does the organization concept have an explicit entity to support its implementation?

rq.v. Can the structure of the system (in any dimension, *e.g.* the normative one) evolve at MAS execution time?

rq.vi. Does the implementation have a conceptual model to guide its specifications?

Table 4. Results of DynaCROM combined with third-party norm enforcers

	rq.i	**rq.ii**	**rq.iii**	**rq.iv**	**rq.v**	**rq.vi**
DynaCROM_Moses	✓	V	X	✓	✓	✓
DynaCROM_SCAAR	✓	X	V	✓	✓	✓