3
Dynamic Contextual Regulation Information Provision

Open MAS can be extremely dynamic due to heterogeneous agents that migrate among those systems for obtaining resources or services not found locally. In order to prevent malicious actions and to ensure agent trust, open MAS should be enhanced with normative mechanisms. However, it is not reasonable to expect that *foreign* agents will know in advance all the norms of the MAS in which they will execute. Thus, this chapter presents the DynaCROM approach. DynaCROM stands for *Dynamic Contextual Regulation Information Provision in Open MAS* and it aims to bridge the gap between the theoretical work on norms and practical normative systems, providing a solution to operationalize regulative norms in NMAS.

In the following section, the DynaCROM methodology developed to support the system developer in the tasks of implementation, management and evolution of the norms of his NMAS is explained. The methodology includes the phases of contextualization, concretization, representation and composition of norms. Then, the process of norm incorporation in the agents of a DynaCROM NMAS is presented. The chapter is concluded with a discussion about the advantages and drawbacks in the use of DynaCROM.

3.1.
The DynaCROM Methodology for System Developers

3.1.1. From Abstract to Concrete Norms in NMAS

A major challenge in NMAS is how norms can be effectively applied to their agents and, then, easily managed and evolved. These tasks are arduous because norms are usually written for general purposes, hindering a more precise regulation.

In [Gaertner *et al.* 2007], the authors of the paper propose to extend the coordination level of a MAS with a normative level, so that, norms can be integrated during the design and execution time of the system. This thesis follows

their proposition but, furthermore, it also proposes to extend the normative level with, what is called, a *contextual normative level*. In this level, abstract norms are concretized (*i.e.*, embodied) with domain values according to the context wherein they hold. The proposition for contextual classification of norms follows the ideas first proposed by Dignum in [Dignum, 2002] and, then, refined in [Grossi and Dignum, 2004]. However, their works mainly address formal issues while this thesis addresses the practical ones, providing DynaCROM – an implemented solution as a proof-of-concept for the ideas proposed.

In order to illustrate the proposal of this thesis, Figure 3 presents the *Coordination*, *Normative* and *Contextual Normative Levels* of a simplistic supply-chain scenario in which activities (illustrated by linked ellipses) are represented on the three layers (connected by dashed arrows). Norms (illustrated by vertical arrows) are applied at the second and last levels. Contextual norms (illustrated by diagonal arrows) are applied at the last level.

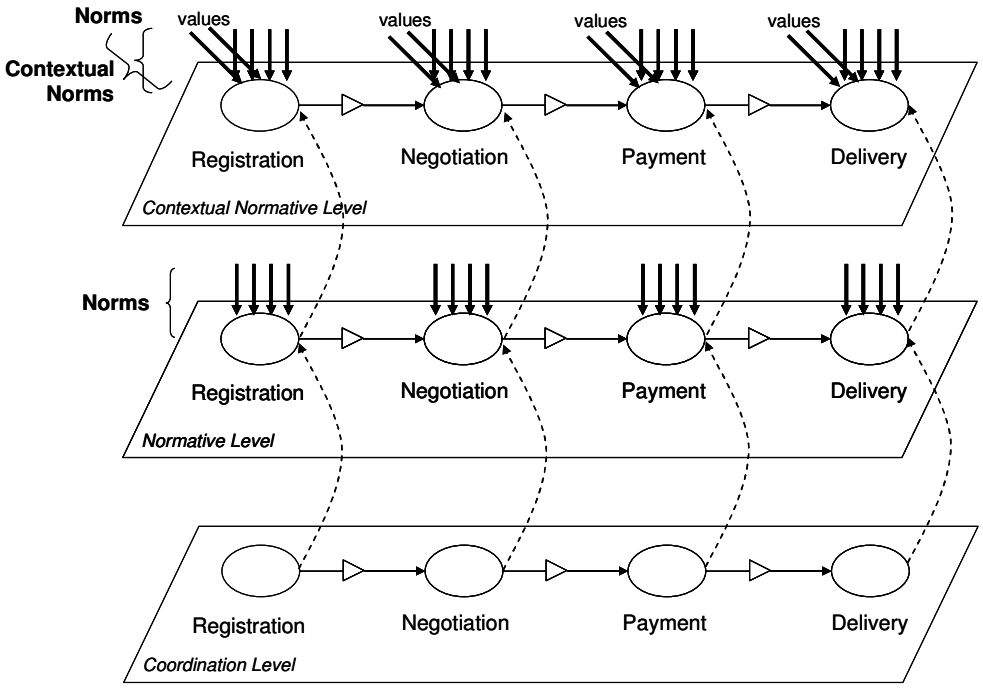

Figure 3 – Activities, Regulated Activities and Contextual Regulated Activities, based on [Gaertner *et al.*, 2007]

In order to exemplify how norms are concretized in normative levels, the *Negotiation* activity of Figure 3 is considered. A *Negotiation* summarizes a set of more specific activities performed between customer and seller agents (*e.g.*, a customer asks a seller how much a product costs; the seller states his price, with

or without discounts; the customer accepts the seller's price). The *Negotiation* activity is linked to the *Payment* one and both activities might be translated in the normative level to:

> *A Payment Norm for Effecting a Negotiation*: Negotiations are obliged to be paid by using the national currency of the seller's country.

The payment norm presented above is abstract and vague, and therefore, applied for general purposes. In order to cause any effect in a regulated system, abstract norms must be translated into concrete norms [Grossi and Dignum, 2004]. Thus, the abstract payment norm might be contextualized, by the system developer, as an environment norm and, then, concretized in its NMAS. For example, in *American* and *Japanese* supply-chain domains, the environment norm is concretized with the following instantiations:

> *A Concrete Environment Norm for Effecting a Negotiation:* Negotiations are obliged to be paid (i) in *USA*, with *American dollars (USD)*; and, (ii) in *Japan*, with *Japanese Yen* (*JPY*).

In the contextual normative level, the classificatory reading of '*counts-as*' from [Grossi *et al.*, 2006] is applied. The reading states that if "A counts-as B in context c", then, it is interpreted as "A is a sub-concept of B in context c". In this sense, '*counts-as*' statements work as '*contextual classifications*'.

For instance, considering the payment norm exemplified above, its reading is done as follows: "*USD* counts-as a *valid currency* in the context of the *USA environment*"; and its interpretation is done as follows: "*USD* is a sub-concept of a '*valid currency*' concept in the context of the *USA environment*".

Figure 4 illustrates part of the *Contextual Normative Level* of Figure 3 in which the '*valid currency*' variable of the *Negotiation* activity is instantiated for the payment norm according to the *American* and *Japanese* environments.

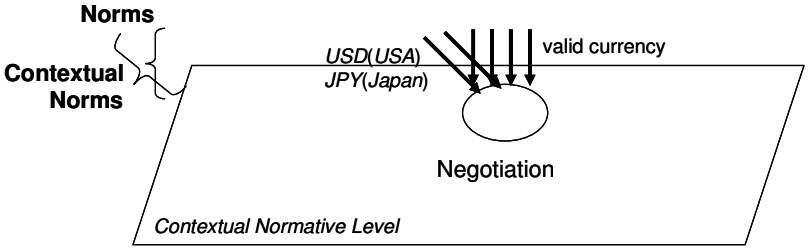

Figure 4 – Contextual classifications for a payment norm

Moreover, besides the instantiation of contextualized variables, it is considered that activities, in the contextual normative level, can also have different

predefined conditions. For instance, *Give Discount* (a sub-activity of *Negotiation*) states that, in an organization, discounts can be given (a) by subtracting *10%* of the price value for *orders paid in cash*, or (b) by subtracting *15%* of the price value for *products bought in bundles*.

3.1.2. Contextual Norm Classification

Basically, a MAS consists of environments, organizations and agents playing roles and interacting [Jennings, 2000]. As environments, organizations, roles and agent interactions are important concepts for the understanding of the text, thus, the meaning in which they are used in this thesis is characterized below.

Environments [Weyns *et al.*, 2007] are discrete computational locations, similar to places in the physical world, which provide conditions for agents to inhabit it. Environments can have refinement levels, such as a specialization relationship (*e.g.*, country and state), but there cannot be overlaps (*e.g.*, there cannot be two countries in the same place). An environment can also have many organizations.

Organizations [Ferber *et al.*, 2003] are social locations in which groups of agents play roles. An organization can embody many sub-organizations, but each organization belongs to only one environment [Silva and Lucena, 2004b]. Agents can execute in different organizations and they can also migrate among environments and organizations in order to obtain resources or services not found locally.

Roles [Thomas and William, 2005] are abstractions that prescribe a set of related tasks, which agents must perform in order to achieve their designed goals. Roles are defined by organizations independently of agents' individual identities.

An agent can interact with any other agent in a MAS, for example, by exchanging messages.

Environments, organizations, roles and interactions suggest different contexts for regulation in MAS. *Context-aware computing* means to be software that "*adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time.*" [Schilit and Theimer, 1994]. However, although contexts are tacitly known by most people, they are normally hard to be identified.

In order to help the system developer in his task of norm contextualization, DynaCROM follows directions taken by research in context-aware applications that suggest top-down architectures for classifying contextual information [Khedr and Karmouch, 1995; Henricksen and Indulska, 2005].

DynaCROM defines that norm information should be classified in a MAS according to the following contexts: *Environment*, *Organization*, *Role* and *Interaction*, which are differentiated by the boundaries of their data (*i.e.*, norms). *Environment Norms* are applied to all entities in a regulated environment. Likewise, *organization norms* are applied to all entities in a regulated organization; *role norms* are applied to all agents playing a regulated role; and, *interaction norms* are applied to all agents involved in a regulated interaction.

Figure 5 illustrates an example scenario in which entities of a MAS are influenced by the application of norms (represented by arrows) from different levels of abstractions (represented by dashed boxes). In the *Environment Normative Level* (upper level), environment norms are directly applied to environment instances (*e.g.*, *USA* and *Japan*). Likewise, in the *Organization Normative Level*, organization norms are directly applied to organization instances (*e.g.*, *Dellie*, *HPie* and *DellieJapan*); in the *Role Normative Level*, role norms are directly applied to role instances (*e.g.*, *ADellieSeller*, *AHPieSupplier* and *ADellieJapanManufacturer*); and, in the *Agent Normative Level*, interaction norms are directly applied to the agents that are interacting.

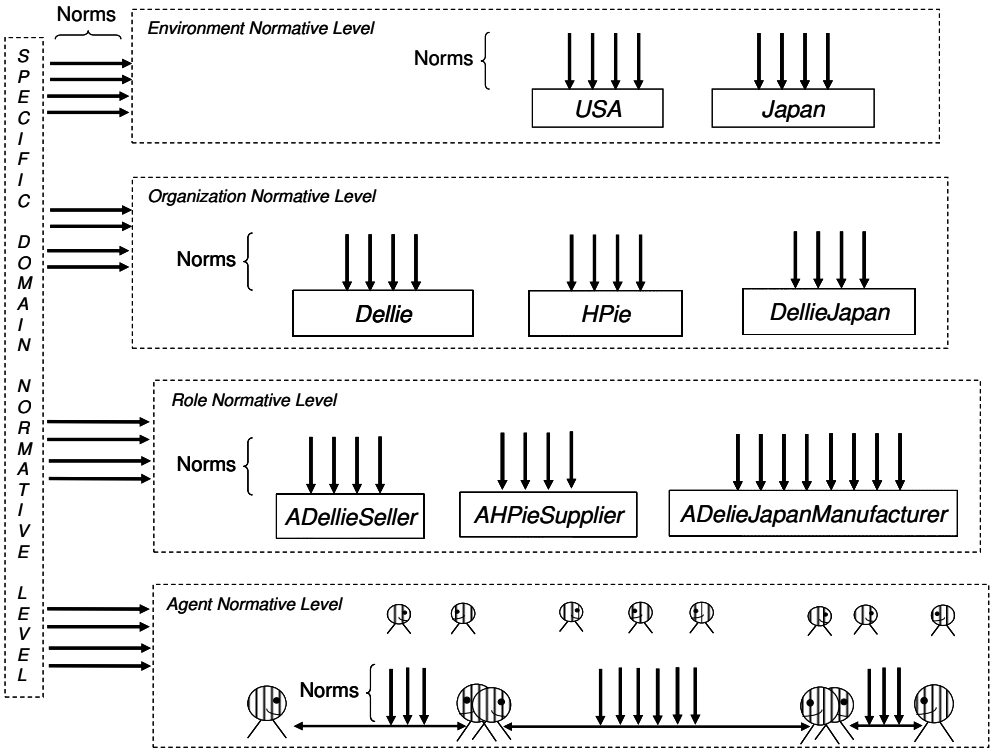

Figure 5 – Norms from different contextual normative levels

The four predefined normative contexts of DynaCROM are not targeted to a particular application domain; by so, they rather represent a basic set for a general regulation in NMAS. For a more precise regulation, this set should be improved through additions and refinements of application domain normative contexts and their respective norms. An example of a domain normative context and its norm might be, in the *Catholic* domain, a *Religious* concept that holds a (religious) norm stating that "*marriage is prohibited in the case that the man and/or the woman to be married made perpetual vows of chastity in a religious institute*".

Specific[6] domain norms (*e.g.*, political norms) can be directly applied in any other normative level, as also illustrated in the Figure 5 (by the horizontal arrows from the vertical rectangle in the left side of the figure). For instance, the political norm: "*American and Japanese organizations are forbidden to deal with each other when their countries are undergoing political crisis*" can be directly applied in the *Organization Normative Level* of *American* and *Japanese* organizations (*e.g.*, in the *Organization Normative Level* of *Dellie* and *HPie*, and in the *Organization Normative Level* of *DellieJapan*, respectively).

[6] '*Specific*' meaning '*particular*' and having '*general*' as its antonym (definition from the Roget's New Millennium™ Thesaurus [OnLineDictionary, URL]).

The extra effort of the system developer to classify norm information in more precise levels of abstraction is rewarded during the management phase and can also provide a fine-grained mechanism for norm enforcement solutions. Norms concretized in the contexts that directly affect the different MAS entities can be more easily found and updated because information is decoupled in pre-defined levels for norm classification. Besides that, each norm from the normative level of environments, organizations, roles or interactions can hierarchically influence each other, and each specific domain norm can transversally influence any norm from the other normative levels of a NMAS.

3.1.3. Contextual Norm Representation

In order to represent norms in a meaningful way for heterogeneous agents, the formalism to be used in a NMAS needs to be chosen. This choice must balance two major characteristics: *expressiveness* versus *efficiency*, considering that, from a software engineering perspective, agent responses in MAS should be *quick*, *automatic* and *reliable* [Breitman *et al.*, 2004].

Speed is a requirement intrinsic to most systems. A *quick* response means that, once a request is sent by an agent, its response should be given at system runtime, even if the request may have been sent to multiple recipient agents, which compete for time of response.

Interoperability among agents in MAS lead to executions that must be *automatic*, *i.e.*, that cannot count with user intervention. One reason for that is, while the designer of a MAS is a domain expert, its human users may not be. Moreover, in open systems, a minimum level of *reliability* is mandatory in order to build trust for its participants.

The number of related norms involved in a negotiation among agents can be extremely high. In this case, it is not reasonable to expect that all system norms will be investigated in each negotiation, not in a reasonable time frame.

For the goals of this thesis, it is accepted that the provision of a sub-set of relevant norms, where relevance is characterized by both agents' current contexts and actions performed, is a reasonable result if it is attained quickly, automatically and within reliable limits (predefined by the system developer). This way, information does not need to be stored since the relevant norms are provided, each time, at system/agents' requests.

3.1.3.1. Norm Representation by Using OWL

The *Web Ontology Language* (OWL) [Bechhofer *et al.*, URL], a Web standard from the *World Web Consortium* (W3C), was analyzed in order to verify its applicability for norm representation in open MAS. OWL was chosen mainly because of the two following reasons.

The first reason is because OWL represents information in a meaningful way (*i.e.*, with a common understanding) for heterogeneous agents, supporting them in their processes of data retrieving and integration with different sources. This way, information can be understood by computer applications, instead of only by humans.

The second reason for choosing OWL is because it provides three sublanguages, which are differentiated by their levels of expressiveness: OWL Lite, OWL DL (includes OWL Lite) and OWL Full (includes OWL DL).

OWL Lite was designed for easy implementation and to provide users with a functional subset that permits a classification hierarchy and simple constraints.

OWL DL (where DL stands for *Description Logic*) was designed to support those users who want the maximum expressiveness without losing computational completeness (*i.e.*, all entailments are guaranteed to be computed) and decidability (*i.e.*, all computations will finish in finite time) of reasoning systems. OWL DL is so named due to its correspondence with description logics [Baader *et al.*, 2003], a field of research that has studied a particular decidable fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems.

OWL Full is meant for users who want expressiveness with no computational guarantees. In this case, OWL Full relaxes some of the constraints on OWL DL so as to make available features which may be of use to many database and knowledge representation systems, but which violate the constraints of Description Logic reasoners. Thus, it is unlikely that any reasoning software will be able to support every feature of OWL Full.

In order to meet the software engineering requirements for responses in NMAS (*i.e.*, expressiveness in computational completeness and decidability) and based on the characteristics of each OWL sublanguage, OWL DL was the chosen sublanguage for representing the domain data of the usage scenarios pre-

sented in this thesis. Therefore, *formal* versus *non-formal* issues about those usage scenarios are restricted to the available properties of the sublanguage.

3.1.3.2. Declarative Specifications of Concrete Norms

DynaCROM proposes a *contextual normative ontology* for declarative specifications of norms, providing information with a common understanding about well-defined system regulation to heterogeneous agents.

An *ontology* is a conceptual model that embodies shared conceptualizations of a given domain [Gruber, 1993]; a *contextual ontology* is an ontology that represents localized domain information [Bouquet *et al.*, 2003] (*e.g.*, *USD* is the national currency of *USA*); and, a *contextual normative ontology* is a contextual ontology that has a *Norm* concept as its central asset. The *Norm* concept should be instantiated with norms contextualized differently according to basic MAS entities (*i.e.*, environments, organizations, roles and agent interactions) or specific domain entities.

The DynaCROM contextual normative ontology, hereinafter the DynaCROM ontology, defines the following five related concepts, all in the same hierarchical level: *Role*, *Organization*, *Environment*, *Norm* and *Action*, as illustrated in Figure 6[7]. These concepts must be instantiated according to the application domain of a NMAS proposed.

In the DynaCROM ontology, the *Role* concept encompasses the instances of all regulated roles of the system and each role instance is associated with its norms (via the *hasNorm* property) and with its organization (via the *isPlayedIn* property). The *Organization* concept encompasses the instances of all regulated organizations and each organization instance is associated with its norms, with itself (via the *hasMainOrganization* property for representing its main organization) and with its environment (via the *isIn* property). The *Environment* concept encompasses the instances of all regulated environments and each environment instance is associated with its norms and with itself (via the *belongsTo* property for representing its owner environment). The *Norm* concept encompasses the instances of all norms and each norm instance is associated with its regulated

[7] For readability purposes, ontology is presented graphically by using the Ontoviz graph plug-in for OWL [Ontoviz, URL].

actions (via the *regulates* property). The *Action* concept encompasses the instances of all regulated actions of the system.

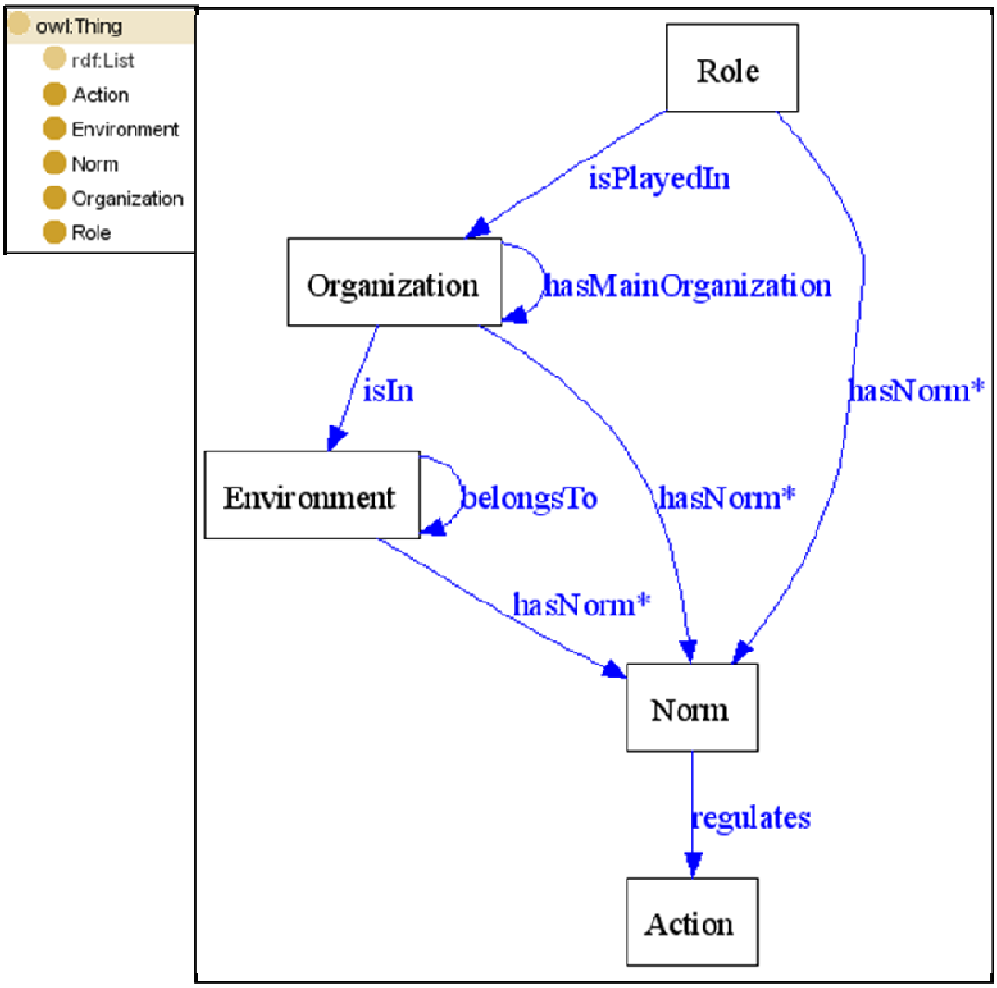

Figure 6 – The DynaCROM ontology

The DynaCROM ontology is an extensible one, *i.e.*, its basic concepts can be extended and/or new domain concepts can be created, both for representing classified contextual domain information. More precisely, the representation of a concrete norm in a DynaCROM ontology should be done by extending existing concepts or by creating new ones, then, instantiating the concept with norm information and, at last, linking the regulated instances to its related abstract norm (represented as a created norm instance).

For example, Figure 7[8] illustrates the *OblToPayWithNationalCurrency* norm instance that represents an abstract payment norm. The norm is concretized in each environment by instantiating its domain datatype property *hasNationalCurrency* (*e.g.*, *JPY (Japanese Yen)* in *Japan* and *USD (U.S. Dollar)* in *USA*), which extends the DynaCROM *Environment* concept (originally presented in Figure 6).

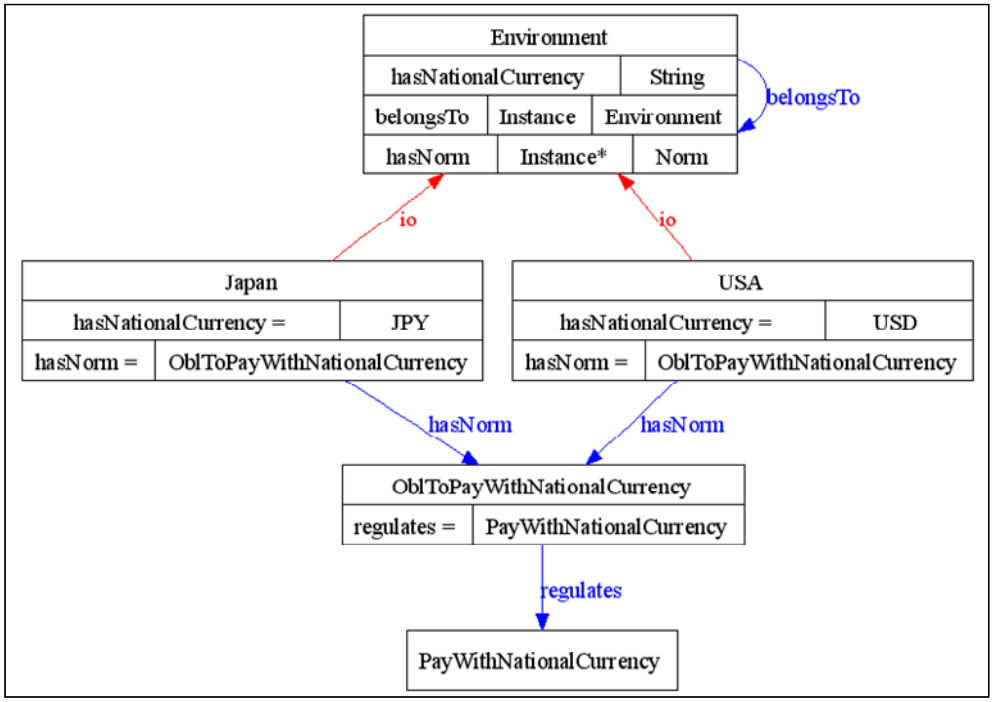

Figure 7 – An abstract payment norm concretized in *Japan* and *USA*

For a more precise regulation in NMAS, specific domain contexts should be represented in an application domain DynaCROM ontology through additions and refinements of their related concepts and norms. An example of an application domain context and its norm can be:

> *A Political Norm for Regulating Deals:* organizations are prohibited from dealing with each other when their countries are undergoing political crisis.

The political norm presented above is an abstract interaction norm. Interaction norms should be concretized in a DynaCROM domain ontology by instantiating its *Norm* sub-concept, which must be already created for linking the other

[8] In the Ontoviz graph plug-in, '*io*' means *'instance of'*; '*io'* is the label given for the link between a concept and its instance. Just for anticipating already, '*isa*' is the label given for the link between a concept and its sub-concept.

concepts from the relation (*i.e.*, reification of relationship). This solution follows the representation pattern presented in [Noy and Rector, URL].

Figure 8 illustrates the abstract political norm represented by the *PrhToDealWith* (a *Norm* sub-concept) and concretized in *AmericanOrganizations* by the *PrhToDealWithJapaneseOrganizations* norm instance. The concrete norm prohibits *AmericanOrganizations* to deal with *JapaneseOrganizations* when their countries are undergoing political crisis.

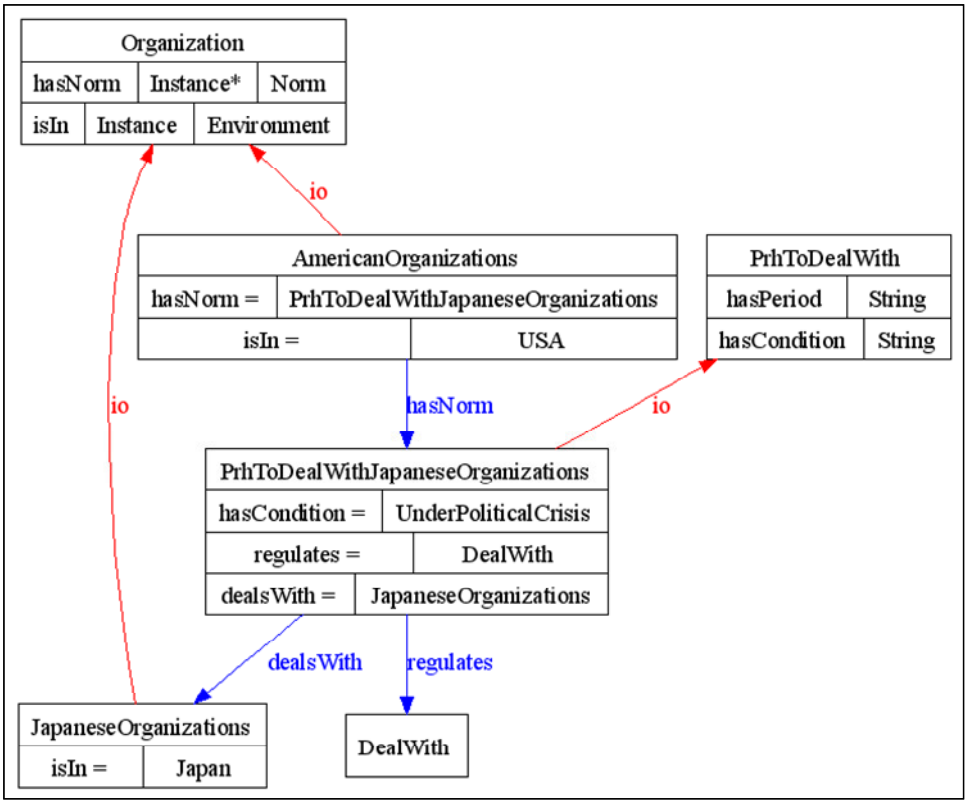

Figure 8 – A political norm concretized in *American* organizations

3.1.4. Contextual Norm Composition

After classifying and representing norms in precise levels of abstractions, contextual norms can be composed during system execution since, at any given moment, an agent may be related to norms defined at one or more normative contexts. Compositions of related contextual norms result in sets of independent norms, in which the semantic of one norm can influence the semantics of the others. For instance, the environment norm presented below is considered:

A Concrete Environment Norm for Calculating Prices: a state corporate income tax rate of *6.25* in *Missouri* is obliged to be imposed on all sales.

Figure 9 illustrates the environment norm for calculating prices in *Missouri*. Although *Missouri* and *USA* are hierarchical environments (defined via their *belongsTo* relationship), a mechanism should be used in order to effectively compose their norms in a DynaCROM NMAS.

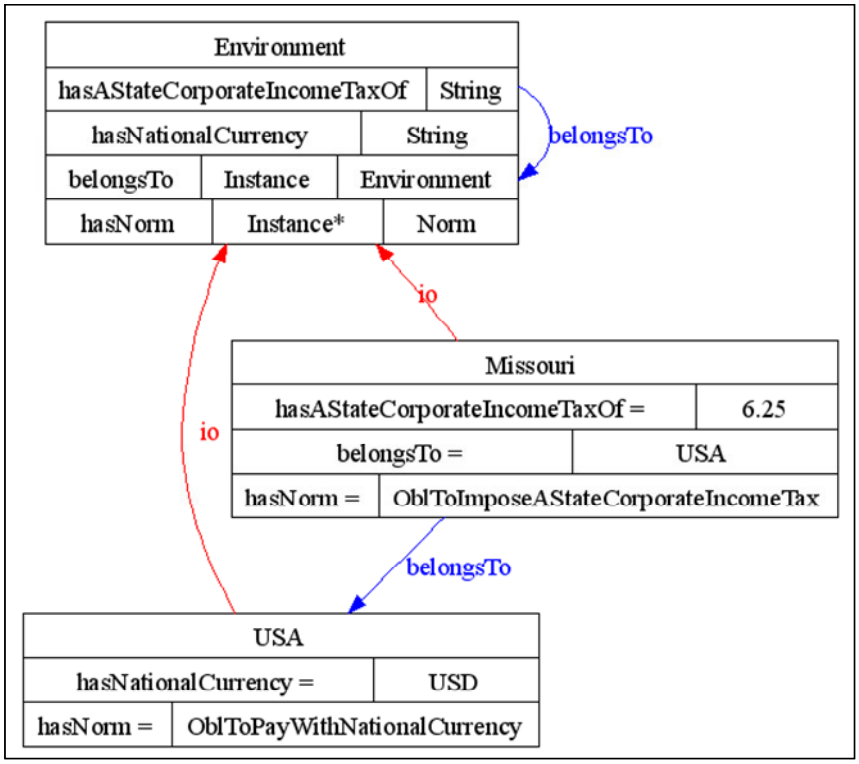

Figure 9 – The *Missouri* and *USA* hierarchical environments

DynaCROM follows rules to compose contextual norms. DynaCROM rules are *ontology-driven* rules, *i.e.*, they are created by the system developer, according to the ontology structure, and they are limited to the related concepts to which each concept is linked to.

Code 1[9] presents an example of rule that recursively compose the norms of hierarchical environments as, for instance, the norms of the *Missouri* and *USA* environments. More precisely, considering *Missouri* as an example of the given environment, the following composition process is executed, according to the

[9] The rules presented in this thesis are written following a syntax that was simplified for readability purposes. However, in Appendix B, an example of rule is given in the exact way as it was written to be used by the Jena [JENA, URL] rule inference engine.

domain ontology instance illustrated in Figure 9: in (4), the '*?OEnv*' variable is instantiated with the *USA* inferred value, when the '*?Env*' variable is instantiated with the *Missouri* given value; in (3), the '*?OEnvNorms*' variable is instantiated with the *OblToPayWithNationalCurrency* inferred value; and in (2), the inferred norm is added as a new norm of *Missouri*.

The result of the norm composition process is that, in *Missouri*, all negotiations are obliged to be paid with *USD* and increased by a state corporate income tax of *6.25*.

Code 1. A DynaCROM rule to compose the norms of hierarchical environments

```
(1)  [DynaCROMRule_EnvWithOEnvNorms:
(2)    hasNorm(?Env,?OEnvNorms)
(3)     <- hasNorm(?OEnv,?OEnvNorms),
(4)        belongsTo(?Env,?OEnv)]
```

DynaCROM predefines the rules to compose the norms of hierarchical environments (explained above) and also the others presented in Code 2. Inputs for these rules are domain instances of the *Organization* and *Role* concepts and their outputs are compositions of related contextual norms.

Following a norm composition process similar to the one explained for the '*DynaCROMRule_EnvWithOEnvNorms*' (presented in Code 1), the '*DynaCROMRule_OrgWithMOrgNorms*' (line 5 to 8 from Code 2) states that a given organization will have its norms composed with the norms of its main organization; the '*DynaCROMRule_OrgWithEnvNorms*' (line 9 to 12 from Code 2) states that a given organization will have its norms composed with the norms of its environment; and, the '*DynaCROMRule_RoleWithOrgNorms*' (line 13 to 16 from Code 2) states that a given role will have its norms composed with the norms of its organization.

Code 2. DynaCROM rules to compose the norms of normative contexts

```
(5)  [DynaCROMRule_OrgWithMOrgNorms:
(6)    hasNorm(?Org,?MOrgNorms)
(7)     <- hasNorm(?MOrg,?MOrgNorms),
(8)        hasMainOrganization(?Org,?MOrg)]

(9)  [DynaCROMRule_OrgWithEnvNorms:
(10)   hasNorm(?Org,?OrgEnvNorms)
(11)    <- hasNorm(?OrgEnv,?OrgEnvNorms),
(12)       isIn(?Org,?OrgEnv)]

(13) [DynaCROMRule_RoleWithOrgNorms:
(14)   hasNorm(?Role,?OrgNorms)
(15)    <- hasNorm(?Org,?OrgNorms),
(16)       isPlayedIn(?Role,?Org)]
```

Rules can compose data from the same concept type (*e.g.*, the '*DynaCROMRule_EnvWithOEnvNorms*' and the '*DynaCROMRule_OrgWithMOrgNorms*') or from different concept types (*e.g.*, the '*DynaCROMRule_OrgWithEnvNorms*' and the '*DynaCROMRule_RoleWithOrgNorms*'). Rules can also compose data from concepts directly related (hierarchical form) or indirectly related (non-hierarchical form).

Code 3 presents an example of a rule that compose the norms of the DynaCROM *Role* and *Environment* concepts, which are examples of indirectly related concepts.

Code 3. A rule for composing the norms of two indirectly related concepts

```
(17) [DynaCROMRule_RoleWithOrgEnvNorms:
(18)   hasNorm(?Role,?OrgEnvNorms)
(19)    <- hasNorm(?OrgEnv,?OrgEnvNorms),
(20)       isIn(?Org,?OrgEnv),
(21)       isPlayedIn(?Role,?Org)]
```

For the composition process, DynaCROM employs[10] an inference rule engine that executes the following tasks: (i) read an ontology instance to get data (*i.e.*, concept instances and their relationships), (ii) read a rule file to retrieve the information about how concepts must be composed; and then, (iii) infer an ontology instance based on the previous readings. An overview of this composition process is illustrated in Figure 10.

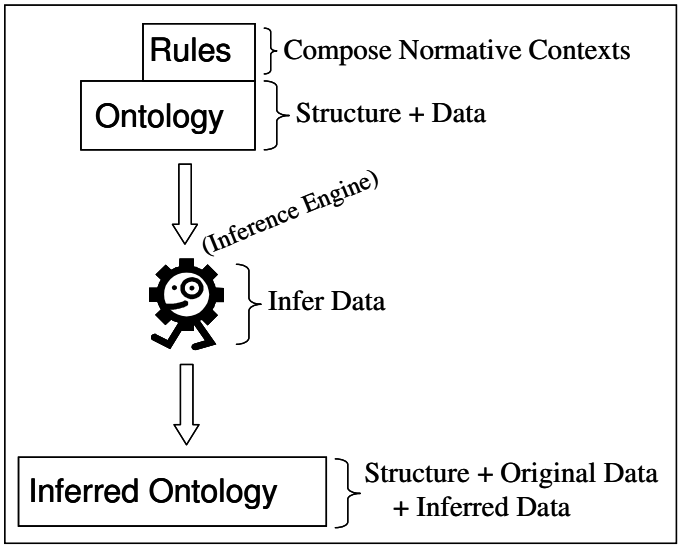

Figure 10 – The DynaCROM composition process

[10] '*Employ*' meaning "*to make use of (an instrument, means, etc.); use; apply*" [OnLineDictionary, URL]).

Once the domain ontology and/or rule file change(s), updated information is automatically forwarded to agents in the next DynaCROM execution. This makes it possible for managements in the system to be done at runtime, providing the dynamicity and flexibility necessary for regulation and also regarding social changes characteristic of MAS. These achievements for norm management are gotten because all norms provided by DynaCROM are applicable at a given moment.

In the current implementation of DynaCROM, the Jena rule-based inference engine [JENA, URL] is used for the composition process, however, other Semantic Web reasoners, like Racer [RACER, URL], Pellet [PELLET, URL] or FaCT [FACT, URL], can also be used.

In the composition process, it will still be the system developer's responsibility to write rules in the exact order he wants to compose his system data. The chosen inference engine will read/interpret those rules in sequence, from the top to the bottom of the rule file.

3.2. The DynaCROM Information Mechanism for Application Agents

3.2.1. DynaCROM Incorporation in Third-Party Agents

DynaCROM is currently implemented as a self-contained JAVA solution, encapsulated as an active JADE behavior. This way, the DynaCROM code is not scattered inside agents or regulated NMAS, fostering modularity. Furthermore, DynaCROM is a non-invasive approach in the sense that agents are developed independently of it; only a normative behavior is spontaneously added inside agents, so that, they become aware of the system norms.

The JADE platform was chosen because it is probably the widest used platform for MAS in academic circles and also because it is in conformance with the FIPA specifications. However, DynaCROM can be applied to agents from other platforms with little effort – it would be only necessary to encapsulate a facade design pattern using the agent implementation unit provided by the chosen platform.

Code 4 introduces part of the JAVA code for the incorporation of the DynaCROM behavior inside agents. The '*addBehavior*' JADE method must be filled

with three parameters. The first parameter is for the agent code itself; the second parameter is for the agent unique name (*e.g.*, "*AMissourianSupplier*"); and the third parameter is for the role that the agent will play in a DynaCROM NMAS (*e.g.*, a "computer-supplier").

Code 4. Adding DynaCROM in a third-party JADE agent

```
(1) public class ThirdPartyAgent extends Agent{…
(2)  protected void setup(){
(3)   addBehavior(new DynaCROM(this,"AMissourianSupplier",
                                "computer-supplier"));
(4) …}
```

3.2.2. Openness in a DynaCROM NMAS

Agents executing in NMAS are heterogeneous, implemented by different third-party developers, with code that is inaccessible. A viable solution for regulation in NMAS should not be hard coded inside agents' original codes and it must allow some flexibility for updating data (*e.g.*, norms) during the system execution [Grizard *et al.*, 2006].

Openness is achieved in a DynaCROM NMAS because only abstract classes and methods for the domain are specified by the system developer. Agents need to choose which role they will play in the NMAS and, according to the chosen role, they need to implement the roles' respective methods (*i.e.*, agents only need to know how to act in the NMAS in which they will perform). The methods from the NMAS can be freely implemented by agent developers upon the condition that they use the exact names given by the system developer for the defined entry parameters and returned values. Thus, the collective acceptance of agent developers, while implementing their agents, is necessary in order to make a NMAS a real application.

In order to execute correctly, *i.e.*, according to updated system information, agents in a DynaCROM NMAS can request DynaCROM about domain data by calling its predefined *getDynaCROMInfo(…)* method. The DynaCROM method can be called with one or two parameters filled.

In the case of only one parameter filled, the *getDynaCROMInfo(String infoRequested)* method is called with its parameter filled with the desired information, which must be represented in the domain ontology instance. Then, DynaCROM returns the parameter values based on the ontology concepts that have their

attributes matching one of the masks "*infoRequested*" or "*has + infoRequested*". Those returned values are related to the agent that called the method.

In the case of two parameters filled, the *getDynaCROMInfo(String infoRequested, String analyzedAgent)* method is called with its second parameter filled with the information about the agent to be analyzed. Then, DynaCROM returns the values of the first parameter related to that agent.

Code 5 exemplifies the use of the *getDynaCROMInfo(infoRequested, analyzedAgent)* method. The method is called inside a domain method that must be implemented by seller agents in order to inform prices in their NMAS. The DynaCROM method is called twice (see lines 3 and 4).

In the first case, the DynaCROM method is used by a seller agent to request DynaCROM about the national currency of the sender of the message received (see line 3). In this case, the '*infoRequested*' parameter is filled with *NationalCurrency*, the exact name that matches the mask "*has + infoRequested*" of the domain ontology instance illustrated in Figure 7.

In the second case, the method is used to request DynaCROM about the state corporate income tax of the sender of the message received (see line 4). In this case, the '*infoRequested*' parameter is filled with *AStateCorporateIncomeTaxOf*, the exact name that matches the mask "*has + infoRequested*" of the domain ontology instance illustrated in Figure 9.

Code 5. A method to be implemented by seller agents

```
(1)public abstract class APlanForSellerAgt{
(2) public String proposeAPrice(…){
(3)  String currency = getDynaCROMInfo(NationalCurrency,
                                              msg.sender);
(4)  String stateIncomeTax = getDynaCROMInfo(AState-
                        CorporateIncomeTaxOf,msg.sender);
(5)  String finalPrice = calculatePrice(currency,
                                           stateIncomeTax);
(6)… return finalPrice; } }
```

Considering, for example, a *Missourian* customer agent that sends a CFP to a seller agent, then, the seller agent will be informed by DynaCROM about the national currency of *USD* and the state corporate income tax of *6.25*, both contextual values concretized according to the *Missouri* environment (see Figure 9).

Code 6 exemplifies a rule, written by the system developer, for adding the information about the national currency of each country from the domain ontology (*e.g.*, *USA*) in its states (*e.g.*, *Missouri*). DynaCROM automatically infers data, following the composition process illustrated in Figure 10.

Once any information is updated in the domain ontology instance, as for example, the migration to *Euros* as the national currency of almost all *European* countries (*e.g.*, in *France*, from *Francs* to *Euros*), the new national currency is automatically forwarded to seller agents in the next execution of DynaCROM, without the need to restart the system. This also happens if any update is done in the domain rule file for new compositions of domain data.

Code 6. Adding the national currency of a country in its states

```
(1)[DynaCROMRule_EnvWithOEnvNationalCurrency:
(2) hasNationalCurrency(?Env,?NationalCurrency)
(3)  <- hasNationalCurrency(?OEnv,?NationalCurrency),
(4)     belongsTo(?Env,?OEnv)]
```

3.3. Discussion

In this chapter, the DynaCROM methodology developed to support the system developer in the tasks of implementation, management and evolution of the norms of his NMAS is explained. The methodology includes the phases of contextualization, concretization, representation and composition of norms.

For the DynaCROM methodology, it was defined: (i) a top-down classification for contextual norms; (ii) a contextual normative ontology; and, (iii) a norm composition process.

The top-down classification for contextual norms proposed by DynaCROM facilitates the tasks of elicitation, organization and management of norms. The DynaCROM contextual normative ontology supports heterogeneous agents with a common understanding about the system norms. The norm composition process defined by DynaCROM makes it easy to update system regulation by both evolving norms in a unique resource (an ontology) and/or by customizing particular rules for different compositions of contextual norms.

In this chapter, the process of norm incorporation in the agents of a DynaCROM NMAS is also explained. Application agents should include the DynaCROM behavior in order to be aware of the applicable norms, according to their current contexts, of each action performed by them. This way, those agents can execute appropriately, *i.e.*, in compliance with updated system norms.

However, although application agents can be informed about their current (contextual) norms, by using the DynaCROM behavior, agents' developers can implement their agents regardless of this information. In this case, agents need a solution that continuously informs them about system data, according to their cur-

rent contexts, in order to deal with the applicable norms of each action performed by them.

In summary, the extra effort made by the system developer to deal with contextual norms, due to the need to be more precise, is rewarded during the management phase and can also provide a fine-grained mechanism for norm enforcement solutions, as will be presented in the next chapter.