Garcia Marques, ME.; Giret Boggino, AS.; Botti V. (2013). A Model-Driven CASE tool for developing and verifying regulated open MAS. Science of Computer Programming. 78(6):695-704. doi:10.1016/j.scico.2011.10.009.

A Model-Driven CASE tool for Developing and Verifying Regulated Open MAS

Emilia Garcia[1], Adriana Giret[1], Vicente Botti[1]

Universitat Politecnica de Valencia

Abstract

This paper describes a CASE tool for developing complex systems in which heterogeneous and autonomous agents may need to coexist in a complex social and legal framework. Model-Driven Technologies are used to integrate the design of systems of this kind with the verification of the models and with the generation of executable code from these models. The verification module is based on model-checking techniques to check the coherence of a modeled legal context at design time is presented and it is exemplified with a case study.

Keywords:
Multi-Agent Systems, Contracts, Model-Driven Software Development, Model Checking

1. Introduction

In today's bargaining scenarios, the e-Business approach is becoming more and more a "must-have" tool. Over recent years, several works have focused on solving the problem of integrating multi-agent system, service-oriented computing paradigms and normative environments in order to model autonomous and heterogeneous computational entities in dynamic, open environments [1]. In this context, the term *Open* means that external and internal agents of the system can interact with each other. In order to adapt

Email addresses: mgarcia@dsic.upv.es (Emilia Garcia), agiret@dsic.upv.es (Adriana Giret), vbotti@dsic.upv.es (Vicente Botti)

[1]Camino de Vera S/N, 46022, Valencia

systems of this kind to industrial environments, the agent social relationships, organizational behavior, agent interactions, and service interchanges must be regulated.

Over the last few years, the integration of electronic contracts in Organizational MAS is becoming increasingly more important to system architectures for agent behavior regulation [2]. This is because contracts are expressive and flexible. They allow agents to operate with expectations of the behavior of other agents based on high-level behavioral commitments, and they provide flexibility in how the autonomous agents fulfill their own obligations [3]. Furthermore, contracts allow the top-down specification of organizational structures to be integrated with the autonomy of participating agents [4]. For example, the rights and responsibilities that an agent acquires when it is playing a specific role in an organization can be formalized using norms and contracts.

Developing these systems is a very complex task because it requires defining of the global behavior of the system, the individual behavior of each agent, the legal context of each entity, and the social and contractual interactions. Also, many conflicts can arise from the potential combination of organizational norms and the specific restrictions of each agent derived from the commitments of their signed contracts. It is necessary to ensure that each single contract has no conflicts, and also that the composition of all the contracts is itself conflict-free. Therefore, automatic techniques are needed to develop and verify these systems.

In our work, we deal with the problem of engineering Regulated Open Multi-Agent Systems (ROMAS). They are systems in which heterogeneous and autonomous agents may need to coexist in a complex social and legal framework that can evolve to address the different and often conflicting objectives of the many stakeholders involved. Our architecture is based on Open Organizational Multi-agent Systems [5, 6]. In this way, organizations comprise both the integration of organizational and individual perspectives and the dynamic adaptation of models to organizational and environmental changes. In our proposal, agents interact between them by means of *Services* which represent the functionality that agents offer to other entities. Organizations impose limits on the actions that the agents can perform by means of *Norms* and *Contracts*. However, agents maintain their autonomy, so they can choose the actions to do next and select with whom to perform them. A complete description of this architecture and a formal meta-model for representing it can be found in [7].

Based on Model-Driven techniques, we have developed a CASE tool that models ROMAS systems, validates these models using model-checking techniques, and is prepared to automatically generate executable code for agent platforms such as THOMAS [8] or Electronic Institutions [9]. This paper presents the main architecture and functionalities of this tool.

The rest of the paper is organized as follows: Section 2 presents some background and related works in the context of Model-Driven Architectures and in the context of the Formal Verification of multi-agent systems (MAS) using model checking. Section 3 exemplifies our proposal by means of a case study and details the verifier module that checks the coherence of the legal context at design time. Finally, Section 5 presents some conclusions and future work.

2. Background and related work

2.1. Model-Driven Architecture and Eclipse technology

In the software engineering field, a model-driven software development process (MDD) should be clearly defined by specifying all the phases of the development lifecycle. Furthermore, CASE tools should be provided as support for the different tasks in the model-based design, such as analysis and verification of models or the automatic transformation from one specification language to another in a transparent and simple way.

The Model Driven Architecture initiative (MDA) [10] has proposed a standard for the metamodels of the specification languages used in the modeling process, which is known as the Meta Object Facility (MOF). This includes a set of requirements for the transformation techniques that will be applied when transforming a source model into a target model. This is referred to as the Query/View/Transformation (QVT) approach [11]. Basically, MDA proposes an approach to software development based on modeling and on the automated mapping of source models to target models. The models that represent a system and its environment can be viewed as a source model, and code can be viewed as a target model.

Following these MDA standards, the Eclipse Platform [12] is an open source initiative that offers a reusable and extensible framework for creating IDE-oriented tools. The Eclipse Platform itself is organized as a set of subsystems (implemented in one or more plug-ins) that is built on top of a small runtime engine. Plug-ins define the extension points for adding behaviors to

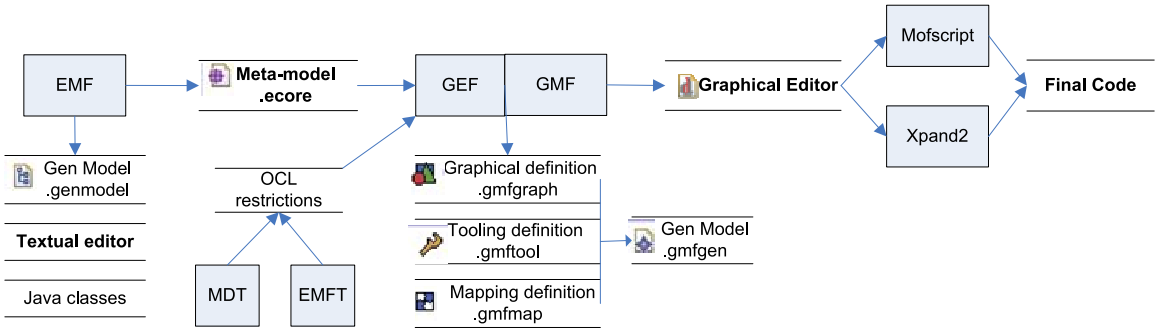

Figure 1: Eclipse plug-in structure

the platform, which is a public declaration of the plug-in extensibility. Figure 1 shows the most common plug-ins used to developed an Eclipse CASE tool:

- The Eclipse Modeling Framework (EMF) plug-in offers a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a metamodel specification described in XMI, Rational Rose, or the ECore standard (a variant of the MOF standard), EMF provides tools and runtime support to produce a set of Java classes for the model. EMF also provides the foundation for interoperability with other EMF-based tools and applications. Moreover, EMF generates a **textual modeler editor** from the metamodel

- The Graphical Editing Framework (GEF) and Graphical Modeling Framework (GMF) plug-ins allow developers to create a rich **graphical editor** from an existing ECore metamodel. These plug-ins allow the definition of the graphical elements that are going to be used in the generated tool. They also allow the definition of several views of the model and the palette of elements of each view. Finally, these plug-ins combine the graphical definition with the metamodel elements and with the different views of this metamodel, creating a complete modeling tool. These new tools are integrated into the platform through plug-ins that allow the definition of models based on the specification of the metamodels.

- The Xpand and Mofscript plug-ins offer a language to define matching rules between the ECore metamodel and another language. A plug-in

generated using Xpand or Mofscript consist in a set of transformations mapping rules between the entities and relationships of a metamodel defined in the ECore language and any other description language. These scripts are executed on an instance of the metamodel, i.e., on a user application model. These scripts have access to each entity and relationship of the model and match this information with the mapping rules defined at metamodel layer to generate the related code. Therefore, users can design their models using the graphical editor and execute this rules to **automatically generate code** from these models.

Since Model-Driven approaches have been recognized and become one of the major research topics in the agent-oriented software engineering community, we present how Eclipse MDA technology can be used for designing and verifying Regulated Open MAS. Some works like [13, 14, 15] show how MDA can be effectively applied to agent technologies. Furthermore, they show how the MDA technology can help to reduce the gap between the analysis stage and the final implementation stage.

2.1.1. Formal verification of MAS using Model Checking

Model checking is an area of formal verification that is concerned with the systematic exploration of the state spaces generated by a system. Model checking was originally developed for the verification of hardware systems, and it has been extended to the verification of reactive systems, distributed systems, and multi-agent systems.

The application of model-checking techniques to the verification of contract-based systems is an open research topic. Some works like [16] model contracts as a finite automata that models the behavior of the contract signatories. Other works represent them as Petri nets [17]. These representations are useful to verify safety and liveness properties. However, adding deontic clauses to a contract allows conditional obligations, permissions, and prohibitions to be written explicitly. Therefore, they are more suitable for complex normative systems like ROMAS. In [18] and [19] a deontic view of contracts is specified using the *CL* language. The work in [18] uses in model-checking techniques to verify the correctness of the contract and to ensure that certain properties hold. The work in [19] presents a finite trace semantics for *CL* that is augmented with deontic information as well as a process for automatic contract analysis for conflict discovery. In the context of Service-Oriented Ar-

chitectures, model checkers have recently been used to verify compliance of web-service composition. In [20] a technique based on model checking is presented for the verification of contract-service compositions.

In the context of verification techniques for MAS, there are some important achievements using model checking. In [21], the SPIN model checker is used to verify agent dialogues and to prove properties of specific agent protocols, such as termination, liveness, and correctness. In [22] a framework for the verification of agent programs is introduced. This framework automatically translates MAS that are programmed in the logic-based agent-oriented programming language AgentSpeak into either PROMELA or Java. It then uses the SPIN and JPF model checkers to verify the resulting systems. In [23], a similar approach is presented but it is applied to an imperative programming language called MABLE. In [24], the compatibility of interaction protocols and agents deontic constraints is verified. However non of these approaches is suitable for ROMAS since they do not consider organizational concepts.

There are only a few works that deal with the verification of systems that integrate organizational concepts, contracts, and normative environments. The most developed approach is presented in the context of the IST-CONTRACT project [2]. It offers contract formalization and a complete architecture. It uses the MCMAS model checker to verify contracts. However, as far as we know, it does not define the organizational normative context or verify the coherence of this context with the contracts.

The approach that we present here is distinct in that it designs and offers a module that allows: (1) the explicit formalization of social and commercial contract templates at design time;(2) the automatic translation of contract and norm descriptions into a verifiable model-checking language; (3) the verification at design time of whether a contract template contradicts the designed normative and legal environment.

3. ROMAS CASE tool

Following the MDA [10] standards by means of the Eclipse technology described in Section 2.1, we have developed a CASE tool for designing ROMAS systems. This tool is based on the ECore specification of the ROMAS formal metamodel described in [7]. This tool has been designed as an exten-

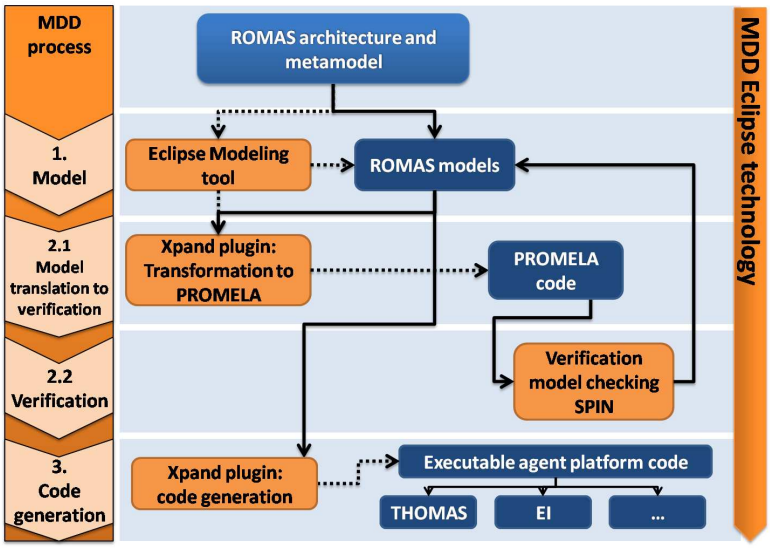

Figure 2: ROMAS development steps

sion of our previous work, the EMFGormas tool[2] [5]. ROMAS CASE tool extends EMFGormas with the notion of contracts and some other semantic entities derived from its metamodel. Moreover, ROMAS CASE tool adds a verification mechanism to check the coherence of the designed models.

This section presents the main architecture and how to use the ROMAS CASE tool to design and verify ROMAS. The main steps of a ROMAS development using this CASE tool are summarized in Figure 2: First, users model their applications by means of a graphical modeling tool based on the ROMAS formal metamodel (Section 3.1). Second, users can verify their designed models in two steps (Section 3.2): (1) Translate their ROMAS models into PROMELA code and LTL formulas which are the language of the model-checker SPIN; (2) Verify the coherence of the legal context at design time using a SPIN plug-in which is integrated in the CASE tool. If any conflict is detected, designers can revise their models and change them by means of the graphical editor. Finally, another Xpand plug-in can be used to generate code to an executable agent platform from the models generated using the graphical editor (Section 3.3).

[2] *http://users.dsic.upv.es/grupos/ia/sma/tools/EMFGormas/*

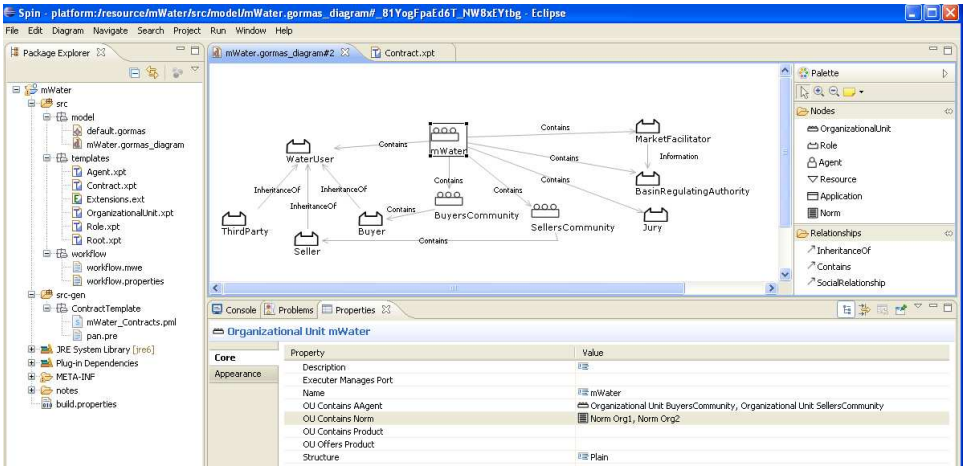

Figure 3: mWater Organization diagram

3.1. ***Model:Analyze and design the system***

As is detailed in [7], ROMAS metamodel can be instantiated by means of four different views that analyze the model from different perspectives. In order to facilitate the modeling tasks, the analysis and design of a ROMAS system is formalized by means of several diagrams that are instances of the ROMAS metamodel views:

- *Organizational External view:* defines the global goals of the organizations, the functionality that the organizations provide and require from their environment and their social structure.

- *Internal view:* defines the internal functionalities, capabilities, beliefs and objectives of each entity (organizations, agents, and roles) by means of different instances of this model.

- *ContractTemplate definition view:* defines *Contract Templates* following the syntax presented in Figure 7. Contract Templates are predefined restrictions that all final contracts of a specific type must fulfill. Contracts are inherently defined at runtime, but contract templates are defined at design time and can be used at runtime as an initial point for the negotiation of contracts.

- *Interaction/Task view:* defines both the interaction protocols and the sequence of activities in which a task is decomposed.

The modeling part of the CASE tool consists of several Eclipse plug-ins that offer one graphical editor for each view of the model. Figure 3 shows a snapshot of the tool in which an organization of a case study is modeled.

Therefore, the first step to develop a ROMAS system is to model it using the graphical CASE tool. This tool will internally generate an ECore model where all the information of the different diagrams will be saved. This ECore model will be used in the next steps to verify the model and to generate code.

3.2. ***Verification of the model***

This step of the process consists in verifying the correctness, completeness, and coherence of the designed model. Although the modeling tool restricts the model to the syntax that is defined in the metamodel, many conflicts such as the coherence between agents' goals and the goals of their organization can arise. At the moment, we deal with the conflicts related to the incoherences between the designed contract templates and the organizational norms. Model-checking techniques are used to verify the models. The verification process is executed in two steps:

1. *Model translation to verification.* The modeled system is translated into a language that can be verified using model checking. As is illustrated in Section 2.1, the Xpand language of the Model to Text (M2T) project [25] helps developers to translate models that are defined using the ECore standard into other languages by means of some mapping rules defined at metamodel layer. Since we use the SPIN model checker [26] to verify the models, the models should be transformed into the PROMELA language and Linear Temporal Logic (LTL) formulas. Based on the Eclipse transformation plug-in Xpand, an Eclipse plug-in called RO2P (ROMAS to PROMELA code transformation) has been developed in order to automatically translate the ROMAS designs into the PROMELA verifiable language and LTL formulas.
   The objective of RO2P is to verify that there is no conflict between the organizational norms, agent norms, and contract template designs. As is presented in [19], conflicts in contracts and norms arise for four different reasons: (1) the obligation and prohibition to perform the same action; (2) the permission and prohibition to perform the same action; (3) obligations of contradictory actions; (4) permissions and obligations of contradictory actions. At the moment, RO2P generates code that verifies the first and the second conflict scenarios. The last

two scenarios need semantic analysis of the ontology which is part of our future work.

Despite that, the SPIN model checker includes a large number of techniques for improving the efficiency of model checking such as state-space compression and partial-order reduction, translating the entire modeled system into a unique PROMELA code from which all the properties of the system could be verified, increase exponentially the complexity of the verification. The RO2P is designed to translate only the part of the model that is related to the property to be verified. The parts of the model that are related to the property *"Coherence between contract templates and their normative context"* are the contract templates and the norms that concern to the normative context of these contracts. The transformation rules for these entities are detailed in Section 4.

Therefore, to generate a verifiable file from the ROMAS models generated with the graphical editor, users only need to select their model and click on the right-click menu "*Generate PROMELA file*". Then, RO2P will transform the ECore file that contains all the models information into a PROMELA program and a set of verifiable LTL formulas.

2. *Execute the model checker.* The SPIN formal verification of the model is directly run from the modeling tool. It is possible thanks to the Eclipse plug-in[3] [27], which has been integrated into our CASE tool. It is used to verify the PROMELA code generated in the step (a). To run the verification the user only need to select the generated PROMELA file and click on the SPIN right-click menu. After the verification, if there is any incoherence, the designer must redesign the application model by means of the graphical modeling editor, and run again the verification process.

3.3. ***Generate the code for the execution platform.***

The CASE tool architecture is prepared to integrate other plug-ins based on Xpand to generate transformation mapping rules expressed at metamodel layer to translate from the models designed with the modeling tool to a programming language of an agent platform. Currently, we are developing a translator from ROMAS models to executable code for the Thomas platform [8] which is an agent platform that supports the description of organizations

[3] *http://lms.uni-mb.si/ep4s/*

and normative environments. However, any other transformation plug-in could be developed to translate the model to other normative agent platforms.

4. Case study and RO2P transformation rules

In order to illustrate the usage of the ROMAS CASE tool, a case study based on a virtual water market has been developed. Besides, a simplification of this case study is used as a running example to detail the Xpand transformation mapping rules between the ROMAS metamodel and the SPIN verifiable language (PROMELA code and LTL formulas).

4.1. ***Model:Analyze and design the system***

The case study is called *mWater* [28]. Let's suppose there is a water market that is an institutional, decentralized framework where users with water rights are allowed to voluntarily trade their water rights fulfilling some pre-established rules. In order to obtain a clear scenario to show how a ROMAS model is verified with the ROMAS CASE tool, a simplified scenario of this case study is presented. A complete description of this case study modeled with ROMAS is presented in [7]. As is described in Section 3.1, a ROMAS model can be defined using different views of the metamodel. This simplified scenario is represented by means of three diagrams:

- *mWater Organization internal view:* Figure 4 shows a simplification of the *mWater organization internal view* diagram where only the entities and relationships that are related to the verification of the normative context are shown. There is an organization called *mWater* that contains two types of roles *Sellers* and *Buyers*. This organization restricts the behaviour of these roles by means of two organizational norms:

  - *Org1 mWater norm*: It specifies that there is a minimum and maximum price for the water, (i.e., it is forbidden to pay more than 50 euro/kL or less than 1 euro/kL).

  - *Org2 mWater norm*: It specifies that any agent who is playing the *Buyer* role cannot offer services to other agents in exchange of water. This rule forces that agents only can exchange water with money.

- *Buying water rights contract:* This contract template indicates that there are two signatory parties who play the role *Buyer* and *Seller*,

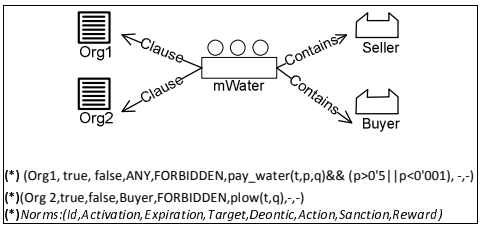

Figure 4: mWater Organizational Internal View

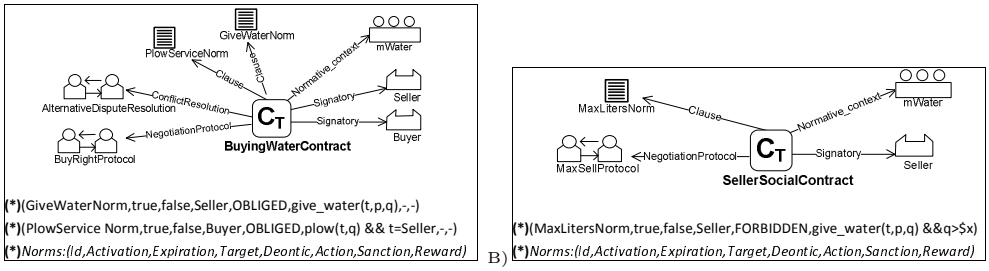

Figure 5: A)Buying Water Contract Template B)Role Seller Social Contract template

respectively, inside the same organization called *mWater*. This contract template has two clauses that specify that the Buyer should plow the field of the Seller in exchange of the water. The formal specification of each norm is presented in Figure 5.A and it follows the syntax defined in Figure 7.

- *Role Seller Social Contract template:* Figure 5.B represents the contract template diagram indicates that any agent who wants to play the *Seller* role must sign a contract that explicitly says the maximum number of liters of water that this agent can sell (Norm *MaxLiters*). The number of liters is defined at runtime during the negotiation between the agent and the organization.

4.2. ***Verification of the model***

The initial objective of this step is to check the coherence between the commitments defined in the contract templates and their normative context. The verification of other properties and the generation of the agent execution platform code are part of our current and future work.

1. *Model translation to verification.* This section presents the Xpand transformation mapping rules between the ROMAS metamodel and the SPIN verifiable language (PROMELA code and LTL formulas). A plug-in based on

Xpand consist in a set of mapping rules defined at metamodel layer. ROMAS metamodel is specified using the ECore language. Thus, the metamodel has a main entity from which all the other entities and diagrams of the metamodel are accessible. The main routine of the RO2P plug-in is presented in Figure 6:

1. Figure 6, lines 4 to 6: Global variables are defined.
2. Figure 6, line 8: the *fillLists* routine is invoked in order to navigate the whole metamodel extracting the entities related to the property to be verified. The property to be verified is *"Coherence between contract templates and their normative context"*, therefore the entities that must be transformed are the *Contract templates*, the *Executers* that are involved in these contracts (agents, roles and organizations), and the *Norms* that concern the normative context of these contracts. The syntax of *Contract templates* and *Norms* is presented in Figure 7. The deontic attribute of the Norms indicates that they can be obligations, permissions or prohibitions. Norms of permission can only produce a conflict if there is a prohibition over the same action. Therefore, to create the verification model, we assume that the agent actually performs the action. This means that permission norms are modeled as obligation norms.
   In our case study there are two contracts (*BuyingWaterContract* and *SellerSocialContract*), two entities (the role *Seller* and *Buyer*), two obligations norms specified in the contract template *BuyingWaterContract*, and three forbidden norms (two organizational norms and one defined in the contract template *SellerSocialContract*).

```
1  «IMPORT romas »
2  «DEFINE root FOR romas_model»
3     «REM» /*********   VARIABLES  **************/ «ENDREM»
4     «LET (List[ContractTemplate]) {} AS contractList »
5     «LET (List[Executer]) {} AS executerList »
6     «LET (List[Norm]) {} AS nForbiddenList »
7     «LET (List[Norm]) {} AS nObligedList »
8     «REM» /*********   Prepare the lists **********/ «ENDREM»
9     «EXPAND fillLists(contractList, executerList,nForbiddenList,nObligedList) FOR this»
10    «REM» /*********   Translate Forbidden Norms **********/ «ENDREM»
11    «FILE "LTL_Norms" + ".pml" »
12        «EXPAND writeForbiddenNorms FOR nForbiddenList »
13    «ENDFILE »
14    «FILE "PromelaFile" + ".pml" »
15        «EXPAND writeContracts FOR contractList »
16        «EXPAND writeExecuters FOR executerList »
17        «EXPAND writeInit(contractList, executerList)»
18     «ENDFILE »
19 «ENDLET » «ENDLET » «ENDLET » «ENDLET » «ENDDEFINE»
```

Figure 6: Xpand script: Main routine

```
<ContractTemplate>::=<ID>[<Description>][<Activation>][<Expiration>]
                      <Normative_context><Signatory_parties><Clauses>
                     [<NegotiationProtocol>][<ExecutionProtocol>][<ConflictResolutionProtocol>]
<Normative_context>::={organization_id}
<Signatories_parties>::= {<entity>}
<entity>::=<agent>:<role>:<organization>|<role>:<organization>|<organization>
<agent>::= NONE | ALL | agent\_id
<role>::= NONE | ALL | role\_id
<organization>::= NONE | ALL | organization\_id
<Clauses>::= {<Norm>}

<Norm>::=<ID>[<Activation>][<Expiration>]<Target><Deontic><Action>[<Sanction>][<Reward>]
<Deontic>::=OBLIGED | FORBIDDEN | PERMITTED
<Target>::= {<entity>}
```

Figure 7: Contract Template syntax

```
65 «DEFINE writeForbiddenNorms FOR List[romas::Norm]»
66   «FOREACH this AS item»
67     /*Norm «item.Id»:: «item.Description» */
68          ltl «item.Id» {[]! («item.Activation» && (!«item.Deactivation») && «item.Action»)
69   «ENDFOREACH»
70 «ENDDEFINE »
```

Figure 8: Xpand script: writeForbiddenNorms routine

3. Figure 6, lines 11 to 13: the routine *writeForbiddenNorms* is invoked in order to translate the norms whose deontic attribute indicates prohibition as LTL formulas. These formulas are saved in a file called *"LTL_Norms.pml"*. The code of the routine *writeForbiddenNorms* is presented in Figure 8. Figure 8 line 68 indicates that never occurs that the action that the norms forbids occurs when this norm is active and it has not been deactivated.
   For example, the norm Org2 mWater norm which is formalized as: (Org 2,true,false,Buyer,FORBIDDEN,plow(t,q),-,-) considering the following syntax: (Id, Activation, Expiration, Target, Deontic, Action, Sanction, Reward), is translated to LTL as: Org2 []!(Buyer_task==plow). This norm does not have activation or expiration conditions, so the LTL formula only express that is not possible that a *Buyer* executes the task *plow*.
4. Figure 6, lines 14 to 18: Executers involved in the process translation, Contracts templates translations and the PROMELA initial process are saved in the file *PromelaFile.pml*.

   - Each **Executer** is represented by an active process (Fig.9 line 119). The core of this process is a loop that checks its pending tasks and simulates its execution. Each party stores its pending tasks in a channel, which is a global variable that is accessible for all the processes (Fig.9 line 113). If an agent is obliged to

```
110 «DEFINE writeExecuters FOR List[romas::Executer]»
111  #define max_tasks 5
112 «FOREACH this AS execu»
113 chan «execu.Name»_Pending = [max_tasks] of {mtype, mtype}
114  mtype «execu.Name»_task;
115  mtype «execu.Name»_param1;
116  mtype «execu.Name»_param2;
117
118 /*Signatory party «execu.Name»::«execu.Description»*/
119 proctype «execu.Name»(){
120   end:
121   do
122         :: «execu.Name»_Pending ?? «execu.Name»_normId , «execu.Name»_Action
123   od;
124 }/*end «execu.Name»*/
125 «ENDFOREACH»     «ENDDEFINE »
```

Figure 9: Xpand script: writeExecuters

```
23 proctype Buyer(){
24 end:
25         do
26         :: Buyer_Pending ?? Buyer_task , Buyer_target;
27         od;
28 }/*end Buyer*/
```

Figure 10: mWater Buyer role in PROMELA

execute a service, it is supposed to do that. Thus, the action of the norm to the channel of the corresponding agent that is simulating the execution. Figure 10 shows an the PROMELA code of the executer role *Buyer*.

- Each *Contract Template* is specified as a PROMELA process (Fig.11 line 83). Figure 12 presents the PROMELA code for the *BuyWaterRightContrat template*. The status of a contract is represented with a global variable (Fig.11 line 81 - Fig. 12 line 31). The **Expiration** condition of a contract is represented as the escape sequence of an *unless* statement which includes all the tasks of the contract. This means that if the expiration condition is satisfied, the contract will interrupt its execution (Fig.11 line from 89 to 93 - Fig. 12 line 41). Each obligation and permission clause adds the action of the norm to the channel of the corresponding Executer that is simulating the execution (Fig.11 line 96 - Fig. 12 lines 37 and 38).
- The **Init process** is the first process that is executed in a PROMELA code. This process launch the executers processes and the contract processes when they are activated. Figure 13 shows the Xpand code to generate this process.

2. *Execute the model checker.*
After generating the PROMELA code. The SPIN model checker is

```
78  «DEFINE writeContracts FOR List[romas::ContractTemplate]»
79  mtype ={no_initated, executing, finished, interrumped}
80     «FOREACH this AS cont»
81  mtype «cont.Id»_state=no_initiated;
82  /*ContractTemplate «cont.Id»::«cont.Description»*/
83  proctype «cont.Id»(){
84     «cont.Id»_state=executing;
85    {«LET (List) cont.clause AS normList»
86     do«FOREACH normList AS normaL2»
87     «LET (Norm) normaL2 AS norma »
89     «IF norma.Deontic.toString()=="OBLIGED" || norma.Deontic.toString()=="PERMITTED" »
90     ::«IF norma.Activation !=null»«norma.Activation»->«ENDIF»
91     «FOREACH norma.Target AS targ»«targ.Name»_Pending ! «norma.Id» «norma.Action» 0;«ENDFOREACH»
92                 «ENDIF»
93     «ENDLET»    «ENDFOREACH»
94     od;«ENDLET»
95    «cont.Id»_state=finished;
96     }unless(«cont.Expiration»);
97  } /*end contract «cont.Id»*/
98  «ENDFOREACH»  «ENDDEFINE»
99
```

Figure 11: Xpand script: writeContracts

```
31  mtype BuyWaterRightContract_state=no_initiated;
32  /*ContractTemplate BuyWaterRightContract::Description*/
33  proctype BuyWaterRightContract(){
34          BuyWaterRightContract_state=executing;
35    {
36     do
37    ::true->Seller_Pending ! norm1 give_water 0;
38    ::true->Buyer_Pending ! norm2 plow 0;
39    od;
40    BuyWaterRightContract_state=finished;
41    }unless(false);
42  } /*end contract BuyWaterRightContract*/
```

Figure 12: mWater BuyWaterRightContrat in PROMELA

executed from the modeling tool to verify that these contracts and norms are coherent with each other.

The execution of the SPIN model checker with the the generated code from our case study dectects a conflict. The LTL formula *Org2 mWater norm* is violated, i.e, the verification shows that this contract is not correct since the organizational norm is not fulfilled. This norm specifies that agents playing the *Buyer* role cannot provide other services, whereas the contract specifies that the *Buyer* must provide the service

```
141 «DEFINE writeInit (List[ContractTemplate] contractList, List[Executer] executerList) FOR romas_model »
142  init(
143 «FOREACH executerList AS execu»
144         run  «execu.Name»();
145 «ENDFOREACH»
146 «IF !contractList.isEmpty»
147         do
148         «FOREACH contractList AS contr»
149         :: («IF contr.Activation !=null» «contr.Activation» && «ENDIF»«contr.Id»_state==no_initiated) ->
150 run «contr.Id»();
151         «ENDFOREACH»
152         od;
153 «ENDIF»
154     )
155   «ENDDEFINE »
```

Figure 13: Xpand script: Init process

Plow to the *Seller*. Therefore, the designer should revise the design of the system and execute again the verifier module.

5. Conclusions and Future work

Engineering complex systems that are composed of heterogeneous and autonomous entities that coexist in a complex social and legal framework is a hard task, therefore, there is a need for guidelines, modeling environments and verification facilities. In this paper, we have dealt with the problem of engineering systems of this kind. A Model-Driven architecture and tool that are based on a formal metamodel for developing these systems is presented. In addition, a verifier module to check the coherence between contracts and their normative contexts has been developed using the SPIN model checker. A case study is presented to illustrate the use of this verifier module. Model-checking techniques, and more specifically the SPIN model checker, have been proved to be good mechanisms to verify the coherence of contracts in ROMAS.

The model and the verification of these systems are integrated into a CASE tool based on Eclipse technology. Moreover, an automatic executable-code generation plug-in is currently being developed. It will generate code for a normative-agent platform. In the near future, we plan to improve our verification module by adding a semantic verification using the information of the contract's ontology. We are also analyzing other properties in order to offer a complete design-time verification module. Finally, we are developing scalability tests to check the real effectiveness and usability of this verifier module for large systems.

6. Acknowledgments

This work is partially supported by the TIN2008-04446, TIN2009-13839-C03-01, PROMETEO 2008/051 projects, CONSOLIDER INGENIO 2010 under grant CSD2007-00022 and FPU grant AP2007-01276 awarded to Emilia Garcia.

References

[1] R. F. Fernandez, I. G. Magarinyo, J. J. Gomez-Sanz, J. Pavon, Integration of web services in an agent oriented methodology, Journal In-

ternational Transactions on Systems Science and Applications 3 (2007) 145–161.

[2] N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, S. Miles, Towards a formalisation of electronic contracting environments, Coordination, Organizations, Institutions and Norms in Agent Systems IV: COIN 2008 International Workshops (2009) 156–171.

[3] J. Vázquez-Salceda, R. Confalonieri, I. Gomez, P. Storms, S. P. Nick Kuijpers, S. Alvarez, Modelling contractually-bounded interactions in the car insurance domain, in: Proceedings of the First International ICST Conference on Digital Business -DIGIBIZ 2009-, 2009.

[4] V. Dignum, J. Meyer, F. Dignum, H. Weigand, Formal Specification of Interaction in Agent Societies, Formal Approaches to Agent-Based Systems (FAABS) 2699 37–52.

[5] E. Garcia, E. Argente, A. Giret, A modeling tool for service-oriented open multiagent systems, in: International Conference on Principles and Practice of Multi-Agent Systems (PRIMA), Vol. 5925 of LNAI, Springer-Verlag, 2009, pp. 345–360.

[6] E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, M. Rebollo, An Abstract Architecture for Virtual Organizations: The THOMAS approach, Knowledge and Information Systems (2011) 1–35.

[7] E. Garcia, A. Giret, V. Botti, Regulated Open multi-agent Systems based on contracts, in: The 19-th International Conference on Information Systems Development (ISD 2010), Springuer, 2010, pp. 235–246.

[8] N. Criado, E. Argente, V. Botti, THOMAS: An Agent Platform For Supporting Normative Multi-Agent Systems, Journal of Logic and Computation (2011) In press.

[9] C. Sierra, J. A. Rodríguez-Aguilar, P. Noriega, M. Esteva, J. L. Arcos, Engineering multi-agent systems as electronic institutions, UPGRADE The European Journal for the Informatics Professional V (4) (2004) 33–39.

[10] R. Soley, the OMG Staff Strategy Group, Model driven architecture. ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf.

[11] Meta object facility (mof) 2.0 query/view/transformation specification. ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf, Object Management Group.

[12] Eclipse - an open development platform. http://www.eclipse.org/ (2011).

[13] S. Rougemaille, F. Migeon, C. Maurel, M.-P. Gleizes, Model driven engineering for designing adaptive multi-agents systems, in: Engineering Societies in the Agents World VIII: 8th International Workshop, ESAW 2007, Revised Selected Papers, Springer-Verlag, 2008, pp. 318–332.

[14] H. Hachicha, A. Loukil, K. Ghedira, Mamt: an environment for modeling and implementing mobile agents, in: Sixth International Workshop From Agent Theory to Agent Implementation (AT2AI-6), 2008, pp. 75–82.

[15] M. Morandini, D. Nguyen, A. Perini, A. Siena, A. Susi, Tool-supported development with tropos: The conference management system case study, Vol. 4951, Springer, Springer, 2008, pp. 182–196, 8th International Workshop, AOSE 2007, Honolulu, HI, USA, May 2007.

[16] E. Solaiman, C. Molina-Jimenez, S. Shrivastav, Model checking correctness properties of electronic contracts, in: Service-Oriented Computing - ICSOC 2003, Vol. 2910 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 303–318.

[17] F.-S. Hsieh, Automated negotiation based on contract net and petri net, in: E-Commerce and Web Technologies, Vol. 3590 of Lecture Notes in Computer Science, 2005, pp. 148–157.

[18] G. Pace, C. Prisacariu, G. Schneider, Model checking contracts a case study, in: Automated Technology for Verification and Analysis, Vol. 4762 of Lecture Notes in Computer Science, 2007, pp. 82–97.

[19] S. Fenech, G. J. Pace, G. Schneider, Automatic conflict detection on contracts, in: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing, ICTAC '09, 2009, pp. 200–214.

[20] A. Lomuscio, H. Qu, M. Solanki, Towards verifying contract regulated service composition, Autonomous Agents and Multi-Agent Systems (2010) 1–29.

[21] C. D. Walton, Verifiable agent dialogues, Journal of Applied Logic 5 (2) (2007) 197 – 213, logic-Based Agent Verification. doi:DOI: 10.1016/j.jal.2005.12.009.

[22] R. H. Bordini, M. Fisher, W. Visser, M. Wooldridge, Verifying multi-agent programs by model checking, in: Autonomous Agents and Multi-Agent Systems, Vol. 12, Kluwer Academic Publishers, Hingham, MA, USA, 2006, pp. 239–256.

[23] M. Wooldridge, M. Fisher, M.-P. Huget, S. Parsons, Model checking multi-agent systems with mable, in: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2, AAMAS '02, ACM, 2002, pp. 952–959. doi:http://doi.acm.org/10.1145/544862.544965.

[24] N. Osman, D. Robertson, C. Walton, Run-time model checking of interaction and deontic models for multi-agent systems, in: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, ACM, New York, NY, USA, 2006, pp. 238–240. doi:http://doi.acm.org/10.1145/1160633.1160674.

[25] Eclipse - xpand plug-in. http://www.eclipse.org/modeling/m2t/?project=xpand (2011).

[26] G. Holzmann, Spin model checker, the: primer and reference manual, Addison-Wesley Professional, 2003.

[27] T. Kovše, B. Vlaovič, A. Vreže, Z. Brezočnik, Eclipse plug-in for spin and st2msc tools-tool presentation, in: Proceedings of the 16th International SPIN Workshop on Model Checking Software, 2009, pp. 143–147.

[28] A. Garrido, A. Giret, P. Noriega, mWater: a Sandbox for Agreement Technologies, in: CCIA 2009, Vol. 202, IOS Press, 2009, pp. 252–261.