A Model for Verification and Validation of Law Compliance of Smart-Contracts in IoT Environment

Flora Amato, Giovanni Cozzolino, Francesco Moscato, Vincenzo Moscato, Fatos Xhafa

***Abstract*—The interest of Industry 4.0 in Smart Contracts and blockchain technologies is growing up day by day. Smart Contracts have enabled new kinds of interactions whereby contractors can even fully automate processes they agree on. This technology is really appealing in Internet of Things (IoT) domain because smart devices generate events for software agents involved in a smart contract execution, making full automation possible. However, smart contracts have to comply with national and international laws and accountability of participant's actions. Soundness of a smart contract has to be verified in terms of law compliance. Here we propose a model for verification and validation of law compliance of smart-contracts in IoT environments. The main goal of this work is to propose a formal model (based on multi-agent logic and ontological description of contracts) for validating law compliance of smart contracts and to determine potential responsibilities of failures.**

***Index Terms*—Industry 4.0, Multi-Agent Systems, IoT, Smart Contracts, Blockchain.**

I. Introduction

SMART Contracts (SCs, hereafter) have been introduced since early 1990s, however, they are becoming increasingly of interest to researchers and developers from academia and industry after the emergence of Internet of Things (IoT) and blockchain technologies [1], [2]. Indeed, the conjunction of SC and blockchain can guarantee trust, reliability and security features of SCs. In contrast to usual contracts, which are formulated in a natural language of contractors, SCs should be expressed in formal languages that are able to address automation of all, or a large part of, contractors' actions. SCs can therefore be seen as an event-based description of activities that agents enact. In particular, software agents automatically execute actions when specific conditions between contractor parties are verified and hold. SCs can be automatically (or semi-automatically) verified and enacted by using software agents. However, the design of SCs is a complex task due to the need of ensuring SC's law compliance. Moreover, the presence of software-based systems as actors in SCs introduces the problem of having *third-parties* agents in contract enactment, that could be hidden in the chain of responsibility that legal aspects have to managed when any legal issue is raised. For instance, in IoT applications where devices and sensors could act as agents. From all the above, proper techniques are needed to support parties in the definition of SCs and assure compliance with current laws, rules and requirements to be stipulated in the contract. Often, agents making the agreement do not have a proper view of all legal issues related to the enactment of a smart contract –if they have legal validity (and this is the case of most countries), they have to be compliant to laws and rules. In this context, it is appealing to have a methodology able to analyze legal aspects as well as technical soundness of SCs, because they are now considered valid-by-law in many countries. For example, recently, car insurances make agreements where IoT devices, like environmental and car sensors, are used to profile and personalize SCs for insurances contractors, whereby, collected events and data are used as evidences for liability assignments in the case of car accidents [3]. In this work we address law-compliance validation and verification of SCs through a model transformation approach. More precisely, we use a Multi-Agent System (MAS) model and a workflow language to define SCs. MAS model has been selected as it enables an easy modeling of agents interactions and is particularly useful in IoT applications. Further, we use an ontology for formal modeling of common laws and rules to verify and validate the SCs.

The rest of the paper is organized as follows. We present in Section II a motivating example from auto insurance industry. The methodology for studying the properties of SCs is outlined in Section III, which is the core part of this work. In Section IV, we develop the example of Section II into a case study to show the feasibility of the SCs approach and methodology. We end the paper in Section VI with some conclusions and indications for future work.

II. A Motivating Example

This section introduces a motivating example for our approach from the domain of Auto Insurance Industry. IoT devices are each time more present in cars to improve efficiency, security. autonomous driving, etc. They are also being sought as possible means for the creation of dynamic Usage-Based Insurances (UBI) [3], where insurances monitor drivers' behaviors according to their consent. On the one hand, the premium can be evaluated depending on real driver attitudes and driving styles and, on the other, a SC can automatize the whole insurance process. Recent research [4] has shown that blockchain can improve security in Mobile Ad-hoc Networks (MANETs) and, in particular, in Vehicular Ad-hoc Networks (VANET). The example here focuses on SCs in the presence of IoT devices. Without loss of generality, we consider three agents in a contract: a human user ($\mathcal{U}$), who is the insurance contractor and car driver; a car (C) with several sensors onboard that record vehicle information from gear, braking and

F. Amato, G.Cozzolino and V. Moscato are with the CINI - Consorzio Interuniversitario Nazionale per l'Informatica, Rome, Italy, email: {flora.amato, gianni.cozzolino, moscato.vincenzo @gmail.com}

F. Moscato is with DIEM - Univ. of Salerno, Italy, email: fmoscato@unisa.it

F.Xhafa is with the Department of Computer Science - Technical University of Catalonia, Barcelona, Spain, email: fatos@cs.upc.edu

other systems, as well as sensors that collect driver's biological and behavior information (e.g. heart rate, temperature, head position, etc.); an insurance ($\mathcal{I}$) system, which automatically prepares documents and information for the user. Fig. 1

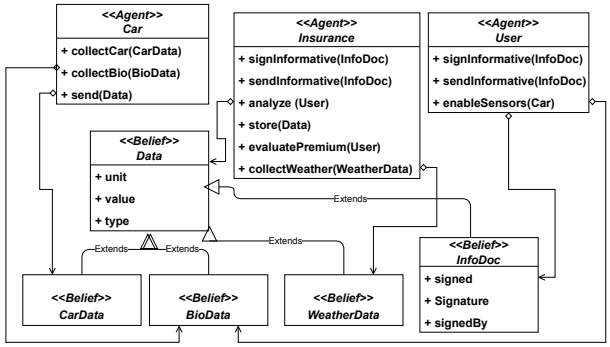


Fig. 1. Agents involved in a SC

represents an UML diagram with agents involved modeled as Classes and their actions as methods. It also contains a definition of information exchanged by agents during the enactment of the contract. They all inherit the generic *Data* class and include data from IoT sensors in the Car (*CarData*), biological and behavior data (*BioData*), weather-related information (*WeatherData*) and, finally, Insurance Documents (*InfoDoc*). Agents that own or manage data are linked to them by proper associations.

The definition of an SC comprises the following steps: (1) $\mathcal{I}$ signs an informative (requiring authorization to use User data) and sends it to $\mathcal{U}$ for countersigning; (2) $\mathcal{U}$ enables sensors on $\mathcal{C}$ to collect data; (3) $\mathcal{C}$ iterates data collection on the vehicle and $\mathcal{I}$ collects weather data related to $\mathcal{U}$'s positions; (4) at each step, $\mathcal{C}$ stores data into a repository on the Cloud; (5) $\mathcal{I}$ periodically evaluates the insurance premium for $\mathcal{U}$ by analyzing collected data. Fig. 2 represents these steps in a diagram similarly to an UML Sequence with these properties: (1) Objects are related to agents; (2) the lifetime of an object is not considered; (3) messages take place asynchronously; (4) messages are sent by agents whereby messages names refer to *methods* of senders. See also [5] for details of messaging among agents in a MAS model. With regard to collected data, we require that it is stored in an high availability storage system in a data center and are digitally signed to guarantee non-repudiation.

III. Methodology

This section describes the methodology to define the properties of SC, comprising the definition of: (1) a model of interacting parties (human or software agents) in an SC; (2) a model for requirements definition in an SC, stipulating laws, rules and other requirements that parties define in the contract. (3) a language to address abstract composition in SCs, which is agnostic of middleware used for SC definition and enactment;

The methodology therefore consists of the following procedures:

1) **MAS model definition for smart contract**: this step uses a UML-based language to define parties' behaviors in the contract as well as their interaction during the contract.
2) **Law Ontology definition**: in this step lawyers use Ontology Web Language (OWL) to formalize laws to be verified as requirements of SCs.
3) **Model Transformation and Planning**: models defined in step (1) are translated into analyzable models (namely Planning models). We address the translation of the MAS model into a workflow-based definition of an SC.
4) **Analysis**: in this step translated models from the previous step are analyzed. We focus here on analyses of law-based requirements, defined in step (2).

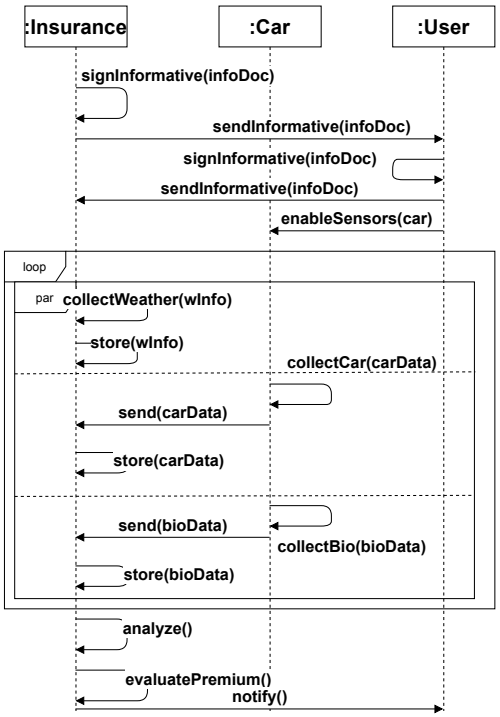


Fig. 2. Steps in the definition of an SC.

Next, we describe in more detail the above procedures.

A. MAS Model

The MAS model describes behaviors and interactions among SCs parties. The language combines MAS Beliefs, Desires and Intentions (BDI) logic and First Order Logic (FOL) [6]. The environment is modeled by resources (that are agents too), properties that are true in the environment (in terms of logical predicates), and events that may happen. In general, agents may have an incomplete or incorrect knowledge of the environment, or of other agents properties, states and knowledge (referred to as *belief*). However, in this work we assume that agents have exact knowledge on all beliefs they own but may have partial information.

An Agent, in a static setting, is defined as a triple:

$$Agent = (Actions, Beliefs, Goals)$$

where *Actions* is the list of actions an agent can perform; *Beliefs* is a set of formulae representing an agent's knowledge;

Goals is the set of formulae to be asserted in order to reach its goals. An *Action* is defined by a triple:

$$Action = (name, Precondition, Effects)$$

where *Precondition* and *Effects* are respectively, one formula and a set of Timed Computational Tree Logics (TCTL) formulae, where atomic components are asserted by RDF triples from the domain/law ontology. We denote the Preconditions and Effects with the name of *Conditions*. In addition *Events* are particular formulae that are expected as precondition, or asserted as effect of an action.

Let $\mathcal{W}orld$ be the set of evaluations of conditions of all beliefs in the environment and all the agents states (i.e. the beliefs that are true for agents in the $\mathcal{W}orld$). In order for an action act to be executed, $\mathcal{W}orld$ should meet all its preconditions. The execution of act leads to a change of the state space adding Effects of act to $\mathcal{W}orld$. In order to describe agents we use a modeling language inherited by Unified Modeling Language (UML). The definition of a contract, requires the declaration of agents structures and a specification of contract interactions. To that purpose, we exploit a formalism extended from UML Class and Sequence Diagrams. For convenience, we refer to Fig. 1 described in Sec.II. Proper stereotypes (*Agent* and *Belief*) identify MAS elements in the diagram; methods in the class diagram are the Agents available actions, and attributes in beliefs define their contents. In the example, *Car, Insurance* and *User* are the three agents acting during contract execution. The car agent owns the Beliefs (i.e. information) *CarData* and *BioData* that respectively represent data acquired from vehicle sensors and data related to the (human) driver. The Insurance agent owns InfoDoc and WeatherData, which represent the information for the use of User's data and the data about weather collected during contract execution. Finally, the User owns its BioData and Informative data. All data inherit from the generic Belief *Data* that has three attributes representing Belief's structure identified by a triple (unit, value and type). Methods listed in each Class, are the actions available to the agents.

Preconditions and effects of actions are listed in Table I that we explain in the next section. The Sequence Diagram in Fig. 2 defines the dynamic view of agents participating in contract execution. As introduced before, messages in the diagram represent actions that one agent uses to communicate with other agents, or to execute inner actions (reflexive messages). For a description of this diagram see Sec.II.

B. Law Ontology Definition

The presented methodology requires a formal definition of laws to verify as requirements of smart contracts. The definition contains laws, rules and other requirements that are applicable to the domain of smart contracts. We used Resource Description Framework (RDF) and Ontology Web Language (OWL) to define an ontology. The ontology is defined once, and its elements can be re-used and integrated while defining models related to the same domain. Fig. 3 describes the main elements of the ontology. We distinguish here the main classes in the ontology as follows: (1) **Laws** descriptions, (2) actions

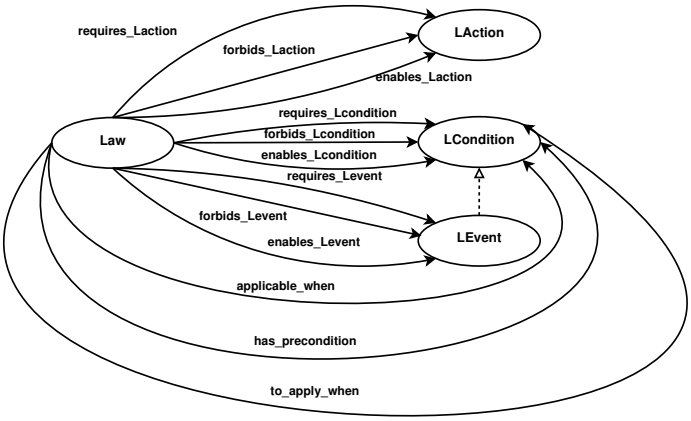


Fig. 3. Laws Ontology

(**LAction**) that agents can enact, (3)conditions (**LCondition**) with predicates to check in order to execute actions, and (4) events (**LEvents**) enabled by actions executions (that in turn may enable or forbid other actions in the same or different law). An LEvent is-a LCondition: this means that events too can be preconditions or effects of laws enactments. These classes define the ability of a law to enable or forbid the execution of an action. The *require_∗* properties define actions, and other elements that are requirements for the application of a Law. The *forbids_∗* properties declare the elements that a law forbids and finally, The *enables_∗* properties list the elements enabled by a law.

Some other properties also define logic conditions that enable laws (*applicable_when*), as well as their preconditions (*has_precondition*) and events required for activation (*to_apply_when*). For example, by using our ontology, sentences in the following structures are able to define laws in a formal way:

*This **law** declares that, the following pre-**conditions** are required to execute this **action***

or

*When the **event** occurs, the **law** defines that an **action** **forbids** another **action***

C. Model Transformation and Planning

Once the ontology and the contract model have been created, the methodology applies the **Model Transformation** [7] and generates an intermediate model to enact the analysis. The first model transformation generates a **planning model**. It merges ontological information that matches with that defined in the MAS model. As plans are sequences of actions enacted by agents to reach their goals, if these coincide with smart contracts goals, we can state that all agents participating in a contract, works together in order to reach goal contracts. During contract execution, some actions or events may enable the execution of malicious of incorrect actions, leading to a state where the contract is considered failed. A **planning model** is a transition system identified by agents actions and their connections, expressed in terms of a precedence graph.

Algorithm 1: Planning Model Transformation

```
Input: Agents Class and Sequence Diagrams
forall messages in Sequence do
    Create an Activity a in the Planning model relate a to the sender Agent in the sequence relate a to
      the action with same name in the sequence.
    forall blocks in the Sequence do
        if the block is a Par Block then
            Create a parallel pattern in the Planning Model with a route activity at the beginning
              and at the end of the parallel; link first activity in parallel alternatives to the outgoing
              transitions of the route activity link last activities of parallel alternatives to the
              incoming transitions of the last route activity
        if the block is a Loop then
            Create a loop pattern in the Planning Model with a route activity at the beginning and
              at the end of the loop;
    end
    Link consecutive Activities in the order their related Actions appear in the Sequence.
end
```

We adopt the planning model from workflow-based models, whose main components are:

Activities: an activity is an action executed by an agent;

Transition: a transition defines a precedence between two actions. It is basically a directed edge linking two activity nodes;

Transitions Conditions: A transition condition is a predicate that has to evaluate true in order to allows the activity on the end point of the transition, to be activated after the completion of the activity on the starting point;

Join and Split Conditions: these elements allows for the definition of complex workflow patterns, like sequences, synchronization points, parallel branches etc.

Activities contain the description of *Agents* performing the *Action* related to each activity, as well as its *Precondition*, *Effects* and waited and raised *Events*. A *Transition* specifies a *Transition Condition* that, if evaluated true, enables the execution of the activities on the directed edge of the transition. The *Join* and *Split Conditions* in the activities allows for specification of different control points, which are used to define flows of activities in the Planning Model. In particular, an AND Split is a point where a single thread of control split in (one), two or more threads executed in parallel. An AND Split, with transition conditions, also enable conditional choices of paths. An XOR Split is eventually a decision point, where only one condition on outgoing transitions can evaluate true. An OR Split is like an XOR, except that more than one outgoing transition can fire if their conditions evaluate true. From the Join (incoming) point of view, It controls if and when the related activity can execute. An AND Join executes the activity only when all its incoming transitions have fired; An XOR Join executes the activity in the case only one among its incoming transitions have fired. As can be seen in Fig. 4, we use boxes to represent Activities and arrows for Transitions. split and join conditions are respectively reported at the beginning and at the end of the activity in the directed graph. Eventually Transition Conditions are reported on the top of transitions. For clarity and brevity sake, we use in this work the name of activities are the name of agents' actions enacted during the activity. We can apply a simple **Model Transformation** procedure (Alg. 1) to the MAS model in Fig. 2 to create a Planning model for the smart contract depicted in the figure.

Fig. 4 (without considering the *C.encrypt* activity) is the output of Alg. 1 applied to the Sequence Diagram in Fig. 2.

Algorithm 2: Propagation of Conditions

```
Input: Plan Graph
Result: List of Lists of Conditions on Activities
Initialisation :
For each activity create a List of Conditions (Conditions is a List itself)
forall Activity in the Plan Graph do
    Assigning Conditions of previous activity to new Activity
    if Effects on activity do not change Conditions in the lists then
        Insert Effects into the List of Conditions
    else
        Delete Conditions in the new list
    end
    if Split Condition of the Activity is XOR or OR then
        forall Activities on outgoing Branches do
            Insert other Conditions FOR EVERY BRANCH (these are alternatives for others)
        end
    else if Split Condition is AND then
        forall Activities in outgoing Branch do
            if Conditions on Branches do not interfere then
                Assume Worst Case: The branch fires first.
                Add All Conditions into Activities
            else
                Assume both the Best Case and Worst Case Principles:
                For all Conditions, insert the The List Built as in the previous case, but with
                  Conflicting activities in both order
            end
            Include an alternative List of Conditions FOR ALL BRANCHES
        end
    if Join Condition is AND then
        Melt Conditions of incoming activities in a unique list
    else if Join Condition is XOR or OR then
        Maintain all Conditions from incoming activities in the list of Condition of the actual Activity.
        They are all Conditions that can appear at the beginning of the activity
end
```

D. Analysis

(Law-based) Correctness analysis of a SC requires the creation of a state-based model, where conditions and beliefs represent the state of contract execution. Activity enactment changes the state of the system, producing effects and changing the beliefs.

In order to build this state-based model, where we can apply model checking techniques, we need to: (1) produce a Planning Model where Effects are propagated over the whole Plan; (2) translate the new model into a state-based model. Alg. 2 summarizes the execution of the Condition Propagation on the planning model graph. Its application to the planning model results in propagation highlighted in Fig. 4, where there is one parallel branch (starting after the **collect (route)** activity).

Effects of each activity are in the last column of Tab.I (we use here the short name ***Ex***). Under each activity we reported the list of properties asserted by Effects. For example, the list of effects of **I.SignInformative** contains only an empty list. Then, the effect of this latter activity is added to the list of **I.SendInformative**, which has only one Condition in the list, the ***E1***.

Branches on transitions outgoing from ***collect*** route activity execute in parallel. This means that subsequent activities in the graph propagate their effects independently from other branches. Effects are merged on the ***join*** route activity. For convenience, in the figure we renamed two subsets of conditions into $\boldsymbol{K}$ and $\boldsymbol{J}$ (with $J \subset K$). After Condition Propagation, it is possible to check if the lists of conditions on each activity matches its own precondition. It easy to prove that here all propagated conditions match activities preconditions, which are listed in Table I. This implies that activities, enacted when scheduled by the SC middleware, can be always activated resulting in a correct and runnable contract.

Reachability Analysis is another kind of analysis that involves both model transformation and Planning Graph (PG) exploitation. By means of model transformation, reachability of states in SCs is translated into a reachability problem of

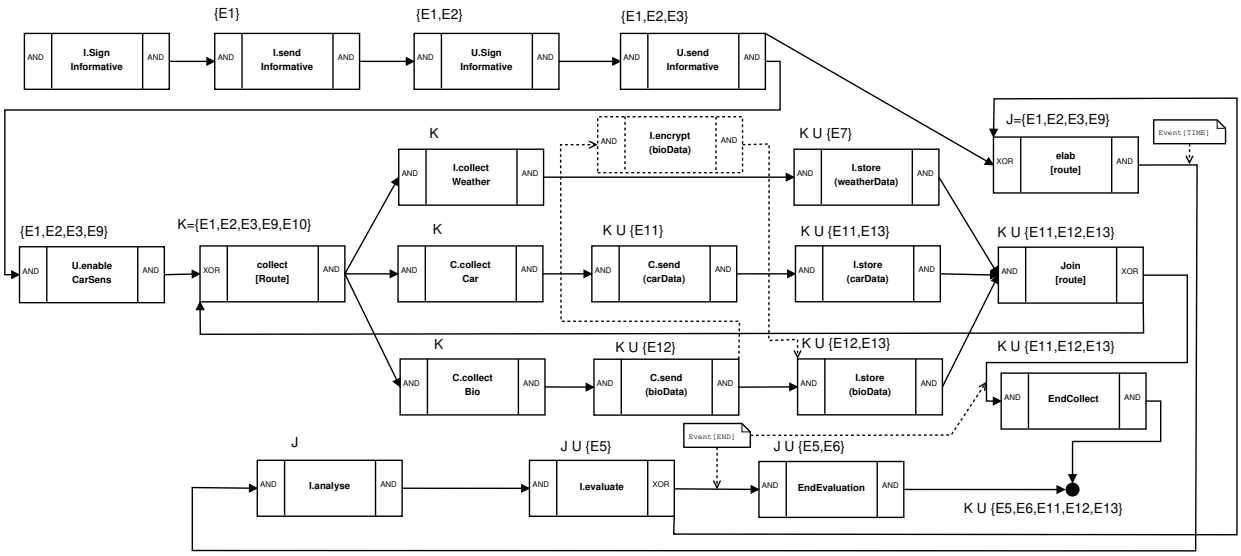


Fig. 4. Plan for Insurance smart Contract

Algorithm 3: Reachability Model Transformation

Input: Agents Class Diagram and Planning Graph Model
forall $a \in Agents$ **do**
  Create a TA t_a: ; **forall** $method \in a$ **do**
    add one state to t_a with an incoming (waited) event for entering the state and one outgoing (asserted) event when exiting the state after method execution
  end
  Create a Synchronization TA (syncTA) from PG: **forall** *activity in PG* **do**
    create a state in syncTA with an (asserted) event with the name of the activity and a one outgoing (waited) event;
    forall *parallel branches* **do**
      Create a sub-automaton for the branch with syncTA rules
    end
  end
end

Timed Automata (TA) [8], which are the target model of our transformation.

The analysis of correct termination of contracts and other reachability problems becomes the analysis of properties expressed in TCTL logic like: "Is it always true that a collaboration will ever reach contract goals respecting legal requirements?". The model transformation produces both the model and the expression of the formulae to analyze. Fig. 5 shows some of the TA generated by Alg. 3 from models in figures 1 and 4. In particular, at the bottom left there is the TA for the *Car* Agent; at the top there is part of the TA generated from the Planning graph in Fig. 4, where for simplicity we reported the parts related to the first two activities of the SC, as well as the part included between the ***collect*** and ***join*** activities (dotted arrows represent cut elements in this TA); at the bottom right there is one of the sub automaton generated to implement one of the parallel branches in Fig. 4 (the one reported in the middle of parallel pattern).

Notation: We use a dotted notation in names of states and events. In addition, we use the common TA notation for events where asserted events in sub automata are followed by an exclamation point, while waited events are followed by a question mark. Dotted notation in events naming highlight the name of the agent and the name of the action that generate the event. The events terminating with "Done" are asserted when an action ends. The Car TA automaton reports clock variable that can be used for temporal analysis, where *ex1, ex2 and ex3* can be considered as timeouts for Car's actions. As example, the waited event ***C.collectCar?*** is generated by the sub automaton of the parallel branch reported at bottom right in the figure.

The use of results from PG analysis we described before, contribute in reducing complexity of generated TA. Previous analysis assures that all preconditions hold for all activities and it is possible to ignore properties in preconditions and effects when checking reachability.

IV. Case Study

Table I summarizes the agents' actions with their preconditions and effects. Notice that all predicates on the last two columns of the table must be defined in the Law and Domain Ontology (in fact their structure is similar to a RDF triple). It should be noted here that hasData and hasCarData (as well as hasBioData) are object properties in the ontology, where the latter is a specialization of the former (i.e. if there is a car data, there is data too). For the presentation of this case study, we omit details of Beliefs (environmental, bio-metrical and data from car sensors are the beliefs we consider in this case study). Other beliefs refer to the state of informative and data (sent, signed, etc.), while Goal is represented by a belief that states that insurance premium has been evaluated by data analysis.

Fig. 4 shows the related plan model for the smart contract. We use the classical dotted notation in order to define the agents and the action it is performing. Let us describe the plan without considering the dotted-lined activity I.encrypt, that it is not yet defined in Table I.

By applying the methodology presented in Sec. III to this plan, we achieve to meet all preconditions for any action, in any state. It might be the case that contractors has made an omission, namely, they did not consider explicitly some laws about data protection, say, the General Data Protection

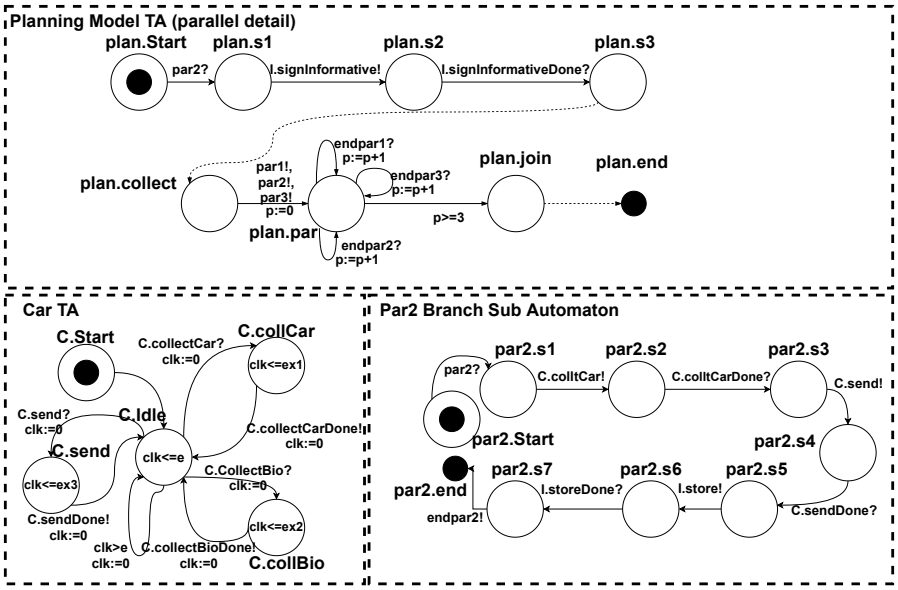


Fig. 5. Generated TA structure

TABLE I
AGENTS, ACTIONS, PRECONDITIONS AND EFFECTS

Agents			
Name	***Actions***	***Precondition***	***Effects***
Insurance (I)	signInformative (infoDoc)		E1:signed(infoDoc,I)
	sendInformative (infoDoc)	signed(infoDoc,I)	E2:hasSignedInfo(I,InfoDoc)
	store (data)	hasData(I,data)	E3:stored(data,I), E4:dataStored(I,data)
	analyse(U)	dataStored(I,U)	E5:dataAnalyzed(I,U)
	evaluate(premium,U)	dataAnalysed(I,U)	E6:premiumEvaluated(I,U)
	collectWeather(Wdata)	hasWaterData(I,Wdata)	E7:hasData(I,Wdata)
User(U)	signInformative(infoDoc)	hasSignedInfo(I,InfoDoc)	E8:signed(infoDoc,U)
	sendInformative(InfoDoc)	signed(infoDoc,U)	E9:hasData(I,infoDoc)
	enableCarSens(C)	signed(infoDoc,U)	E10:carEnabled(U,C)
Car(C)	collectCar(carData)	carEnabled(U,C)	E11:hasCarData(C,carData)
	collectBio(bioData)	carEnabled(U,C)	E12:hasBioData(C,bioData)
	send(data)	hasData(C,data)	E13:hasData(I,data)

Regulation (GDPR)[1], which states that Biometric Data has to be stored encrypted.

Since Beliefs address Biometric data too, therefore this have to be dealt with in the law domain ontology. In particular, we have to address these elements:

- isInstance(GDPR, Law);
- forbids_condition(GDPR, (storageEncryption(data, !crypted)));
- requires_action(GDPR, (encrypt(data)));
- applicable_when(GDPR, (isDataStored(data,storage)));
- applicable_when(GDPR, (dataType(data,biometric)));
- requires_condition(GDPR, (isDataStored(data,storage) AND dataType(data,biometric) AND strageEncryption(data,crypted)));

By linking this information too in the Planning Model, the activity I.store(bioData) does not meet preconditions added by GDPR. The clause applicable_ when allows to introduce the precondition to the activity in the plan; the *requires* clauses define preconditions to add. In order to meet these preconditions, an action should be added to I, which encrypts data before the storagetakes place (as depicted in the figure, with the activity I.encrypt inside the dotted lined box). In addition we can study reachability of the finalization of an SC. Following the methodology for reachability analysis, this problem is the same of analyzing the reachability of ***plan.end*** state in the TA of Fig. 5. This problem is solved by assuming that the planning model is correct in terms of enactment of activities and by applying model checking techniques to analyse a property in TCTL logic on the TA that, for the reachability of end point of the SC is:

$$AF(plan.end)$$

[1]https://gdpr.eu/, site accessed Jan 9th, 2021

We use the UPPAAL model checker[2] to check the asserted formulae. In addition, the model transformation in Alg. 3 produces TA representation in UPPAAL formalism. The property referred to above, for this concrete case study, evaluates to true.

V. Related Work

In the last year, many surveys in scientific literature analyzed challenges, advances and platforms related to the topic of smart contracts (e.g. [1], [2]). Considerable efforts by the research community are being done for improving security and analysis of correctness. IoT-SC intersection is interesting for sharing services in IoT ecosystems and of automating existing time-consuming workflows in Industry 4.0 domain, by introducing some blockchain-related benefits such as the use of encrypted and verifiable transactions [9]. While most of the scientific works in the literature deal with verification of byte-code and other low-level representation of smart contracts, some methodologies are now addressing high-level languages for definition of SCs [1], [2], [9], [10]. In particular when dealing with Law-compliance of smart contracts [11], [12], [13], the general approach is to provide a language that is simple and abstract enough to be used by both lawyers and programmers. This high level representation of contracts would then be translated (by using model transformation) into low-level and byte code languages, as well as into other intermediate representations for formal verification.

With regard to analysis, we distinguish here the most used frameworks for analysis of SCs are Zeus [14], ContractLarva[15], Ergo [10], Beagle [11] and VERISOL [16]. All of them produce low level representation of SCs, that are in turn described by higher level languages: some of these are based on templates of contracts with explicit definition of policies and laws to verify (like Zeus or Beagle), or functional languages that are used to express legal agreements (like Ergo). In ContactLarva, requirements of SCs are translated (by experts) into predicates to verify on a state-based model. VERISOL follows a similar approach, using monomial predicate abstraction to apply bound model checking techniques in verification.

Zeus, Beagle and VERISOL are commonly used in other works [1], [2] to analyze properties of SCs defined by high level languages. They use techniques like Depth First Search or model checking on a state-based model. Our approach is similar, namely, we build a graph based model from SCs specifications, and then we apply model transformation to build a state based model after model transformation to analyze law compliance.

The main difference with the approach presented in this paper, is that we manage law compliance at a semantic level and the analysis is performed directly on the abstract graph obtained by model transformation, differently from previous approaches that execute analysis at low-level. In this way, we are able to maintain an high level of abstraction in verification, which is useful for lawyers that have to manage compliance issues.

Other works take directly into account law compliance as requirements to verify in SCs [17], [11], [12], [13]. These works mainly focus on the high-level language used to describe SCs and the requirements to be verified. In [13] the authors discuss on the use of Business Process Model and Notation (BPMN) or Finite State Machines (FSM) as contract modeling languages, while in [12] authors describe a modeling language for General Data Protection Regulation (GDPR) based on Unified Modeling Language (UML) and a Query and Answer (Q&A) methodology. Finally, the approach of *OpenLaw*[3] uses a markup language (the Legal Markup Language) to define both law requirements and SC templates of known legal interactions.

From the point of view of the modeling language, we used a workflow language (like BPMN discussed in [13]), and a model based on Multi-Agent System (MAS) to define requirements to verify. In this way we are able to generate an intermediate (graph-based) model that can be used for further verification activities. In addition, we have only formal languages at definition level, so we ca implement automatic model translation. Definition of Law-dependent requirements is based on Semantics (ontology): laws descriptions follow structures forced by ontology, and preconditions and effects of laws are checked as properties on the intermediate model. The MAS model is powerful to express IoT and services behaviors [9]. In [18] the authors provide a model for smart contracts validation. It is based on Multi-Agent Models, but it lacks the ability of catching semantics of actions, rules and laws that we address explicitly in our paper.

From all the above, to the best of our knowledge, our approach distinguishes for its ability to model SCs and legal requirements from an high-level perspective. Moreover, our approach is general enough to take into account IoT devices actions in contracts workflows.

VI. Conclusion

In this work we have presented a methodology to define and analyze smart contracts. Our methodology explicitly addresses the problem of compliance with current laws and rules, in a context where both humans and software can interact, whereby accountability of liability is relevant. We reported here the definition of a formal model and of a procedure to analyze workflows of smart contracts. The model is based on MAS models and BDI logic. We apply Model Driven Engineering and Model Transformation in order to implement proper model checking techniques, including a Condition Propagation algorithm to be enacted before the check. We applied the methodology to an IoT-based case study from the domain of Industry 4.0. In the case study, a smart contract is used to define a procedure through which insurance companies can collect data from drivers in order to adapt premium prices to driving styles and weather conditions. We show that the existence of common laws (e.g. GDPR EU framework) requires proper management of the contract. The application of the methodology is able to identify issues in bad contracts by showing the laws they do not match and where it happens

[2]https://uppaal.org/

[3]https://openlaw.io

in the smart contract. In future works we envisage to apply the methodology to blockchain based contracts, and include the definition of a more complex and complete ontology for Laws and Rules in smart contracts.

ACKNOWLEDGMENT

Fatos Xhafa's work is supported by Research Project, "Efficient & Sustainable Transport Systems in Smart Cities: Internet of Things, Transport Analytics, and Agile Algorithms" (TransAnalytics) PID2019-111100RB-C21/AEI/ 10.13039/501100011033, Ministerio de Ciencia e Innovación, Spain.

REFERENCES

[1] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[2] M. Almakhour, L. Sliman, A. E. Samhat, and A. Mellouk, "Verification of smart contracts: A survey," *Pervasive and Mobile Computing*, p. 101227, 2020.

[3] P. Baecke and L. Bocca, "The value of vehicle telematics data in insurance risk selection processes," *Decision Support Systems*, vol. 98, pp. 69–79, 2017.

[4] R. Shrestha, R. Bajracharya, A. P. Shrestha, and S. Y. Nam, "A new type of blockchain for secure message exchange in vanet," *Digital communications and networks*, vol. 6, no. 2, pp. 177–186, 2020.

[5] F. Moscato, "Model driven engineering and verification of composite cloud services in metamorp (h) osy," in *2014 International Conference on Intelligent Networking and Collaborative Systems*. IEEE, 2014, pp. 635–640.

[6] D. Burfoot, J. Pineau, and G. Dudek, "Rrt-plan: A randomized algorithm for strips planning." in *ICAPS*, 2006, pp. 362–365.

[7] T. Mens and P. V. Gorp, "A taxonomy of model transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, no. 0, pp. 125 – 142, 2006, proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005), Graph and Model Transformation 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066106001435

[8] K. Quaas, M. Shirmohammadi, and J. Worrell, "Revisiting reachability in timed automata," in *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2017, pp. 1–12.

[9] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *Ieee Access*, vol. 4, pp. 2292–2303, 2016.

[10] A. Chepurnoy and A. Saxena, "Multi-stage contracts in the utxo model," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2019, pp. 244–254.

[11] W.-T. Tsai, N. Ge, J. Jiang, K. Feng, and J. He, "Beagle: A new framework for smart contracts taking account of law," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2019, pp. 134–13 411.

[12] M. Corrales, P. Jurčys, and G. Kousiouris, "Smart contracts and smart disclosure: coding a gdpr compliance framework," in *Legal Tech, Smart Contracts and Blockchain*. Springer, 2019, pp. 189–220.

[13] J. Ladleif and M. Weske, "A unifying model of legal smart contracts," in *International Conference on Conceptual Modeling*. Springer, 2019, pp. 323–337.

[14] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts." in *NDSS*, 2018.

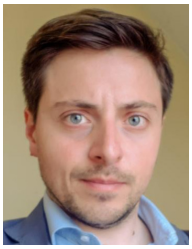
[15] S. Azzopardi, J. Ellul, and G. J. Pace, "Monitoring smart contracts: Contractlarva and open challenges beyond," in *International Conference on Runtime Verification*. Springer, 2018, pp. 113–137.

[16] Y. Wang, S. K. Lahiri, S. Chen, R. Pan, I. Dillig, C. Born, and I. Naseer, "Formal specification and verification of smart contracts for azure blockchain," *arXiv preprint arXiv:1812.08829*, 2018.

[17] J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," *IEEE Access*, vol. 7, pp. 77 894–77 904, 2019.

[18] D. Calvaresi, A. Dubovitskaya, J. P. Calbimonte, K. Taveter, and M. Schumacher, "Multi-agent systems and blockchain: Results from a systematic literature review," in *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer, 2018, pp. 110–126.

Flora Amato , PhD, is Assistant Professor and she works with CINI research consortium. Her research topics include Formal Modeling, Knowledge Management and Cloud Computing. She is author of more than 150 papers, published on Int. Journals and Conf. Proc. She coordinated many national and international research projects. She is in the committee of many int. conferences and she served as editor of Int. Journals.



Giovanni Cozzolino Main research interests of Giovanni Cozzolino, PhD, are Computer Forensics, Knowledge Management, Information Extraction and Data Integration. He published on dozens of International Journals, he participated to some European projects and he collaborated both with Italian Public Administrations and Companies.

Francesco Moscato , PhD, is Associate Professor at University of Salerno. His research activities are mainly centered on formal modeling and verification of reliable and critical systems. He is author of many articles published on international journals, conf. proceedings and books. He was involved and leaded many national and international (EU funded) research projects.

Vincenzo Moscato , PhD, is Associate Professor and he works with CINI research consortium. He is in the executive comm. of CINI ITEM Res. Lab. and one of the leaders of PICUS research group. He was involved in many national and international projects and coordinated some of the them. He was in the program committees of many int. conferences and served as reviewer and editor of many int. journals. He authored about 170 papers on international journal, conference proceedings and book chapters.

Fatos Xhafa is Full Professor at the Technical University of Catalonia (UPC), Barcelona, Spain. Prof. Xhafa has widely published in peer reviewed international journals, conferences/workshops, book chapters and edited many books and proceedings in the field. He is awarded teaching and research merits by Spanish Ministry of Science and Education, by IEEE conference and best paper awards. Prof. Xhafa has an extensive editorial and reviewing service for international journals and books. He is Editor in Chief of the Elsevier IoT Journal and of IJGUC Inderscience Journal. He is Editor in Chief of the Elsevier Book Series Intelligent Data-Centric Systems and of the Springer Book Series Lecture Notes in Data Engineering and Communication Technologies. He is a member of IEEE Communications Society, IEEE Systems, Man & Cybernetics Society and Emerging Technical Subcommittee of Internet of Things. His research interests include parallel and distributed algorithms, massive data processing and collective intelligence, IoT and networking, P2P and Cloud-to-thing continuum computing, machine learning and data mining.