GDPR Modelling for Log-Based Compliance Checking

Colombe De Montety, Thibaud Antignac, Christophe Slim

GDPR Modelling for Log-Based Compliance Checking

Colombe de Montety[1,3], Thibaud Antignac[1][0000-0003-0670-3439], and Christophe Slim[2]

[1] CEA List, Software Safety and Security Laboratory, PC174, 91191 Gif-sur-Yvette, France
[2] CEA, Agreements and Intellectual Prop. Service, PC144, 91191 Gif-sur-Yvette, France
[3] DANTE, UVSQ, 3 rue de la division Leclerc, 78280 Guyancourt, France
{colombe.de-montety,thibaud.antignac,christophe.slim}@cea.fr

Abstract. Since the entry into force of the General Data Protection Regulation (GDPR), public and private organizations face unprecedented challenges to ensure compliance with new data protection rules. To help its implementation, academics and technologists proposed innovative solutions leading to what is known today as privacy engineering. Among the main goals of these solutions are to enable compliant data processing by controllers and to increase trust in compliance by data subjects. While data protection by design (Article 25 of GDPR) constitutes a keystone of the regulation, many legacy systems are not designed and implemented with this concept in mind, but still process large quantities of personal data. Consequently, there is a need for "after design" ways to check compliance and remediate to data protection issues. In this paper, we propose to monitor and check the compliance of legacy systems through their logs. In order to make it possible, we modelled a core subset of the GDPR in the Prolog language. The approach we followed produced an operational model of the GDPR which eases the interactions with standard operational models of Information Technology (IT) systems. Different dimensions required to properly address data protection obligations have been covered, and in particular time-related properties such as retention time. The logic-based GDPR model has also been kept as close as possible to the legal wording to allow a Data Protection Officer to explore the model in case of need. Finally, even if we don't have a completed tool yet, we created a proof-of-concept framework to use the GDPR model to detect data protection compliance violations by monitoring the IT system logs.

Keywords: Privacy, Logic, Model, Accountability, Compliance.

1 Introduction

Formal methods are a feasible solution to ensure compliance with data protection rules with a high level of trust. In his paper [1], Daniel Le Métayer calls for the use of these techniques to bridge the gap between legal and technical means in reducing privacy risks related to persons whose data are processed. His work provides a formal framework of legal concepts which can serve as a basis for modelling entities' rights and duties. However, he points out the subtlety of real-life situations, that cannot be replicated in programs, because they often go beyond mere dual and static situations. The recently entry into force of the GDPR aims at protecting data subjects at a high level, creating many obligations for data controllers and rights for data subjects regarding any processing of personal data. We believe a translation of core GDPR rules into a logic programming language can be performed to help and support legal

practitioners in their work and help addressing the inherent complexity of the law – though this might not completely bridge the gap between legal and technical domains. Indeed, data protection is known for a long time to be difficult to handle in requirements engineering processes [2].

Our work provides a way for the data controller to comply with its accountability through the demonstration of its compliance (Article 24 of GDPR [3]) while carrying any current or future data processing. To that end, we propose a methodology for building a compliance checker that will point out the issues, for any given processing of personal data, that should be remedied. Thus, this checker is aimed at assisting the Data Protection Officer, whose role is notably is to ensure that its organization processes the personal data in compliance with the applicable rules and will facilitate the compliance verification, which is the starting point for data controllers' accountability. As it will not be possible to give a definitive compliancy diagnosis in each and every case, we adopted a conservative approach by enabling the compliance checker to raise alerts when in doubt. Consequently, a raised alert does not mean there is a non-compliance, but all non-compliances raise alerts. This approach, though increasing the number of false alerts, ensures a high level of trust in the diagnosis.

Many previous works have shown that legal rules can be interpreted sometimes according to very different paradigms than logic rules (see [4] for a data flow diagram-based description for instance). Our work is nevertheless very different from previous developments because of the large concepts and ambiguous writing of some articles of the GDPR. This is why we chose, as a first step, to narrow the scope of our work by focusing on two key articles: Article 5 (principles on processing of personal data) and Article 6 (lawfulness of processing), which cover the main obligations to be considered as compliant for any given personal data processing. Moreover, as they constitute general principles, this creates a core which can be further extended to cover other aspects of the GDPR in the future. Our main contributions are:

1. A model of the core of the GDPR as a Prolog-based logic model which can be further extended to cover more data protection principles;
2. A coverage of timing properties to cover obligations such as retention time;
3. A set-up of a compliance checking architecture that detects violations based on logs of actions having occurred in an IT system.

In Section 2, we will describe the method and tools used to carry out this work. The way the IT system and the GDPR are modeled are developed in Section 3 while we detail how this comes together to form the basis of a compliance checker along with a small example in Section 4. Finally, we refer to related works in Section 5 and conclude by stating our plans for further developments in Section 6.

2 Method and Tools

Systems can be modelled in many different ways and the choices made depend on the objectives targeted. We will discuss the modelling choices made and their motivations in Section 2.1 and briefly present the most prominent features of Prolog in Section 2.2.

2.1 Modelling Choices and Rationales

Working on the GDPR entails dealing with a large quantity of material. For this reason, we choose to strictly translate the words from the GDPR, so that we can keep track of the provisions in the program. Also, we can make sure we don't go beyond the legal obligations and principles stated in the regulation.

In such circumstance, the first step is to choose and narrow the scope of the regulation. For instance, we choose not to deal with the situation where a data processor is actually processing the personal data; we only focus on the data controller, who is liable for the processing of personal data. Indeed, the GDPR gives a key role to the controller, who determines the means and purposes of the processing. Thus, the data processor is subject to obligations that are deeply linked with the data controller's ones. As a result, provisions regarding the data processor may be added up to the rules already translated, when such a data processor is involved. This is a situation where a false alert can be raised.

Another benefit of a close translation of the GDPR is to allow the involvement of a person, such as a DPO, whose role is to ensure compliance with data protection issues. Our work aims at being a tool to aid for compliance checking, rather than providing all right answers in order to achieve compliance. In case of inconsistency between the log and the regulation, a data protection practitioner is required to interpret any answer given by the program, and to implement the knowledge base of the program.

Our methodology is driven by Maxwell and Antón's methodology, which implies rights, obligations and permissions as a systematic step, when translating legal rules [5]. However, our approach differs from theirs as it considers the appropriate rules by identifying the involved elements (processings, personal data, data subjects, or other persons, for instance) as well as their relationships to one another. This step implies identifying the conditions for an obligation to be applicable to the data controller as well as any disjunctions. Then, we implement the initial translations of the legal rules into actions and states, to make them closer to how an abstract machine works. For this reason, we adopted a concrete vision of every obligation and constrained compliance to some specific actions or states of the data processing. We also included some deontic propositions to express obligations for the controller. In addition, a pattern is dedicated to verify the conditions of success of a request from the data subject about one of his or her rights.

2.2 Logic Modelling in Prolog

To express legal rules, we use the Prolog language, that deals with implication and instantiation of variables through concrete elements. This language is declarative,

meaning we need to declare a knowledge base and define a set of rules in order for the program to resolve a given query. Through a solving strategy called backtracking, which is automatically handled by Prolog, the program will reply using instantiation of the variables used in the rules. For instance, two rules can be defined as depicted in Fig. 1 below.

```
A ← B, C.
A ← ¬D.
```

Fig. 1. Example of Prolog rules.

In the first proposition, we declare that in order to verify A, we have to check whether B and C can be proved. The arrow means the implication relation between A on one hand, and B and C on the other hand: if B and C are proved (considered as true by Prolog), then A is true. The comma means a conjunction between B and C: if B is proved but C is not, then A is not true. The period means the end of the proposition. In these two rules, A is the "head" of the two rules; but the two "bodies" differ. As a consequence, there is two ways to prove A: the period at the end of the propositions implies a disjunction. A can be proved if both B and C are proved, but also if the negation of D is proved. The negation is marked "¬" (or "\+" in Prolog), meaning that in the second proposition, A is proved if the program cannot prove D.

To prove the rules, the program uses the knowledge base we defined. A, B, C and D are variables that the program will try to instantiate to constants that are stated in the knowledge base, i.e. replace the variables by constants. The unification is the operation of instantiating the constants where the variables are declared. Finally, backtracking allows the program to go back to higher rules when an answer is given as "false", in order to try every solution to reach a "true". In the end, the result replied by the program is an affirmation or refutation of the original query [6].

We consider Prolog as an effective programming language that is suitable to translate legal rules in a formal language. Its operation requires to declare a knowledge base, and a set of rules; which is appropriate to achieve legal modelling. Indeed, to check for legal compliance, the controller needs some inputs (a knowledge base) and some obligations to fulfil with (a set of rules). However, Prolog has its own limits, that we need to avoid .As an example, a legal provision doesn't describe all the notions and elements involved, whereas Prolog would need a reference to every element concerned in this provision to translate the legal rule. Consequently, in our work, the constraints due to Prolog syntax impacted the way we translated the legal rules.

As a first step, the modelling of the IT system and of the GDPR allowed us to build the elements we need as inputs to check compliance of an IT system based on its logs.

3 IT System and GDPR Modelling

In Fig. 2, we present the elements on which our work is based. First, the environment is composed of devices and networks on one hand, and the GDPR's provisions and other sources on the other hand. These elements help us create two models, the IT system model and GDPR model.

Firstly, the IT system represents any information system that is processing data, under the responsibility of a data controller. Then, we provide a model of this system that describes the evolution of the data processing, through actions and states.

On the other hand, the GDPR provides broad provisions and obligations that data controllers need to comply with, when processing personal data. Our approach focuses on Articles 5 and 6, and other articles that relate to these ones. However, the GDPR itself is not sufficient to understand the concepts stated (in the provisions) and their scopes; to this end, we need to read other sources, such as opinions of the Article 29 Data Protection Working Party, or the French supervisory authority opinions' (CNIL) or take into consideration the recitals of the GDPR.

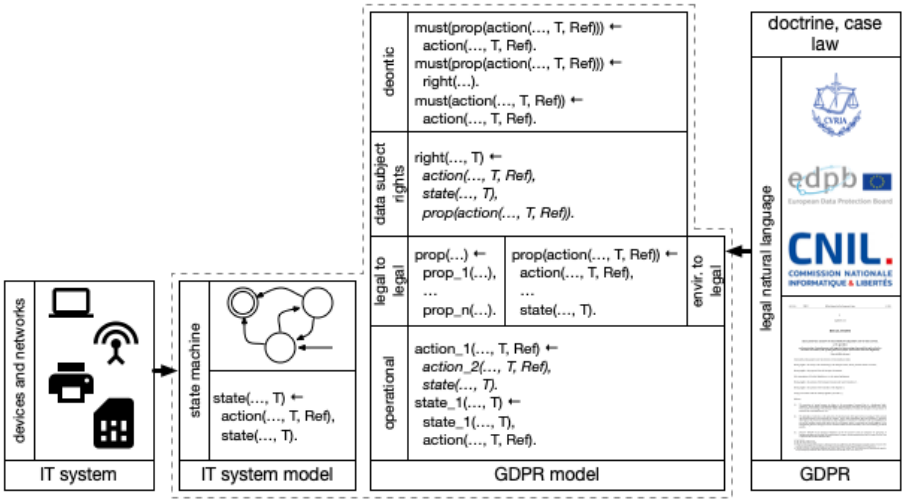

Fig. 2. IT system and GDPR models.

Section 3.1 describes the model and syntax used in the IT system, providing some examples; and Section 3.2 focuses on the model and syntax of the GDPR's rules.

3.1 IT System Modelling

We consider any information system that is processing data. We assume that the data is personal data (Article 4(1)). This system is modelled so that actions and states express the evolution of the system, i.e. the evolution of every action happening about the data processing. One of the main contributions of this work is the use of the "time" indicator, in the IT system model and in the GDPR model. A first section will present the state machine, and two following sections will describe how we consider actions and states in this system model.

State Machine. The state machine describes the evolution of the machine, through the states of that machine. In order to change of state, an action must happen. For instance, if an action in the processing happens, a state may be reached. But, to reach a state, another state may be required. This idea is expressed in Fig. 3.

```
<state>(…, T) ←
    action(…, T, Ref),
    …
    <state>(…, T).
```

Fig. 3. Pattern for state machines modelling.

We use the "<" and ">" quotation marks to indicate that the word will be substituted in the language. The next two sections explain how we define actions and states in our work.

Actions. All actions are written following the pattern shown in Fig. 4: an action (action) consisting in an Operation, is performed by someone, identified as a Doer. This action happens at time (Time) and holds a reference (Reference) to allow a direct reference later. For instance, the reference consists in a document the data controller must record in order to prove a consent has been given. This reference can then be expressed later when the data subject withdraws his or her consent.

```
action(Doer, Operation, Time, Reference).
```

Fig. 4. Pattern example of actions modelling.

States. On the contrary, a state is expressed with a different pattern as presented in Fig. 5.

```
<state>(PersonConcerned, Object, PersonalData, Time).
```

Fig. 5. Pattern example of states modelling.

The state (state) concerns a person (PersonConcerned), who is directly involved in the situation which is described; for example, the data controller or the data subject. This state relates to an element (Object),for instance, a request; and is linked to personal data (PersonalData). In addition, the state is reached at a time (Time). Even though actions are written with the word "action", here the word "state" doesn't appear in this syntax: it is always replaced by an expression describing the state, including a present perfect verb. For instance, "isProvided" or "isMinimised" are expressed as states, and replace the word "state". This choice of a present perfect verb is made in order to faithfully express a state of the system.

Therefore, the model of the IT system expresses that if an action and/or a state is proved by the program, then a state is reached in the system. Fig. 6 illustrates the conditions for a state to be reached; here, for a consent to be considered as withdrawn. If the two actions are completed, then the state is considered as reached:

```
isWithdrawn(DataSubject, document(ProofConsent), PersonalData, Time_Withdrawal) ←
    action(DataSubject, withdrawsConsent(DataSubject, Purpose, PersonalData),
        Time_Withdrawal, ReferenceProcessing),
    action(DataController, recordsDocument(document(ProofConsent, DataSubject),
        Time_Withdrawal, _).
```

Fig. 6. Rule for withdrawn consent.

To verify this state, two actions must happen: the data subject withdraws his or her consent, and the data controller records a document related to the consent and its withdrawal.

Actions and states are also used in the GDPR model, at an operational level. Indeed, this modelling of the GDPR provisions is structured into several levels to articulate the rules.

3.2 GDPR Modelling

The GDPR provides rules to ensure data protection for subjects whose personal data are processed. As stated before, we face difficulties reading the GDPR's rules; the scope and the wording of the text bring us to adopt an approach that remains close to the terms. We choose not to provide too much details in our model, leaving a supervisory authority or a competent person the power to decide on the meaning behind a given word (for instance, what "a task carried out in the public interest" in Article 6(1)(e) may refer to).

The other difficulty for us is to derive logic rules from the European regulation: the syntax we use has to reflect the rules in their extent, without translating beyond or above the legal text. To achieve this goal, we are driven by previous works on legal modelling, that focus on expression of rights, permissions, obligations, but also on implied rights, permissions and obligations drawn from the provisions [7].

Furthermore, these logic rules need to be articulated in order for a logic organization to be designed between them. For instance, we need to know what obligation relates on another one. The GDPR does not systematically provide such logic articulation between the provisions, but introduces some cross-references. Thus, modelling the GDPR leads us to separate the rules into several levels, following a hierarchy that is deduced from the text. We then propose an architecture that is drawn from the GDPR's rules.

However, as stated before, the GDPR model must be expressed using the syntax used for the state machine, i.e. the syntax expressing actions and states. As a consequence, an operational level in the GDPR model is expressed through actions and states. These actions and states will then be linked to the GDPR's provisions.

In the following, we will detail each level from the GDPR model from Fig. 2.

Deontic Level. This level in our model aims at expressing obligations for the controller, when an action occurs in the system. Fig. 7 provides an example of the use of deontic concepts about the GDPR's rules.

```
must(<prop>(action(..., T, Ref))) ←
    action(..., T, Ref),
    \+ <prop>(action(..., T, Ref)).
```

Fig. 7. Pattern for obligations modelling.

This pattern expresses a deontic obligation for a doer, when an action occurs in the system, and the obligation on this action is not fulfilled yet. If the two conditions can be proved by the program, then the controller needs to comply with this top-obligation. First, we explain the use of "must" in these propositions.

Must. Deontic obligations are implied in the GDPR, but we formulate them in order to detect the obligations that the data controller has to comply with, when an action happens. This deontic idea is expressed through the word "must", meaning the data controller must comply with a legal obligation, if the relevant action happens, as shown in Fig. 8. Thus the deontic level uses the actions recorded in the system to infer obligations for the controller.

```
must(processesCompliant(action(DataController, processes(DataSubject, Purpose,
        PersonalData), Time_Process, Reference))) ←
    action(DataController, processes(DataSubject, Purpose, PersonalData),
        Time_Process, Reference).
```

Fig. 8. Rule for compliant processing obligation.

Data Subjects Rights Level. Another part of the program deals with requests from data subjects concerning their rights. The idea is that the controller must accept the request concerning a right if the conditions are proved by the program. To this aim, an action may be required, or a state, to verify that the conditions are checked.

Property. A property pertaining to an action is denoted by "prop". For instance, an action representing a data processing might be "processesCompliant" or not, which is a property. If an action is not verified (ie is not compliant), then the data controller may have an obligation to accept a request from a data subject as shown in Fig. 9.

```
<isToSucceedRequest>(…) ←
    action(…, T, Ref),
    <state>(…, T),
    \+ <prop>(action(…)),
    \+ action(…, T, Ref).
```

Fig. 9. Pattern for success for data subject request conditions modelling.

For example, we express the conditions for a request about the right to object to be successful in Fig. 10.

```
isToSucceedObjectionRequest(DataSubject, objection(Reference), PersonalData, _) ←
    action(DataController, processes(DataSubject, Purpose, PersonalData),
        Time_Process, Reference),
    action(DataSubject, asksForRight(objectsProcessing(DataController, Purpose),
        Time_Request, Reference)),
    Time_Process =< Time_Request,
    action(DataSubject, motivatesRequest(DataSubject, Purpose, PersonalData),
        Time_Request, Reference),
    \+ action(DataController,
assertsLegitimateGroundsForProcessingOverrideRightsFreedoms(DataSubject,
        Purpose, PersonalData), Time_2, Reference),
    Time_Request =< Time_2.
```

Fig. 10. Rule for success of the right to object requests.

Three actions have to be recorded in the log, and one has to not be proved by the program. Precisely, the first action is a processing of personal data, the second one is a request from the data subject about the right to objection ("asksForRight"); the third one expresses the data subject motivating its request ("motivatesRequest"). Then, the last proposition refers to an action from the controller who asserts that he finds legitimate grounds for the processing which override the rights and freedoms of the

data subject. Indeed, if this action is not proved, then the controller can't refuse the objection to the processing, he must accept the request.

Translating these conditions for the right to objection to succeed, enables us to state later in the model that the controller must stop the processing when this property is proved.

"Legal to Legal" Level. This part in our model aims at expressing how the legal rules are articulated between each other. In that way, several sub-obligations are needed to verify if a higher obligation is considered as fulfilled (see Fig. 11 below).

```
<prop>(action(…, T, Ref)) ←
    <prop_1>(action(…, T, Ref)),
    <state>(…, T),
    action(…, T, Ref),
    <declaration>(…),
    …
    <prop_n>(action(…, T, Ref)).
```

Fig. 11. Pattern for legal to legal rules modelling.

Again, we use "prop" to express a property upon an action. This property is proved if other properties upon the same action are proved. The obligation ("prop") and sub-obligations ("prop_1", "prop_n") are always linked to the same action performed by a doer (Doer). But a state or an action may also be required to verify the top property.

Finally, we need to add some declarations ("declaration") that bring values we need to resolve the query.

For instance, Fig. 12 illustrates the conditions for a transparent processing to be proved by the program. Here, the sub-obligation to be fulfilled is "providesTransparentInformation", and an action and a state ("isDemonstrableCompliance") are also required.

```
processesTransparent(action(DataController, processes(DataSubject, Purpose,
        PersonalData), Time_Process, Reference)) ←
    action(DataController, processes(DataSubject, Purpose, PersonalData),
        Time_Process, Reference),
    providesTransparentInformation(action(DataController, processes(DataSubject,
        Purpose, PersonalData), Time_X, Reference)),
    Time_X =< Time_Process,
    isDemonstrableCompliance(demonstratesTransparency(document(ProofProcess)),
        DataController, Reference, PersonalData, Time_Compliance).
```

Fig. 12. Rule for transparent processings.

In addition, we introduce the use of time indicators. We state in Fig. 12. that the time of the process ("Time_Process") has to be greater (the symbol "=<") than the time for the controller to deliver the information ("Time_X"). If not proved, this time constraint would terminate the query resolution with a false reply.

"Environment to Legal" Level. This level in the architecture deals with the actions and states that will prove an obligation is fulfilled. In Fig. 13, we show the pattern: to verify a property is fulfilled, an action or a state is required, and some declarations are added to provide more details about the elements involved in the rule.

```
<prop>(action(…, T, Ref)) ←
```

```
    action(…, Time_n, Ref),
    <declaration>(…),
    …
    <state>(…, Time_n).
```

Fig. 13. Pattern for environment to legal rules modelling.

For instance, the example below in Fig. 14 illustrates the obligation for the controller to process personal data fairly ("processesFair").

```
processesFair(action(DataController, processes(DataSubject, Purpose,
        PersonalData), Time_Process, Reference)) ←
    action(DataController, assertsFairInformation(DataSubject, Purpose,
        PersonalData), Time_X, Reference),
    action(DataController, assertsFairProcessingTowardsDataSubject(DataSubject,
        Purpose, PersonalData), Time_Process, Reference),
    Time_X =< Time_Process,
    isDemonstrableCompliance(demonstratesFairness(document(ProofProcess)),
        DataController, Reference, PersonalData, Time_Compliance).
```

Fig. 14. Rule for fair processings.

For this obligation to be verified, two actions performed by the controller must happen, and a state has to be reached. The first action is an assertion from the controller who confirms that the information provided to the data subject is fair ("assertsFairInformation"). This action is linked to the transparency principle of processings, but we choose not to bind transparency and fairness together here. The second action refers to a fair processing towards the data subject ("assertsFairProcessingTowardsDataSubject") and does not concern transparency.

Indeed, the translation of the fairness principle led us to some difficulties. The fairness principle appears to have a key role among data protection rules, according to its repetition in the GDPR. However, the regulation doesn't give any detail nor condition to consider a fair processing. In order to translate "fairness" into the two actions described earlier, we read other sources, such as the Guidelines of the Data Protection Working Party on transparency under the GDPR [8]. We understand that fairness is not an autonomous principle. In the Guidelines, fairness is deeply linked to transparency of the information delivered to the data subject[1]. However, Recital 39 in the GDPR states that "natural persons should be made aware of risks, rules, safeguards and rights in relation to the processing of personal data" to fulfil the fairness principle. Yet, we chose not to link fairness to transparency, and remain close to the Recital 39, because the two principles are clearly divided in Article 5.

Consequently, we express two actions by the controller; the first one reflects the relation between fairness and transparency of the information; the second one refers to the substance of the information. In doing so, we assume that the controller or the DPO will perform a positive action to assert that fairness of the processing and fairness of the information are satisfied. The last proposition in the rule refers to the accountability obligation: the state "isDemonstrableCompliance" would be reached if the controller can demonstrate he is compliant about fairness.

[1] "transparency is intrinsically linked to fairness and the new principle of accountability under the GDPR", p.5.

Operational Level. The last level in this architecture is an operational one, that links actions and states. The pattern shown in Fig. 15 allows us to define how a state may be reached in the system: when actions and states are proved by the program.

```
<state>(…, T) ←
    action(…, T, Ref),
    <state_1>(…, T),
    <declaration>(…).
```

Fig. 15. Pattern for operational rules modelling.

As an operational level example, Fig. 16 provides the definition of a processing necessary for the performance of a task carried out in the public interest.

```
isNecessaryProcessingForPerformanceTaskPublicInterest(DataController,
        TaskPublicInterest, PersonalData, Time_X) ←
    action(DataController, processes(DataSubject, Purpose, PersonalData),
        Time_Process, Reference),
    taskPublicInterestByDataController(TaskPublicInterest, DataController),
    action(DataController,
        assertsProcessingNecessaryForPerformanceTaskPublicInterest(DataSubject,
        Purpose, PersonalData), Time_X, Reference),
    Time_X =< Time_Process.
```

Fig. 16. Rule for public interest processings.

Two actions are required in order to reach this state. The first action concerns the processing of personal data undertaken by the controller; the second action relates to the controller asserting the processing is necessary for the task he is carrying out in the public interest ("assertsProcessingNecessaryForPerformanceTaskPublicInterest"). Through a declaration, we are able to express which task is carried out by this data controller in the public interest ("taskPublicInterestByDataController"). Finally, the program will compare two "time" indicators: the assertion of the controller must happen before the beginning of the processing.

Consequently, we have two models, the IT system model that must reflect exactly the state of the IT system, and the GDPR model, that must reflect exactly the data protection rules from the regulation. Then, the compliance checker we propose will compare a log from the IT system (the list of actions that happened in the system), with the two models. This dynamic approach is carried out to check for compliance about a processing of personal data.

4 Log-Based Compliance Checking

When checking for potential violations under the GDPR's provisions (Fig. 17), the compliance checker gathers the two models, and takes as input the log of every actions that happened in the system. However, we need additional information, about the context of the processing. This information is provided through a complementary source, and does not depend on the log: the set of declarations. Then, the compliance checker will reply a query about a potential violation of the GDPR's rules translated in the models. Finally, a diagnosis is provided, and determines if the processing undertaken presents any violation.

Section 4.1 will present the context and the log of the system as inputs to the compliance checker; Section 4.2 will describe the compliance checker, illustrating a query. Finally, Section 4.3 will give a small example to show how violation can be detected.

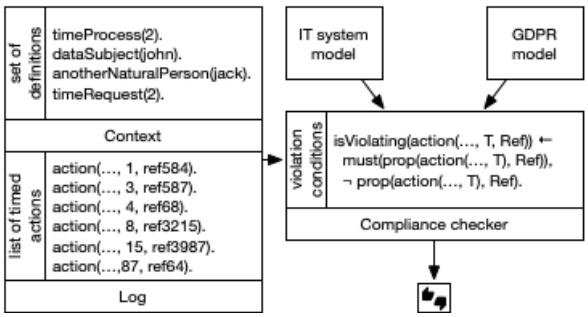

Fig. 17. Compliance checking architecture.

4.1 IT System Logs

As input to the compliance checker, the log of the system is composed of the actions that happen about a certain processing of data. Here, the values of the actions are concrete, i.e. instantiated. Therefore, the log represents the knowledge base of the compliance checker. In addition, the actions are recorded in the log with the time when they happen.

```
action(google, givesInformationRightToWithdrawConsent(jack), id_jack, 5,
privacy_form).
```

Fig. 18. Example of an instantiated action.

Fig. 18 illustrates an example of an action recorded in the log; the data controller (Google) provides information about the right to withdraw the data subject's (Jack) consent, at time 5, including a reference to the form used to provide this information ("privacy_form"), and the personal data being processed ("id_jack").

Furthermore, the compliance checker relies on another knowledge base, which describes the concrete context of the processing of personal data. These declarations permit to add some information to the actions from the log. Fig. 19 illustrates three declarations.

```
taskPublicInterestByDataController(police, intelligenceService).
taskOfficialAuthorityByDataController(dataProtection, supervisoryAuthority).
legitimateInterestOfDataController(fraudPrevention, anyAuthority).
```

Fig. 19. Example of an instantiated context declaration.

These declarations allow us to know that "police" is designated as a task carried out by the "intelligenceService" which is the data controller here. In the same way, the "supervisoryAuthority" carries out a task that is "dataProtection". The third declaration states that "fraudPrevention" is a legitimate interest defended by the data controller ("anyAuthority").

As a consequence, the log and the context are used as a knowledge base to detect violations with data protection's provisions.

4.2 Compliance Checker

The compliance checker gathers the two models, and receives as input the knowledge base presented in Section 4.1. The checker will then try to solve to the query we address. In order to do so, the compliance checker will try to verify the rules, relying on the knowledge base. One important assumption here is the log soundly reflects the IT system behavior: all actions appearing in the log should have actually happened in and all actions actually happening should appear in the log.
Using Prolog's instantiation of the variables, the checker replies if a processing of personal data performed through the system is violating the rules, or not. To this aim, the query is defined as it follows in Fig. 20 below.

```
isViolating(action(…, T, Ref)) ←
    must(<prop>(action(…, T, Ref))),
    \+ <prop>(action(…, Time, Ref)).
```

Fig. 20. Pattern for violation modelling.

The checker can thus be queried about an obligation for the controller to fulfil upon an action ("must"). This part refers to the deontic level we exposed earlier. The second proposition asks if the property upon the same action is not proved ("\+ prop"). If the obligation exists and if the property is not proved (i.e. the conditions cannot be met by the program), then the program will reply "isViolating", meaning that the action is violating the GDPR's provisions.

For instance, in Fig. 21, we express that if the data controller has an obligation to process personal data compliantly ("processesCompliant"), and if this processing is not proved as compliant ("\+ processesCompliant"), then it is violating the relevant data protection rules.

```
isViolating(action(Doer, processes(DataSubject, Purpose, PersonalData),
        Time_Process, Reference)) :-
    must(processesCompliant(action(Doer, processes(DataSubject, Purpose, Personal
        Data), Time_Process, Reference))),
    \+ processesCompliant(action(Doer, processes(DataSubject, Purpose, Personal
        Data), Time_Process, Reference)).
```

Fig. 21. Rule for compliance violation detection.

Therefore, the compliance checker's role is to detect when a processing doesn't fulfil the obligations needed to achieve compliance. The time of actions are taken into consideration, as part of the compliance checking.

We do not claim to substitute entirely the task and experience of a person who is competent for checking compliance with data protection (as a lawyer), so the checker does not provide an answer that is to be taken for granted, our aim is to increase the level of trust in the compliance of the system.

4.3 Example

In this section, we give an example of a rule that is analyzed by the compliance checker. The rule described below in Fig. 22 expresses the three actions needed for an information provided by the controller to be proved as being transparent. (These conditions are the clarity, the conciseness, and the accessibility of this information.)

```
isProvidedTransparentInformation(DataController, Reference, PersonalData, Time_X) ←
    action(DataController, givesClearInformationDataSubject(DataSubject,
        Purpose, PersonalData), Time_X, Reference),
    action(DataController, givesConciseInformationDataSubject(DataSubject,
        Purpose, PersonalData), Time_X, Reference),
    action(DataController, givesAccessibleInformationDataSubject(DataSubject,
        Purpose, PersonalData), Time_X, Reference).
```

Fig. 22. Rule for information transparence.

On the other hand, the logs of the actions in the IT system are recorded as shown in Fig. 23 (with the data controller being a bank).

```
action(bank, givesClearInformationDataSubject(john, archiving, johnBankDetails),
1, archives_X).
action(bank, givesConciseInformationDataSubject(john, archiving, johnBankDetails),
7, archives_X).
action(bank, processes(john, archiving, johnBankDetails), 28, process_john).
```

Fig. 23. Log excerpt.

In this setting, we prompt the query "isViolating" about the processing of John's personal data, with the instantiations from the above as depicted in Fig. 24.

```
?- isViolating(action(bank, processes(john, archiving, johnBankDetails), 28,
process_john)).
```

Fig. 24. Violation query prompt.

The program will then verify whether there is an obligation to process the personal data compliantly, and then check for every condition to achieve compliance. Considering the transparency rule in Fig. 22, the program will try to instantiate the variables of the rule. But as we can see in Fig. 23, only two actions – in top of the processing action itself – are recorded in the log, whereas the rule requires three actions. Indeed, "givesAccessibleInformation" doesn't appear in the log. As a consequence, the program will reply "true" and thus deduced that the processing is not compliant. To be able to capture the condition that is not fulfilled, the "trace" module of Prolog provides instrumented ways to guide the auditor to the missing elements.

5 Related Works

Formal methods, in spite of their weaknesses, are largely accepted as a solution to translate legal rules into a modelling even though they face limitations and should thus be adequately tailored to address well-identified domains. Tschantz and Wing give several examples in which these methods are relevant to check privacy violations [9]. In our approach, we focus on "a posteriori" verification of compliance to data protection rules, when processing of personal data has already happened. However, many works relate to privacy by design [10], for instance to implement data minimization into engineering systems. Also, in this work, we chose to focus on the GDPR only, though a few other documents were also used to help analyze the concepts that are ambiguous in the regulation. Our aim was to get a model as close as possible to the legal wording of the GDPR. Other work proposed approaches relying on many sources, not only legal

ones [11], which would require more work to be appropriately handled through formal methods.

The Prolog language appears to be a relevant programming choice for expressing legal rules; we can relate to former works that chose this programming logic to resolve queries about compliance to legislations. Maxwell and Antón [5], for instance, give a methodology for production rule models concerning HIPAA based on Prolog. They have been able to cover rights, obligations, permissions and declarations, and implied rights, obligations, and permissions. They provide a model that checks the requirements, so that inconsistent requirements are flagged and then rechecked with legal domain experts. We adopted the same method that flags potential issues that must be analyzed by a Data Protection Officer (DPO). The main difference between our work and theirs is that they aim at facilitating communication between requirements engineers and legal domain experts. Also, previous work focuses on short parts of legal texts, that is straightforward and documented.

When translating legal rules into a formal language, Breaux, Vail and Antón bring a methodology for automatically implying an obligation upon the counterparty from a right [7]. We took inspiration of this work to help understand all the implications of the GDPR's rules, but have been limited by the expressivity of the GDPR, compared to legal requirement coming from HIPAA. As we aimed at remaining close to the Regulation itself, we decided not to go beyond and neither states new rights nor obligations that are deducible from them.

Furthermore, even if previous works about legal modelling of data protection rules provide interesting methods to translate them, we cannot rely entirely on these methods because the GDPR's rules are highly nested and interconnected. Our approach is different from previous works: the syntax is new, mainly divided into actions and states (leading to a more formal semantics than what is proposed in [12] for instance; and organizing the rules, from the "top" one, reasoning on legal implications, to go "down" to the actions required in order to verify the legal rules. Finally, our approach brings a new element, the variable for the time of the action. Not only this allows us to consider whether an action takes place before or after another one but we can also consider durations and detect violations of time limitations.

Working on logs as a mean to secure compliance and accountability is not new [13]. In this paper, the authors discuss what information must be included in the logs: only essential information, but also contextual ones, as we also used. They also rise two challenges we met in our work: the ambiguity in the logs, when the log is not explicit enough; and the need for human verification in complement to the log analyzer. However, their formalism is further from the GDPR than ours and would require extensions to address the same data protection properties.

6 Conclusion

In conclusion, we built two concurrent models – one for the IT system and the other one for the GDPR – that are used by a compliance checker. This checker is able to provide an answer about which compliance aspects may not be properly covered by the data controller by relying on a log which records all actions having occurred in the IT system. The checker will thus raise flags for points which are to be checked by a DPO.

These alerts may be false alerts because of a lack of contextual information, which can then be added by the DPO, or because of actual violations, for which the DPO can proceed to a remedy, helped by the compliance checker showing which requirement is not met (ie., which action was expected in the log and not found). A prototype of the core principles of these models has been implemented. Future work will consist in evaluating, improving our contribution, and implementing it through a bigger scale proof of concept.

The requirements that we stated above, both legal and technical, brought us to provide an extensible architecture that organizes how the rules from the GDPR model can be operationalized through an interface between technique-oriented levels (modelled as state machines) and legal-oriented levels (modelled as property definitions). These models are then used by a deontic-based compliance checker to reason about possible violations.

Nevertheless, we faced limitations, both legal and technical. The main legal difficulty concerns the elusive notions that the GDPR states inviting us to rely on soft law. Then, the heart of the technical difficulties lies at the articulation of rules and exceptions, that can lead to long sets of rules, especially when some of them might have to be repeated to several places because of cross-references. Better handling of this could have been possible at the cost of more elaborate logic patterns and of more distance from the GDPR structure.

We plan to extend this work by developing more properties taking benefit of the time indication embedded in the actions (for instance by handling obligations concerning notifications in case of a breach). In addition, we also plan to implement a way to prioritize violations in order to help a DPO to take the best action possible. The prioritization strategy should be parametrizable, depending on the wish of the DPO. This could be, for instance, based on a list of principles which have been deemed to be of particular importance at corporate level, for a given sector, or based on the output of a data protection impact analysis.

References

1. Le Métayer, D.: Formal methods as a link between software code and legal rules. In: Barthe G., Pardo A., Schneider G. (eds) Software Engineering and Formal Methods. SEFM 2011. Lecture Notes in Computer Science, vol 7041.
2. Guarda, P. and Zannone, N. (2009): Towards the Development of Privacy-Aware Systems. Information and Software Technology, 51(2):337-350.
3. EU Parliament, Council of the EU: REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). Official Journal of the European Union, L119/1, May 4 2016.
4. Mina Deng, Kim Wuyts, Riccardo Scandariato, Bart Preneel, and Wouter Joosen: A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements: Requirements Engineering Journal, 16(1):3–32, 2011.
5. Maxwell, J. C., Antón, A. I.: Developing production rule models to aid in acquiring requirements from legal texts. 17th IEEE International Requirements Engineering Conference 2009, vol., pp. 101-110.

6. Lloyd, J. W.: Foundations of logic programming. 1st edn. Springer-Verlag, Berlin (1984).
7. Breaux, T. D., Vail, M. W., Antón, A. I.: Towards regulatory compliance: extracting rights and obligations to align requirements with regulations. 14th IEEE International Requirements Engineering Conference 2006, pp. 46-55.
8. Article 29 Data Protection Working Party, Guidelines on transparency under Regulation 2016/679, 9 November 2017.
9. Tschantz, M.C., Wing, J.M.: Formal methods for privacy. In: Cavalcanti A., Dams D.R. (eds) FM 2009: Formal Methods. Lecture Notes in Computer Science, vol 5850.
10. Gürges, S., Troncoso, C., Diaz, C.: Engineering privacy by design. 2011
11. Visser, P., Bench-Capon, T., van den Herik, J.: A method for conceptualising legal domains: an example from the Dutch unemployment benefits act. In: Artificial Intelligence and Law (1997) vol. 5: 207-242.
12. Palmirano, M., Martoni, M., Rossi, A., Bartolini, C. Robaldo, L.: PrOnto Privacy Ontology for Legal Compliance. In Proceedings of 18th European Conference on Digital Government, 2018.
13. Butin, D., Chicote, M., Le Métayer, D.:Log Design for Accountability: 2013 IEEE Security and Privacy Workshops.