Regulatory Services to Automate Compliance with Ex-post Enforcement

Lu-Chi Liu[1], Mostafa Mohajeri Parizi[1], L. Thomas van Binsbergen*[1], and Tom van Engers[2]

[1] Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
[2] Leibniz Institute, University of Amsterdam/TNO, Amsterdam, The Netherlands

This version of the contribution has been accepted for publication after peer review but is not the Version of Record and does not reflect post-acceptance improvements or corrections. The final publication is available at Springer

Abstract. Software systems are expected to comply with norms from a variety of sources. However, the increasing volume and the dynamism of norms make it rather difficult to make and keep software systems fully compliant by design. This paper proposes an architecture of (norm-)regulated software systems that facilitate compliance with high-level policies through regulatory services. Our architectural model has the flexibility to adapt to norm modifications and to norm violations by including both ex-ante as well as ex-post enforcement mechanisms. The proposed solution is assessed using a case study and a prototype implementation in the eFLINT and ASC2 languages for modeling norms and software/social agents respectively.

Keywords: Regulated systems· Executable specifications· Runtime compliance· Actor-oriented programming· Distributed systems· Regulations· Law· Contracts.

1 Introduction

Law, policy, regulations, contracts, and standards contain norms that guide behavior and raise expectations for parties to act accordingly. Software systems with automated (or autonomous) decision-making procedures need also to be compliant with those norms and are expected to faithfully capture the (organizational or governmental) policies behind the services they implement.

Ideally software systems are "compliant by design" and only admit permitted behavior. However, norms may contain so-called 'open concepts' and qualification of concepts reflecting reality, i.e. mapping such concepts onto institutional concepts is a challenge and may require judgment from authorities (e.g. a judge). Furthermore, designing fully compliant software systems is complicated due to the large volume and dynamic nature of norms [4]. In order to achieve compliance we need services/mechanisms that provide legal reasoning, monitoring, and enforcement.

As part of a collaboration between several research institutes and industry partners, we investigate data-sharing architectures for systems in which norms from a variety of

* Corresponding author: ltvanbinsbergen@acm.org

sources can be embedded as so-called regulatory services supporting both ex-ante and ex-post enforcement enabling subjective responses (as well as autonomous responses) to occurrences of violations, producing traces from which detailed reports (audit trails) can be constructed for the purposes of *diagnosis*. These regulatory services are derived from (high-level) specifications written in a domain-specific language for expressing norms: eFLINT [3]. As we will show, such regulatory services help to achieve transparency in the norms applied and hence improve the trust that data-sharing and data processing are legitimate (avoiding liabilities and costly punishments in case of norm violations). We opt for an actor-oriented approach[3] in which norms are enforced by actors that deliver rewards or punishments, resembling regulated *social* systems, and compliance is not necessarily hard-coded into the system.

In this paper we present the outline of our approach and we will demonstrate some of the tools and languages we use to develop prototypes and perform case studies. In particular, we show how normative specifications written in eFLINT [3] and agent specifications written in the ASC2 language [27] form executable models of regulated systems. In this paper we will present:

- An architectural model of regulated systems that involve high-level policies such as regulations and contracts, normative services for reasoning about policies, and enforcement services for responding to policy decisions and violations;
- A demonstration of the eFLINT language and reasoner enforcing high-level policies in a distributed system governed by norms changing over time, and;
- A discussion on (partially) automated ex-post enforcement and the real-world scenarios that could potentially benefit from this design.

2 Related Work

This paper contributes to the idea of regulatory services and regulated systems. To ensure that compliance is properly anchored, it is crucial to have a clear set of rules, a properly configured framework, and implementable associated processes in a regulated system. One typical example is policy-based management systems, mainly used for access control[4]. Such systems use various policy-related components in their framework to evaluate whether a certain *Access Request* is allowed for the requested *Target Resource* and to grant access to the requesting party if it is allowed. The access request is evaluated against a given set of access control policies. Widely-used access control protocols, for example XACML [31] and SAPL [21], contain components that share similarities to what we have in our regulatory services. While access control systems aim at access control (policies) and consider full compliance in their context, our approach depicts a more general architecture for regulated systems and includes ex-post enforcement.

We are not the first authors to suggest the usage of ex-post enforcement. Literature addressing the topic of modeling and enforcing e-contracts often stresses that it

[3] The term "actor" is used both for software components and human users of a software system.

[4] We argue that access control systems are too limited to comply with the demands from e.g. the GDPR as one should control access as well as other data processing (which should meet some well-defined purpose), hence we rather speak about usage control.

is important to deal with violations as it is possible to have contract breaches, especially in a business setting. In [7], Chiu et al. map the contract clauses to ECA (event-condition-action) rules to provide operational semantics, specifying what action to take under what condition/event. They also proposed an implementation outline to illustrate a communication protocol for parties to publish and collect event information. The work in [28] introduces a contract compliance checker that observed B2B interaction events and inferred whether they are contract compliant based on its observation. Clauses related to violations are converted into exceptions in their model. Similar ideas were employed also in the field of multi-agent systems (MAS). For example, a formal framework inspired by supervisory control theory (SCT) is presented in [10], proving that regimentation-, sanction-, and repair-based enforcement can be modeled as special supervisors from SCT. Although sharing the same concept of ex-post enforcement, this paper focuses on compliance with high-level policies and considers not only violations but also modifications of norms.

The connection between norms and multi-agent systems has been studied extensively in the field of normative MAS (nMAS) [5]. In a complex system with heterogeneous agents that may have conflicting goals, norms can provide an efficient way of coordination between agents typically in the form of agent organizations [25]. Most of the studies in nMAS are concerned with the theoretical aspects like the interactions between governing norms and individual agents' desires and goals [8,11,29]. However, there are also works that introduce practical approaches for embedding some degree of control in a MAS by the enforcement of norms. An overview of these approaches can be found in [9], analyzing norm enforcement architectures based on criteria like supporting automatic enforcement, different levels of norms that describe actions and/or state of affairs, dynamicity of the norms in the sense that they can be adopted, dropped or changed. They also introduce practical criteria like execution efficiency and the centralized and decentralized nature of enforcement.

The authors of [9] also introduce the MaNEA, a norm-enforcing architecture aimed at controlling norms in open MAS. This architecture utilizes a distributed model of agent organizations that includes norm manager nodes with prescriptive norms in the form of deontic notions (obligation, permission, and prohibitions) and norm enforcer nodes that can observe the system, detect violations and respond to them with a punishment/reward method. Their approach differs from ours firstly because they focus on controlling a MAS as a whole, they assume full information of every action and message within the system from the perspective of enforcers nodes, while in our architecture —and arguably in real life– this is not the case. Secondly, they only take into account prescriptive norms in the form of deontic notions, while our approach goes above that by also utilizing norms as a coordination mechanism between agents.

Another thread of research that is close to this work is Compliance Management Frameworks (CMF). In any organization, there is always the concern to verify if the business activities of the organization are compliant with the governing rules and policies. This typically includes encoding the relevant rules into some normative specifications and utilizing a reasoner for automated verification of normative specifications over business process specifications. There are multiple CMFs introduced in the literature [12,19,20]. One example of a CMF is the COMPAS framework presented in

[12] that is designed for service-oriented architecture-based systems. In COMPAS, Linear Temporal Logic (LTL) is used to specify compliance requirements of a system and Business Process Management and Notation (BPMN) to identify the business process model. Then, formal model-checkers are used for verification of the compliance specifications over business process specifications. Like CMFs, we are also concerned with the compliance of a business process (albeit implicit in this work) with relevant norms. What makes COMPAS —and most other CMFs — different from our architecture is that the focus in a CMF is on one organization and its business, however, by modeling our system as a MAS, we are also concerned with possible interactions that may happen between multiple actors with possibly conflicting goals in the presence of multiple normative sources.

Lastly, we would like to mention the relevance of run-time verification, particularly run-time verification with reparations, to our work. Run-time verification is a mature field that aims to ensure the dependability of software systems during execution. It addresses various approaches to monitor the behaviors of a running system and to check whether the observed system behaviors satisfy certain properties. Other than some algorithmic properties (e.g., race condition and deadlock freedom), this paper relates more to formally specified properties (e.g., deontic logics and normative specifications). Similar to our ex-ante and ex-post enforcement mechanisms, there are two families of techniques taken in this field. The first one *run-time enforcement* targets the prevention of misbehaviors and failures, while the second one *healing* concerns the reactions towards observed undesired behaviors (e.g., reverting to a correct state). Run-time verification could be applied to a great number of domains, for example to verify system and business requirements in [14], contracts and behaviors of agents in [1], smart contracts running on Ethereum [16], and more in [17,33]. While most of them focus on one of the two families for specific applications, our work adopts a hybrid method, attempts to provide a versatile model, and considers the amendment of norms.

3 Modeling Norms, Software, and Social Agents

Motivated by the advent of computational systems, philosophers, legal theorists, and logicians from various backgrounds have attempted to formalize norms and automate normative reasoning. The problem with the existing formalizations is that they typically conflate the behavioral view on norms (concerning agents that act lawfully or not) [11,29] with a principled view on norms (concerning what is right) [18]. The frameworks in which norms are typically expressed consider primarily situations that can be captured by propositions [6]. However, in legal jurisprudence, Hohfeld, inspired by earlier work by Salmond [32], provided an alternative framework making relevant actors, their normative positions, and their actions first-class [22].

To formalize norms in this work we use the domain-specific eFLINT language [3] that implements the Salmond/Hohfeldian conceptualization. Hohfeld's approach is much more suitable for our work as we consider regulatory systems that involve social and software actors, and their interactions (e.g., different types of enforcement). The eFLINT language can be used to describe norms from various kinds of sources (e.g., GDPR, data sharing agreements among healthcare sections, internal policies from companies, etc.).

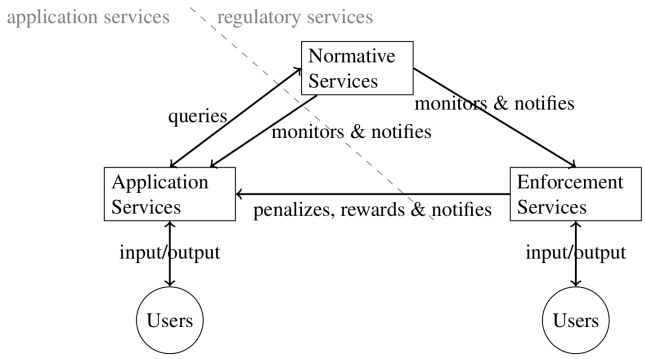

Fig. 1: High-level, schematic overview of regulated systems. The arrows describe relations between different types of actors, not necessarily lines of communication.

To increase robustness and persistence, eFLINT knowledge bases can be externalized and backed up. Finally, the language provides mechanisms to formalize (modifications of) norms modularly [2].

To model both the social actors and software components of a system, we utilize the belief-desire-intention (BDI) model of agency.The BDI model, typically used for representing humans' reasoning processes, is utilized to create computational agents that exhibit rational behavior. Furthermore, the BDI model has been investigated as the basis to create agents capable of reasoning with norms typically referred to as *normative agents* [8,11,29]. In this work, we model the agents via ASC2 [26,27], a BDI framework that utilizes an agent-programming language close to AgentSpeak(L) [30] which meets our technical requirements and can interface with the eFLINT reasoner. Although using ASC2, our approach is conceptually independent of how actors are implemented, assuming that an interface for monitoring, notifications, and queries can be established.

4 Regulated Systems

In our approach, we distinguish three types of actors based on their role within the regulated system and describe a minimal communication protocol between these actors. The following explanation is visualized in Figure 1. This diagram is not exhaustive nor prescriptive: other classifications of actors/components are possible, actors/components can be of more than one class and additional lines of communication may be present. A regulated system is composed of two main types of services: *application services* and *regulatory services*. The application services realize the core functionality of the system, i.e. the application. Some **application services** receive input from and send output to users of the system (e.g. clients) while others do not (e.g. servers). There could be multiple (different) instances of each service in a regulated system. These services are implemented as ASC2 actors. Regulatory services consist of normative and enforcement services. Each **normative service** administers a set of norms within the system and represents a particular interpretation and instantiation of these norms.

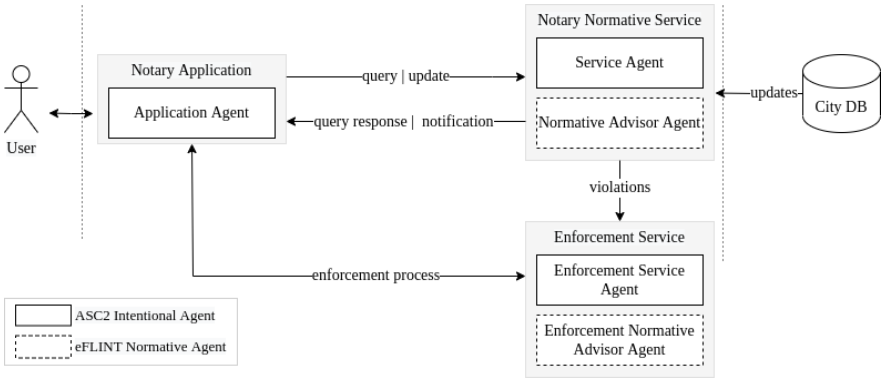

Fig. 2: This architecture for our case study demonstrates the proposed approach by adopting application, normative, and enforcement services. Each component is implemented as an ASC2 agent, furthermore, the normative and enforcement services also include an eFLINT instance.

An application service may directly query a normative service to, for example, gain insights into whether they hold specific duties or powers. Conversely, normative services may query application services in order to confirm specific facts. For example, a healthcare provider could be asked to verify a receipt in order to ensure that an individual has the power to demand a refund from their insurance company. The integration of normative actors and enforcement actors with the eFLINT and ASC2 languages is demonstrated in a case study in the next section.

5 Case Study

The case study involves a notary automating the process of maintaining the registration of real estate properties and mortgages with a policy around national insurance on mortgages that is updated from time to time, making it difficult to ensure and preserve compliance. This section is designed to represent a variety of issues that concern a regulated system and architectural requirements to address those issues. The study is divided into four subsections, each narrating a different architectural aspect.

Figure 2 shows the overall architecture of the system. The notary application is used by the notary to process property data in a certain city which includes a database of information about clients, properties, mortgages, etc. To make sure that all processes are in compliance with relevant regulations, the notary has implemented an automated normative service which includes a normative reasoner (an eFLINT instance) that maintains the institutional state of the system. Listing 1 shows an excerpt of the eFLINT norms specifications of the notary service.

At a technical level – through ASC2 – the normative service exposes endpoints that the application can utilize to update and/or query the institutional state. In our example this includes `+fact(Fact)` to report relevant data, `+performed(Act)` to report the

```
Fact article-one
  Identified by citizen * property
  Holds when first-property(citizen, property)

Fact article-two
  Identified by property
  Holds when property.value <= property-mean-value

Act issue-nim
  Actor notary
  Recipient citizen
  Related to mortgage, property
    When mortgage.citizen == citizen && mortgage.property == property
  Conditioned by article-one(), article-two()
  Creates nim-covered(mortgage)
  Holds When !nim-covered(mortgage)

Act register_occupant
  Actor municipality
  Recipient citizen
  Related to property
  Conditioned by owned_property(property) && !occupant()
  Creates occupant()
  Terminates (Foreach occupant : occupant
    When occupant.citizen == citizen && occupant.property != property)
  Holds when property
```

Listing 1: Snippet of notary's normative service eFLINT specification

performance of acts, and `?fact(Fact)`/`?enabled(Act)` to respectively query if a certain fact (or any sub-types such as acts and duties) holds and if a certain act is enabled according to the institutional state of the service. As the internal state of the normative service is updated, there may be inferred knowledge that will be sent to the application service as events; this includes assertion and retraction of facts, (violated) duties, and (violated) actions. The full code is publicly available online[5].

5.1 Querying the Normative Service

The first and most intuitive requirement for a regulatory service is the ability to respond to normative queries from the rest of the system. This means the service should implement an interface to provide normative information when queried, e.g., if an action is permitted in the current state or not. As specified in Listing 1, the act of issuing collective insurance on mortgages (NIM) is encoded as the act `issue-nim` and is preconditioned by: (Article 1) the mortgage pays for the first property of the receiver of the mortgage and (Article 2) the value of the property is less than the mean value of all properties in the city the property is located.

To see how the notary can use this service, assume a citizen Alice and a property with the address A1 and a value of 500k which is more than the current average property value. Also assume that this is Alice's first property (information provided by City DB in Figure 2) and that there is already a mortgage on this property not covered by NIM. Alice believes that because this is her first property, without being aware of Article 2,

[5] `https://github.com/mostafamohajeri/jurix-notary`

she can get covered by NIM. After she sends a request to the notary service, the service agent (automatically) queries the normative service as:

```
?enabled(issue-nim(Notary,Alice,
    mortgage(Alice,property(A1,500k)),property(A1,500k))).
```

to which the service will reply with `false`. Then, the application service can reply to Alice that issuing a NIM is not possible, with any extra argumentation that it is configured to provide. However, if the service agent decides to accept a NIM request without consulting the normative service, a message `+performed(issue-nim(...))` is sent to the normative service. In the case of Alice, this results in a `+violation(...)` message sent in response, indicating the performance of an illegal act. This example illustrated how the regulatory service can be queried to provide normative information about the current institutional state of the system. An important aspect of this architecture is the stateful nature of eFLINT's instances that allows the service to keep track of the institutional state of the system instead of just hosting static regulations or policies.

5.2 Normative Event Notifications

Alongside the ability to respond to queries, another requirement for a regulatory service is its ability to proactively notify other services about normative events that may occur within the institutional state of the system. Continuing with the case study, in the current situation, Alice cannot get NIM coverage. Fortunately, the notary provides a pre-registration service –external to the regulations– that notifies citizens about changes to their eligibility. As time goes on and the city develops, more expensive properties are registered, until eventually Alice's property falls under Article 2. At this moment, the instance of `article-two` involving Alice is automatically created and the event `+enabled(issue-nim(...))` relating to Alice is generated. The normative service agent includes the following abstract plan in its ASC2 script:

```
+enabled(issue-nim(N,C,Mortgage,Property)) :
    preregister(C) => #inform("Application",
        enabled(issue-nim(N,C,Mortage,Property))).
```

that informs the application service about this newly enabled act, which in turn can then automatically (or manually by a human operator) issue the NIM. This action is reported to the normative service as `+performed(issue-nim(...))` to update the institutional state of the normative service and possibly affect the normative positions of other citizens. Note that although the pre-registration service is not part of the regulations, its execution is based on the institutional state, i.e. on updates to normative positions. This example illustrates how a regulatory service can not only respond to normative queries, but also be programmed to proactively push normative notifications to the system.

5.3 Adapting to Changes

Norms are subject to change and it is pivotal for a regulatory service to take adaption to future changes into account. Consider the new situation in which the regulations around

national insurance on mortgages have been amended by the national government with an extra requirement that NIM insurance can only be issued on properties of which the owner is a registered occupant. If a citizen relocates, but still owns the property at their old address, they are obliged to cancel the NIM coverage of the mortgage on that property within a certain time window. The following specifications are added (at run-time) to represent the amendment:

```
Fact article-three
  Identified by citizen * property
  Holds When occupant(citizen, property)
            && mortgage(citizen, property)
Extend Act issue-nim
  Conditioned by article-three()
Duty duty-to-cancel-nim
  Holder citizen
  Claimant notary
  Related to mortgage, occupant
    When citizen == mortgage.citizen
         && citizen == occupant.citizen
  Violated When undue-cancel-nim()
  Holds When nim-covered(mortgage)
    && mortgage.property.address != occupant.property.address
```

With the new specification, new normative concepts and modified norms are introduced, and the normative service generates a new state in which the amendment is now also considered. The Act `issue-nim` now has not only the original two articles as its preconditions but also `article-three` by applying the keyword `Extend` in eFLINT. The previous state (although originating from a different specification) is then used to produce the new state reflecting the amended norms. If Alice registers at another address, the municipality (represented as City DB in Figure 2) will provide this information to the normative service with the following ASC2 plan:

```
+!register_occupant(Citizen,Property) =>
    #coms.achieve("NotaryService",perform_normative(
        register_occupant(Self,Citizen,Property),true)).
```

Reasoning from the new state and this information, a duty to cancel Alice's NIM coverage as `+duty(duty-to-cancel-nim(...))` is created. The normative service could then notify the notary application about this duty, and also start a timer to track the window in which this duty ought to be fulfilled. Then, the service will generate an `+undue-cancel-nim()` event when that time is over, possibly resulting in an automated `+violated(...)` event being raised in case the duty is not terminated in time by canceling the insurance. The following ASC2 plan notifies the enforcement service:

```
+violated(duty-to-cancel-nim(C,N,Mortgage,Occupant)) =>
    #inform("Enforcer",cancellation-violation(C,N,Mortgage)).
```

This example represents a simple case of laws changing and the regulatory service adapting to such changes. However, modification of laws is a complex matter; in Section 7 we provide a brief discussion of the adaptation/modification process.

5.4 Automatic Enforcement

The final part of our case study illustrates two concepts (1) distribution of control, and (2) ex-post enforcement. In our regulatory services architecture, there could be multiple normative service points, as illustrated in Figure 2, the enforcement service which is in charge of handling violations is a separate service that includes its own internal normative reasoner. Also, unlike the notary normative service that mainly implements ex-ante rules to govern other processes, the enforcement service takes action only after a violation has occurred, making it an ex-post enforcement service. The normative reasoner within the enforcement service is instantiated with the following (partial) specification to handle the cancellation of NIM:

```
Act enforce-cancel-nim
  Actor enforcement-service
  Recipient notary
  Related to citizen, mortgage
  Creates processing-fee()
  Holds When cancellation-violation()
```

When the enforcement service receives a NIM cancellation violation event, according to its institutional state, it will have the power to cancel the NIM (e.g., by instructing the application service), and to generate a processing fee to be billed to the client (e.g., again through the application service), as is implemented by the following ASC2 plan:

```
+enabled(enforce-cancel-nim(Self, C, N, Mortgage)) =>
    #achieve("Application",cancel-nim(N,C,Mortgage));
    #achieve("Application",issue-cancellation-bill(N,C,Mortgage)).
```

This example also shows another interaction between institutional notions (regulations) and extra-institutional notions (desires, goals). While the enforcement service has the power to enforce cancellation, it has no duty to do so. Instead, the behavioral specification determines that when given the power, the service desires to invoke cancellation.

6 Evaluation

This section gives a self-evaluation of the proposed solution based on a set of criteria/requirements that we consider important and applicable to this study. To determine the evaluation criteria for this work, we surveyed a number of papers (including [15,19,23,13,20]) and picked out the criteria that we consider relevant to our approach.

6.1 Evaluation Criteria

Compliance Strategy This criterion refers to various compliance checking strategies, where compliance is evaluated either during the initial stages of process design (i.e., design-time/pre-execution time), when processes are in operation (i.e., run-time/execution-time), or after the fact through audits that review process logs (i.e., auditing/post-execution) [20]. As argued before, we need a combination of these strategies to have full coverage and we will use this criterion as an initial classification to distinguish different approaches.

Norms Modeling The literature has emphasized the significance of norms modeling [15,19,20,23]. This modeling methodology extracts and represents requirements from legal texts and models them into a certain form of representation. The purpose of norms modeling is to ensure that the activities performed during the execution of a business process are aligned with these normative specifications and to facilitate automated verification of compliance.

Multiple Sources of Norms Organizations may need to comply with norms from a variety of sources in a single process. For example in a data-sharing scenario, the systems may need to adhere not only to GDPR but also to some internal policies and data-sharing agreements between other parties.

Adaptation to Changes of Norms Since both norms and business processes will change from time to time, regulatory services should take the evolution of norms into account and support mechanisms to reflect and adapt to these changes.

Human Intervention Software may perform actions that legal experts find incorrect or hard to comprehend. This lack of involvement by legal practitioners can make the resulting systems challenging for them to use as evidence in compliance disputes, and therefore, difficult to be accepted by users in the legal community. Despite the benefits of automation in terms of efficiency and convenience, it is crucial to permit human intervention when required.

Violation Handling and Reparation In Section 6.1, we mention that compliance can be assessed before, during, or after process execution. Similarly, violations can be handled by mechanisms that vary in multiple design dimensions [13,24].

Explanation As stated in previous subsections, modeling norms is a complicated task because the meaning of legal inputs may not be captured accurately enough, not to mention that those inputs could arise from various sources that may have different interpretations and may be updated occasionally. Hence, it is desirable for software to provide explanations for their behaviors and decisions (e.g., to explain what violation was detected and what caused the violation).

6.2 Evaluation Results

In the subsequent paragraphs, we will examine the extent to which our approach aligns and meets each criterion. A summarized version can be seen in Table 1.

Compliance Strategy The approach we present in this paper emphasizes the run-time aspects of compliance using queries to determine the compliance of actions before committing to them and notifications of violating actions to enable ex-post enforcement. We argue that our approach can also be combined with a design-time compliance strategy as well as with methods for reporting compliance as part of auditing. However, we argue that ex-post enforcement at run-time and as part of auditing processes are still required for several reasons. We view our approach as a hybrid approach that covers the full spectrum of compliance strategies. The offline compliance strategies are to be discussed in further detail in future work.

Table 1: Summary of the self-evaluation

Criterion	Summarized Evaluation Results
Compliance Strategy	Run-time strategy, but broadly speaking can be a hybrid approach
Norms Modeling	eFLINT supports the Hohfeldian concepts of powers and duties
Multiple Sources	Our implementation provides ways to support this feature, yet manually
Adaptation to Changes	Norms can be amended by adding new types/modifying existing ones
Human Intervention	Human involvement is allowed; the autonomy of human is guaranteed
Violation Handling	The architecture holds ex-ante and ex-post enforcement strategies
Explanation	Basic explanations are available

Norms Modeling We have used eFLINT to successfully express permission, duty, power, and prohibition in our case study, and check the correctness with the involved experts. More research is planned to evaluate the capability of this language in comparison with alternatives, e.g., by formally comparing the underlying logics.

Multiple Sources of Norms In Section 5.4, we demonstrate the possibility to distribute control by applying different sets of norms to different actors. Even though the two sets of norms reside in two different actors, the connection of these norms can be found in the related ASC2 plan. This ASC2 plan not only expresses the link between the two sets of norms but also how they interact (e.g., given a power when a duty is unfulfilled). In addition to ASC2 plans, eFLINT itself can also be used to express the cross-reference idea between different sets of norms. Multiple sources of norms can be integrated into systems following our architecture by instantiating multiple normative services and by interconnecting multiple normative sources before applying them.

Adaptation to Changes of Norms As exemplified in Section 5.3, we support making amendments to the interpretation of norms applied within our systems by submitting code extensions. The extensions can contain new types to add powers, duties, events and facts. Moreover, existing types can be extended to add pre-conditions, post-conditions, violation conditions and derivation rules.

Human Intervention In Figure 1, the *Users* actor indicates where human actors could be involved in a regulated system. According to the proposed architecture, human users can take part in both ex-ante and ex-post enforcement. For example, when an ex-ante check determines that a particular action within the application is considered non-compliant, the application can still give the user the freedom the execute the action (possibly subjected to a warning). For ex-post enforcement, an enforcement service can propagate the notification of a violation to a user to let them determine how to respond to the violation. Our architecture reflects that agents, whether they are software or human, possess their own set of beliefs and have the autonomy to make their own decisions.

Violation Handling and Reparation While most approaches adopt only one type of enforcement, we include both ex-ante and ex-post mechanisms. Having both of them involved increases the coverage in run-time compliance strategies. For ex-ante compliance, the architecture provides query-based permission checking and normative notifi-

cations to help prevent non-compliant behaviors; for ex-post enforcement, compliance is achieved through monitoring, automatic enforcement, and human intervention.

Explanation By making use of ASC2 and eFLINT, we can provide some explanation about the normative results generated by the reasoner. For example, when observing a violation, the normative service not only sends the name of the violation but also the pre-condition (can be interpreted as the causes of this violation) and post-condition (can be interpreted as the effects of this violation) to the receiver.

7 Discussion

We have presented a foundational architecture of regulated systems, a prototype instantiating the proposed architecture for a case study, and an evaluation examining the benefits and limitations of our approach. As a proof of concept, we conducted some case studies to demonstrate the different components and their interactions focusing on the different requirements of a regulatory service. Particularly noteworthy is how we can amend norms dynamically, for example by adding (pre-)conditions to an existing specification. In our implementation this is realized by adopting an extended set of rules on an existing knowledge base capturing the institutional state. The architectural model we propose is adaptable in that it can be implemented in a variety of ways, with our prototype giving one example. The model is flexible in that design choices could be made differently.

8 Conclusions

This paper presented a regulatory-compliant architecture for regulated systems to help automate compliance with (for example) regulations and policies through normative services and enforcement services. A novelty of our approach is that, besides ex-ante enforcement, we include ex-post enforcement mechanisms in our design for increased flexibility and adaptability, and to attempt to bridge the legal-technological gap. We demonstrated the advantage of our approach in a case study. In particular, we have shown the ability to amend norms and bring software systems closer to legal practice by introducing (human or automated) responses to violations in (ex-post) enforcement services. We also listed extensive relevant literature, hoping to draw a clearer picture of this broad topic. In future work we intend to demonstrate and evaluate the benefits of our approach further.

Acknowledgments This research is funded by the Dutch Organisation for Scientific Research (NWO) under contracts 628.009.014 (SSPDDP project) and 628.001.001 (DL4LD project) and the AMdEX Fieldlab project supported by Kansen Voor West EFRO (KVW00309) and the province of Noord-Holland.

Disclosure of Interests The authors have no competing interests to declare that are relevant to the contents of this article.

References

1. Aranda García, A., Cambronero, M.E., Colombo, C., Llana, L., Pace, G.J.: Runtime verification of contracts with themulus. In: de Boer, F., Cerone, A. (eds.) Software Engineering and Formal Methods. pp. 231–246. Springer International Publishing (2020)
2. van Binsbergen, L.T., Kebede, M.G., Baugh, J., van Engers, T., van Vuurden, D.G.: Dynamic generation of access control policies from social policies. Procedia Computer Science **198**, 140–147 (1 2022)
3. van Binsbergen, L.T., Liu, L.C., van Doesburg, R., van Engers, T.: eFLINT: A Domain-Specific Language for Executable Norm Specifications. In: Proceedings of the 19th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. p. 124–136. GPCE 2020, Association for Computing Machinery (2020)
4. Boella, G., Humphreys, L., Muthuri, R., Rossi, P., van der Torre, L.: A critical analysis of legal requirements engineering from the perspective of legal practice. In: 2014 IEEE 7th International Workshop on Requirements Engineering and Law (RELAW). pp. 14–21. IEEE Computer Society (2014)
5. Boella, G., van der Torre, L., Verhagen, H.: Introduction to normative multiagent systems. Computational and Mathematical Organization Theory **12**(2-3 SPEC. ISS.), 71–79 (2006)
6. Broersen, J., van der Torre, L.: Ten Problems of Deontic Logic and Normative Reasoning in Computer Science, pp. 55–88. Springer Berlin Heidelberg (2012)
7. Chiu, D.K., Cheung, S.C., Till, S.: A three-layer architecture for e-contract enforcement in an e-service environment. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences - Track 3 - Volume 3. p. 74.1. IEEE Computer Society (2003)
8. Criado, N., Argente, E., Noriega, P., Botti, V.: Towards a normative BDI architecture for norm compliance. CEUR Workshop Proceedings **627**, 65–81 (2010)
9. Criado, N., Argente, E., Noriega, P., Botti, V.: Manea: A distributed architecture for enforcing norms in open mas. Engineering Applications of Artificial Intelligence **26**(1), 76–95 (2013)
10. Dastani, M., Sardina, S., Yazdanpanah, V.: Norm enforcement as supervisory control. In: An, B., Bazzan, A., Leite, J., Villata, S., van der Torre, L. (eds.) PRIMA 2017: Principles and Practice of Multi-Agent Systems. pp. 330–348. Springer International Publishing (2017)
11. Dignum, F., Kinny, D., Sonenberg, L.: Motivational attitudes of agents: On desires, obligations, and norms. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **2296**(Section 2), 83 (2002)
12. El Gammal, A.: Towards a comprehensive framework for business process compliance. Ph.D. thesis, Tilburg University, School of Economics and Management (2012)
13. El Kharbili, M.: Business process regulatory compliance management solution frameworks: A comparative evaluation. In: Proceedings of the Eighth Asia-Pacific Conference on Conceptual Modelling - Volume 130. p. 23–32. APCCM '12, Australian Computer Society, Inc. (2012)
14. Elakehal, E.E., Montali, M., Padget, J.: Run-time verification of msmas norms using event calculus. In: Proceedings of the 2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems Workshops. p. 110–115. SASOW '14, IEEE Computer Society (2014)
15. Elgammal, A., Turetken, O., van den Heuvel, W.J., Papazoglou, M.: On the formal specification of regulatory compliance: a comparative analysis. In: Service-Oriented Computing: ICSOC 2010 International Workshops, PAASC, WESOA, SEE, and SOC-LOG, San Francisco, CA, USA, December 7-10, 2010, Revised Selected Papers 8. pp. 27–38. Springer (2011)
16. Ellul, J., Pace, G.J.: Runtime verification of ethereum smart contracts. In: 14th European Dependable Computing Conference (EDCC). pp. 158–163. IEEE Computer Society (2018)

17. Falcone, Y., Mariani, L., Rollet, A., Saha, S.: Runtime failure prevention and reaction. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification: Introductory and Advanced Topics, pp. 103–134. Springer International Publishing (2018)
18. Gibbs, J.P.: Norms: The problem of definition and classification. American Journal of Sociology **70**(5), 586–594 (1965)
19. Hashmi, M., Governatori, G.: Norms modeling constructs of business process compliance management frameworks: a conceptual evaluation. Artificial Intelligence and Law **26**(3), 251–305 (2018)
20. Hashmi, M., Governatori, G., Lam, H.P., Wynn, M.T.: Are we done with business process compliance: state of the art and challenges ahead. Knowledge and Information Systems **57**(1), 79–133 (2018)
21. Heutelbeck, D.: Demo: Attribute-Stream-Based Access Control (ASBAC) with the Streaming Attribute Policy Language (SAPL). In: Proceedings of the 26th ACM Symposium on Access Control Models and Technologies. p. 95–97. SACMAT '21, Association for Computing Machinery (2021)
22. Hohfeld, W.N.: Fundamental legal conceptions as applied in judicial reasoning: and other legal essays. Yale University Press (1923)
23. Lam, H.P., Hashmi, M.: A comparative study of compliance management frameworks: Penelope vs. pcl. Knowledge **2**(4), 618–651 (2022)
24. Liu, L.C., Sileno, G., Van Engers, T.: Digital Enforceable Contracts (DEC): Making Smart Contracts Smarter. In: Legal Knowledge and Information Systems, pp. 235–238. IOS Press (2020)
25. Luck, M., d'Inverno, M., et al.: A normative framework for agent-based systems. Computational & Mathematical Organization Theory **12**(2), 227–250 (2006)
26. Mohajeri Parizi, M., Sileno, G., van Engers, T.: Seamless Integration and Testing for MAS Engineering. In: Engineering Multi-Agent Systems. pp. 254–272. Springer International Publishing (2022)
27. Mohajeri Parizi, M., Sileno, G., van Engers, T., Klous, S.: Run, Agent, Run! Architecture and Benchmarking of Actor-Based Agents, p. 11–20. Association for Computing Machinery (2020)
28. Molina-Jimenez, C., Shrivastava, S., Strano, M.: A model for checking contractual compliance of business interactions. IEEE Transactions on Services Computing **5**(2), 276–289 (2012)
29. Pandžić, S., Broersen, J., Aarts, H.: Boid*: Autonomous goal deliberation through abduction. In: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems. p. 1019–1027. AAMAS '22, International Foundation for Autonomous Agents and Multiagent Systems (2022)
30. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: de Velde, W., Perram, J.W. (eds.) Agents Breaking Away. pp. 42–55. Springer Berlin Heidelberg (1996)
31. Rissanen, E.: eXtensible Access Control Markup Language (XACML) Version 3.0 (Jan 2013)
32. Salmond, J.W.: Jurisprudence: Or, The Theory of the Law. Stevens and Haynes (1902)
33. Sánchez, C., Schneider, G., Ahrendt, W., Bartocci, E., Bianculli, D., Colombo, C., Falcone, Y., Francalanza, A., Krstić, S., Lourenço, J.M., Nickovic, D., Pace, G.J., Rufino, J., Signoles, J., Traytel, D., Weiss, A.: A survey of challenges for runtime verification from advanced application domains (beyond software). Formal Methods in System Design **54**(3), 279–335 (2019)