Reflections on the design and application of eFLINT

L. THOMAS VAN BINSBERGEN, Informatics Institute, University of Amsterdam, The Netherlands

ACM Reference Format:

1 INTRODUCTION

Following advancements in autonomous and distributed computing, software systems are increasingly more integrated with social systems. Compliance with laws, regulations and (contractual) agreements regulating such systems is a top priority for many organisations and regulators, as is evidenced by the impact of the EU’s privacy regulations (GDPR) and the anticipated impact of the forthcoming regulations on the use of AI. In various project, the University of Amsterdam is experimenting with approaches to automate compliance in software systems through the integration of so-called regulatory services tasked with enforcing explicit, formal interpretations of relevant norms. This presentation discusses the design of eFLINT, a domain-specific language for formalising norms used within these projects.

Compared to existing languages, eFLINT is novel in several respects and is most similar to languages based on the event calculus such as Symboleo [13] and InstAL [10]. A significant body of work exists concerning the formalisation, analysis and enforcement of specific kinds of norms [7] such as policies for access control [14], network policies [1] (e.g. firewall configurations) and contracts [3, 12, 13]. Instead, eFLINT is designed for describing a wide variety of normative sources such as laws, regulations, policies and contracts. Other formal languages for expressing norms are based on deontic logics [5], action logic [8] and defeasible logic [4, 9]. An important aspect of eFLINT is that the language is action-based and supports the legal concept of power – the ability to change the normative positions of (other) actors. The benefit of the action-based approach is that checking the compliance of software systems is simplified because they are inherently action-based.

Together, these features enable eFLINT for various types of applications requiring online or offline compliance-checking, monitoring, traceability and explainability.

This presentation will discuss the connection between fundamental notions in computer science and the normative theory forming the basis of our approach. This analysis reveals interesting similarities between the processes of drafting regulations and software engineering. In particular, the importance of modularity, inheritance, versioning and specialisation are discussed. The presentation reflects on the first phases of eFLINT’s development, each widening the scope within which eFLINT is applicable, and lays out the plans for the next phase, in which usability is the primary concern.

2 REFLECTIONS

Figure 1 provides an example specification.

Phase 1: Legal and computational concepts. Inspired by earlier work on FLINT [18, 19], the first version of eFLINT (executable FLINT) connects the normative concepts of ‘power’ and ‘duty’, as described by Hohfeld [6], to the concepts of configuration and transitions over configurations in Plotkin-style transition systems [11]. As described in [16], an eFLINT program consists of a collection of type declarations (a specification) and a sequence of statements (a scenario). The type declarations determine the structure of the transition system, with every configuration a set of
A scenario describes a trace in the transition system, which may be action-compliant and/or duty-compliant depending on the preconditions and violation conditions of action-types and duty-types respectively. For a scenario to be action-compliant, every transition labelled with an instance of sign-certificate, for example, must have been performed by a doctor (see Listing 2). For a scenario to be duty-compliant, every instance of a duty in the knowledge base must be terminated before a configuration is reached in which the enforcing acts of that duty are enabled. Violation conditions can also be associated with duties directly. In the example, the existence of a birth certificate gives a mayor the power to place a duty on the birth-parent of the child. The duty is terminated by registering the child; if this is done by the second parent, this is considered as recognising the child with the person becoming a co-parent of the child. Undue delay in the registration gives the mayor the power to register the child without co-parent. Because the action observe-late-registry is listed as an enforcing act of duty-to-register, the duty is considered violated.

The tools supporting phase 1 of the development of eFLINT make it possible to automatically assess scenarios for compliance and to perform rudimentary simulations by iteratively stepping through the underlying transition system by choosing transitions to execute. These tools require the 'domain of discourse' to be bounded such that every type has a finite number of possible instances and every configuration a finite number of outgoing transitions. This is problematic for checking the compliance of running software as, for example, the set of users of particular application is not know in advance.
Phase 2: Dynamic specification and assessment. The second version of eFLINT can be used to check the compliance of running software. The semantics of the `Foreach` operator, (implicitly and explicitly) used to enumerate the instances of a type, was modified such that it enumerates all the instances of the type when possible or enumerates those instances that hold true in the existing knowledge base. This pragmatic design decision ensures that enumeration terminates in static scenarios (with a bounded domain of discourse) and dynamically produced scenarios.

The second version of eFLINT also has a more flexible syntax that enables type-declarations and statements to be mixed freely. As a result, both scenarios and specifications can be developed incrementally. This language extension was performed by applying the principled approach to REPL-style interpreters presented in [17]. The resulting interpreter can be embedded as a service in service-oriented software system – an important step towards realising the regulated systems mentioned in the introduction. At any time, the eFLINT services can be queries about active duties and permissions. Moreover, eFLINT services inform enforcement actors (human or otherwise) about any violations, potentially triggering these actors to invoke their powers to enforce norms.

Phase 3: Modularisation and specialisation. In the third phase, eFLINT was evaluated with respect to important software engineering principles such as reuse, separation of concerns, and modularity. In [15], a case study was performed within the health-care domain, resulting in additional extensions to the language. Within this case study, a consortium agreement between hospitals ‘imports’ concepts from the GDPR, the European Union’s privacy regulation [2]. The paper demonstrates how the connection between the two documents is formalised in eFLINT and how this is achieved with a GDPR specification that does not anticipate how it used within other specifications. In this version of eFLINT, alternative interpretations of norms and open terms such as ‘undue delay’ can be specified as replaceable parts (i.e. with versioning). Moreover, interpretations can be composed and can be reused across applications with different specialisations of certain concepts, e.g. a banking application considers clients as data subjects according to the GDPR, whereas a healthcare application considers patients as such.

Phase 4: Usability and application. This is how eFLINT exists today. The next phase is aimed at improving the usability of the language and at demonstrating its usage in pilots with industrial partners. The features introduced in the second and third phase made eFLINT very flexible. As a negative consequence, users can now easily ‘break’ a specification, e.g. by overriding type definitions so that other types are no longer well-defined. The aim is to build a higher-level language on top of the existing language. This language should have a basic module-system, additional static analyses for discovering inconsistencies and verifying properties, a clearer separation between normative and computational concepts, and a development and testing environment suitable for use by legal experts. Moreover, we wish to achieve interoperability with formalisms and tools designed for knowledge representation and knowledge derivation, such as those used within Semantic Web communities.

3 CONCLUSION

Experiments with the eFLINT language have demonstrated its potential to serve as a language with which to formalise interpretations of norms from a variety of sources and to use these interpretations to automatically assess the compliance of software statically (off-line) and dynamically (on-line). This presentation provides an overview of the development of the language, the experiments performed, and the lessons learnt. The next phase is to improve the usability of the language and to demonstrate its usage within regulated systems as part of significant pilot projects.
ACKNOWLEDGMENTS

This work is part of the EPI project supported by NWO in the Commit2Data – Data2Person program (628.011.028), the SSPDDP project supported by NWO in the Big Data: Real Time ICT for Logistics program (628.009.014) and the AmDEx Fieldlab project supported by Kansen Voor West EFRO (KVW00309) and the province of Noord-Holland.

REFERENCES


