Reflections on the design and application of eFLINT

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

ACM Reference Format:
L. Thomas van Binsbergen. 2022. Reflections on the design and application of eFLINT. In *Informal proceedings of Programming Languages and the Law 2022.* ACM, New York, NY, USA, 4 pages.

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

ProLaLa '22, January 16, 2022, Philadelphia, PA
2022.

```
Fact person     Identified by String
Fact doctor     Identified by person
Fact mayor      Identified by person
Fact registered Identified by person

Placeholder parent For person
Placeholder child  For person

Fact natural-parent-of
  Identified by parent * child
  Holds when birth-parent-of(parent,child)
           ,co-parent-of(parent,child)

Fact birth-parent-of
  Identified by parent * child
  Holds when birth-certificate()
Fact birth-certificate
  Identified by doctor * parent * child
Fact undue-delay
  Identified by birth-certificate

Fact co-parent-of
  Identified by parent * child
  Holds when child-recognized()
Fact child-recognized
  Identified by parent * child
```

Listing 1. Knowledge represention with fact-types.

```
Act sign-certificate
  Actor doctor Recipient parent Related to child
  Conditioned by doctor
  Creates birth-certificate()
  Holds when True

Act observe-birth
  Actor mayor Recipient parent Related to child
  Conditioned by mayor
  Creates duty-to-register()
  Holds when birth-certificate()

Duty duty-to-register
  Holder parent Claimant mayor Related to child
  Enforced by observe-late-registry

Act register
  Actor person Recipient child
  Conditioned by !registered(child)
  Creates registered(child)
         ,child-recognized(parent=person)
           When !birth-parent-of(parent=person)
  Holds when birth-certificate()

Act observe-late-registry
  Actor mayor Recipient parent Related to child
  Conditioned by mayor, undue-delay(birth-certificate())
  Creates registered(child)
  Holds when duty-to-register()
```

Listing 2. Act-type and duty-type declarations.

Fig. 1. Example inspired by the treatment of legal parenthood and child registration in The Netherlands.

facts (a knowledge base) and the transitions determined by the (postconditions of) the action-types. Listing 1 in Figure 1 shows fact-type definitions that establish atomic facts (e.g. `person`), predicates (e.g. `doctor`) and relations (e.g. `natural-parent-of`) of which instances are considered to 'hold true' if they are in the current configuration.

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 GPDR, 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

[1] E. S. Al-Shaer and H. H. Hamed. 2004. Modeling and Management of Firewall Policies. *IEEE Transactions on Network and Service Management* 1, 1 (2004), 2–10. https://doi.org/10.1109/TNSM.2004.4623689

[2] Council of the EU. 2016. General Data Protection Regulation. *Official Journal of the European Union* 59 (2016).

[3] G. Governatori, F. Idelberger, Z. Milosevic, R. Riveret, G. Sartor, and X. Xu. 2018. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law* 26, 4 (2018), 377–409. https://doi.org/10.1007/s10506-018-9223-3

[4] G. Governatori, M.J. Maher, G. Antoniou, and D. Billington. 2004. Argumentation Semantics for Defeasible Logic. *Journal of Logic and Computation* 14, 5 (10 2004), 675–702. https://doi.org/10.1093/logcom/14.5.675

[5] H. Herrestad. 1993. Norms and Formalization. In *Proceedings of the 3th International Conference on Artificial Intelligence and Law (ICAIL 1993)*. ACM, 175–184. https://doi.org/10.1145/112646.112667

[6] Wesley Newcomb Hohfeld. 1917. Fundamental legal conceptions as applied in judicial reasoning. *The Yale Law Journal* 26, 8 (1917), 710–770. https://doi.org/10.2307/786270

[7] A.A. Jabal, M. Davari, E. Bertino, C. Makaya, S. Calo, D. Verma, A. Russo, and C. Williams. 2019. Methods and Tools for Policy Analysis. *Comput. Surveys* 51, 6, Article 121 (2019). https://doi.org/10.1145/3295749

[8] A.J.I. Jones and M. Sergot. 1996. A Formal Characterisation of Institutionalised Power. *Logic Journal of the IGPL* 4, 3 (06 1996), 427–443. https://doi.org/10.1093/jigpal/4.3.427

[9] D. Nute. 2003. Defeasible Logic. In *Web Knowledge Management and Decision Support*, O. Bartenstein, U. Geske, M. Hannebauer, and O. Yoshie (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 151–169.

[10] J. Padget, E. Elakehal, T. Li, and M. De Vos. 2016. *InstAL: An Institutional Action Language*. Law, Governance and Technology Series, Vol. 30. Springer Verlag, 101.

[11] Gordon D. Plotkin. 2004. A Structural Approach to Operational Semantics. *Journal of Logic and Algebraic Programming* 60–61 (2004), 17–139. https://doi.org/10.1016/j.jlap.2004.05.001 Reprint of Technical Report FN-19, DAIMI, Aarhus University, 1981.

[12] P.L. Seijas, A. Nemish, D. Smith, and S. Thompson. 2020. Marlowe: implementing and analysing financial contracts on blockchain. In *Workshop on Trusted Smart Contracts (Financial Cryptography 2020)*.

[13] S. Sharifi, A. Parvizimosaed, D. Amyot, L. Logrippo, and J. Mylopoulos. 2020. Symboleo: Towards a Specification Language for Legal Contracts. In *28th IEEE Int. Requirements Engineering Conf. (RE 2020)*. IEEE.

[14] OASIS Standard. 2013. eXtensible Access Control Markup Language (XACML) Version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[15] L. Thomas van Binsbergen, Milen G. Kebede, Joshua Baugh, Tom van Engers, and Dannis G. van Vuurden. 2021. Dynamic generation of access control policies from social policies, In The proceedings of the 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare. *Procedia Computer Science*.

[16] L. Thomas Van Binsbergen, Lu-Chi Liu, Robert van Doesburg, and Tom van Engers. 2020. eFLINT: A Domain-Specific Language for Executable Norm Specifications. In *Proceedings of the 19th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE 2020)*. ACM, 124––136. https://doi.org/10.1145/3425898.3426958

[17] L. Thomas van Binsbergen, Mauricio Verano Merino, Pierre Jeanjean, Tijs van der Storm, Benoit Combemale, and Olivier Barais. 2020. *A Principled Approach to REPL Interpreters*. ACM, 84–100. https://doi.org/10.1145/3426428.3426917

[18] R. van Doesburg, T. van der Storm, and T. van Engers. 2016. CALCULEMUS: Towards a Formal Language for the Interpretation of Normative Systems. In *AI4J Workshop at ECAI 2016 (AI4J 2016)*. 73–77.

[19] R. van Doesburg and T. van Engers. 2019. The False, the Former, and the Parish Priest. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law (ICAIL 2019)*. ACM, 194–198. https://doi.org/10.1145/3322640.3326718