Logic-based Methods for Software Engineers and Business People

Joost Vennekens

KU Leuven, Dept. of Computer Science
Campus De Nayer, St-Katelijne-Waver, Belgium

Abstract. Both software engineers and business people tend to be reluctant to adopt logic-based methods. On the one hand, this may be due to unfamiliarity with "scary" logical syntax. To ameliorate this, we developed an API for a state-of-the-art logic system, using only standard Python syntax. On the other hand, logic-based methods might have more impact if they could be used directly by business people. The recent DMN standard that might help in this respect.

Logic-based methods have great potential to improve current software engineering practice. However, this potential is often not evident for people from industry. For instance, when demoing a state-of-the-art knowledge base system to programmers, we typically get comments stating that, e.g., they can program the same functionality with "a couple of for-loops". To properly appreciate the benefits of a logic-based approach (greater flexibility, modularity, reliability and maintainability), it is necessary to understand in detail how the representation works, at least in the context of a small example. Here, syntax often seems a bottleneck, with programmers getting hung up on details they dislike.

To reduce this effect, we developed an API [3] that allows the state-of-the-art IDP knowledge base system [1] to be used from the Python programming language. Crucially, this API allows logic formulas to be added to the knowledge base using *standard Python syntax.* For instance, suppose that a mapping from users to roles is given by the following Python data structure, consisting of a list of tuples, and that a list of access rights is given in a similar way:

$$InRole = [\ (\text{'John', 'Student'}), (\text{'Ann', 'Admin'})\ ] \tag{1}$$

$$Allowed = [\ (\text{'Student', 'PublicData'}), (\text{'Admin', 'Passwords'})\ ] \tag{2}$$

Suppose we now want to verify that certain users indeed have access to certain resources, e.g., $Access$ = [('Ann', 'Passwords')]. The following Python expression then checks that Ann is indeed allowed to access the password data:

```
all(any((u,r) in InRole and (r,res) in Allowed for r in Role) for (u,res) in Access)
```

Our API accepts the same Python expression, but allows it to be used in different ways: for instance, if the programmer does not assign a value to $InRole$ herself (i.e., she omits statement (1)), then the IDP system will itself compute an assignment of users to roles that ensures that all of the required accesses are possible (i.e., *Ann* will be placed in the role *Admin* to ensure that she can access the password data). In this way, the flexibility of a declarative approach can be experienced without any new syntax. We therefore believe that this API may be a useful tool to introduce programmers to the power of logic-based approaches.

In many contexts, however, not programmers need to be convinced, but business people. Despite the significant technical differences between traditional software engineering and knowledge-based methods, business users may simply see a choice between either paying a programmer to deliver a piece of software or paying a knowledge engineer to deliver a piece of software. Here, we believe that the unique selling proposition of knowledge-based methods is that they may allow to “eliminate the middle man”, by giving ownership of the domain knowledge back to the business instead of to an IT departement.

The recent Decision Model and Notation (DMN) standard[1] published by the Object Management Group has been developed specifically to allow domain experts without any background in logic or computer science to construct a formal model of a decision process. Using this notation, a business expert could define the decision logic for access control by the following three tables:

C	Input	Output
	User	Role
	“John”	“Student”
	“Ann”	“Admin”
	“Ann”	“Student”

C	Input	Output
	Role	Resource
	“Student”	“PublicData”
	“Admin”	“PublicData”
	“Admin”	“Passwords”

U	Input	Output
	Role	AccessGranted *default: false*
	Request	true

Here, the “C” in the top left of the first table represents the “Collect” hit policy, meaning that a user belongs to *all* of the roles that are mentioned for her in the table. The second table expresses that the user has access to all resources that correspond to one of the roles to which she belongs. Finally, the third table expresses that when a user requests a resource to which she has access, the access should be granted, and otherwise it should not.

The tabular notation is not only intuitive for business people, it also allows the completeness and correctness of the decision procedure to be easily checked. Currently, most reasoning systems for DMN have evolved from rule-based expert systems and they typically only allow forward propagation. However, since a DMN model expresses a purely declarative piece of knowledge, there is no reason why other inference tasks could not be applied to it, for instance, by translating such a model to input for the IDP system, as done (currently still manually) in [2]. We suspect that the DMN language might easily be extendable to allow more complex kinds of knowledge to be expressed, while retaining the ease-of-use for domain experts. In this way, more of the power of logic-based systems could be put directly at the finger tips of domain experts and business decision makens, eliminating the need for an intervening programmer or knowledge engineer.

[1] `https://www.omg.org/spec/DMN/`

References

1. M. Bruynooghe, H. Blockeel, B. Bogaerts, B. De Cat, S. De Pooter, J. Jansen, A. Labarre, J. Ramon, M. Denecker, and S. Verwer. Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with IDP3. *Theory and Practice of Logic Programming*, 15(6):783–817, 2015.
2. I. Dasseville, L. Janssens, G. Janssens, J. Vanthienen, and M. Denecker. Combining dmn and the knowledge base paradigm for flexible decision enactment. In *Supplementary Proceedings of the RuleML 2016 Challenge*. CEUR-WS. org, 2016.
3. J. Vennekens. Lowering the learning curve for declarative programming: A python API for the IDP system. pages 86–102, 2016.