Implementing rules without a rules engine

By Ed Mullen

Published on October 9, 2018

data access public benefits technical guides

Follow 18F

18F on GitHub

18F on Twitter

18F on LinkedIn

RSS feed

Many government programs establish *rules* that define the way the program will be implemented. This can include eligibility rules that are defined in regulations and policy, as is the case with many federal health and human services programs. Or it can include rules established to improve the quality of data, such as federal spending submitted to USAspending.gov under the DATA Act. In order to operate these programs, the rules are turned into business logic that drive technology systems used on a daily basis and automate much of the work.

Frequently when these systems are developed in government, there is the assumption that if you've got rules, you're going to need a *rules engine*. But this is ultimately a false assumption. **Business rules can be (and frequently are) implemented as just another module in the code, *without the use of a rules engine*.** This is how we've done it at 18F.

Business Rule Engines, or BREs, facilitate the process of writing, managing, and executing business rules. Often, they're also rolled up into tightly coupled products with other business intelligence features and offerings. These "solutions" can be much more complex than what's needed for the core task: implementing rules. And **adopting this complexity comes with a cost.** You limit the pool of people who can help maintain your rules over time, increase your dependency on a vendor, and incur proprietary licensing costs. (While we love open source software, even open source rules engines can be an overly complex solution to a problem more easily solved.) Even the core utility of writing and managing rules with a business rules engine can lead to the need for specialized expertise or roles to operate an extremely niche product.

But there is a simpler way to approach this challenge: **policy people and engineers can effectively work together to code the logic *without needing the mediating rules engine product at all*.**

Implementing rules with modern programming languages

Many modern programming languages emphasize the ability to express business logic elegantly, through straightforward abstractions, concise and meaningful APIs, and embedded, domain-specific languages. Modern programming languages have all the features, flexibility, and expressivity needed to implement a rules-based system, without the need for a specialized, proprietary product.

Two recent projects illustrate our approach. In both cases, we had to define rules so that multiple parties could submit data, have it run against the rules, and then receive the results. We could have used a rules engine for these scenarios. But in both cases, we decided instead to write our business rules using SQL.

We chose SQL to encode our rules because it's:

- Designed to operate over data
- Application-independent
- Very well understood by a wide variety of people (including non-developers)
- Fairly legible for non-experts
- A mature language and an ANSI standard

Here's how that choice played out on two distinct projects.

Data validation for the DATA Act

Under the DATA Act, a variety of federal agencies are required to submit detailed spending information to a central store for exposure to the public at www.usaspending.gov. We needed to collect and validate financial data submitted as simple CSVs. A number of validation rules are applied to incoming data submissions to guard the accuracy of the data.

The DATA Act team evaluated a number of rules engines and technologies to implement these validations, and settled on expressing each rule in SQL. The team chose SQL for its expressiveness, speed, familiarity of both the development and business teams, and for not restricting other development choices. The shared, well-understood language shaved off significant startup cost for everyone involved and will simplify future maintenance.

Evaluating eligibility via a central eligibility rules service

Our recent eligibility rules service project was specifically aimed at answering the question of whether it would be possible to build a single, central rules web service that states could use to help them determine eligibility for health or human services programs that are managed by federal agencies but administered by states. To help answer this question, we built a prototype eligibility web service for a sample federal program. The rules service receives anonymous application data, analyzes the data against the eligibility criteria, and sends back a response that expresses whether the applicant meets the criteria.

The eligibility rules service allows program staff to define a set of rules describing eligibility criteria (rules) for benefit programs, then apply those rules to applicant information to help determine eligibility. We needed a rules format that could express the logic of a wide variety of rules, yet that would also be easy for program owners to read. We chose to create a simple centralized web service that applies SQL rules to submitted data and returns eligibility results. By doing so, we've avoided licensing costs, dependencies on specialized BRE expertise, and created a more simplified product that can be easily managed and audited. At the same time, we've created a path forward for states to break their own rules engine dependency and reduce their technology management burden.

On both of these projects, building our rules in SQL was ultimately the preferred option because it's a well and broadly understood language, avoids procurements, and allows for greater openness. And through both of these projects, we learned that it's not only possible, but *actually preferable* to do so, even when working within the context of a complex government program.

The advantages of a lighter-weight approach

Unlike using a complex, off-the-shelf rules engine product, writing rules in a well-understood programming language (such as SQL, Python, or Ruby) has several distinct advantages.

It facilitates cross-functional collaboration on your team.

Rules engines make claims about the ability of non-developers such as program or policy staff to adjust rules without the need for a developer's help. In practice, however, a subject matter expert on the rules engine *product itself* is generally required, adding additional layers between the policy and the programmers.

Working in cross-functional teams (where product, design, engineering, policy, and business experts work closely together as a tight-knit team) is a critical contributor to the success of our work at 18F and on agile software projects in general. This principle extends to codifying rules in systems. When you include the policy and business people on the development team from the start, and they work collaboratively with the software engineers to program the rules, there is greater assurance that the rules are interpreted and implemented correctly while avoiding the overhead of a rules engine. It just takes some effort to build those bridges and get the conversations rolling.

It's supported by a robust ecosystem of talent.

Using a common, well-understood programming language such as SQL allows you to draw upon the resources and expertise of expansive communities built around these widely-used programming languages and opens up access to a much larger ecosystem of support.

Any challenges you face with one of these languages has likely been faced and addressed many times over by the community. These languages are well documented, with a community of expert knowledge online. The costs and effort of hiring vendors or individuals are more reasonable, as you are not competing for scarce expertise. By contrast, when you're dealing with a smaller, proprietary system, you're left with a smaller, potentially expensive pool of experts to draw upon.

It fosters openness and accountability.

When you use common programming languages to describe the rules in code, you open yourself to the benefits of open source software: improved quality, security, reusability, and trustworthiness. The code can readily be shared through a public code repository, as you can see with the DATA Act rules on GitHub. Federal oversight bodies as well as the public can review your code and see exactly how the rules are being implemented, creating confidence that the policies set by the federal and state agencies are being accurately applied. Non-governmental partners can view the rules as implemented and integrate public APIs or reuse your rulesets to aid their clients in determining their best options. Government partners can repurpose the rules without the need to procure the same rules engine. Legislators could use the code to run tests against test data to investigate impacts of potential legislative changes.

Take the easier approach

If you're building a rules-based system, don't assume that you need a separate business rules engine product. Rules can be implemented more easily and with less overhead by cross-functional teams working to describe the rules and policy directly in code using general purpose programming languages like Python, Ruby, etc. This approach opens you to well-supported communities that can more readily be accessed to support the maintenance of the rules and systems over time. It also creates opportunities for openness that can increase reuse, collaboration, trust, and integrity.

18F is always available to help work through questions like this. Feel free to reach out to our team at inquiries18F@gsa.gov to set up a time to talk.

This post is largely a distillation of ideas from a number of 18F alumni, including Tony Garvan, Catherine Devlin, Becky Sweger, and benefitted from the guidance of CM Lubinski, Greg Walker, Ryan Hofschneider and Vraj Mohan.

‹ Previous post

Am I doing this right?: Antipatterns in agile contracting

Next post ›

Exploring a new way to make eligibility rules easier to implement

Modular contracting and working in the open

Working in the open is a key component of building trust between governments and vendor partners. Read about how the State of Alaska is using openness and code sharing to foster greater trust between government project teams and vendor teams as part of a large legacy system overhaul.

Exploring a new way to make eligibility rules easier to implement

When federal agencies issue a policy change, say income eligibility guidelines, that policy gets communicated down to the states as text on the Federal Register or via PDF. This translation of federal policy into many state systems creates opportunities for implementation errors.

The 18F Public Benefits Portfolio reflects on the last year

Pairing our deepening domain knowledge of the unique nuances of benefits administration and delivery across programs and levels of government with our core expertise in modern technology and digital service delivery, 18F's Public Benefits Portfolio team helped empower our partners to take some important leaps forward to rise to the critical challenges of the current moment, and we're thrilled to highlight some of their achievements from this past year.