Lineate

Automating Decision Making for a Leading FinTech Company

Empowering credit analysts at an online lending company by giving them control over automated decision engines

Contents

Lineate

Intro: Keeping a dynamic decision engine up to date

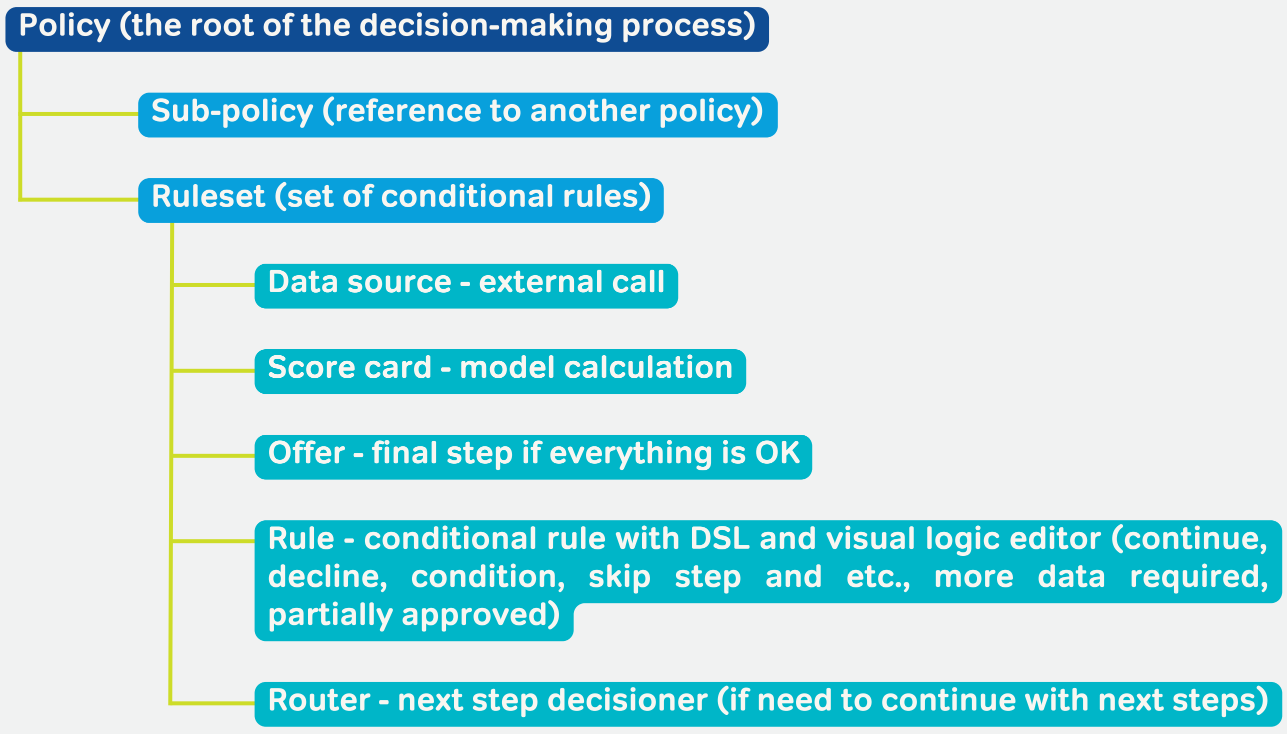
Our client, a leading FinTech company specializing in online lending, employs credit analysts who need to make rapid and even same-day decisions to respond to their credit inquiries. They do this using sophisticated automated processes to evaluate credit risk and score applicants. The company's decision engine takes into account hundreds of different factors, such as credit scores, external documents, and reports, and applies scoring models using its own internal proprietary processes. The client's decision logic is complex but needs to be transparent, testable, correct, and smart enough to allow for large numbers of edge cases.

The company constantly evaluates loan performance and tunes its decision engine to reflect varying circumstances. This is a costly and time-consuming process, requiring specialized technical expertise to effect changes. We knew there had to be a better way, one that empowered credit analysts to improve and iterate the credit scoring process themselves. Our challenge was to build a system to meet these needs.

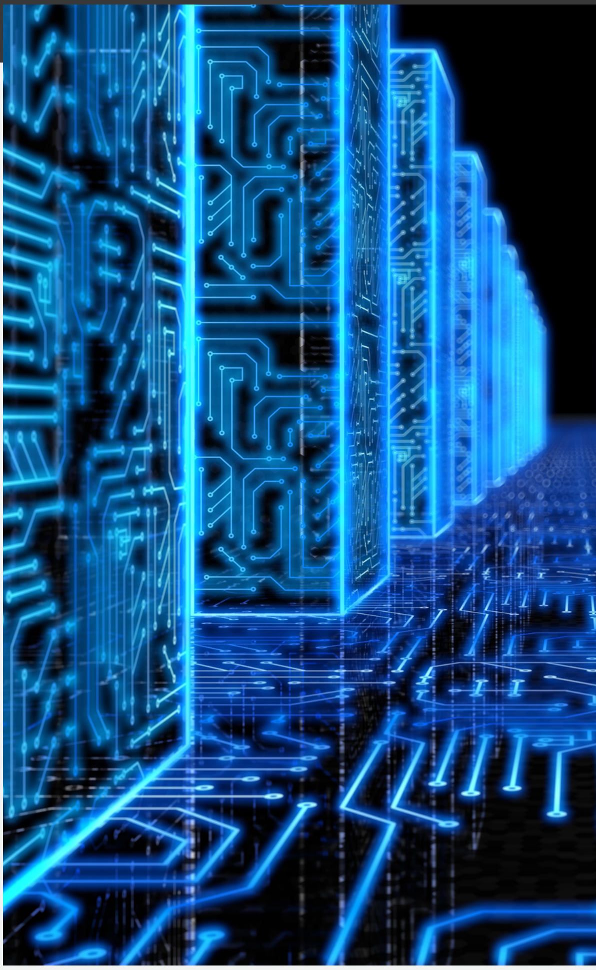
Using configuration, not code

Although our client’s credit scoring system was composed of what are essentially software applications, the logic behind it needed to be transparent to external experts in order to validate and test. Moreover, the logic will need to change often because of new discoveries and changing business conditions. For that reason, we extracted the specific rulesets from the applications into configuration files. This allowed for different logic to be backtested by swapping config files without rebuilding and redeploying the application, and for changes to be clearly versioned and auditable.

In modeling out the credit decision process, we settled on several core concepts for the decision engine:


Policy (the root of the decision-making process)
Sub-policy (reference to another policy)
Ruleset (set of conditional rules)
Data source - external call
Score card - model calculation
Offer - final step if everything is OK
Rule - conditional rule with DSL and visual logic editor (continue, decline, condition, skip step and etc., more data required, partially approved)
Router - next step decisioner (if need to continue with next steps)

Supporting partner banks and handling data integration complexity



We built a decision engine with rapid decision capabilities for processing loan applications. These capabilities were very attractive to the clients' partner banks, which were interested in leveraging this platform. As dozens of large banks began to integrate, more sophisticated challenges in maintaining the automated rule sets came into play. The core decision-making logic was similar across partner banks, but not the same. Each partner may have had different third-party data sources or differences in scoring formulas, or even different branches in the decision-making tree.

The business logic around these decisions started to grow quite complex, and it was being extended in unpredictable directions that were difficult to encode in configuration files. Our challenge was to create a configuration system flexible enough to model arbitrarily complex logic but clear enough for credit analysts to understand and apply the models without bringing in a programmer as an intermediary.

We eventually met all those challenges for our client. The first evolutionary step was to replace JSON config files with something that could operate in different ways on the data structures familiar to the credit analysts. We considered using one of the open source business rules management systems available on the market, such as Drools, because we had used this type of system before. However, this case had many business levels, in addition to significant security and UX requirements. These conditions, in combination with the inherited business logic implementations led us to build a new decision engine and management system.

We used the JMESPath query language as a base to design a simple but flexible domain-specific language. This language allowed us to make decision-making rules readable by non-technical people, yet flexible enough to support arbitrary changes in many dimensions. A further iteration could be to move this language to Kotlin, which provides additional perks such as syntax validation, compilation, and unit testing, and allows the development of unique syntax that can call special functionality.

Visualizing the decision engine workflows

With the client's decision engine complete, data integration began. But as dozens of integrations started to take shape, it became too difficult to manage the configurations, even with the domain-specific language we had designed. There was neither syntax validation nor real-time error checks, and it was very easy to make a mistake — we figured this out only after deploying the config and having to debug through log files. At this point, we decided to build an integrated development environment (IDE) for credit analysts.

Initially, we built a user interface to manage the rule sets and visualize the decision-making tree, along with the ability to use a visual editor to drag and drop logic steps and branch logic. The idea of rules visualization was so successful that the system grew into a full-fledged rules management system.

With the full user interface that visualizes and manages business logic, you can quickly test logic against test data. And with the UI you can create many policy parts using visual constructor components, or you can clone and reuse certain policy steps. Even the user interface itself is configurable, presenting different capabilities to different banks.
In short, the domain-specific language was our solution to dealing with the increasing complexity of rules, and the visualization and management tools we built upon it handle larger numbers of bank data integrations and enable business experts to participate in review and testing.

Following are some, but not all, features of the system were:

Visual editor that shows a tree representation of the decision-making process and allows you to create new steps with rules or conditional steps

Visual editor with on-the-fly rules and logic validation, so any syntax or logical error is immediately highlighted for your

Policies, which are the root of any decision-making process. Inside a policy you can add a rule set, rule, logic branching, third party system call, or even a task for a human.

- If during the loan estimation process a condition or step requires a human analyst to review, approve, or allow something, the system will automatically create a task for the analyst. The human analyst input is then automatically used in the next decision-making steps.

- You can refer another policy inside a policy, so you can implement more complicated decision making with “reusable” decision-making trees. This allows you to stack decision-making policies like Legos.

- All policies are versioned and “draft” mode is supported. This enables you to deploy, test, or try multiple scoring and decision-making models in production. You can also promote a draft version to production when it passes all tests and user acceptance testing.

- Policies are exportable to a JSON format, so you can export and import a policy, for example from staging to dev for testing. Exporting allows you to quickly clone the whole decision-making process. The average exported policy file consists of about 4,500 lines of code, or a few hundred decision-making logic steps. You can imagine how complicated it would be to manage such policies across multiple partners as textual configuration files or application source code.

Visual logic representation, which is the coolest feature from our perspective.

- The visualization allows credit analysts to not only edit but also review loan estimation results.

- The decision engine generates a final decision, but the visualization allows analysts to quickly see why such a decision was made, at what step the application was declined, or what documents are missing, why the loan amount is so small, and so on.

- The visual logic representation feature tremendously helps not only in debugging production data, but also in testing decision-making logic by nontechnical people.

- This is not a full list of system features, but it shows the scale of the system and the evolution of decision-making logic from simple config files to complex rules management UI.

Conclusion

At Lineate, we always emphasize building simple MVPs so we can quickly bring our clients' products to market. Accomplishing this task can be made simply by building rules logic directly into code. Building complex configurations and domain-specific languages can be complicated, and the resulting system requires users to face a learning curve before they can start using the system efficiently. But for critical FinTech functions, building out a full domain language can make sense sense because of compliance and auditing needs, as well as the fact that business expertise is spread across many people and the majority of them are nontechnical.

We discovered the limits of that approach, however, when we released this flexible platform to be the back end for several of the clients' large partner banks.. The increasingly complex automated rules required engineers to make more and more changes to the configurations, which meant that many of the advantages of the domain-specific languages were not being realized. Engineers were not only maintaining the rule sets, they were also tasked with internally spreading knowledge about how things worked.

In the end, we built a highly maintainable solution for our FinTech client:

a fully managed decision engine and a process visualization system that enables credit analysts to understand the system "at a glance" and customize rule sets to support dozens of partner banks. A single user experience supported all partners and eliminated the need to migrate thousands of rules for each partner.

Thank you.

Can we help you with your ambitious goals?

Talk to us today at
lineate.com/contact

Lineate