White Paper

Decision engines. Powering financial services.

Utsav Kapoor
Vice President, Cloud and API Platforms
United States Information Solutions

September 2021

Contents

Businesses are constantly seeking ways to acquire new customers, as this metric is a leading indicator of business growth. Driving higher valuations in the pre-pre-IPO stage and top-line growth in public companies is an aspirational goal of all successful businesses.

As financial institutions evaluate ways to find new customers or retain existing, fierce competition is breaking out in the marketplace. Companies must think outside the box about how to target and acquire customers while managing risks and maintaining healthy margins. It's a fine balancing act leading more companies to look at deploying decision engines to streamline and automate their customer acquisition, retention, and risk management processes.

Whether it's a bank striving to acquire new credit card customers or an online mortgage lender finding new home buyers, the user experience is visibly different in cases where decision engines are used in place of traditional manual processes.

What is a decision engine?

A decision engine is a technology platform that automates the execution of well-defined business rules to serve a specific use case or set of use cases using a standard input and output payload. Decision engines can be real-time transactional or batch-based systems.

There are various use cases, ranging from simple to complex. See a very simplistic example in Diagram 1, below.

Diagram 1: The most basic form of a decision engine implementation

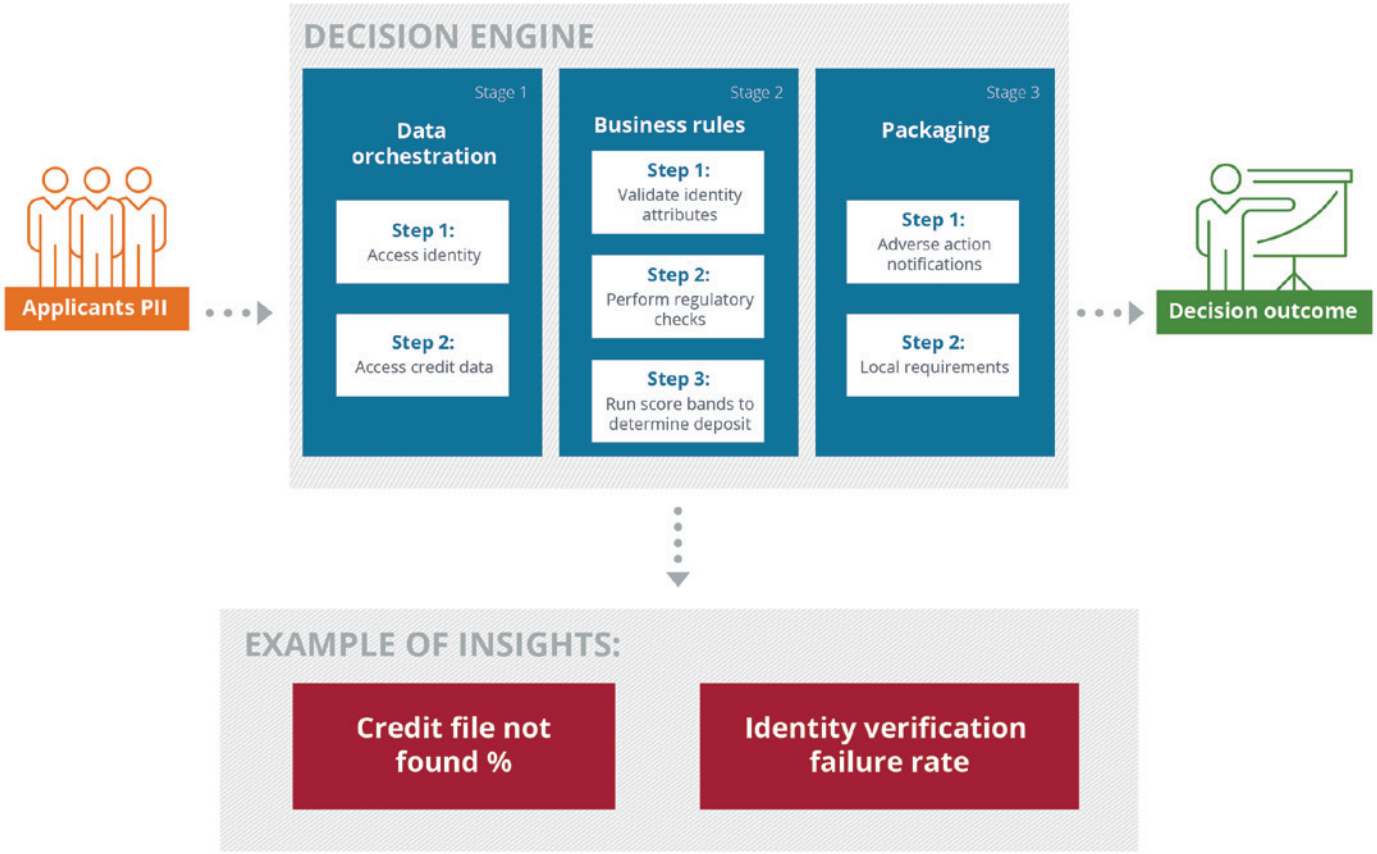
Over the years, decision engine implementations have evolved to varying degrees. Companies are finding creative ways to automate processes like data orchestration, risk analytics, and customer acquisition. Demand for more sophisticated features is on the rise, and decision engine providers are constantly evolving and innovating platforms — sometimes going past the tipping point.

This white paper provides a deep dive into the architecture and usage of decision engines, with best practices from successful deployments at financial services companies.

Use Case 1: Using a decision engine for new consumer identity and credit check

A regional utility company using a decision engine to verify identity and perform a credit check on a new consumer's application for connection.

Diagram 2



In this example, the decision engine serves multiple purposes, all tied to a common use case for the utility company.

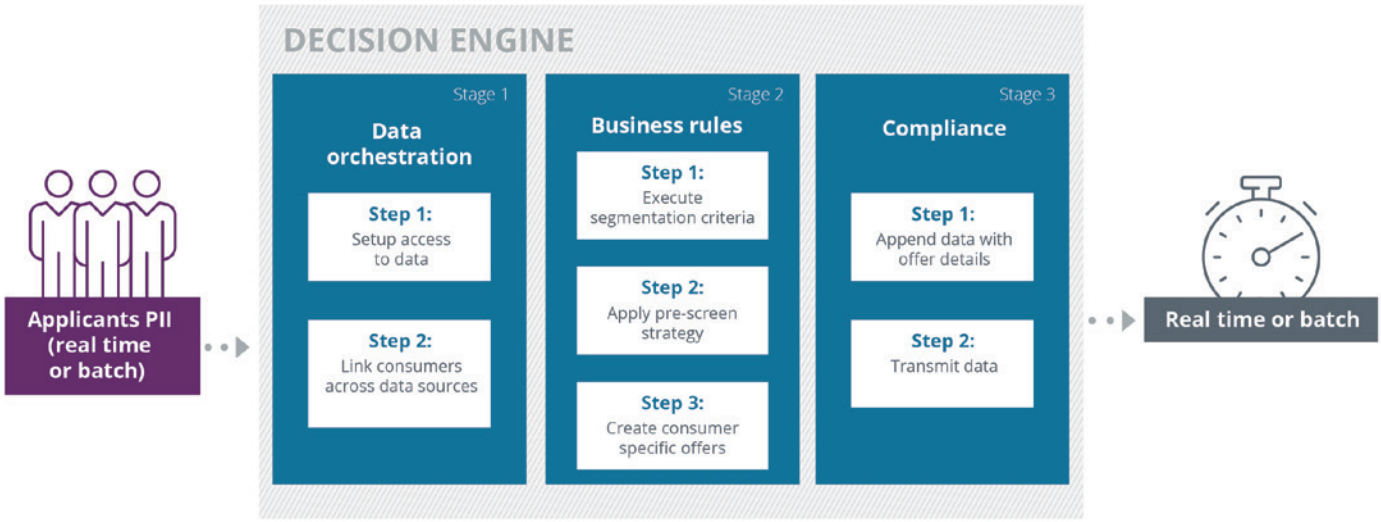
Purpose 1: Orchestrate data access from identity verification providers and credit bureaus
Purpose 2: Execute business rules to verify consumers identities or request more data
Purpose 3: Perform credit risk assessment to determine if a deposit is required
Purpose 4: Meet regulatory requirements like FCRA and adverse action notifications
Purpose 5: Give insights to the utility company on things like fraud rates and credit invisibles

The utility company streamlines the application process and can use the new insights to adjust its fraud or credit risk strategy and explore alternate data assets to improve approval rates.

Use Case 2: Using a decision engine for building a consumer base for a marketing campaign

A credit union is using a core processor software to run its banking operations. The software can integrate with a decision engine, which can help the FI create marketing campaigns, target new customers, and prescreen them in advance to speed up the offer acceptance process.

Diagram 3



Here, the decision engine is attempting to solve two distinct use cases: marketing and risk management. Without a decision engine, the credit union would have resorted to traditional methods of receiving batch files from the data providers, and then running direct mail campaigns. Instead, the decision engine automates several purposes.

Purpose 1: Setup access to credit and demographic data for target marketing
Purpose 2: Link entities across disparate data sources
Purpose 3: Apply predefined segmentation criteria to narrow down the target pool
Purpose 4: Run business rules on target pool to assess risk
Purpose 5: Create consumer-specific offers based on predefined criteria
Purpose 6: Append consumer data and attributes with offers to pass back to the credit union
Purpose 7: Transfer large quantities of data between organizations

Decision engines can optimize and automate workloads across different use cases that are interdependent. One of the unstated benefits of using a decision engine is handling Class 5 data (PII). The customer doesn't have to worry about data privacy and protection laws involved in this process, as the decision engine takes care of those requirements — and the PII sent to the customer is encrypted.

We can now understand how the scope of decision engines is expanding. While the core functions are still in play, data transmission techniques including encryption and packaging could be considered ancillary features.

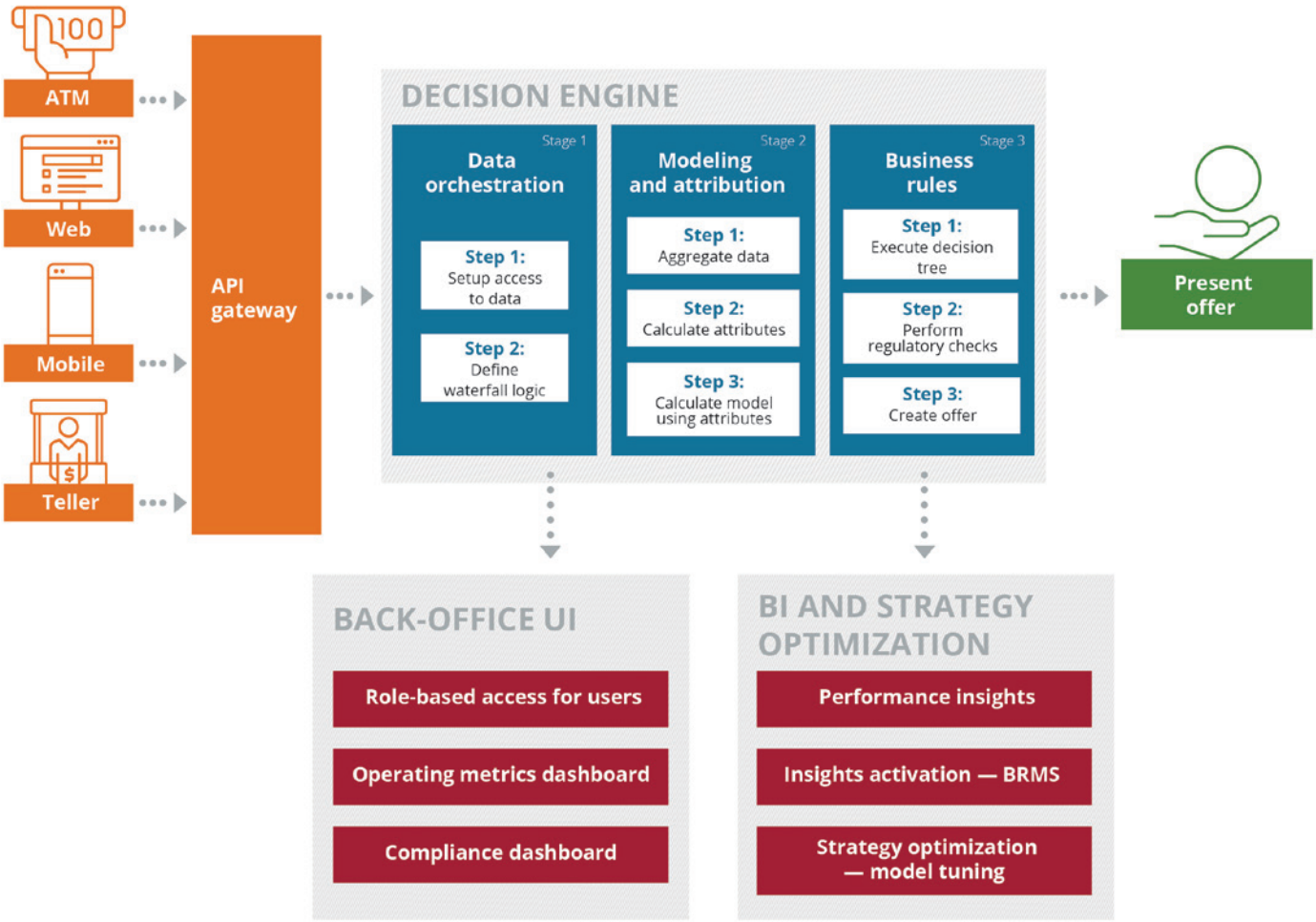
Use Case 3: Using A decision engine for prescreen offers and back-office BI capabilities

Decision engines can involve much more complex deployments. A large bank wants to integrate its digital channels with a decision engine in the background to generate real-time offers for its customers. Since some products involve extending a large line of credit, the bank wants to use a more sophisticated risk assessment strategy.

The bank wants support for all its digital channels, including POS (point-of-sale) terminals at a branch. The bank's risk officer wants to use custom attribution and modeling in their risk assessment and wants BI capabilities to tune their strategy over time. The operations team wants reporting and UI capabilities to meet compliance requirements.

The decision engine is becoming more of an enterprise platform.

Diagram 4



Since different channels adopt different intake fields, a layer of abstraction is required, as the input to the decision engine must be standard. For example, on digital channels, all PII fields may be requested, but for ATM transactions, the bank may have to accept only the debit card number and internally look up the customer's PII to pass on to the decision engine.

An integral part of a mature decision engine is a Business Rules Management System (BRMS), which allows business users to not only define the business rules that support their risk strategy, but also perform data attribution and model implementation. These are advanced capabilities that often get highlighted as the differentiating factors when comparing decision engines.

Purpose 1: Set up access to data sources with built-in waterfall logic
Purpose 2: Aggregate raw data for attribution
Purpose 3: Provide the ability to create attributes on raw or aggregated data
Purpose 4: Use attributes in model implementation and score calculation
Purpose 5: Execute business logic as decision trees
Purpose 6: Execute regulatory checks to meet compliance requirements
Purpose 7: Create real-time offers for customers and pass back via the channel of origination
Purpose 8: Provide operations reporting and support for compliance requirements
Purpose 9: Activate insights activation using transactional data
Purpose 10: Optimize strategies to improve take rate or approval rates

Some features in this example can be common causes for the downfall of a decision engine and should be evaluated carefully.

Ingress — support for multiple channels and varying payloads

Decision engines require a standard set of input parameters for optimal performance. When there are variations in input, additional work is required to accommodate those variations and not result in unhandled exceptions. For example, a digital channel captures full PII including SSN, while mobile may only capture the last four digits of SSN and a ZIP code.

The abstraction layer needs to add additional data to complete the partial input from the mobile channel before making a callout to the decision engine. Another option is to allow the decision engine to accept partial input but add supplemental data before making a callout to data sources.

If left unaddressed, this could lead to errors from the data source and unhandled exceptions in the decision engine.

Modeling and attribution — IDE for authoring attributes and models and performing audits and model validations

Most large FIs expect the decision engine to have a business user-friendly Integrated Development Environment (IDE), where policy managers can author new attributes and build new models, and audit and validate their work before deploying to production. Since modeling and attribution is an analytical exercise, the IDE should support various modeling techniques and compatibility with other modeling tools like SAS, R, etc.

Insufficient mature analytical tools will likely lead to a poor user experience and could potentially introduce erroneous code into a live environment, causing significant damage.

Product managers should take this into account. In many cases, decision engine providers opt to partner with third-party vendors with niche products or tools that play in this space. Building these integrations allows companies to put their best approach forward by offering combined capabilities.

Back office UI — user interface to support back-office operations

Many decision engines provide back-office user interfaces to allow customers to generate operational reports, view metrics, or research specific transactions or the underlying data. While this is not uncommon, decision engine providers must carefully evaluate how the customer intends to use these capabilities before making them available.

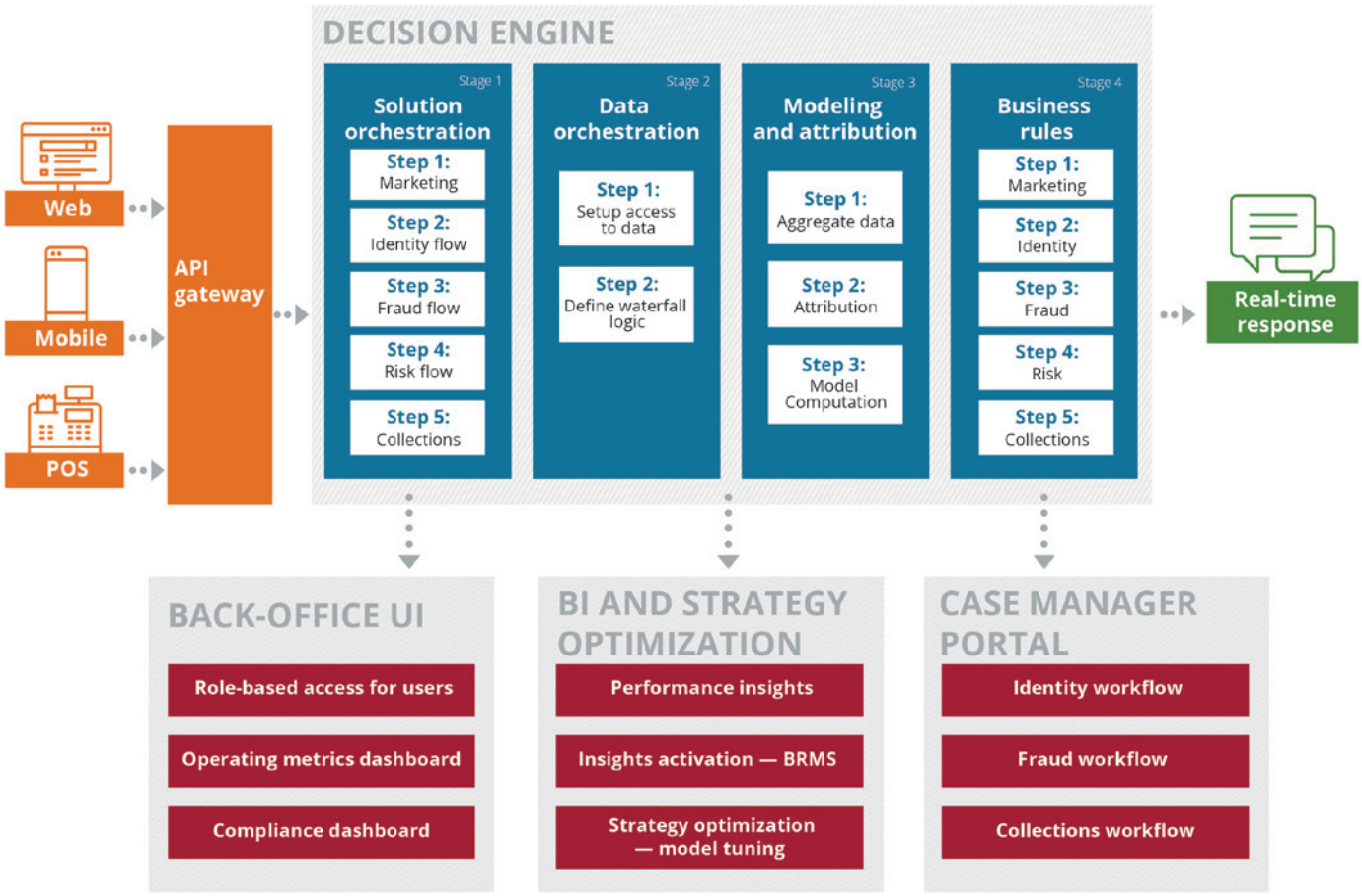
In many cases, the customers' operations team could be a contact center. The user interface may not scale, heavy user activity may slow incoming transactions, or you may find yourself customizing the UI based on individual customer requirements. Several scenarios play out here, and while modern cloud-based technologies address some of these concerns, upfront consulting and end-user training are essential to communicate the value of these capabilities and their proper use.

Use Case 4: Using a decision engine for multiple processing during wireless customer onboarding

Decision engines can also be an end-to-end transactional solution. One of the largest telecommunication companies in the country wants to use a decision engine to target new customers, verify their identity, mitigate fraud, and manage risk and collections.

Decision engine implementation is highly complex, spanning multiple interrelated use cases. While many would argue customers should use RESTful APIs to solve individual use cases, often customers end up choosing a decision engine as it optimizes the end-to-end process and minimizes upfront investment by reducing the number of integration points.

Diagram 5



The scope of the decision engine expands in this example. While the it serves its purpose, these types of implementations often require ongoing tuning and management, so organizations should consider the ongoing operating costs.

Core components of a decision engine

Decision engines have several essential components that give the provider a competitive edge and a better shot at success.

Business Rules Management System (BRMS)

At a minimum, the decision engine needs a Business Rules Management System (BRMS). This is where a business user can implement rules for the use case that the decision engine solves. This is also where many BRMS offer users the ability to build process flows and plug in business rules at logical points within those process flows.

A business rule is a set of conditions, which, when met, trigger an action — a simple "if-then-else" format. Business rules in mature BRMS can offer additional capabilities, like allowing to define some pre-conditions that must be met for the business rule set to trigger, or a set of post-actions that are triggered after successful execution of the business rules. A group of logically aligned rules can be grouped into what a BRMS refers to as a rule set.

A good BRMS offers the ability to use predefined models from a library of rule definitions and functions to create new rules and process flows. A BRMS should also test standalone rules and sub-processes. Finally, a good BRMS should deploy rules in a test or live environment. **Change management associated with a BRMS plays an integral part in determining the user experience.**

BRMS often simulates Champion/Challenger scenarios (also known as A/B testing). This is a useful feature that allows **customers to implement challenger strategies and test the performance** of these strategies against existing strategies. Companies use this to optimize what's running in production by making incremental tweaks and throttling a fraction of the volume through the changes to evaluate performance. This has proven to be an extremely successful capability in BRMS.

Decision engine providers are starting to use AI and ML (machine learning) to train the BRMS to make optimizations and drive efficiency and improve performance.

Ingress — Egress

Decision engines have Ingress and Egress payloads. Like any process, a decision engine requires a degree of standardization in the input parameters and output parameters. Even in environments involving big data, some data normalization is required, and a schematic must be developed to use a decision engine.

Why do the payloads require standardization? A **decision engine works best if the underlying rules are well-defined.** To do this, the BRMS needs a business object model that dictates the rule definitions, conditions, and actions. As one would imagine, when building a business object model that the business rules can use, the parameters need to be well-defined. Let's use an analogy here to drive the point home. Content written using words that are not defined in any dictionary would only qualify as gibberish. For the same reasons, business rules written using definitions and functions not defined in a business object would mean nothing. Worse, they would likely introduce run-time errors in the system.

Diagram 6: A sample process flow diagram in a BRMS implementation

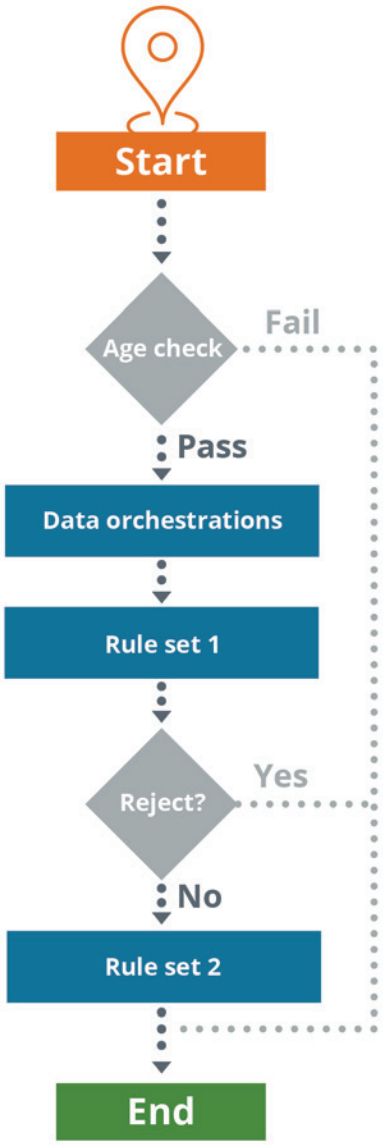
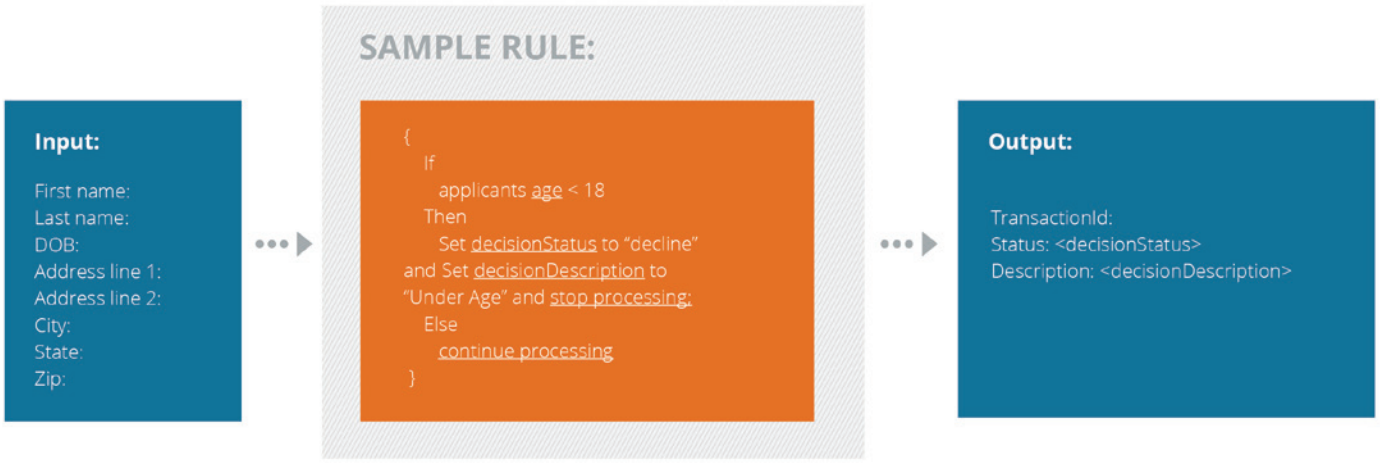


Diagram 7: Sample Ingress / Egress payloads and how they're used in business rules



Modern decision engines provide RESTful API with JSON payloads. JSON is a more human-readable format that makes interpretation of the Ingress/Egress payload easier. **Using JSON payloads and RESTful API allows browser compatibility and ease of integration with digital channels.**

While standardizing the payload is essential for a decision engine to function optimally, more and more data processed via decision engines is becoming unstructured. Additionally, customers often want to pass internal data to a decision engine and use it in the business rules. Decision engines must have the ability to pass additional data in the input and pass additional fields in the output.

A poorly designed Ingress framework could lead to the failure of the decision engine, so this aspect needs to be carefully architected and implemented.

Differentiating components of a decision engine

Core components aren't the only important factor of a decision engine. Most customers are looking for more than just an API-enabled BRMS system. Decision engines can have many components, depending on the provider.

Solution orchestrator

Decision engine power users often use the same implementation to accomplish multiple use cases. To put it simply, customers use the same instance of the decision engine, with the same input and output payloads, but tweak the orchestration code to execute varying business rules.

The decision engine architecture should support a solution configuration tool, or a solution orchestrator. **This tool can be an internal fulfillment tool or an external self-service tool** (depending on the type of decision engine). A solution orchestrator can configure different use cases tied to the solution by assigning a unique orchestration code. When triggered, the decision engine knows which orchestration to execute and what rules to implement.

This capability allows customers to orchestrate multiple purposes, optimizing their investment in a decision engine and driving up efficiency.

Data orchestrator

In the financial services industry, **companies often pull different data sources and**

execute business rules to accomplish end-to-end use cases. While customers could choose to feed raw data from different data sources to the decision engine, this design is clunky and most decision engines will not have native support. Today, most data sources are available via RESTful API. **Customers prefer that the decision engine directly access the API corresponding to the data source** and pull in the necessary data to apply the business rules.

Not only is **this design pattern more optimal and scalable,** but it also reduces the size of the payload, improving system performance. Additionally, this architectural pattern results in most decision engines needing a data orchestration layer, where different data sources can be configured for access.

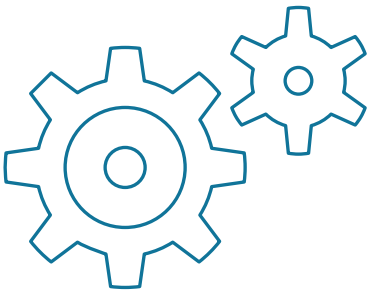
A data orchestrator not only provides the ability to source and enable access to data sources but also allows users to sequence data sources or define waterfall criteria.

Modeling engine

One of the more sophisticated features of a decision engine is the ability to implement analytical models. **Using score models allows companies to adopt a statistical approach to solving business problems.**

Model engines must provide the ability to ingest source code of models developed in various languages and provide hosting and run time capabilities. A modeling engine also needs to enable customers to code new models and scorecards. Developing a modeling engine that provides statisticians with the ability to develop new models and perform model validations is a much more sophisticated task that needs to be carefully evaluated. There are many industry-leading model development tools, like SAS, that are more mature and better suited to perform the task.

A poorly designed modeling engine could quickly become one of the failure points of a decision engine, as it may end up providing a poor customer experience. Any errors in the underlying libraries could introduce errors in a live environment, costing decision engine providers big fines and/or losses.



A poorly designed modeling engine could quickly become one of the **failure points** of a decision engine.

Attribution engine

Most decision engines consume vast amounts of data to make accurate decisions. **The raw data being fed to the decision engine from the source often needs to be aggregated to perform modeling and analytics.** For example, a business rule calculating the debt-to-income ratio needs to be fed the sum of total debt and the sum of total income as represented across every consumer account. This concept of summing debt and income is accomplished via an attribution engine.

Attribution engines provide a library of functions that allows users to author new attributes. The engine has a pre-baked business object model, which represents the data that the engine can consume. Users can use the object model to access fields from the data source payload and apply functions on top of it. Further, **the attribution engine allows users to create variables and definitions** that support the calculation of a more complex attribute.

Since attribution is a specialized skill, the tool needs to offer users ways to perform audits. This ensures attributes are validated before being fed to a scoring model or used directly in business rules.

Like the modeling engine, the attribution engine also has pitfalls. A poorly designed attribution engine will likely lead to a poor user experience or inaccurate calculations, which could prove costly in a live environment. While there are several analytical tools in the market, not all can consume data source payloads in a raw format. Therefore, having a tool that can do both is a differentiating capability that will give the decision engine provider a competitive advantage.

Business intelligence and reporting
Decision engines are often end-to-end transaction processing systems. Over time, users like to tap into and perform analysis on the build-up of transactional data to improve their business performance.

Transaction data is separate from raw data being fed to the decision engine from various data sources. Having the ability to perform analysis and get insights across this universe of raw and transactional data using a business intelligence (BI) tool gives users a significant advantage in improving their process performance.

Decision engines can leverage various cloud providers and associated tools to allow users to create BI reports. Sometimes insights from BI reports end up plugging significant gaps in use cases or leading to the implementation of a new orchestration to solve unaddressed scenarios.

FIs rely on BI and insights to improve their performance, so this capability becomes a key differentiator for decision engine providers to consider on their roadmaps.

Strategy optimizer
A decision engine can automate the consumption of these insights and recommend optimizations to the process flow, or even the end-to-end strategy.

A strategy optimizer is essentially a feature built on top of the existing BRMS that allows users to tweak business rules or the process flow to improve performance or business outcomes. These tweaks are a result of insights generated by the BI tool.

A strategy optimizer also allows users to run a champion/challenger comparison with the tweaks to compare and contrast how the changes are performing. Further, the strategy optimizer should provide the ability to parameterize key fields used in business rules, so they can be easily modified to evaluate the outcome.

The strategy optimizer is a capability that **allows users to perform regression analysis on their end-to-end process before and after making adjustments to the decision rules.**

Case manager
Another popular feature in large complex implementations of decision engines is a case manager. Often the use case implemented involves outcomes that require a manual review from a customer representative. A case manager allows customers to view these transactions and perform a manual verification to close the application with a decision.

Case managers can serve multiple use cases and have varying implementations, which is why they are not a core capability of a decision engine, but a differentiating one. **Decision engine providers must carefully evaluate case management as a capability that is in line with their product strategy.** Case management capabilities involve significant research and design around the user experience, so decision engine providers should factor in that aspect.

Batch
Batch is another way customers use decision engines to process workflows in an offline environment.

The only thing that's different in this model is how Ingress/Egress happens. Instead of sending real-time payloads, customers feed batch files with records formatted in a certain way for the decision engine to process. The output is also a batch file. This model is **popular in marketing use cases where customers want**

In use case #1, the utility company found that **too many new applications were being rejected because the identity verification step was returning mismatched addresses.** On further analysis, the utility company determined the address mismatch rate was high because a chunk of the applicants were students moving residences during the summer. **The utility company expanded the address match to include previous addresses, and by making this change, they improved approval rates.** A strategy optimizer combined with BI tools allows companies to perform analysis and make adjustments to determine if it improves results.

large segments of qualified customers to reach out to via campaigns.

Streaming
Data streaming is an emerging trend. A decision engine could have an API that customers can integrate with and stream real-time updates. This model can be useful in scenarios where customers use decision engines to alert them of any changes in the underlying data associated with the user base or generate triggers.

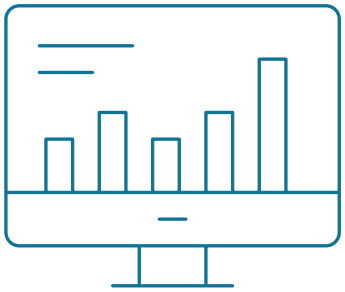
Streaming capabilities are a one-way interaction, where the decision engine replays changes/updates back to a listener set up on customers' systems. **There are several use cases where this capability is extremely useful, including debt monitoring and life events.**

Back-office user interface (UI)
Decision engines cater to many use cases that often involve manual tasks, including case management or order entry. **It is not uncommon for customers to ask for a back-office UI built on top of the system-to-system base that a decision engine uses.** In a contact center environment, the back-office UI could be used to view details of a transaction and perform an override on the existing disposition. A back-office UI could also serve as a case management tool, where a customer rep uploads additional documents associated with manual verification or review various data attributes in more detail to determine a disposition.

A business user could use a back-office UI to generate operational reports and determine user activity and performance metrics or set up new users and control their access levels.

If the customer uses a decision engine in a POS environment to approve potential customers for a line of credit, then the back-office UI could be used manually to enter new applications for an instant decision outcome. For example, a flooring company is running a promo and offering low-interest financing to interested homeowners. A sales rep onsite could use the decision engine's back-office UI to run the transaction on his device and provide an offer to the homeowner.

While there are several benefits of providing a back-office UI, there are potential pitfalls. A poorly designed UI will likely create a poor customer experience, and even if the underlying decision engine is functioning properly, the customer will likely feel frustrated by the experience. Another challenge here is customizations. It's not uncommon for customers to want different fields on an order entry page vs. using something common. **Customizations can break the mold of the product and force decision engine providers to move away from a multi-tenant product to a standalone application**, which will likely lead to higher operating costs.



Decision engines cater to many use cases that often involve manual tasks, including case management or order entry.

Operational metrics dashboard
Decision engines can be used in low- or high-volume scenarios. In a high-volume scenario, customers typically have policy owners or process owners that govern what the decision engine should be doing and monitor the performance. These policy owners are responsible for ensuring that the decision engine is performing optimally and evaluate performance metrics and make tweaks to the system.

Most modern SaaS platforms are architected in a way that involves sophisticated events and logging frameworks. These frameworks measure everything associated with the platform. Some of these events represent operating metrics that could be insightful for the policy owner. Decision engine product managers should consider exposing some of these metrics via an ops dashboard to the policy owner, so they can view everything in one place.

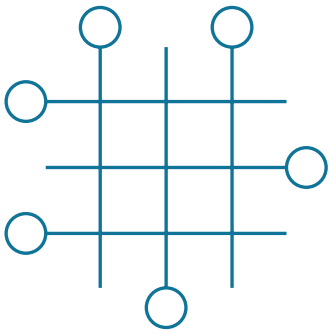
The metrics dashboard is also useful when determining system performance and evaluating it against contractual Service Level Agreements (SLA).

Microservices architecture

When we talk about a decision engine, we should consider implementation and setup. Capabilities including solution orchestration and data orchestration are critical, but for these capabilities to work, we need to lean on a modern microservices architecture.

Most of the decision engine's capabilities need to be configurable and turned on or off easily, based on customers' needs. A microservices architecture allows features to be developed as standalone capabilities that are discoverable in the orchestration layer. This architecture allows product development teams to expand the capabilities of a decision engine and continue to enrich the feature set, while also allowing fulfillment teams to configure these for each customer.

In essence, a microservices architecture allows decision engine implementations to be more configurable, rather than need custom development. Not only does this speed up revenue, but also supports future changes and the compatibility of the solution.



In essence, a microservices architecture allows decision engine implementations to be **more configurable.**

Best practices for success

Even with all the best ingredients, sometimes things go wrong. Decision engine providers should follow several best practices to ensure implementation success and avoid any potential pitfalls.

Well-set customer expectations

With any software project, poorly managed customer expectations will likely lead to high costs, low quality, and undesirable outcomes. While product-based companies can help control expectations by defining boundaries around their product's capabilities, configurability is necessary for a decision engine.

Decision engines have many applications. **While the simplest implementations tend to be smoother due to well-defined parameters, as complexity increases, the amount of configured or customized features increases.** This will inadvertently introduce elements and aspects of customer experience, which start to fall outside the bounds of what the decision engine was meant to do.

Rules that could help manage proper customer expectations:

Rule 1: Education

Don't assume your customer knows what a decision engine is. Educate a bit.

Rule 2: Consultation and design

Consult with the customer at every step. Pay special attention to areas like:

- Ingress/Egress
- Data — Information that the customer intends to use in the decision-making process
- Business Rules — Define conditions, actions, and process flow
- Compliance — Customers often don't know regulatory and compliance requirements
- Capacity — Configure to support projected volume and scale up/down dynamically

Rule 3: Documentation

Standardize your deliverables. Essential deliverables include:

- Business requirements artifact or checklist
- Process flow diagram, inclusive of business rules definitions
- System requirements — capacity, payload, etc.

Rule 3: Fulfillment

Fulfillment should be systematic and dependent on features like:

- Solution orchestrator
- Data orchestrator
- BRMS

Rule 5: Validation

Validation should cover all aspects, including:

- Configured features
- Data configurations and associated payloads
- Ingress/Egress
- End-to-end use case validation
- Customer validations (aka user acceptance)

Rule 6: Duration

Decision engines cannot be turned on with the flip of a switch, so it's important to set expectations on the amount of time required to configure and deliver the solution.

Clearly defined business outcomes

Decision engines can be configured to accomplish multiple tasks, so it's important to **define the business outcomes upfront.** You can tweak these along the way and make necessary adjustments to the implementation, but the starting point cannot be a moving target.

Providers need to lean on superior consulting abilities to better understand what the customer is trying to solve with the decision engine to create well-defined use cases with outcomes.

There is a lot more that would need to be defined for the above scenario, like what risk threshold allows the customer to achieve 80% auto-approval rates, but that is what the provider should uncover during the consulting and design phase of implementation.

The more defined the outcomes are, the more effective the decision engine implementation can be. In the example above, if the customer knew the type of fraud they see most or the identity verification failure rates, it could lead to the provider considering a data analysis step before starting the consultation and design. The data analysis step allows the provider to uncover failure points in the existing process that contribute to high fraud or identity verification failure rates. These failure points can be addressed during the decision engine implementation by leveraging a combination of data and business rules.

What is applicable at a macro level is also applicable at a micro level.

The payload specifications, business rules definitions, and flow requirements must be well-defined for the decision engine to succeed. Gaps in business rules and unaddressed conditions can lead to unexpected runtime errors. Incomplete process flows can lead to transaction failure. If junk characters aren't caught during Ingress, it can lead to failed transactions, or data and transactional charges with no valid match.

Poorly defined outcome:

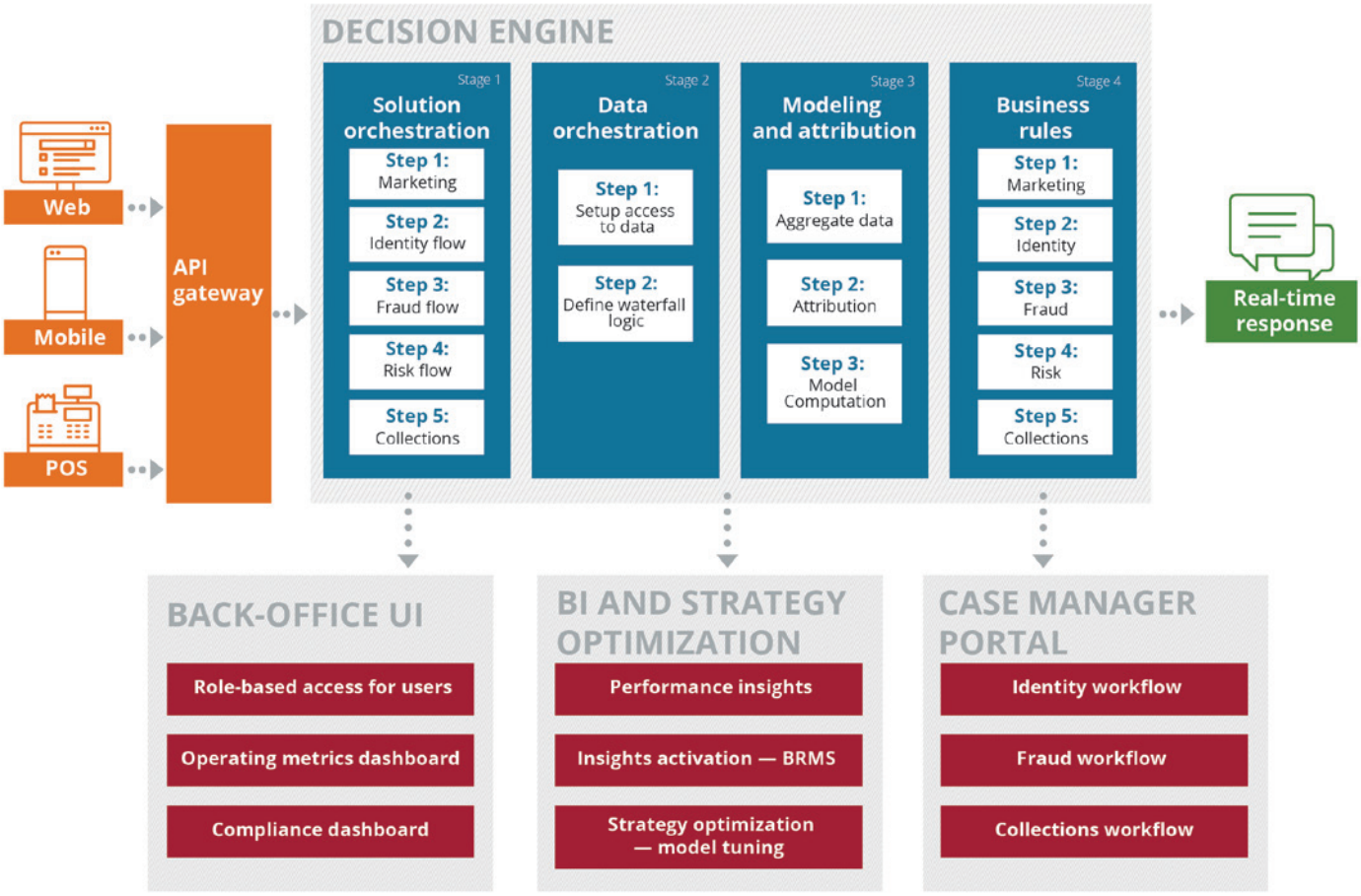
Automate the customer acquisition process and reduce manual costs.

Well-defined outcome:

Automate the customer acquisition process by shifting identity verification, fraud detection, and credit check processes into a single decision engine instance. Target auto-approval rate to be above 80%.

Solution simplification
Decision engines can be leveraged to cover various use cases. They can also combine use cases into one transactional flow. However, providers are encouraged to break transactions down into logical use cases and leverage the orchestration capabilities of the decision engine to provision these.

Diagram 5



In Diagram 5, above and shared earlier in this white paper, the decision engine should be configured to treat each use case (Marketing, Identity Verification, Fraud, Risk, and Collections) as a distinct transaction with an outcome. The provider should also take a prescriptive approach and recommend a predefined strategy (sequence of data and business rules to execute) for each use case that can be easily tweaked to suit the customer's needs.

While it's not uncommon for large companies to want something completely custom, **providers should lead with a consultative approach and, where possible, keep the customer's requirements as close to standard features as possible.** If the situation requires custom-developed features, care and consideration should be taken to ensure that these customizations can be supported and accounted for in the cost model.

Breaking complex scenarios into simple solutions (orchestrations) can reduce system complexity and make the implementation more scalable and supportable. **Simplification allows providers to measure the performance of the customer's instance against established industry benchmarks**.

Go-to-market strategy
Many companies fail to grow their decision engine business because they did not invest in building a good go-to-market (GTM) strategy.

Even having your decision engine built on the most modern tech stack with all the key features isn't enough to be successful.

Decision engines are SaaS platforms. **Providers can offer packaged versions of the platform as a product, or a configurable version.** These decisions must be part of the GTM strategy, as they dictate the amount of salesforce, support systems, and staff required. A configurable version requires a more consultative salesforce, along with some form of professional services — compared to a packaged version where the setup may a lot simpler.

The GTM strategy should also define the scope of the decision engine, so it does not get pitched as something it's not meant to be. For example, pitching a credit decision engine as a CRM platform, where the long-term chances of the decision engine being successful are likely low.

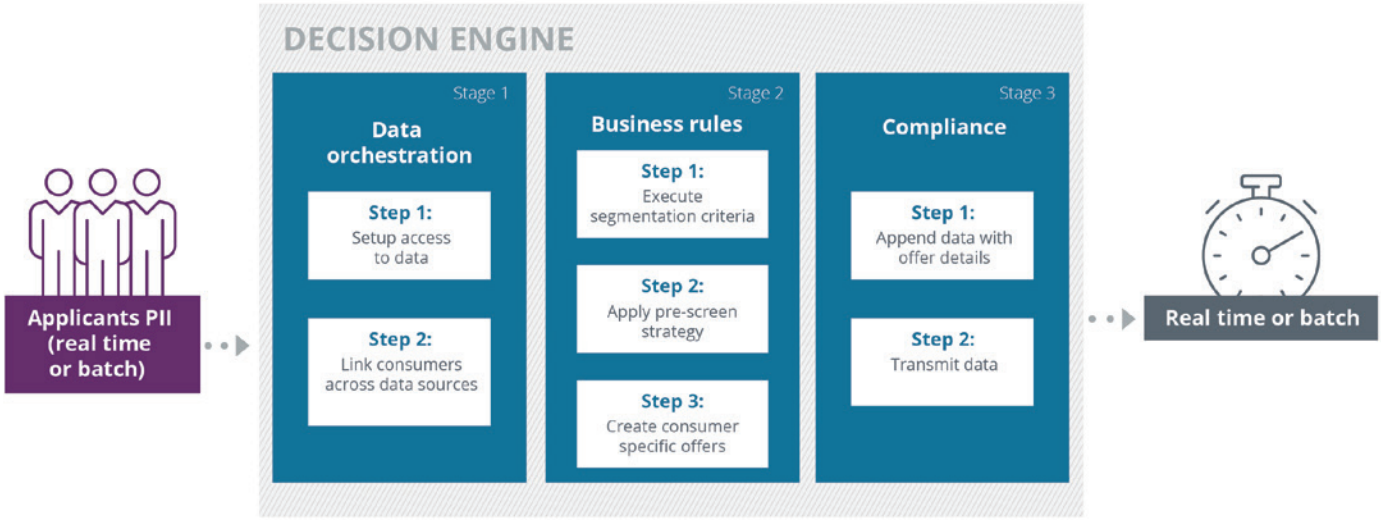
A traditional GTM strategy also ensures internal alignment and funding for all the necessary support functions. Not having this piece figured out up front could lead to the decision engine not being operationally ready for general availability. This will also lead to a poor customer experience, support challenges, and missed SLAs.

A traditional GTM strategy also **ensures internal alignment** and funding for all the necessary support functions.

Data-driven insights for continuous improvement
Decision engines' performance can deteriorate over time, due to changes in data, conditions, or even economics. They should be tuned regularly to ensure the performance stays optimal, which could involve updates to the business process, strategy, application performance, or even the underlying infrastructure.

Evaluate how fine-tuning can help the use case shared previously in this white paper in Diagram 3.

Diagram 3



A credit union is using a decision engine to create marketing campaigns to target new customers and prescreen them in advance to speed up the offer acceptance process.

The credit union monitored offer activity and acceptance rates and observed a decline in the target base over the years. After further analysis, it was determined that, as the local demographics changed and more millennials moved into the area, many more "emerging credit" scenarios came up. The bank realized that conventional credit data would not be sufficient to solve this problem and decided to add alternate data assets to the mix. This required changes in the data access setup, segmentation criteria, and prescreen strategy.

These types of changes are all considered tuning. In this example, the customer had to make changes to their risk strategy. In other cases, tuning could involve simply lowering the threshold in business rules to allow more candidates to come through.

An example of a simpler tuning exercise: During the pandemic, many companies wanted to relax criteria to allow new customers to sign up for their services or reduce collections activity for a defined period. These changes are easy to implement in a decision engine, as they modify the underlying business rules. These changes are also important, as they allow customers to adapt their business processes to the changing environment.

Data analytics needs a purpose and a plan. But as the saying goes, "no battle plan ever survives contact with the enemy." To add another military insight — the OODA loop, first conceived by U.S. Air Force Colonel, John Boyd regarding the decision cycle of observe, orient, decide, and act. Victory, Boyd posited, often resulted from the way decisions are made. The side that reacts more quickly to situations and processes new information more accurately should prevail. **The decision process, in other words, is a loop or — more correctly — a dynamic series of loops.**

Decision engines, like any other software, require occasional maintenance, including keeping binaries up to date and keeping the operating system current. Failure to perform regular maintenance will likely lead to degradation of system performance or a specific component. This could lead to incidents and eventually breach SLAs.

Avoiding pitfalls

Throughout this white paper, I have highlighted several areas that don't cleanly align with a decision engine's core capabilities. Additionally, not following best practices reduces the chances of successful implementation. Let's look at the potential pitfalls that could turn a decision engine implementation into a failure.

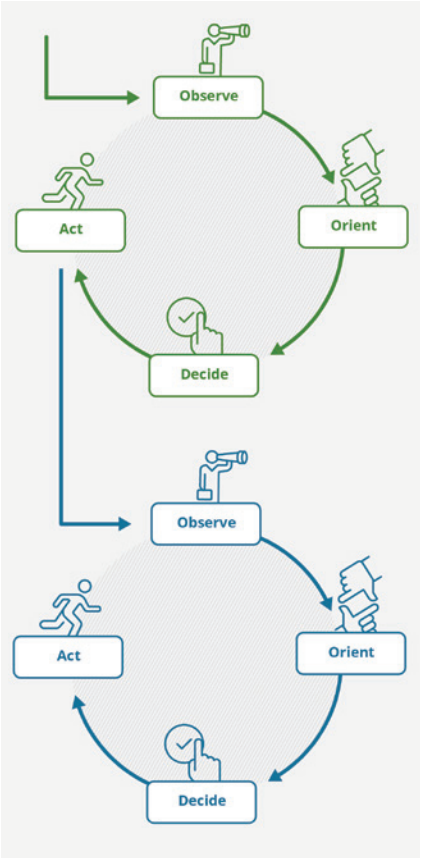
Architecture

The success of a decision engine partially depends on configurability. **While there are many architectural approaches to achieve configurability, the recommendation is to use a microservice architecture** that allows ease of configuration and a lighter deployment package. A microservices architecture also allows new features to be consumed with ease.

Scope of a decision engine

Not having a well-defined GTM strategy often leads to the decision engine scope staying open-ended. This also leads to misaligned expectations. **Providers are encouraged to take a product-centric approach and define the solution patterns and use cases the decision engine can accomplish.** New features should be rolled out as product increments vs. customer-specific solution enhancements.

Diagram 8: The Observe, Orient, Decide, Act framework

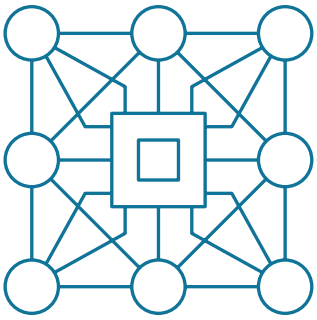


Ingress/Egress
Standardization of Ingress and Egress is critical. **Since decision engines involve business rules that work on well-defined parameters, incoming and outgoing payloads need to be standardized.** This also allows providers to easily scale across web, API, and batch capabilities. Non-standard data can lead to unexpected runtime errors, which are difficult to triage and resolve. As much as possible, providers should attempt to make the decision engine "foolproof."

Modeling and attribution tools
Data modeling and attribution are advanced statistical capabilities that decision engine providers should carefully vet. Since there cannot be a half-baked approach, **providers should consider build vs. buy vs. partner as appropriate.**

User interface
It's common to find decision engines that support a UI. Users typically expect the UI to be customized to their specifications. **Decision engine providers should draw the line to determine what is controllable vs. not controllable.** Adding a few additional fields on the UI or changing the page flows can sound simple, but at the same time, make the instance unique to a customer and increase maintenance tasks. **A standard UI with well-defined scenarios is recommended.** Providers should also take the time to understand how their customers intend to use the UI, so it does not get used in a way that's not supported.

Data cleansing and entity resolution
Data is foundational to a decision engine, and customers often want their decision engine to consume vast quantities of data from different sources and process it through the business rules. On most occasions, the only thing common input between these disparate data sources is the transactional input variables, like PII. So customers can expect the decision engine to perform data cleansing to improve match rates or do entity resolution across data sources. Decision engines can be designed to solve this, but **due consideration should be given before making this a standard feature.** Any discrepancies or mismatches can lead to poor outcomes, and that could lead to failure.



Data is **foundational** to a decision engine

How can you select the right decision engine? Financial institutions should ask several questions to determine the best solution.

Capabilities
- What will the decision engine be used for?
- What payload format does the decision engine support?
- What features can the decision engine be easily configured for?
- What kind of architecture is the decision engine built on?

Data
- What kind of data will be passed to and from the decision engine?
- What kind of data access is required in the use case?
- What data sources are available in the decision engine?
- Will there be a need for data orchestration?
- How does the decision engine store and retrieve data?
- What kind of data retention and archiving policies does the decision engine have?

Rules engine
- How complex will the business rules be?
- Does the decision engine offer a rules engine?
- Will there be a need for rule flows?
- Does the rules engine have the ability to author decision trees and matrices?
- How often will the business rules need to be updated?
- Does the rules engine have the ability to test standalone rules and end-to-end flows?
- How technical are the users of the decision engine?
- Does the rules engine automatically produce rule documentation?
- What kind of regulatory requirements need to be met in the solution?

Reporting
- What kind of reporting requirements need to be met by the solution?
- Does the decision engine offer any BI tools around transactional data?
- Is there any batch reporting capability that comes standard with the decision engine?
- Does the decision engine have online reports?
- Does the decision engine support batch input/output feeds?

Orchestration capabilities
- What kind of orchestration capabilities does the engine offer?
- Does the decision engine offer data orchestration capabilities?
- Is there a customer-facing admin console to make config changes?
- What is the deployment model? Can changes be scheduled for deployment?
- Does the decision engine have pre-baked solution templates?
- Does the decision engine support custom use cases?

Pricing
- What pricing model does the decision engine use?
- Is pricing subscription-based or transaction-based?
- What kind of billing schedule does the decision engine follow?
- How does billing work?
- Are data fees included in the pricing?

User interface
- Does the solution require a front-end user interface?
- Does the decision engine provide support for case management?
- Does the UI have search and update capabilities?
- Can the UI be embedded as a portlet within an external site?
- Is there an admin console through which user management tasks can be performed?

Modeling and attribution
- Does the solution require any analytical models or attributes?
- Does the decision engine offer the ability to implement customer scorecards?
- Does the decision engine offer the ability to audit models and attributes?

Scalability
- Does the decision engine leverage the public cloud?
- Does the decision engine have the dynamic ability to scale up or down?
- What kind of geo-redundancy does the decision engine offer?

Security
- What kind of authentication does the decision engine leverage?
- What kind of data protection and encryption does the decision engine support?
- How does the decision engine handle Class 5 data?
- What kind of infrastructure does the decision engine run on?
- How does the provider address vulnerabilities?

Ease of integration
- How quickly can the solution be configured and set up?
- What kind of test data does the provider offer?
- Does the decision engine have an API front end?
- What are the various connectivity protocols that the decision engine supports?

Maintenance and Service-Level Agreement
- What is the uptime SLA offered by the decision engine?
- What are the typical response time thresholds that the decision engine offers?
- How does the decision engine perform maintenance?
- What is the deployment technique that the decision engine uses?
- How often does the provider roll out upgrades?
- How does the provider handle major version upgrades?

Conclusion

Decision engines have many use cases, but some of the most successful applications are in the financial services industry. There is heavy reliance in this space on data-oriented solutions with regulatory oversight. That, combined with the need for near-real-time outcomes, makes decision engines the perfect choice.

Decision engines not only help companies consume and evaluate large quantities of data, but they also help automate regulatory requirements and offer strategy tools to make processes more efficient. All this packaged in a single platform results in a high degree of automation, which can reduce manual costs significantly when done right. However, when not done correctly, it can lead to costly operational challenges. This white paper intends to educate and guide companies in selecting the right decision engine, setting it up for success, avoiding costly implementation mistakes, and ultimately, driving favorable business outcomes.

equifax.com/business/interconnect