Goldman Sachs

jDMN: A DMN engine in Java

May 2024

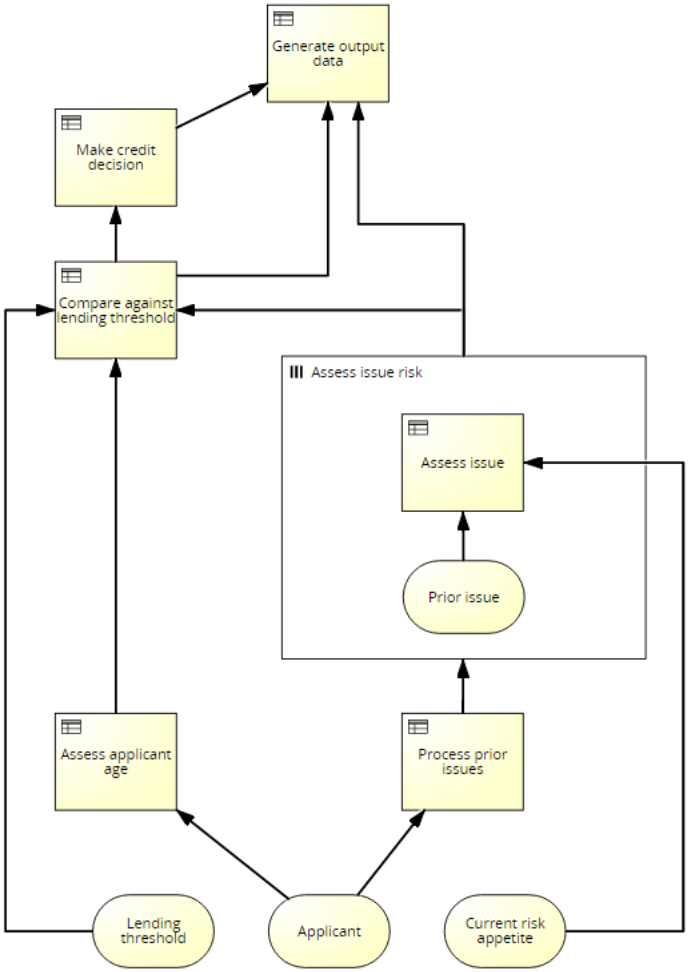
Content

- What is DMN?
- Overall structure of jDMN
- Transpiler to Java / Kotlin / Python
- Code optimisation
- Q&As

What is DMN?

Decision Model and Notation (DMN)

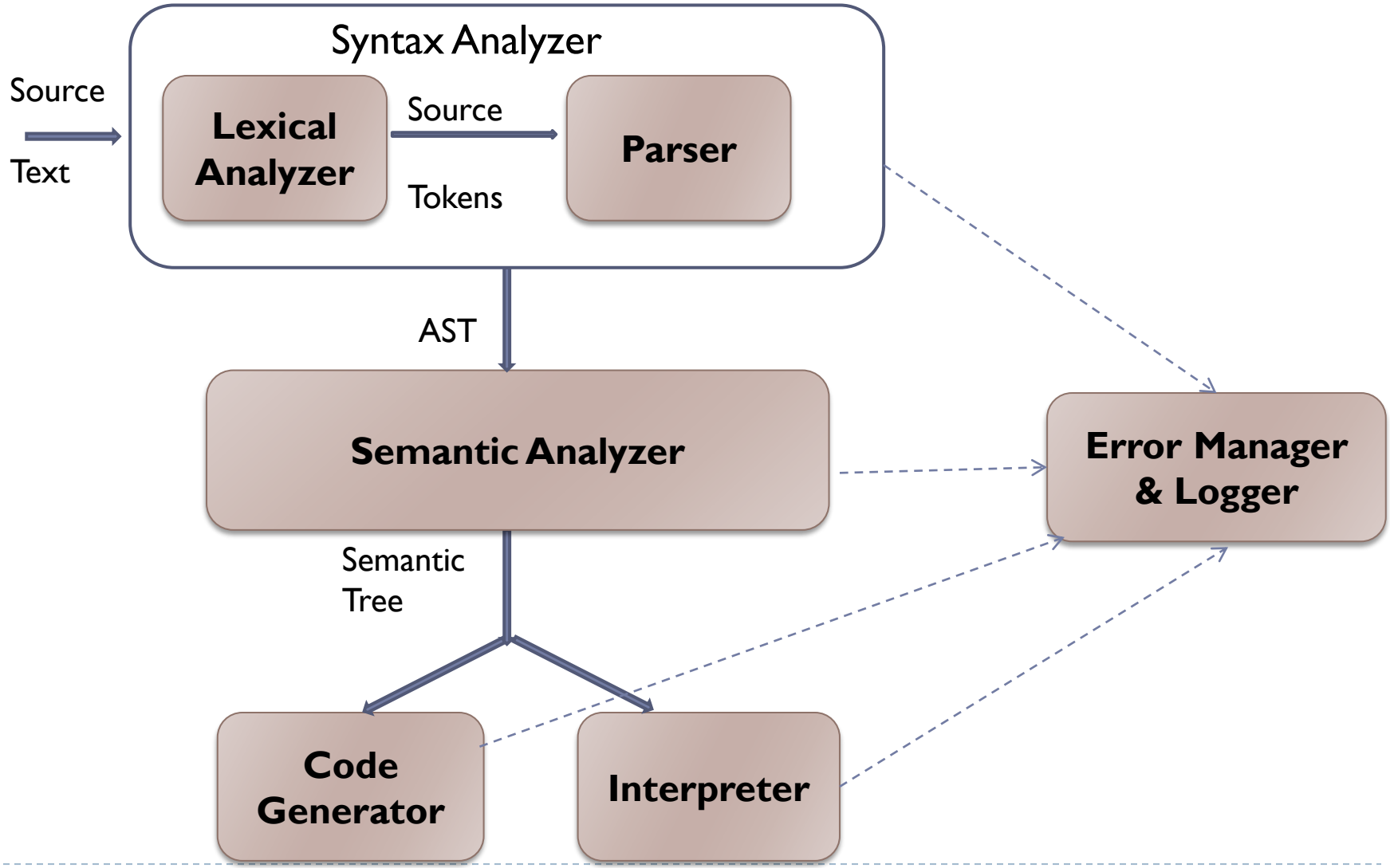
- Standard published by OMG
- Notation to support decision management and business rules
- Users
  - Business People: manage and monitor decisions
  - BAs or Functional Analysists: specify decision models
  - Technical developers: execution and automation
- DSL
  - Diagrammatic notation
  - Templates
  - Expression language FEEL
- Standalone or with BPMN & CMMN



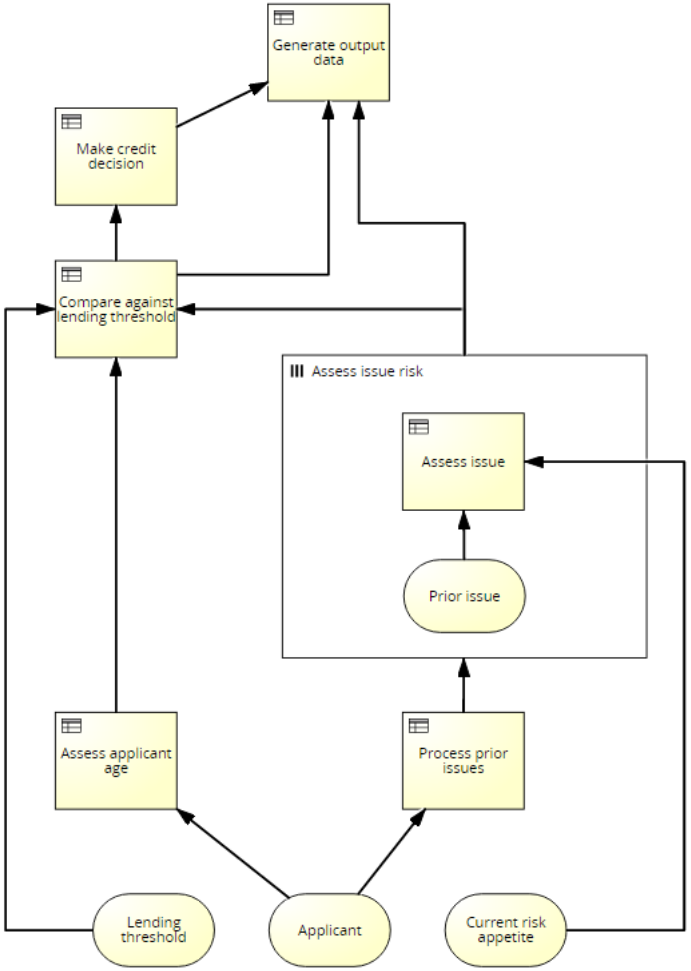
Overall Structure

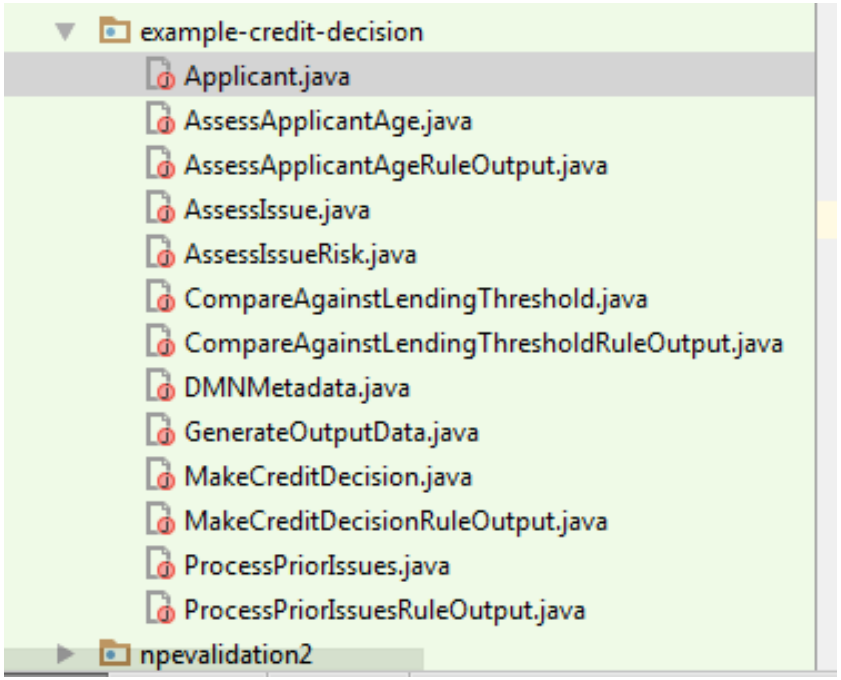
- DMN Processors
  - Reader / Writer
  - Validators
  - Transformers
  - Interpreter
  - Translator
- Dialects
  - DMN 1.1 – 1.5
  - Signavio

Overall Structure

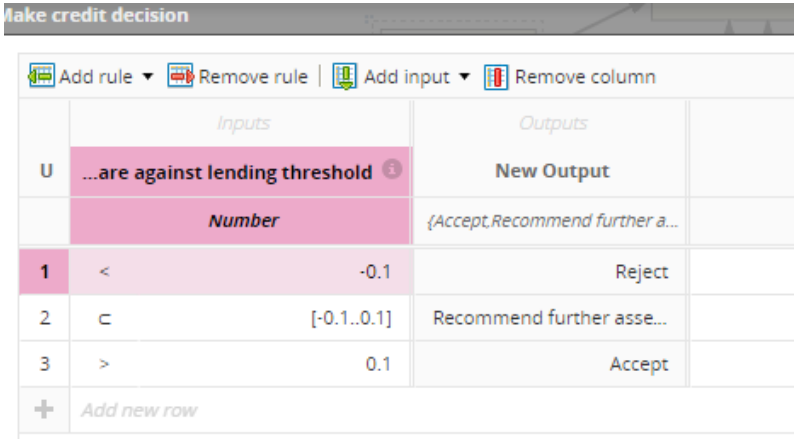


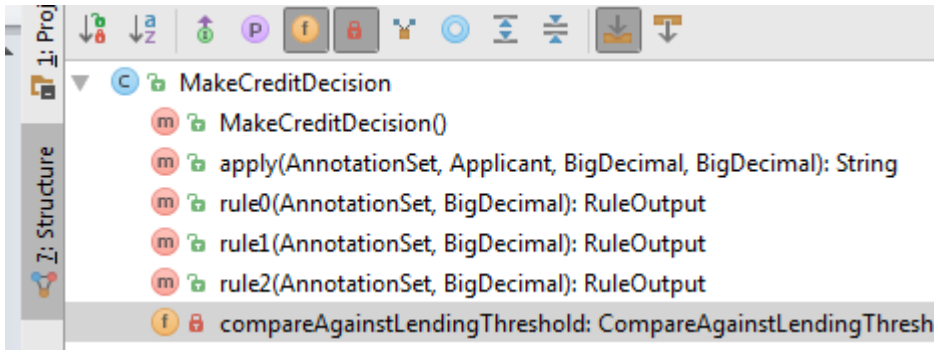
Translation





Translation





Translation

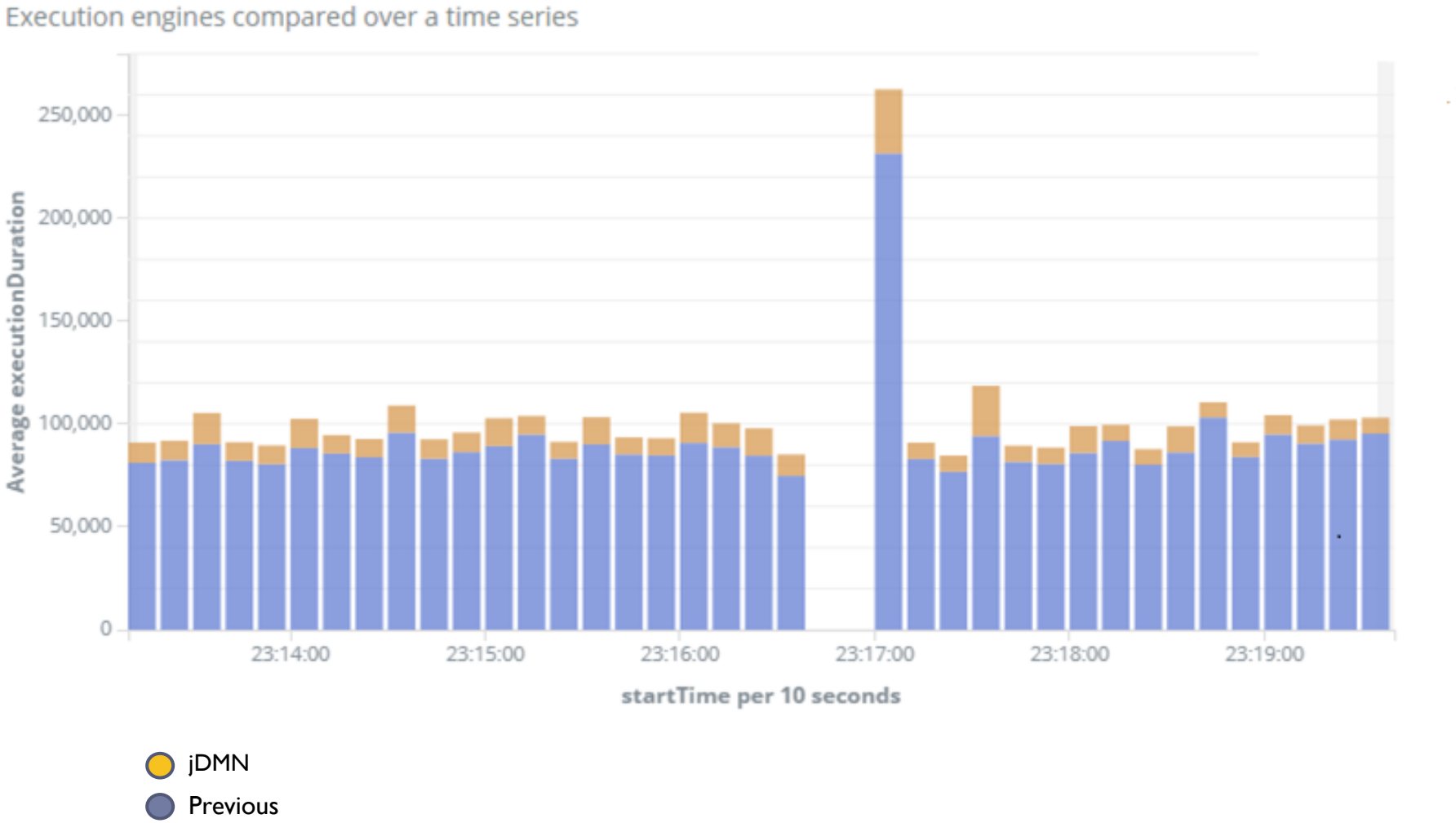
- How did we built it?
  - Syntax-Driven Translation Schematas (SDTS)
  - Based on Knuth's attributed gramars
    - Synthesized attributes
    - Inherited Attributes

```
Expr1 → Expr2 + Term    { Expr1.value = Expr2.value + Term.value }
Expr → Term             { Expr.value = Term.value }
Term1 → Term2 * Factor  { Term1.value = Term2.value * Factor.value }
Term → Factor           { Term.value = Factor.value }
Factor → "(" Expr ")"   { Factor.value = Expr.value }
Factor → integer        { Factor.value = strToInt(integer.str) }
```

Translation

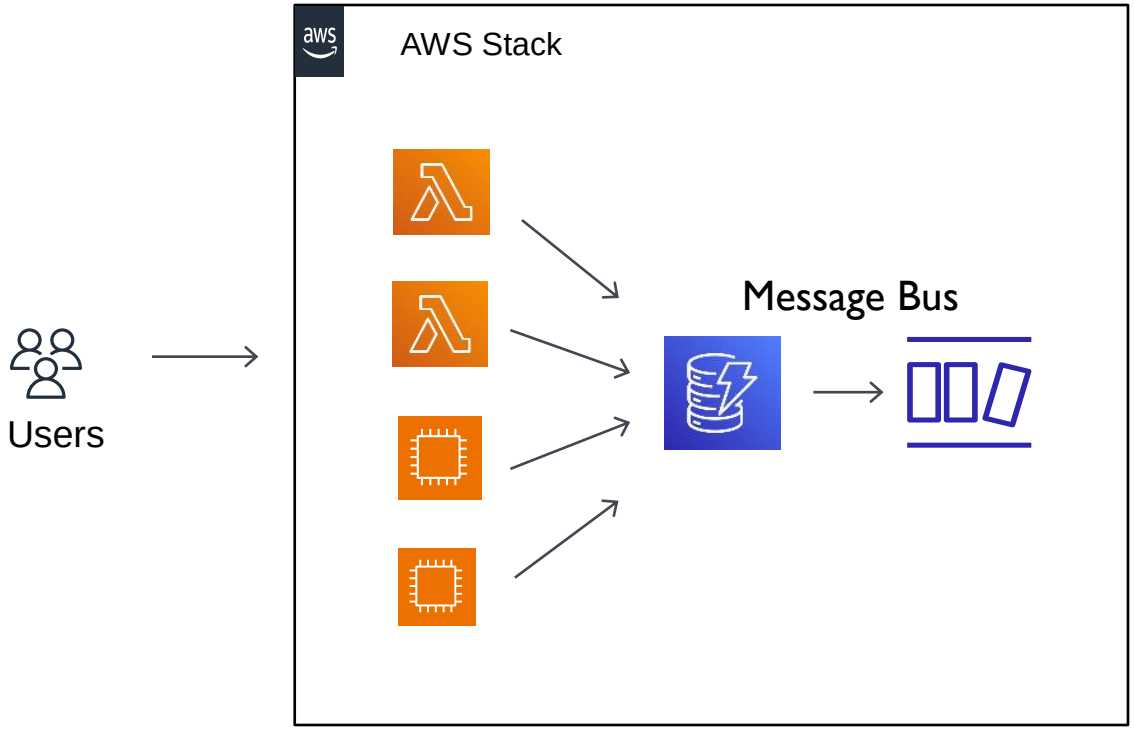
- Advantages include
  - *Performance*: we have seen runtimes 4-10 times faster than previous engine execution in complex decision tests
  - *Stability:* given that we now control the code generation, we are able to resolve issues without relying on the vendor
  - *Functionality:* the fact that we control the code generation means that we are also able to enable more advanced functionality for DMN/Java models

Performance

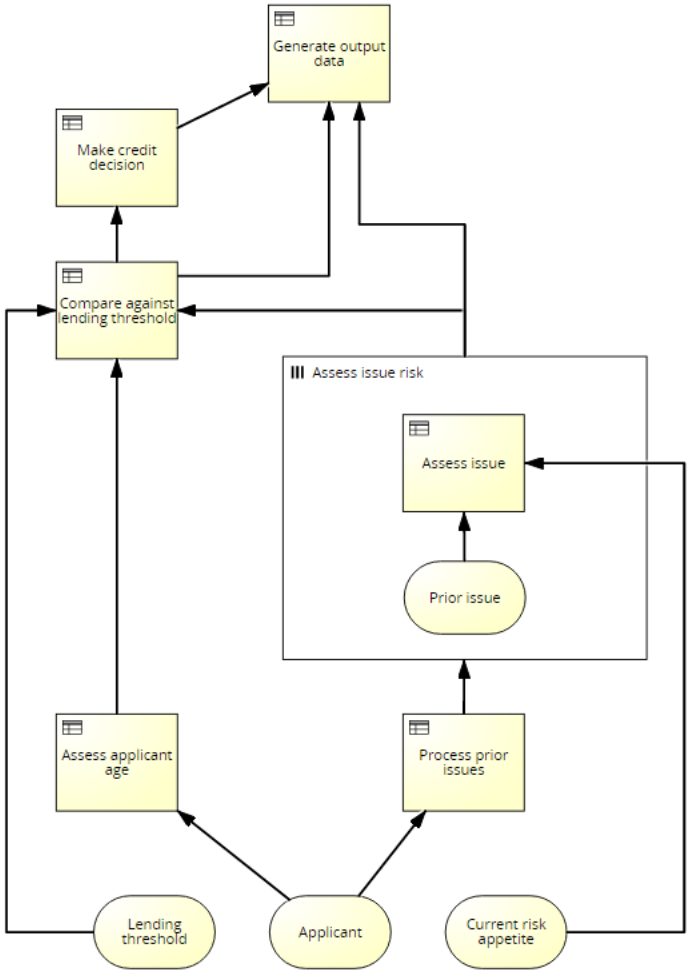


Code Optimisation – Model Level

- AWS Lambda
- gRPC / protobuf



Code Optimisation – DRG Element Level



- Caching
- Map-reduce for
  - MID

Code Optimisation – DRG Element Level

- Lazy evaluation
  - Inner nodes (Decisions, BKMs, Decision Services)
  - Leaves (Input Data)

F	E		
	A	B	*Output*
1	= "1"	= "2"	# C
2	= "2"	-	# D

Code Optimisation – DRG Element Level

Performance Before DRG Optimization				
Decision Name	Request Count	Response Time 90th (ms)	Min Response Time (ms)	Max Response Time (ms)
D1	12899	40	4	5562
D2	361	6676	21	9653
D3	28731	15	1	933
D4	86165	39	4	6367

Performance After DRG Optimization				
Decision Name	Request Count	Response Time 90th (ms)	Min Response Time (ms)	Max Response Time (ms)
D1	12899	35	5	7150
D2	361	315	153	4603
D3	28745	12	1	606
D4	86204	37	4	7736

Code Optimisation at FEEL level

- Native Types
- Built-in functions
- Native Compiler / Interpreter (e.g. JIT compiler)

Recent features

- DM composition
- Cross-translation for Java, Kotlin and Python
- Optimised execution (e.g. map-reduce for MID)
- Support for gRPC / protobuf
- Explanation: Annotations and Tree / Postorder Listeners