**Lab Guide**

IBM Decision Manger Open Edition

Nigel Crowther – ncrowther@uk.ibm.com

Hands-on Lab

Design Patterns for Big Decisions



NOTICES

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive, MD-NC119

Armonk, NY 10504-1785

United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

**TRADEMARKS**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a Registered Trade Mark of AXELOS Limited.

ITIL is a Registered Trade Mark of AXELOS Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

© Copyright International Business Machines Corporation 2020.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

**Table of Contents**

[1 Introduction 5](#_Toc129865225)

[2 DMN Design Patterns for Big Decisions 6](#_Toc129865226)

[2.1 The Anti-pattern 6](#_Toc129865227)

[Figure 1: Initial Habitability DMN Diagram 6](#_Toc129865228)

[Figure 2: Habitability Decision Table 7](#_Toc129865229)

[2.2 The Structured Data Pattern 8](#_Toc129865230)

[Figure 3: PlanetInfo Data Type 8](#_Toc129865231)

[Figure 4: Planet DMN Input defined as a PlanetInfo Data Type 9](#_Toc129865232)

[Figure 5: Decision referencing PlanetInfo data type 9](#_Toc129865233)

[2.3 Lab 2 – The Divide and Conquer Pattern 10](#_Toc129865234)

[Figure 6: Oxygen Table 10](#_Toc129865235)

[Figure 7: Atmosphere Table 10](#_Toc129865236)

[Figure 8: Surface Table 11](#_Toc129865237)

[Figure 9: Habitability 11](#_Toc129865238)

[Figure 10: Divide and Conquer Pattern for Habitability 12](#_Toc129865239)

[2.4 The Tiered Service Pattern 13](#_Toc129865240)

[Figure 11: Tiered Service Pattern 13](#_Toc129865241)

[Figure 12: Atmosphere Decision Service 14](#_Toc129865242)

[Figure 13: Surface Decision Service 14](#_Toc129865243)

[2.5 Lab 5 – Directory Pattern 15](#_Toc129865244)

[Figure 14: Tall thin table 15](#_Toc129865245)

[Figure 15: Hybrid BPMN/DMN solution calling rules and a database 16](#_Toc129865246)

[Figure 16: Manual Telephone Directory 16](#_Toc129865247)

[Figure 17: DMN to split Tall thin table 17](#_Toc129865248)

[Figure 18: FEEL to invoke alphabetically indexed decision services 17](#_Toc129865249)

[Figure 19: Routing rules for Swift Bic starting A-F 18](#_Toc129865250)

[2.6 Conclusion 19](#_Toc129865251)

[3 Labs 20](#_Toc129865252)

[3.1 Lab 0 - The Anti-pattern 20](#_Toc129865253)

[Figure 1: Kie Sandbox Start Page 20](#_Toc129865254)

[Figure 2: Habitability Decision Inputs 21](#_Toc129865255)

[Figure 3: Initial Habitability DMN Design 21](#_Toc129865256)

[Figure 4: Logic Type 22](#_Toc129865257)

[Figure 5: Habitability Decision 22](#_Toc129865258)

[Figure 6: Test Result - Habitable 22](#_Toc129865259)

[Figure 7: Test Result - Uninhabitable 23](#_Toc129865260)

[3.2 Lab 1 –Structured Data Pattern 24](#_Toc129865261)

[Figure 8: PlanetInfo Data Type 24](#_Toc129865262)

[Figure 9: Planet DMN Input defined as a PlanetInfo Data Type 25](#_Toc129865263)

[Figure 10: Decision referencing PlanetInfo data type 25](#_Toc129865264)

[3.3 Lab 2 – The Divide and Conquer Pattern 26](#_Toc129865265)

[Figure 11: Pressure Table 26](#_Toc129865266)

[Figure 12: Oxygen Table 26](#_Toc129865267)

[Figure 13: Methane Table 27](#_Toc129865268)

[Figure 14: Atmosphere Table 27](#_Toc129865269)

[Figure 15: Surface Table 28](#_Toc129865270)

[Figure 16: Habitability 28](#_Toc129865271)

[Figure 17: Divide and Conquer Pattern for Habitability 29](#_Toc129865272)

[3.4 Lab 3 – The Tiered Service Pattern 30](#_Toc129865273)

[Figure 18: Tiered Service Pattern 30](#_Toc129865274)

[Figure 19: Atmosphere Decision Service 31](#_Toc129865275)

[Figure 20: Surface Decision Service 31](#_Toc129865276)

[3.5 Lab 5 – Directory Pattern 32](#_Toc129865277)

[3.6 Lab 6 - Generating DMN from Excel 36](#_Toc129865278)

[3.7 Conclusion 37](#_Toc129865279)

# Introduction

In this Lab we will implement DMN patterns to scale large projects using **IBM Decision Manager Open Edition**. This lab assumes you have mastered the basics of **DMN** and **KIE Sandbox**.

The Lab will start by building a simple decision service that exhibits a design antipattern. We will improve this design by applying the following patterns:

* The **Structured Data Pattern** to simplify input data in DMN Diagrams
* The **Divide and Conquer Pattern** to split a complex table into smaller tables
* The **Tiered Service Pattern** to split a single DMN file into several smaller DMN Files.

By the end of the lab, you will understand how to build large scale decision projects in DMN.

It is worth noting that not all the patterns may be appropriate for your project. The patterns presented are appropriate for large projects, and for small projects with small decision tables, the patterns may not be required.

**Pre-requisites**

To perform this Lab, you need either:

A remote Virtual Machine from IBM TechZone with a pre-deployed PAM-DM Open Edition tooling to use in this Lab. Please follow instructions in Appendix A to provision and access to an instance of this Virtual Machine.

OR

Your own PC with Chrome or Safari pointing at URL <https://sandbox.kie.org/>

# Design Patterns for Big Decisions

In these labs, you will start with a simple decision service that will not scale. It shows how easy it is to fall into the trap of poor design. Once a poor design is approved, it quickly takes hold. At this point the decision service may require a total re write which can be expensive. Therefore, it is important to start with a good design at the beginning.

## The Anti-pattern

For the labs we consider a decision service that decides on planet habitability based on observed planetary data. The decision service is called *Habitability* and takes the following planetary inputs:

|  |  |  |
| --- | --- | --- |
| **Planet Attribute** | **Unit o** | **Description** |
| Surface Pressure | Atmospheres | The air pressure in atmospheres. On earth this is 1. Tolerable values are between 0.5 and 2 |
| Carbon Dioxide | % Atmosphere | Concentrations over 10% will cause unconsciousness or death |
| Methane | % Atmosphere | Concentrations over 1% will cause unconsciousness or death |
| Oxygen | % Atmosphere | Ideal range is 30-40% |
| Temperature | Average day time surface temperature in Degrees Celsius | 0-30C is ideal |
| Gravity | G | 0 to 2 is ideal |

This can be expressed in the following DMN diagram:

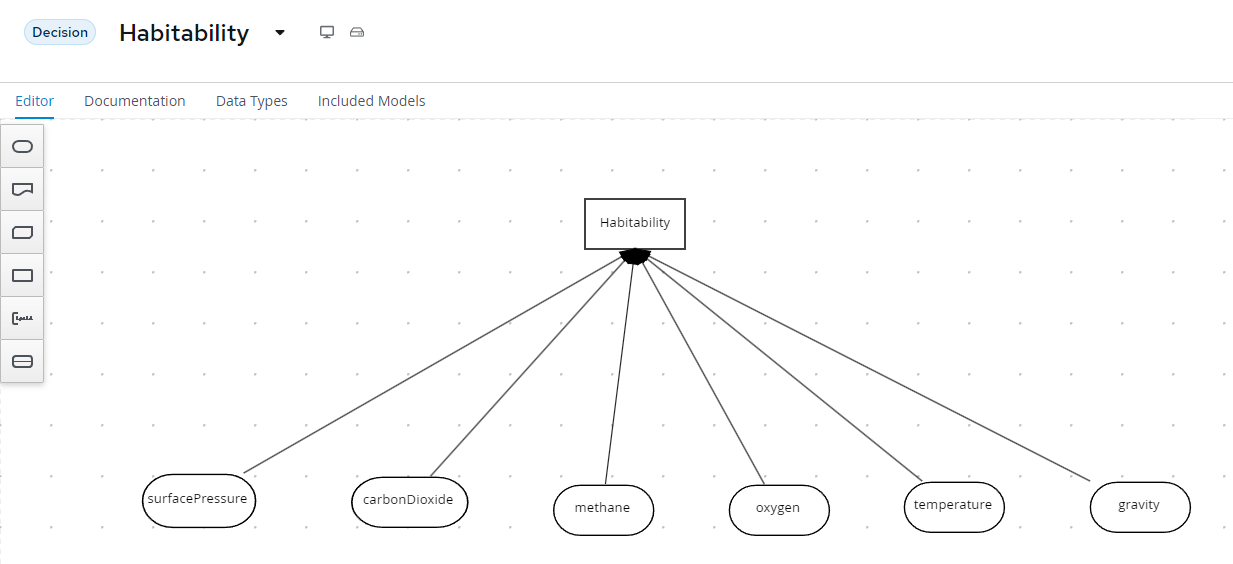


Figure 1: Initial Habitability DMN Diagram

And the decision logic can be expressed as the following decision table:

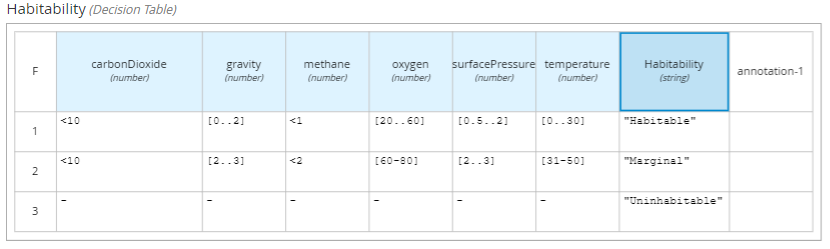


Figure 2: Habitability Decision Table

**What is Wrong with this Design?**

Whilst at first glance, this design looks ok, there are some serious problems.

The most obvious is that we have not created all possible decisions, particularly in the “Marginal” habitability state. Given the number of variables there could be hundreds or even thousands of habitability states which would require a unique row in the table. A thousand row table is not maintainable.

The second problem arises when a complete set of planetary data are added. For example, radiation levels, distance from earth etc. The table will become too wide to fit on the screen and the DMN diagram will have a web of inputs which will become difficult to maintain.

**Welcome to the ‘Swiss Army Knife’ Anti Pattern**

This design falls into the ‘Swiss Army Knife’ anti pattern. This antipattern is so named because one DMN diagram with one decision table tries to implement the whole decision, just like a Swiss army knife that tries to implement an entire toolbox.

The next three labs will show how we can refine the design to allow us to scale the decision service to thousands of decision rows and hundreds of attributes.

## The Structured Data Pattern

If your decision has more than four inputs, it is good practice to move these inputs to a data structure. This simplifies the DMN diagram and allows the structure to be passed as one variable to functions. In our example, planetary data can be subclassified into *atmospheric* and *surface* data:

**Atmosphere**

* Carbon Dioxide
* Methane
* Oxygen

**Surface**

* Pressure
* Gravity
* Temperature

This can be modelled in **Kie Sandbox** data types:

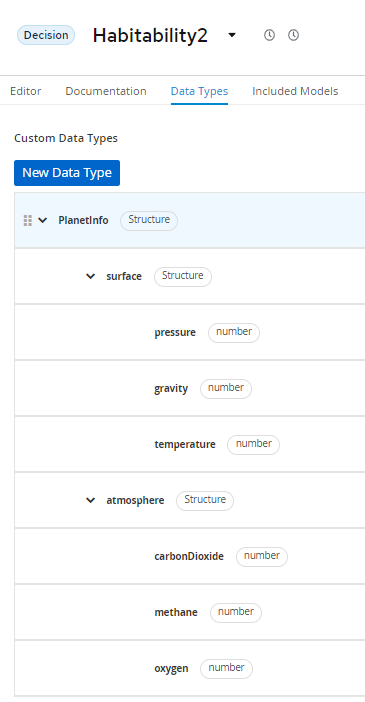


Figure 3: PlanetInfo Data Type

Now we can have just one DMN input called *planet* of type *PlanetInfo*:

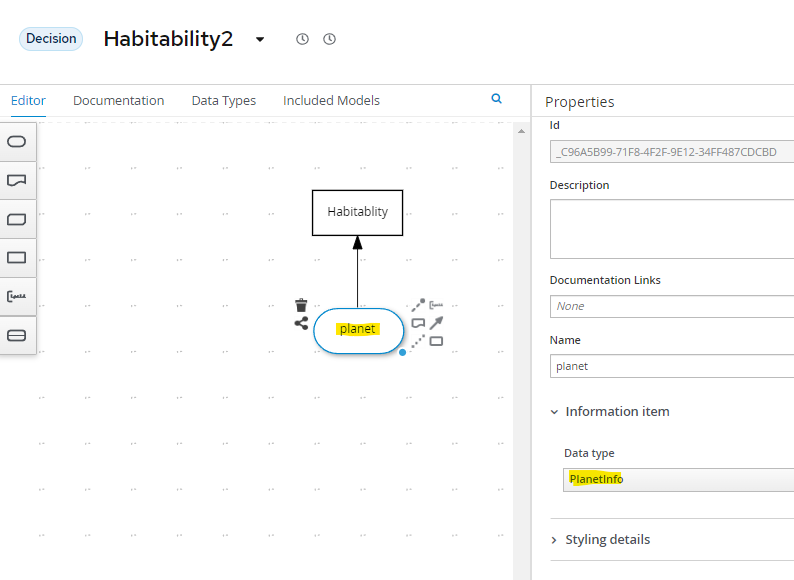


Figure 4: Planet DMN Input defined as a PlanetInfo Data Type

The decision table is the same as the previous lab, but it now references the new data structure:

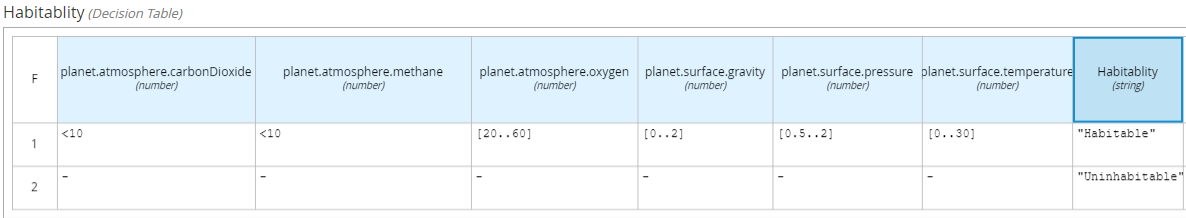


Figure 5: Decision referencing PlanetInfo data type

Although we have simplified the DMN model, the decision table will grow in width and length once more rules and attributes are added. This will be fixed using the *Divide and Conquer Pattern* described in the next lab.

## Lab 2 – The Divide and Conquer Pattern

To reduce the size of a decision table, divide and conquer it.

To do this, create a new decision table for each input attribute. The purpose is to reduce the attribute values to a restricted range of enumerated types. These enumerated types are subsequently used in decisions further up in the DMN model. In our example we are reducing all numeric ranges down to three enumerated types: *Optimal, Bearable and Deadly.* For brevity only the *Oxygen* table is shown. The other tables follow the same pattern.

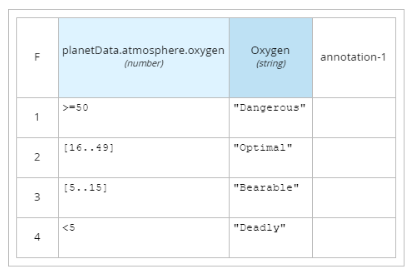


Figure 6: Oxygen Table

Now the decision outputs of the Oxygen, Methane and Carbon dioxide tables are combined to define the *atmosphere* decision:

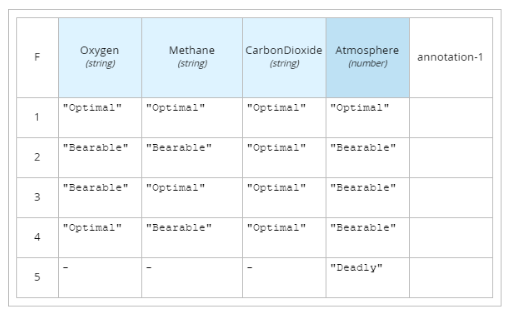


Figure 7: Atmosphere Table

We then apply the same pattern to *Pressure, Temperature* and *Gravity* to create the *Surface* Decision:

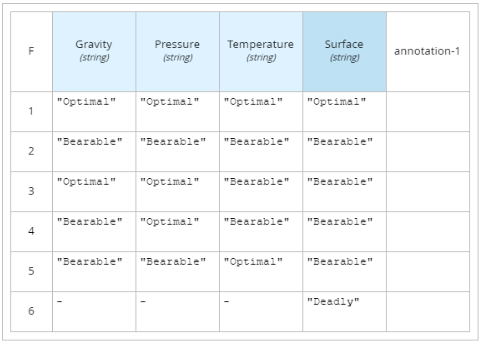


Figure 8: Surface Table

We now join both *Atmosphere* and *Surface* tables to *habitability* which produces the final decision:

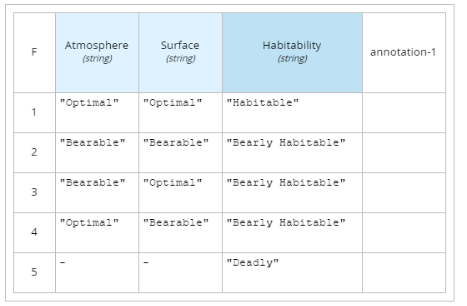


Figure 9: Habitability

All the tables are linked in the DMN diagram as follows:



Figure 10: Divide and Conquer Pattern for Habitability

The **habitability** table has the same behavior as the original DMN design, except it is easier to maintain. Each planet attribute has its own table making it easier to add new attributes and change behavior as there is no dependency on a single table.

The last problem we must address is that all this logic is still inside a single DMN file which means the decision tables cannot be reused or edited by separate people.

## The Tiered Service Pattern

Although the DMN defined in the previous section is maintainable, there is still a design problem. All decision tables are defined within a single DMN file. If additional decision tables were added it would quickly become complex. A single DMN file also prevents decision table re-use and prevents multiple users from making changes at the same time. To improve this, we move the second-tier decisions into their own decision service file and then invoke these decisions from the top tier decision. See figure below, where the second-tier decision services are *AtmosphereDS* and *SurfaceDS* respectively, and the first-tier decision is *Habitability*:

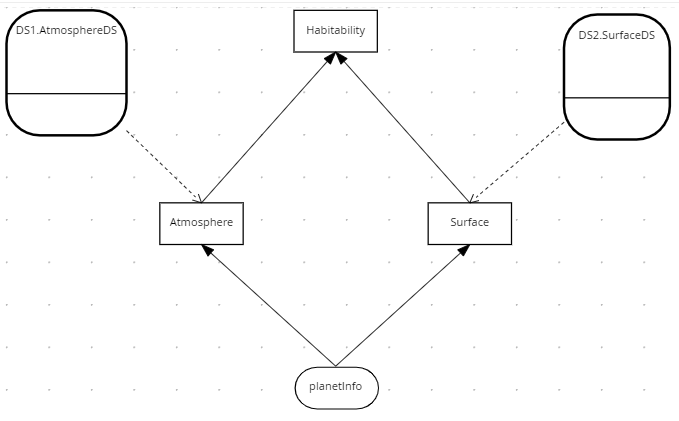


Figure 11: Tiered Service Pattern

The two second tier decision services are *AtmosphereDS* and *SurfaceDS*:

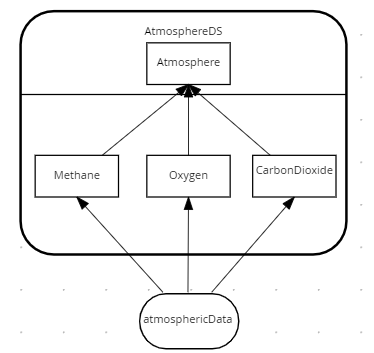


Figure 12: Atmosphere Decision Service

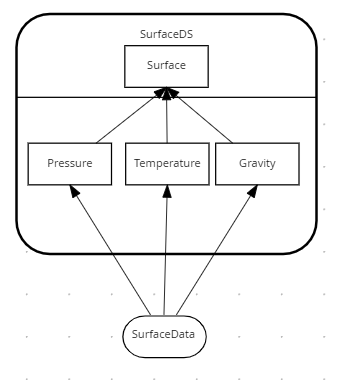


Figure 13: Surface Decision Service

In this implementation we have only two tiers – but in practice the pattern could be applied to an almost unlimited set of tiers, with second tier decision services calling third tier services, and so on. It really depends on the project complexity.

## Lab 5 – Index Card Pattern

Sometimes the Divide and Conquer Pattern described above does not work. This manifests itself on long thin decision tables with few conditions and many rules. An example of this is SWIFT bank interchange. A simplified model of a SWIFT message is as follows:

**BIC** – A Bank Identifier Code

**Receiving Branch** – The bank branch

**Message type** – The type of payment

**Route** – The route the message is sent

Routing rules exist to decide where to route a SWIFT message within the secure international banking network. The routes regularly change due to geopolitical and economic decisions, H

Hence the need to routing tables as rules rather than static data.

A SWIFT routing table in DMN would look like this.

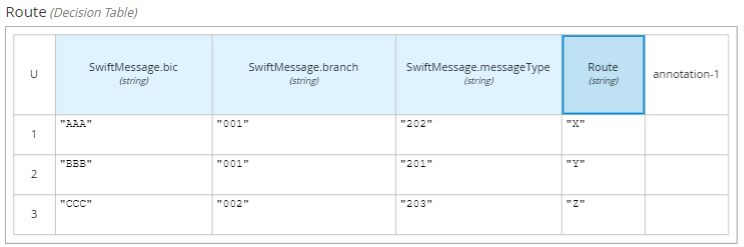


Figure 14: Tall thin table

Only three rows were shown, but in practice it would have thousands of rows based on variates: *Bic, Branch* and *Message Type*.

So how do we scale this tall thin table in a managed way?

One approach is to step back and consider whether this is a decision table at all. Are these ‘rules’ really a reference table? In which case the reference data would be better stored in a database.

You may still wish to use rules for validation, in which case you would have a hybrid solution using BPMN and DMN. This solution would look like this:

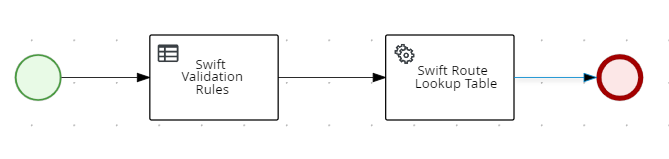


Figure 15: Hybrid BPMN/DMN solution calling rules and a database

If you want to use rules, maybe because a database query would be too complex or you want to govern changes to the routing, then you can apply the **Index Card Pattern**. This pattern splits tall decision tables the same way old-fashioned indexing cards alphabetically index information



Figure 16: Index Cards

The DMN this is modelled like this:

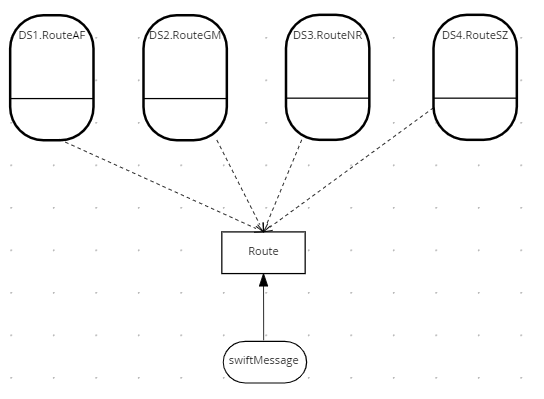


Figure 17: DMN to split Tall thin table

The key logic is in the *Route* decision. It determines the correct decision service to invoke before calling it. This is four times faster than invoking a single table with all the rules:

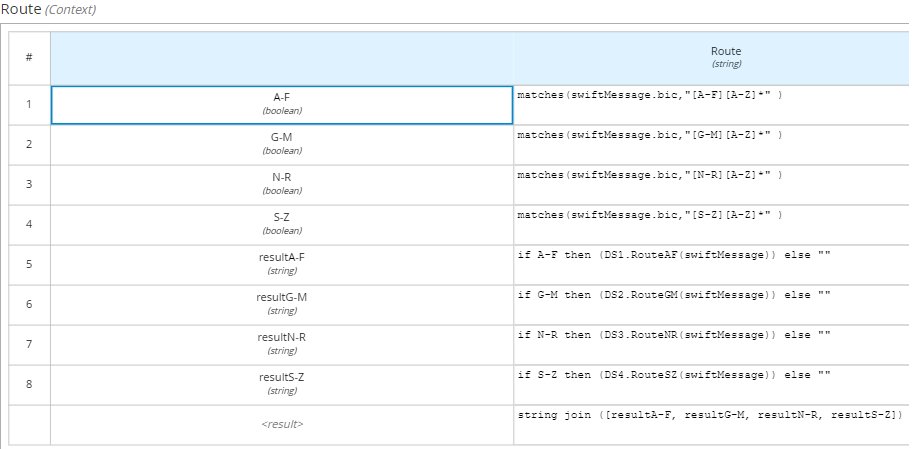


Figure 18: FEEL to invoke alphabetically indexed decision services

The decision service for routes A-F is shown below. The other decision services are not shown as the follow the same pattern.

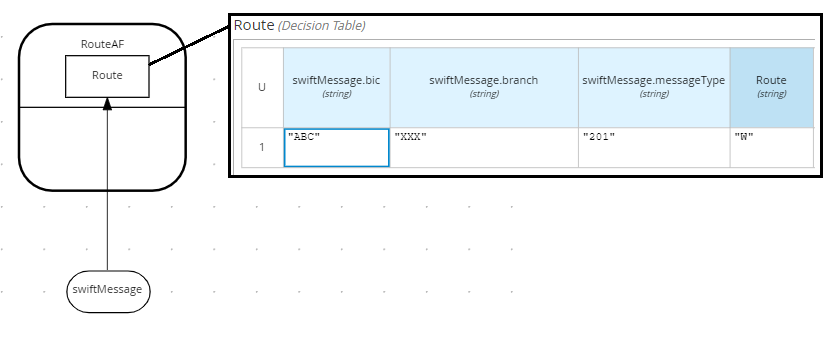


Figure 19: Routing rules for Swift Bic starting A-F

It is worth noting that this pattern can be nested to any depth so that you can reduce the number of rows to 50 per table. For example, if there were 1000 routing rules, the index could have the following structure:

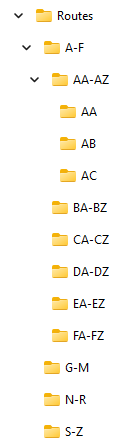


Figure 20: Routing folder structure

## Block pattern

The purpose of the Block Pattern is to avoid rule duplication and reduce the number of decision services.



Figure 1: Block Pattern

## Advantages

The Rule Reuse Pattern enables business users to orchestrate business logic through decision tables. This is easier than creating new paths in DMN diagrams.

## Disadvantages

* This pattern can result in slower execution.
* Additional logic is required to control execution which can complicate the implementation.

## Pattern Example

Consider the example below which determines the car servicing levels: Gold, Silver, Bronze:

## Conclusion

In this lab we presented three design patterns to aid maintainability of large DMN projects. These were:

* The **Structured Data Pattern** consolidates DMN Input Data
* The **Divide and Conquer Pattern** divides a single wide table into smaller thin tables
* The **Tiered Service Pattern** divides a single DMN into smaller DMN Files.
* The **Directory Pattern** divides a single tall table into shorter indexed tables

Consider applying these patterns in your large projects and reap the benefits of more maintainable decision logic!

# Labs

In this section we will implement the patterns described in section 2. The lab material is provided in GIT, all labs except the last can be performed in KIE Sandbox.

## Lab 0 - The Anti-pattern

In this task, you will create a decision service that takes planetary data and decides habitability.

This example will be improved in subsequent labs until we reach a good design.

1. In your Chrome or Safari 16 (or later) browser, open the web site <https://sandbox.kie.org/> :  
     
   Graphical user interface, application, website

   Description automatically generated

Figure 1: Kie Sandbox Start Page

1. Click on the **New Decision** button on the home screen. In the top left corner rename the decision service from Untitled to **Habitability**. Hit the **Enter** button so that the DMN Model Name property is set.
2. Create the following inputs of type **number**: *SurfacePressure, carbonDioxide, methane, oxygen, temperature* and *gravity*:

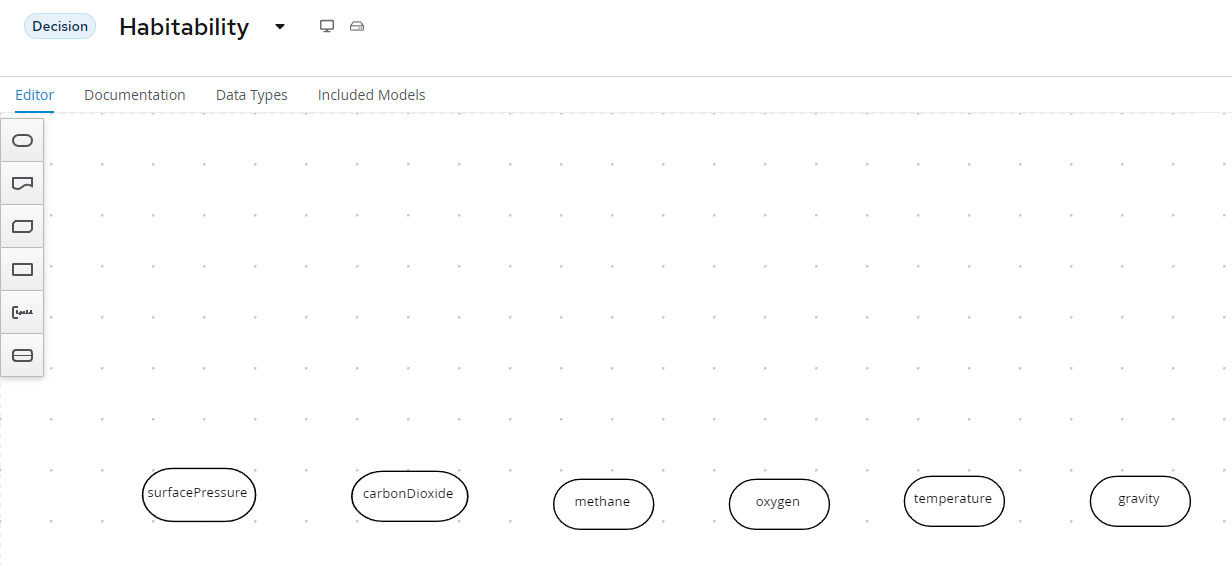


Figure 2: Habitability Decision Inputs

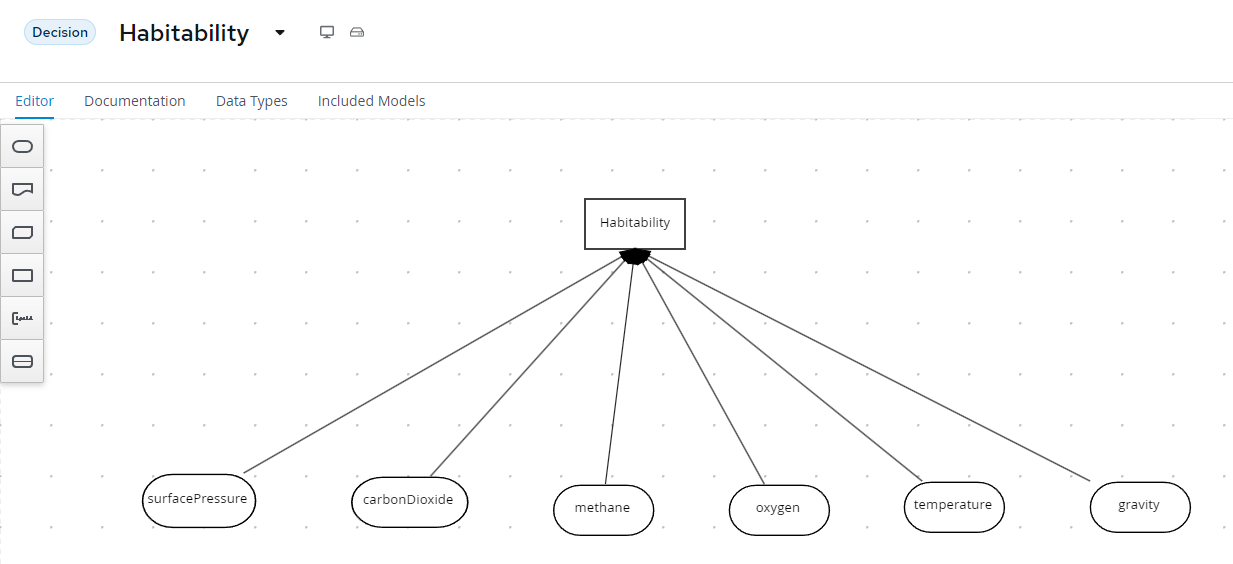
Create Decision *Habitability* and join the inputs to this node:

Figure 3: Initial Habitability DMN Design

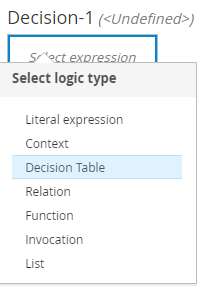
1. Edit **Habitability**. Click on **Select expression** to see a list of available logic types.  
   

Figure 4: Logic Type

1. Select **Decision Table** and enter the following data:

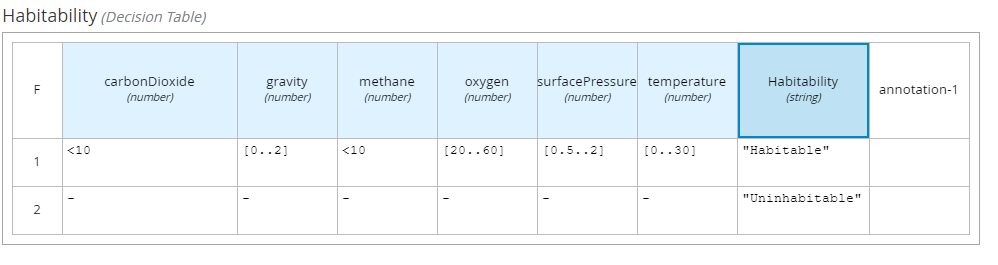


Figure 5: Habitability Decision

1. Test the model with Earth data and the expected result is *Habitable :*

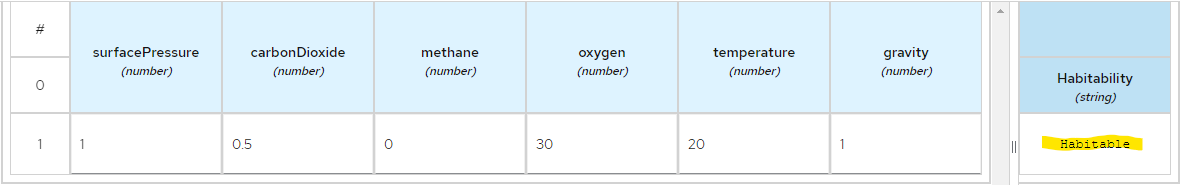


Figure 6: Test Result - Habitable

1. Test the model with Venus data and the expected result is *Uninhabitable :*

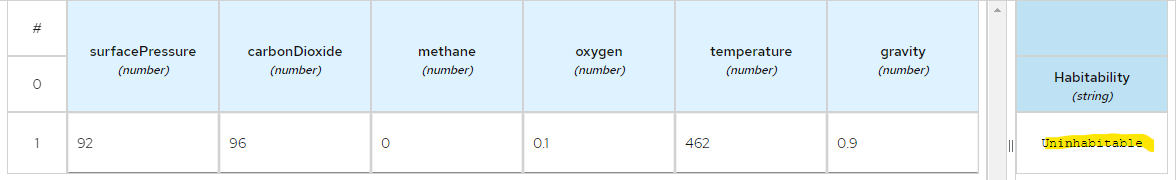


Figure 7: Test Result - Uninhabitable

## Lab 1 –Structured Data Pattern

This can be modelled in **Kie Sandbox** data types:

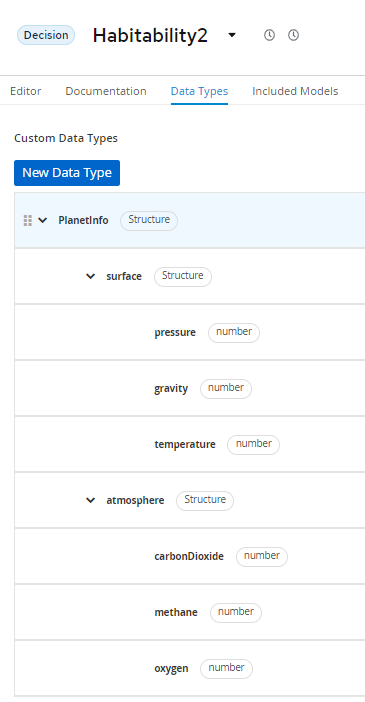


Figure 8: PlanetInfo Data Type

Now we can define just one DMN input called *planet* of type *PlanetInfo*:

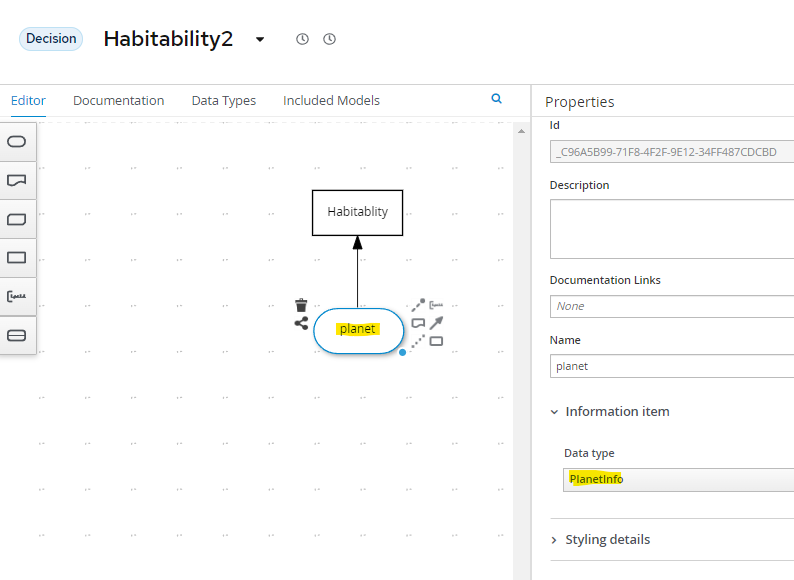


Figure 9: Planet DMN Input defined as a PlanetInfo Data Type

The decision table is the same as the previous lab, but it now references the new data structure:

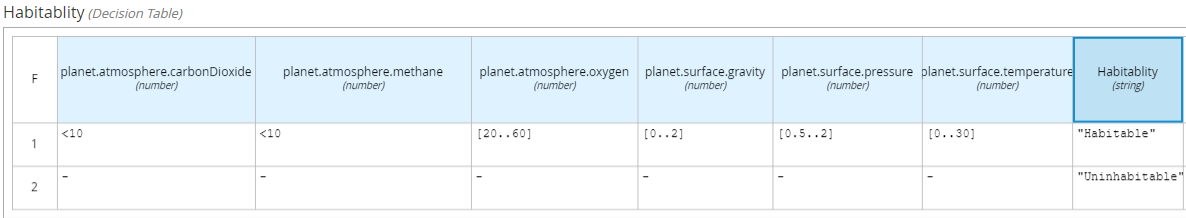


Figure 10: Decision referencing PlanetInfo data type

## Lab 2 – The Divide and Conquer Pattern

To reduce the size of a decision table with many attributes, you can divide and conquer it into smaller parts.

To do this, create a new decision table for each planet attribute. The purpose of these single attribute tables is to reduce the many values of each attribute into a restricted range of enumerated string types. These enumerated types are then applied in aggregated decisions further up in the DMN model. In our example we are reducing all numeric ranges into three enumerated types: *Optimal, Bearable and Deadly.*



Figure 11: Pressure Table

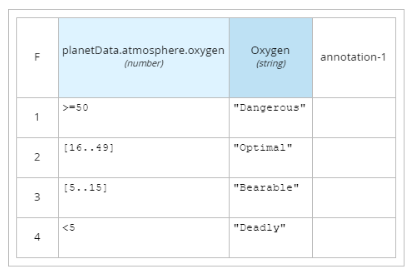


Figure 12: Oxygen Table

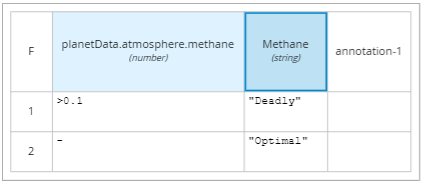


Figure 13: Methane Table

We now combine the decision results of these three tables to define the *atmosphere* decision:

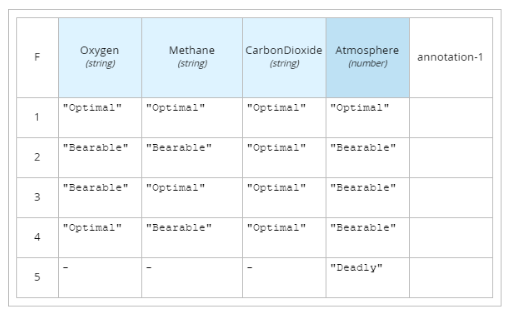


Figure 14: Atmosphere Table

We can apply the same summary pattern to *Pressure, Temperature* and *Gravity* to create the *Surface* Decision:

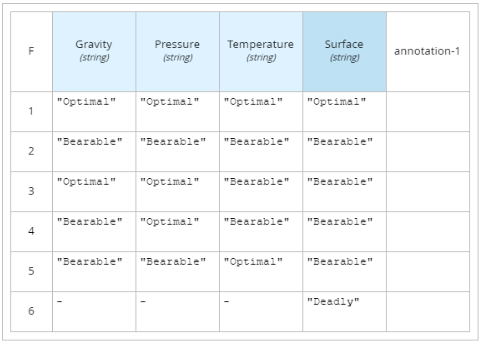


Figure 15: Surface Table

We now join both *Atmosphere* and *Surface* tables to *habitability* which produces the overall decision:

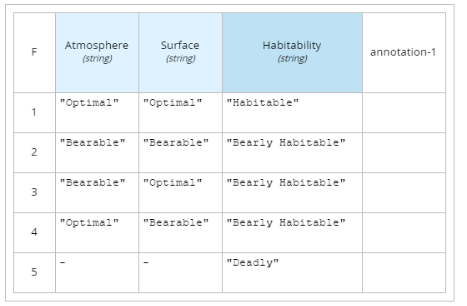


Figure 16: Habitability

All the tables are linked in the DMN diagram as follows:



Figure 17: Divide and Conquer Pattern for Habitability

The **habitability** table has the same behavior as the original DMN design, except now it is easier to maintain. Each planet attribute has its own table making it easier to add attributes and change behavior as there is no dependency on a single table. The last problem we must address is that all this logic is still inside a single DMN file which means the decision tables cannot be edited by separate people and reused in other decisions.

## Lab 3 – The Tiered Service Pattern

Although the DMN defined in the previous section is more maintainable, there is still a design problem. All decision tables are defined within a single DMN file. If additional decision tables and branches were added it would quickly become difficult to maintain. A single DMN file also prevents multiple users from making changes at the same time. To improve on this, we move the second-tier decisions into their own decision service file and then invoke these decisions from the top tier decision service. See figure below, where the second-tier decision services are *AtmosphereDS* and *SurfaceDS* respectively, and the first-tier decision is *Habitability*:

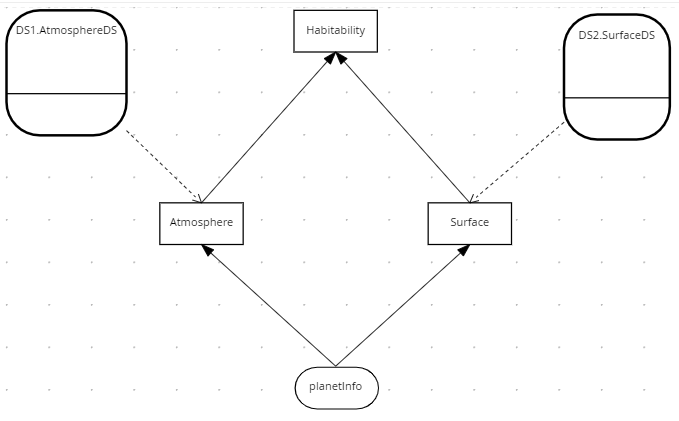


Figure 18: Tiered Service Pattern

The two second tier decision services are *AtmosphereDS* and *SurfaceDS*:

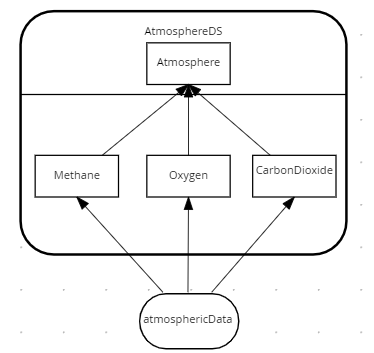


Figure 19: Atmosphere Decision Service

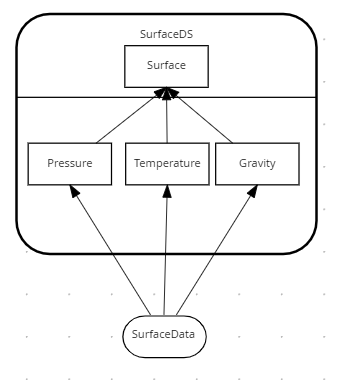


Figure 20: Surface Decision Service

In the Habitability example we have only two tiers – but in practice the pattern could be applied to multiple tiers, with second tier decision services calling third tier services, and so on. It really depends on your project complexity.

## Lab 5 – Directory Pattern

Sometimes the Divide and Conquer Pattern described above does not work. This manifests itself on long thin decision tables with few conditions and many rules. An example of this is SWIFT bank interchange. A simplified model of a SWIFT message is as follows:

**BIC** – A Bank Identifier Code

**Receiving Branch** – The bank branch

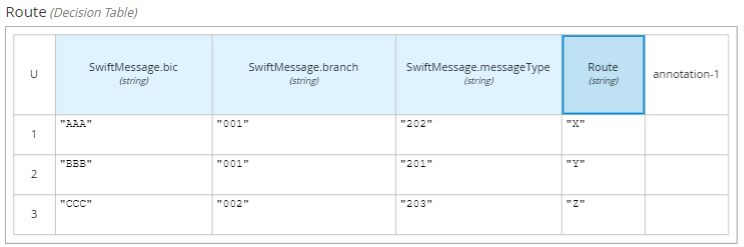
**Message type** – The type of payment

**Route** – The route the message is sent

Routing rules exist to decide where to route a SWIFT message within the secure international banking network. The routes regularly change due to geopolitical and economic decisions, H

Hence the need to routing tables as rules rather than static data.

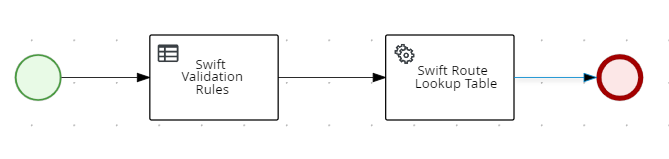
A routing DMN decision table would look like this:



There could be thousands routing rules, based on variates of just these three attributes: *Bic, Branch* and *Message Type*.

So how do we scale this table in a managed way?

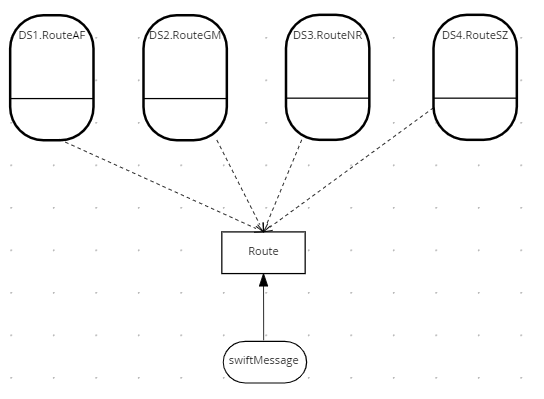
One approach is to step back and consider whether this is really a decision table. Are these ‘rules’ just a reference table? In which case the reference data would be better placed in a database. If rules are required for validation before the database lookup, you could have a hybrid solution using both BPMN and DMN. This solution would look like this:



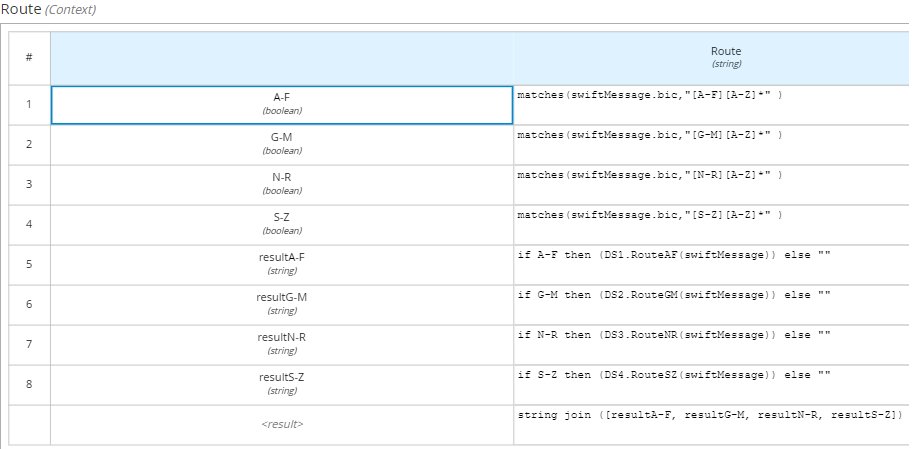
If your rules are definitely rules, maybe because a database query would be too complex or you want to govern changes to the routing, then you can apply the **Directory Pattern**. This pattern splits decision tables the same way an old-fashioned directory alphabetically indexes names and addresses:



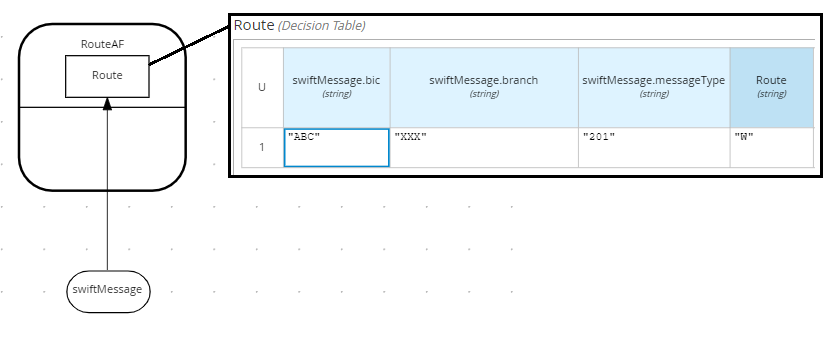
In DMN this is modelled like this:



The key logic is in the *Route* decision. It is complex as it uses regex to determine the correct decision service and then invokes the right one. Note that only the decision service is invoked, not all services. In this case it would be on average four times faster than having a single table with all the rules. Here is the logic:



The decision service for routes A-F is shown below. The other decision services are not shown as the follow the same pattern.



## Lab 6 - Generating DMN from Excel

Many businesses love modelling decisions in Excel. Each worksheet in excel could equate to one or more decision tables. Sometimes the business holds on to their spreadsheet. This can cause maintenance issues as both the spreadsheet and the DMN tables require updating.

There is a solution! The underlying format of DMN is XML and this can be generated. In the following lab we will generate the decision table using a generator that reads Excel data (csv format) and updates the Oxygen table.

## Conclusion

In this lab we presented three design patterns to aid maintainability of large DMN projects. These were:

* The **Structured Data Pattern** to reduce the amount of DMN Input Data
* The **Divide and Conquer Pattern** to divide a single table into smaller tables
* The **Tiered Service Pattern** to divide a single DMN file into several smaller DMN Files.

Consider applying these patterns in your projects and reap the benefits of more maintainable decision logic!