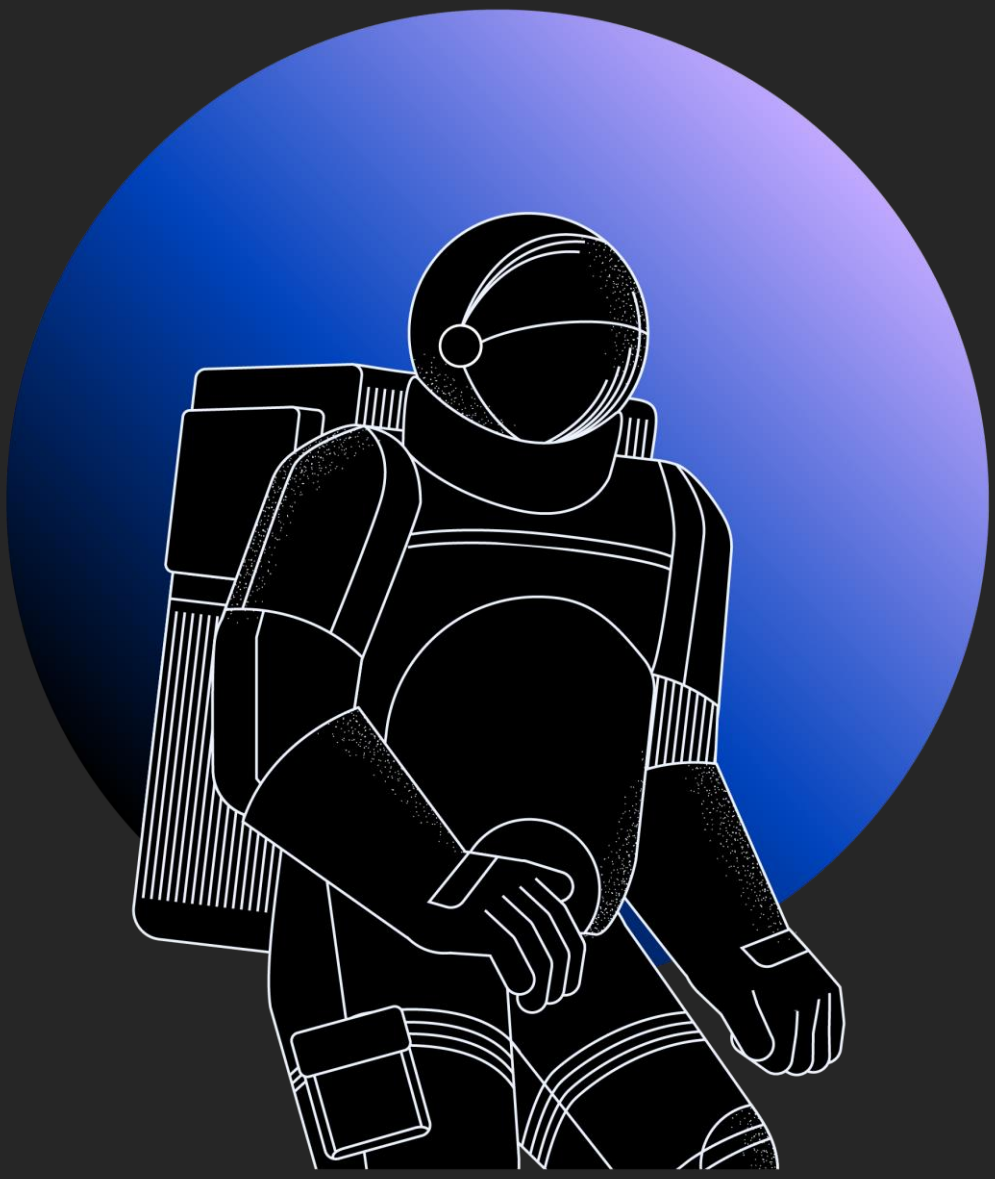
Lab Guide

IBM Decision Manger Open Edition

Nigel Crowther – ncrowther@uk.ibm.com

Hands-on Guide

Scaling Large DMN Projects



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.

Table of Contents

Introduction

The guide presents patterns to scale DMN. These patterns are based on real large rule projects in finance.

1 Design Patterns

1.1 The Divide and Conquer Pattern

1.1.1 Pattern Definition

The Divide and Conquer Pattern[1] divides a single decision table into smaller tables to produce overall result. It does this by splitting one long decision table with many columns into smaller constituent tables. The tables are arranged into a tree, with sibling decisions contributing to the input of its parent. The root is a combination of all decisions from its children and grandchildren. See below.

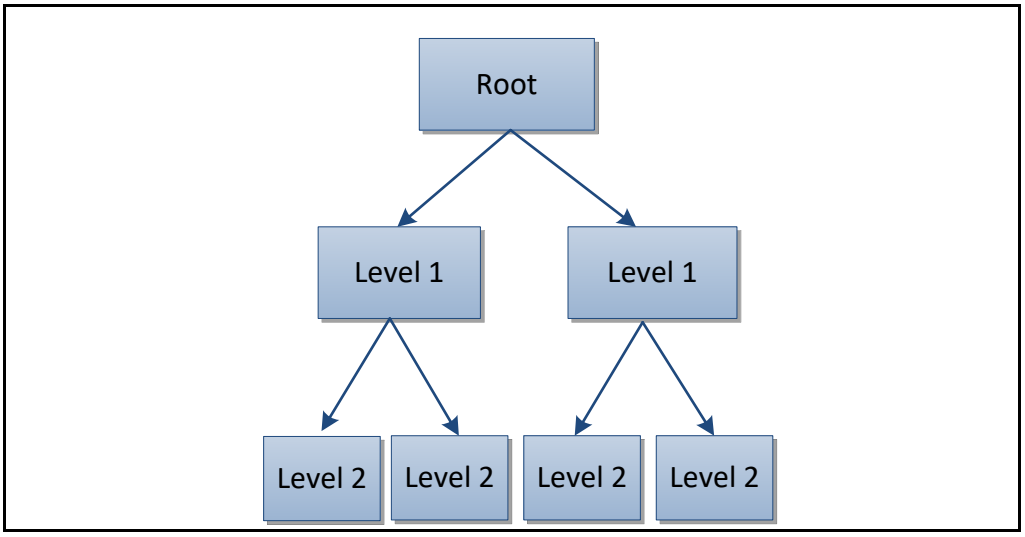


Figure 1: Divide and Conquer

1.1.2 Pattern Advantages

The pattern reduces the size of wide tables, encourages condition reuse, and simplifies business logic.

1.1.3 Pattern Disadvantages

Excessive use of this pattern creates a proliferation of intermediate tables and may affect performance.

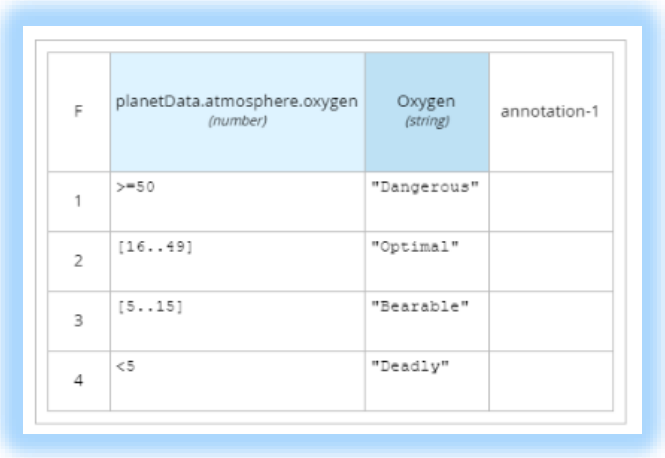
1.1.4 Pattern Example

For this pattern we introduce an example service that determines planet habitability based on planetary data such as atmosphere and climate.

---

[1] See “Real-World Decision Modeling with DMN” by James Taylor and Jan Purchase.

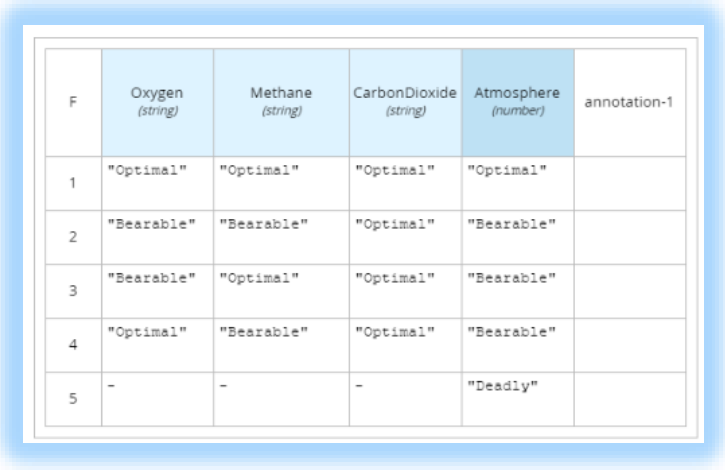
We start by mapping planetary data such as oxygen levels to enumerated types: *Optimal, Bearable and Deadly.* For brevity only the *Oxygen* table is shown below. Tables for Methane and Carbon Dioxide follow the same pattern.



F	planetData.atmosphere.oxygen *(number)*	Oxygen *(string)*	annotation-1
1	>=50	"Dangerous"	
2	[16..49]	"Optimal"	
3	[5..15]	"Bearable"	
4	<5	"Deadly"	

Figure 2: Oxygen Table

Next, the decision outputs of the Oxygen, Methane and Carbon dioxide tables combine to define the *atmosphere* decision:



F	Oxygen *(string)*	Methane *(string)*	CarbonDioxide *(string)*	Atmosphere *(number)*	annotation-1
1	"Optimal"	"Optimal"	"Optimal"	"Optimal"	
2	"Bearable"	"Bearable"	"Optimal"	"Bearable"	
3	"Bearable"	"Optimal"	"Optimal"	"Bearable"	
4	"Optimal"	"Bearable"	"Optimal"	"Bearable"	
5	-	-	-	"Deadly"	

Figure 3: Atmosphere Table

The same pattern is applied to Gravity, *Pressure* and *Temperature* data to create the *Surface* Table:

F	Gravity *(string)*	Pressure *(string)*	Temperature *(string)*	Surface *(string)*	annotation-1
1	"Optimal"	"Optimal"	"Optimal"	"Optimal"	
2	"Bearable"	"Bearable"	"Bearable"	"Bearable"	
3	"Optimal"	"Optimal"	"Bearable"	"Bearable"	
4	"Bearable"	"Optimal"	"Bearable"	"Bearable"	
5	"Bearable"	"Bearable"	"Optimal"	"Bearable"	
6	-	-	-	"Deadly"	

Figure 4: Surface Table

Finally, *Atmosphere* and *Surface* decisions are joined in the *Habitability* table *to* produce the final decision:

F	Atmosphere *(string)*	Surface *(string)*	Habitability *(string)*	annotation-1
1	"Optimal"	"Optimal"	"Habitable"	
2	"Bearable"	"Bearable"	"Bearly Habitable"	
3	"Bearable"	"Optimal"	"Bearly Habitable"	
4	"Optimal"	"Bearable"	"Bearly Habitable"	
5	-	-	"Deadly"	

Figure 5: Habitability

The decision tables are linked as shown in the DMN diagram below:

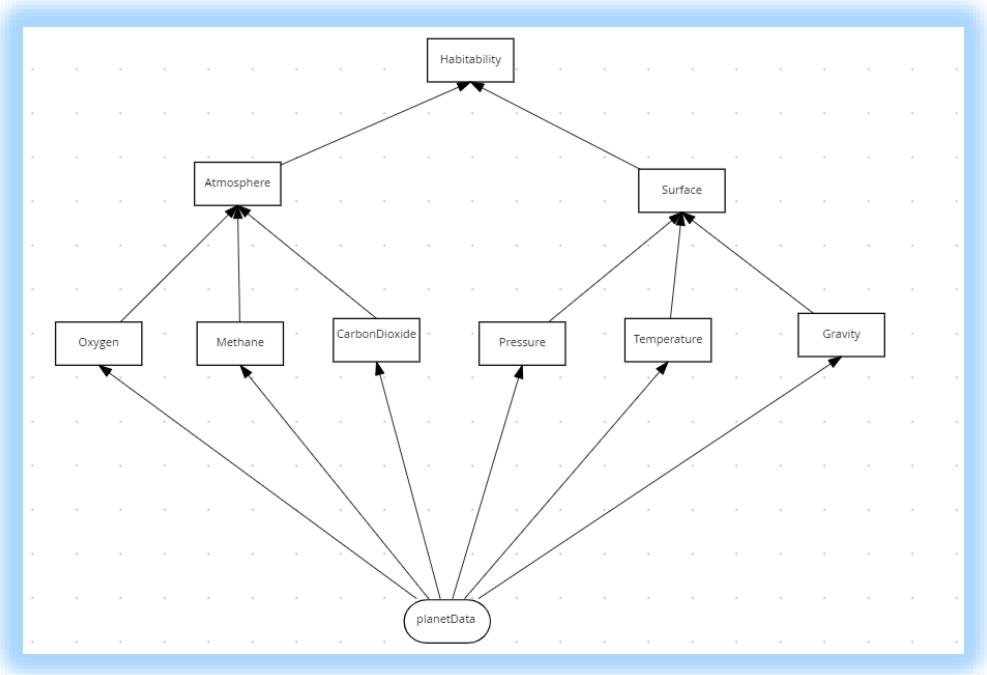


Figure 6: Divide and Conquer DMN Diagram for Habitability

The **habitability** table has the same behavior as the original design, except it will now scale when adding further decisions.

All this logic is still inside a single DRG (Decision Requirements Graph) which means the decision tables cannot be reused or edited at the same time. This is where the Tiered Decision Service Pattern is useful.

1.1.5 Lab

See xxx for lab.

1.2 The Tiered Decision Service Pattern

1.2.1 Pattern Definition

The Tiered Service Pattern splits a single DMN into smaller decision services which combine the result. See below.

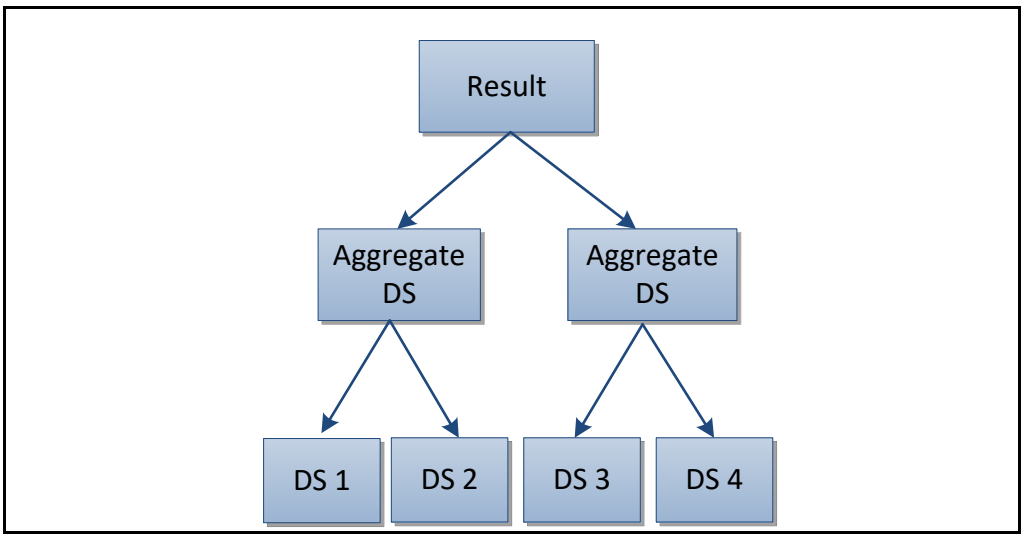


Figure 7: Tiered Decision Service

1.2.2 Pattern Advantages

This pattern splits one large DRG into tiers of smaller DRGs. It encourages decision reuse and simplifies design.

1.2.3 Pattern Disadvantages

Excessive use of this pattern creates a proliferation of DRGs that may affect performance.

1.2.4 Pattern Example

Although the DMN defined in the Divide and Conquer Pattern is maintainable, there is still a design problem. All decision tables are defined within a single DRG. Additional decision tables would quickly make the DRG complex. Having all tables in a single DRG prevents decision re-use and prevents multiple users from making changes at the same time. To improve this, we move secondary decisions into their own decision service and then invoke these decisions from the top tier. See figure below, where the second-tier decision services are *AtmosphereDS* and *SurfaceDS* respectively, and the first-tier decision is *Habitability*:

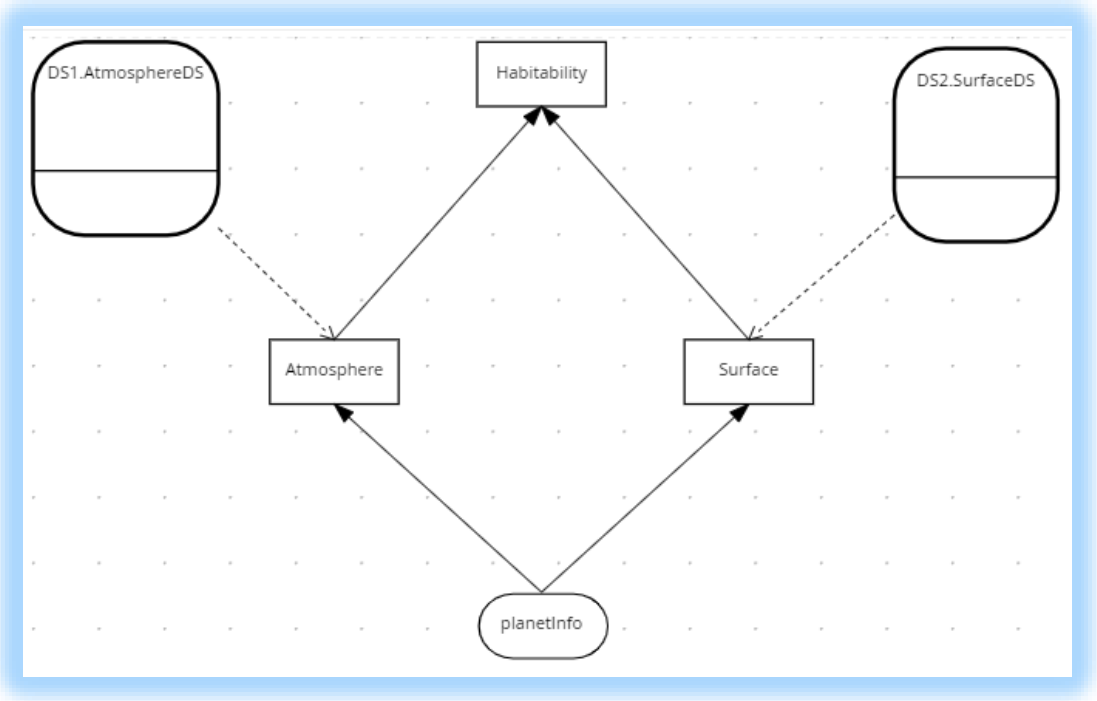


Figure 8: Tiered Decision Service Pattern

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

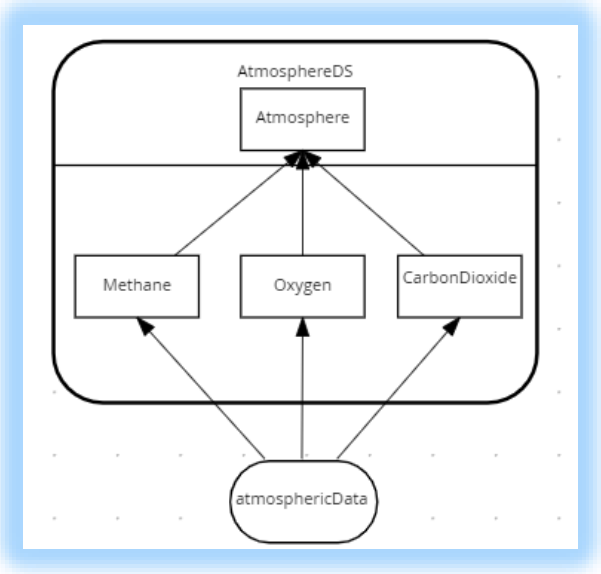


Figure 9: Atmosphere Decision Service

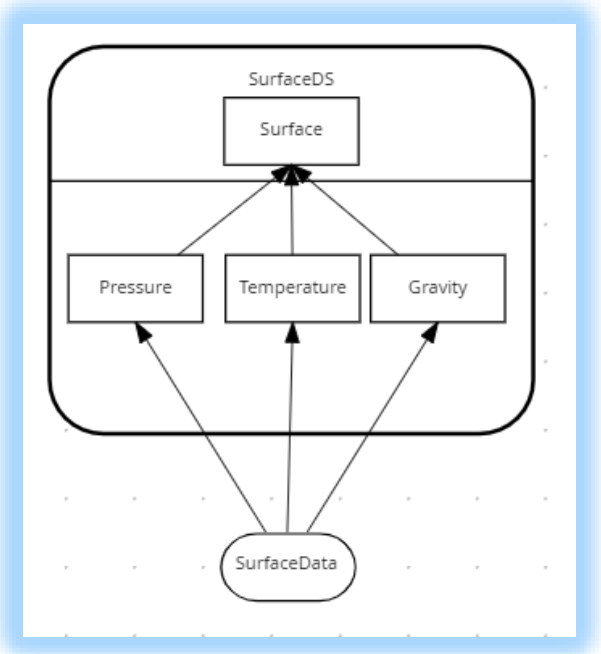


Figure 10: Surface Decision Service

In this implementation we have two tiers – but in practice the pattern could be applied to multiple tiers, with each tier calling further tiers. The number of tiers depends on project complexity.

1.2.5 Lab

See xxx for lab.

1.3 The Index Pattern

1.3.1 Pattern Definition

The Divide and Conquer Pattern does not work for long thin decision tables with few conditions and many rules. Instead, the Index Pattern should be considered. This pattern splits tall thin decision tables using an index:

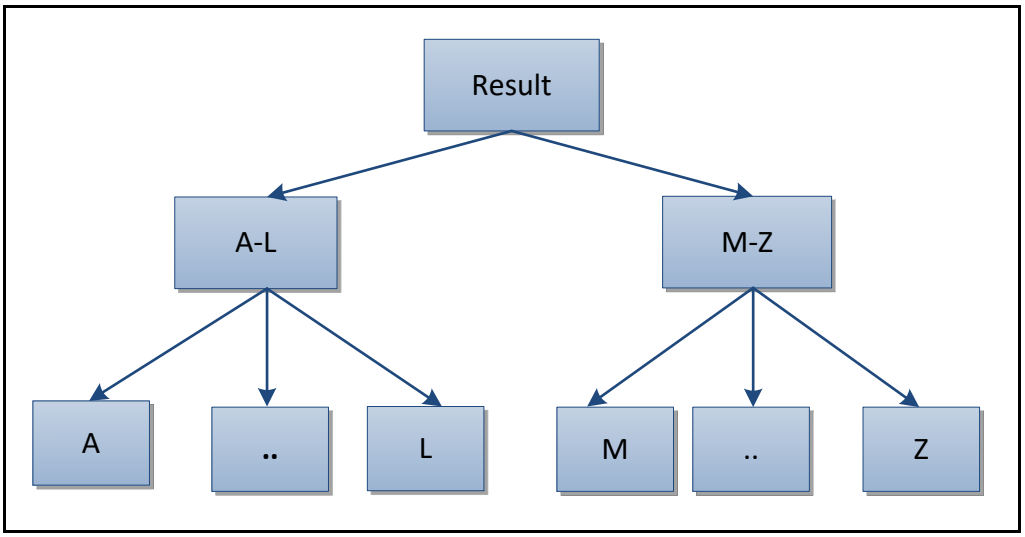


Figure 11: Index Pattern

1.3.2 Pattern Advantages

This pattern splits one large decision table into smaller decision tables using a single indexed attribute. It encourages decision reuse and simplifies DMN diagrams.

1.3.3 Pattern Disadvantages

The index card may be too complex for small DMN projects.

1.3.4 Pattern Example

This example uses routing rules from SWIFT. SWIFT is the mechanism for international bank transfers. A simplified model of a SWIFT message is as follows:

BIC – Bank Identifier Code
Receiving Branch – The bank branch
Message type – The type of payment
Route – The route the message is sent
Amount - The amount of money to transfer

Routing rules decide the route of the message through the banking network. The routes change due to geopolitical and economic decisions, hence the need for routing tables to be rules.

An example rule would be:

> *If the Swift message has a BIC of **AAA**, a branch of **001** and a message Type of **202***
>
> *Then*
>
> *Route the message to **X***

This rule along with two others can be expressed in the following routing table:

Route *(Decision Table)*

U	SwiftMessage.bic *(string)*	SwiftMessage.branch *(string)*	SwiftMessage.messageType *(string)*	Route *(string)*	annotation-1
1	"AAA"	"001"	"202"	"X"	
2	"BBB"	"001"	"201"	"Y"	
3	"CCC"	"002"	"203"	"Z"	

Figure 12: SWIFT Routing Table

Only three rows were shown, but in practice routing tables contain thousands of rows based on variates: *Bic, Branch* and *Message Type*.

So how do we scale this table? The solution is to split the routing table the same way that a long list of names is split using index cards:

Figure 13: Index Cards

In DMN the index cards are modelled like this:

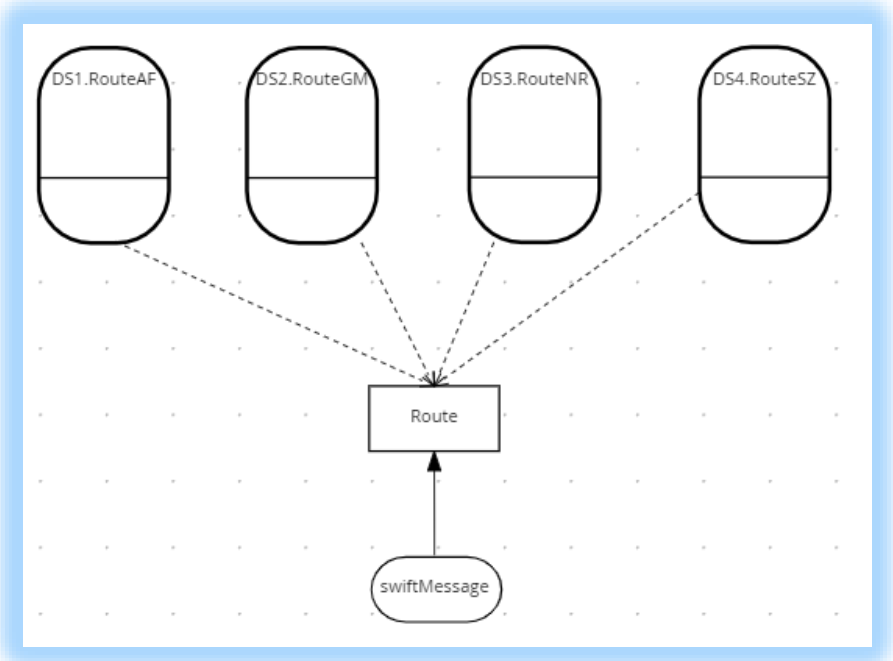


Figure 14: DMN to split Tall thin table

The indexing logic is in the *Route* decision. It determines the decision service to invoke based on the first characters of a key decision attribute. In this example, the key decision attribute is the SWIFT BIC code.

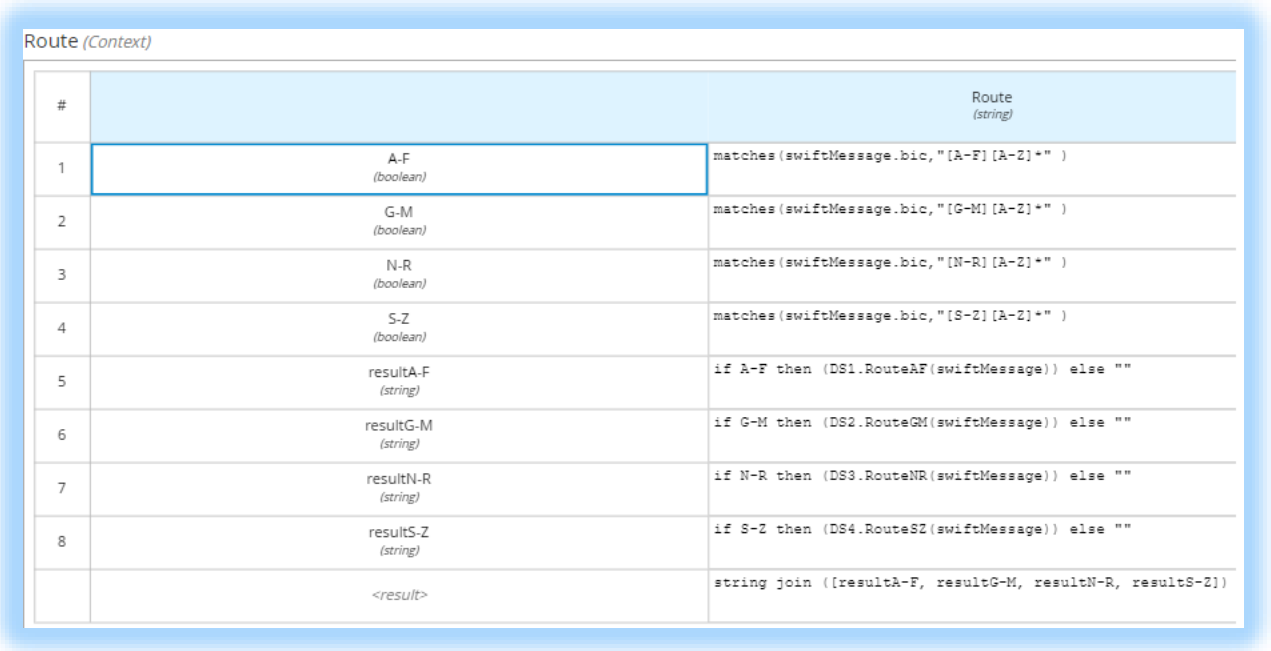


Figure 15: FEEL to invoke alphabetically indexed decision services

The decision service representing a single index card for routes A-F is shown below. Other index cards are not shown as the follow the same pattern.

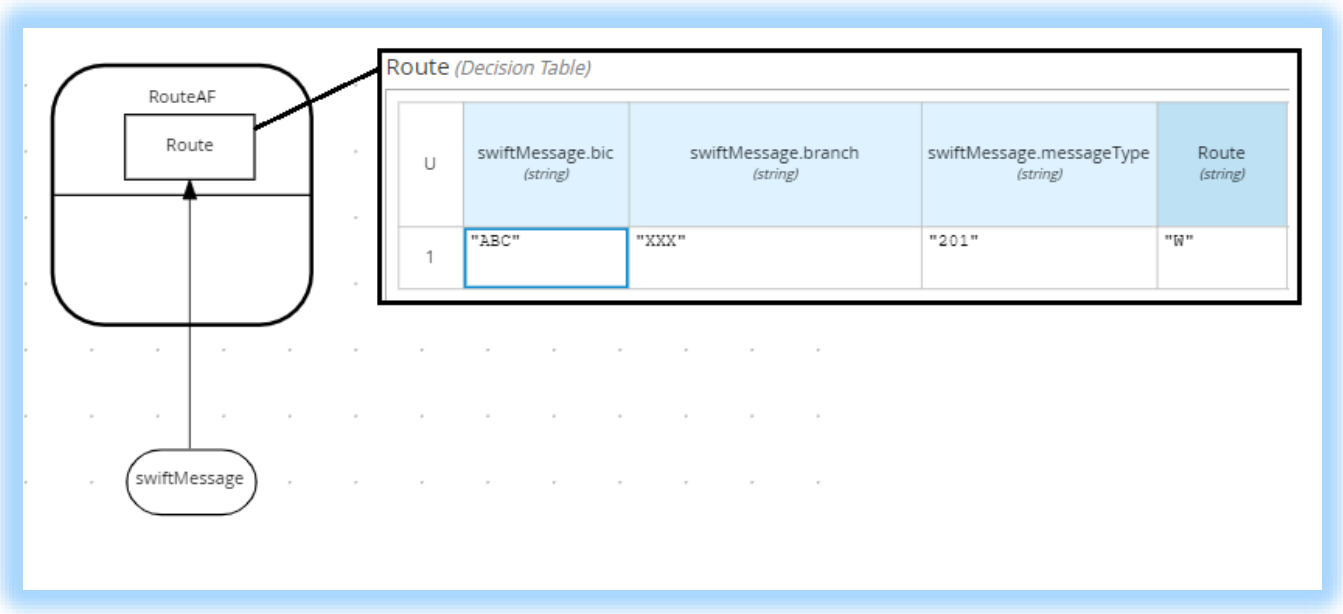


Figure 16: Routing rules for Swift Bic starting A-F

The Index Card Pattern can be arranged hierarchically so that index cards contain child indexes to refine the search. In this way a thousand row table can be separated to a many smaller tables with just a few rows per table. See below.

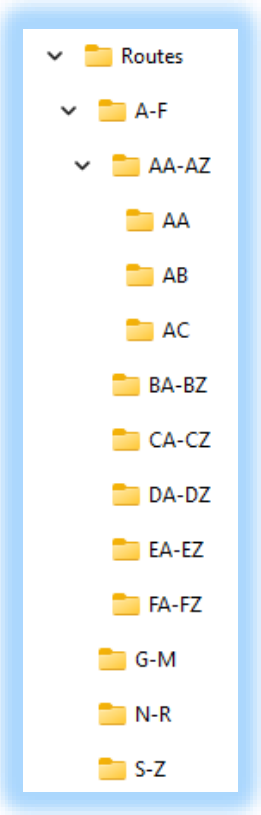


Figure 17: Routing folder structure

1.3.5 Lab

See xxx for lab.

1.4 The Service Level Pattern

1.4.1 Pattern Definition

The *Service Level Pattern* identifies common rule services and permits their reuse. The key construct is the *Service Level* which is associated to one or more *Services*. A *Service* is reused by one or more *Service Levels*. See below:

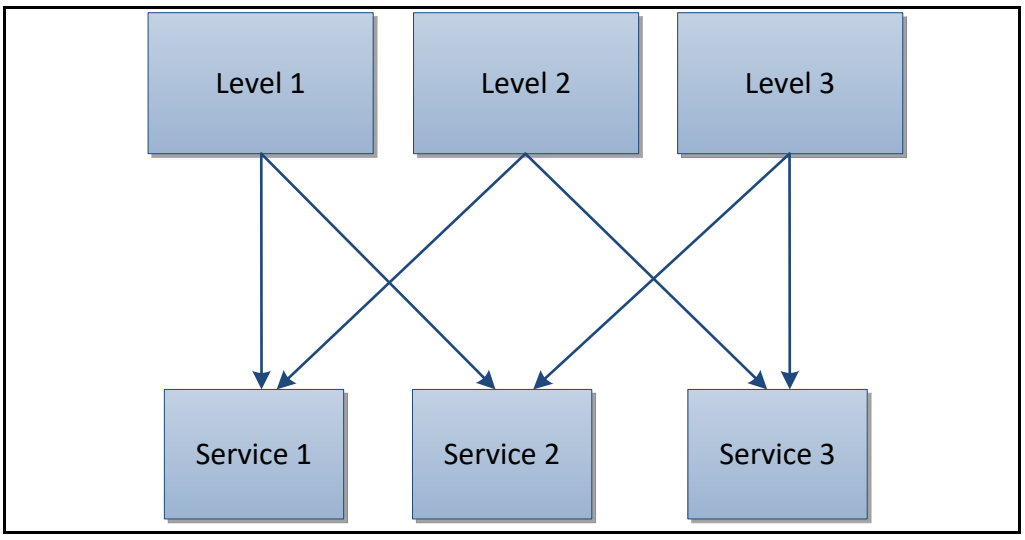


Figure 18: Service Level Pattern

1.4.2 Advantages

The Service Level Pattern places reusable decision logic into a decision service.

1.4.3 Disadvantages

Additional logic is required to control execution of services

1.4.4 Pattern Example

Consider the servicing tasks for a car. There are three service levels: *Gold, Silver, Bronze*. Each level determines the work to be performed. For example, air conditioning is only covered by a *Gold* service level, but an oil change is covered by all service levels.

A bad implementation would be to combine service levels and the services into one decision like this:

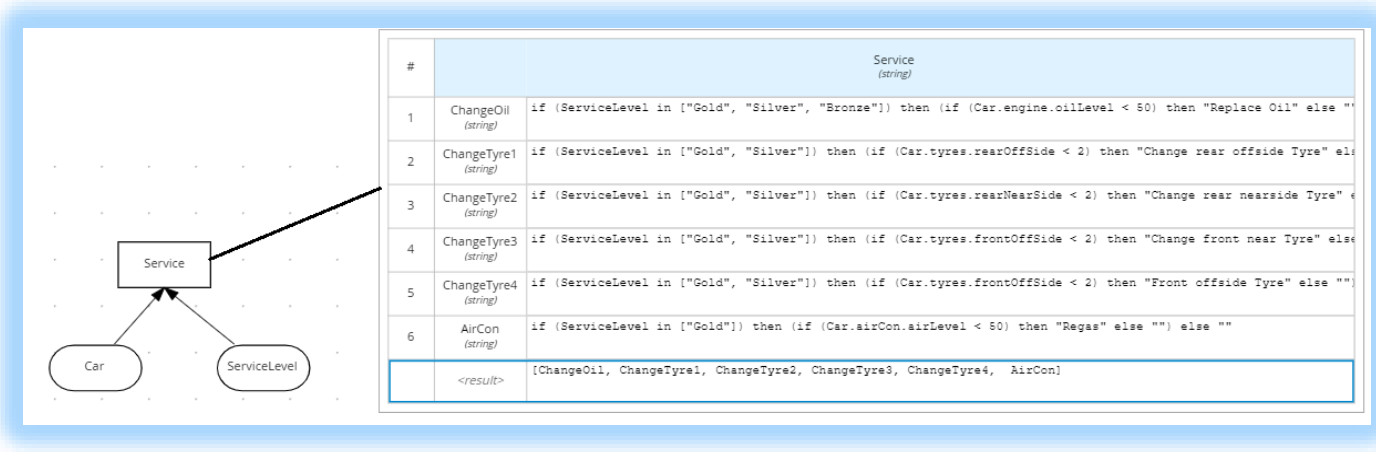


Figure 19: Car Service Anti Pattern

This implementation is difficult to understand and hard to extend. A more scalable solution is to apply the *Service Level Pattern.* In the example below, a service level is mapped to one or more service blocks. Each service block is associated to exactly one decision service.

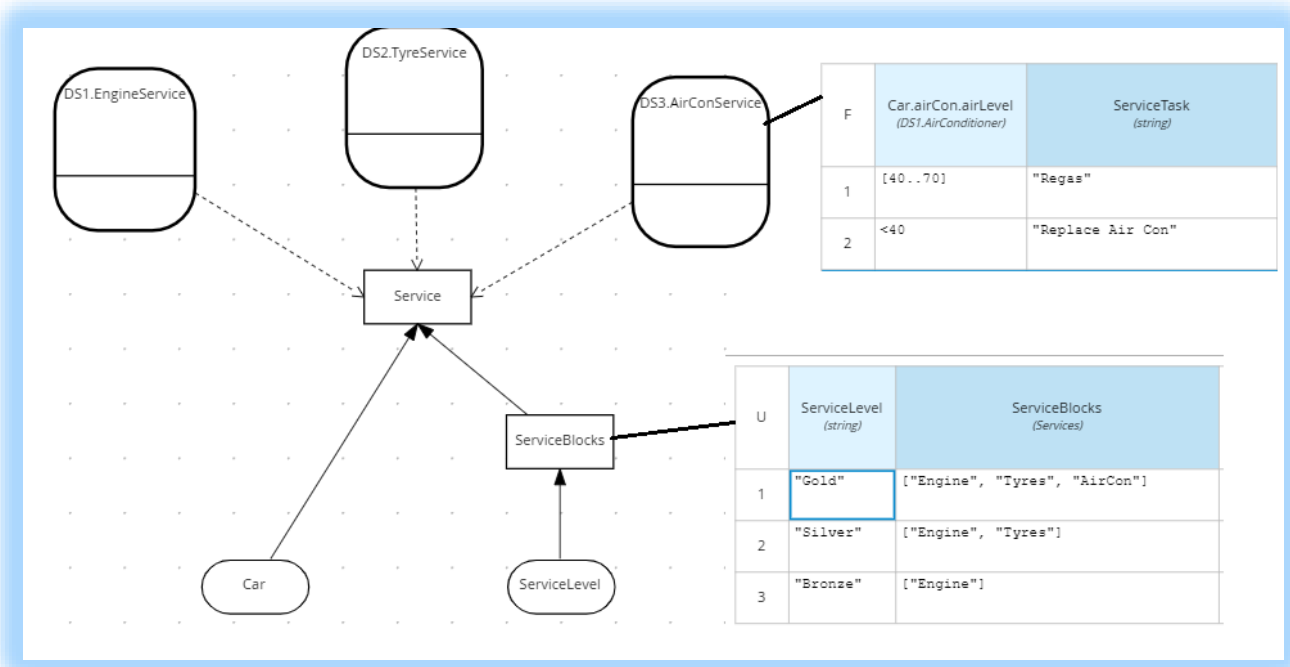


Figure 20: Car Service using Service Level Pattern

The Service Level pattern simplifies DMN for two reasons:

Decoupling. The service level is decoupled from service tasks

Reuse. Decision services could be reused. For example, truck servicing could reuse the Engine and Tyre Service Levels.

1.4.5 Lab

See xxx for lab.

2 Conclusion

In this lab we presented design patterns for building large DMN projects. These were:

- The **Divide and Conquer Pattern** divides a single wide table into smaller thin tables
- The **Tiered Service Pattern** divides a one large DRG into smaller DRGs.
- The **Index Card Pattern** splits a single tall table into shorter indexed tables
- The **Service Level Pattern** reuses decisions and reduce complexity

Consider applying these DMN patterns to help you scale!