**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**

[Introduction 5](#_Toc130063934)

[1 The Divide and Conquer Pattern 6](#_Toc130063935)

[1.1 Pattern Definition 6](#_Toc130063936)

[Figure 1: Divide and Conquer 6](#_Toc130063937)

[1.2 Pattern Advantages 6](#_Toc130063938)

[1.3 Pattern Disadvantages 6](#_Toc130063939)

[1.4 Pattern Example 6](#_Toc130063940)

[Figure 6: Oxygen Table 7](#_Toc130063941)

[Figure 7: Atmosphere Table 7](#_Toc130063942)

[Figure 8: Surface Table 8](#_Toc130063943)

[Figure 9: Habitability 8](#_Toc130063944)

[Figure 10: Divide and Conquer Pattern for Habitability 9](#_Toc130063945)

[2 The Tiered Decision Service Pattern 10](#_Toc130063946)

[2.1 Pattern Definition 10](#_Toc130063947)

[Figure 1: Tiered Decision Service 10](#_Toc130063948)

[2.2 Pattern Advantages 10](#_Toc130063949)

[2.3 Pattern Disadvantages 10](#_Toc130063950)

[2.4 Pattern Example 11](#_Toc130063951)

[Figure 11: Tiered Service Pattern 11](#_Toc130063952)

[Figure 12: Atmosphere Decision Service 12](#_Toc130063953)

[Figure 13: Surface Decision Service 12](#_Toc130063954)

[3 The Index Pattern 13](#_Toc130063955)

[3.1 Pattern Definition 13](#_Toc130063956)

[Figure 11: Index Pattern 13](#_Toc130063957)

[3.2 Pattern Advantages 13](#_Toc130063958)

[3.3 Pattern Disadvantages 13](#_Toc130063959)

[3.4 Pattern Example 14](#_Toc130063960)

[Figure 14: SWIFT Routing Table 14](#_Toc130063961)

[Figure 15: Hybrid BPMN/DMN solution calling rules and a database 15](#_Toc130063962)

[Figure 16: Index Cards 15](#_Toc130063963)

[Figure 17: DMN to split Tall thin table 16](#_Toc130063964)

[Figure 18: FEEL to invoke alphabetically indexed decision services 16](#_Toc130063965)

[Figure 19: Routing rules for Swift Bic starting A-F 17](#_Toc130063966)

[Figure 20: Routing folder structure 17](#_Toc130063967)

[4 The Service Level Pattern 18](#_Toc130063968)

[Figure 1: Service Level Pattern 18](#_Toc130063969)

[4.1 Advantages 18](#_Toc130063970)

[4.2 Disadvantages 18](#_Toc130063971)

[4.3 Pattern Example 19](#_Toc130063972)

[5 Conclusion 21](#_Toc130063973)

# Introduction

In this guide we introduce patterns to help you scale large DMN projects.

The guide offers 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, you will understand how to apply these patterns to build your own large decision projects in DMN.

It is worth noting that the patterns presented may not be appropriate for small projects.

# The Divide and Conquer Pattern

## Pattern Definition

The Divide and Conquer Pattern divides a single decision table into several smaller decisions which combine into the overall result. It does this by splitting one long decision table with many columns into smaller constituent tables. The tables are arranged as a tree, with decisions from siblings contributing to the decision of its parent. The root is a combination of all decisions from the root’s children. See below.



Figure 1: Divide and Conquer

## Pattern Advantages

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

## Pattern Disadvantages

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

## Pattern Example

In our Habitability example, we start by reducing 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.

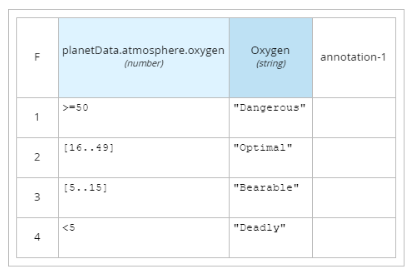


Figure 6: Oxygen Table

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

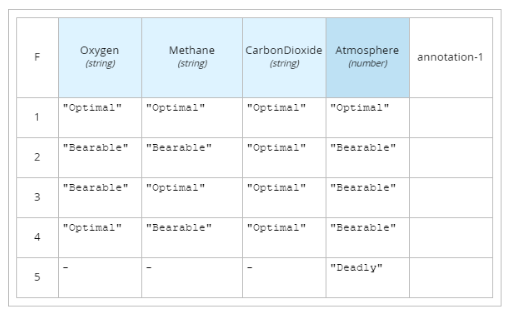


Figure 7: Atmosphere Table

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

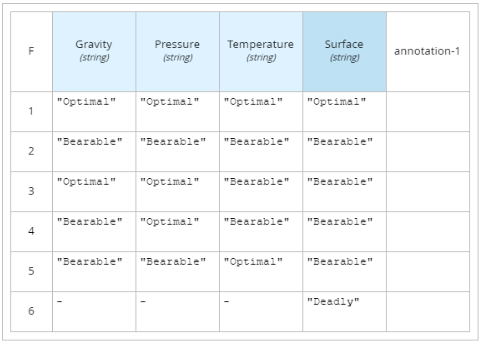


Figure 8: Surface Table

Finally, we join *Atmosphere* and *Surface* to *Habitability* which produces the final decision:

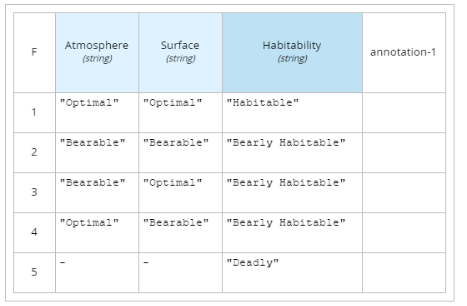


Figure 9: Habitability

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



Figure 10: Divide and Conquer Pattern for Habitability

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

All this logic is still inside a single DMN file which means the decision tables cannot be reused or edited by separate people. This is where the next pattern, the Tiered Decision Service, is useful.

# The Tiered Decision Service Pattern

## Pattern Definition

The Tiered Service Pattern splits one big DMN into smaller decision services which combine the overall result. See below.



Figure 1: Tiered Decision Service

## Pattern Advantages

This pattern splits one large DMN file into a tier of smaller DMN files. It encourages decision service reuse and simplifies DMN diagrams.

## Pattern Disadvantages

Excessive use of this pattern creates a proliferation of DMN diagrams and services and may affect performance.

## 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 DMN file. Additional decision tables would quickly make the DMN complex. Having all tables in 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 second-tier decisions into their own decision service 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*:

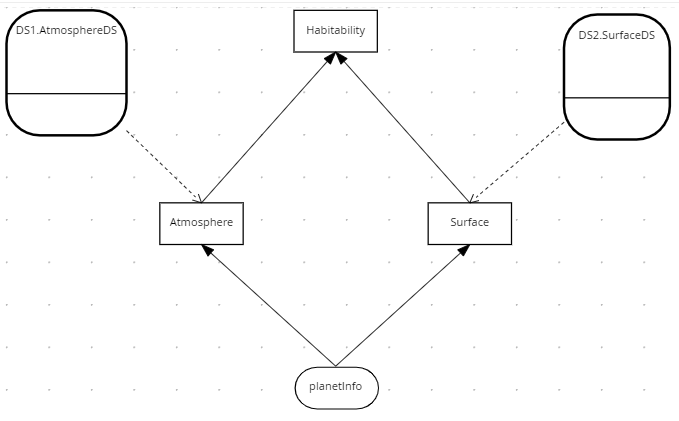


Figure 11: Tiered Service Pattern

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

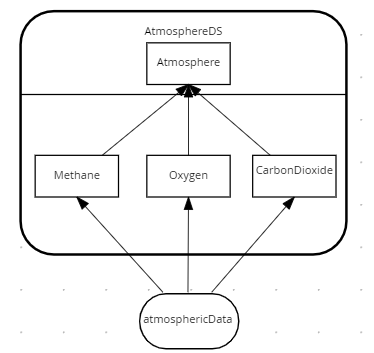


Figure 12: Atmosphere Decision Service

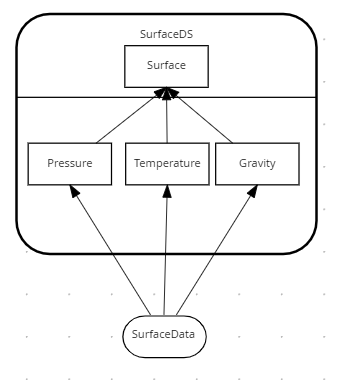


Figure 13: Surface Decision Service

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

# The Index Pattern

## Pattern Definition

The Divide and Conquer Pattern does not work for long thin decision tables with few conditions and many rules. The Index Pattern splits tall thin decision tables into shorter tables using an alphabetical index:



Figure 11: Index Pattern

## Pattern Advantages

This pattern helps split one large DMN file into smaller DMN files where it is impossible to split using the Divide and Conquer pattern. It encourages decision reuse and simplifies DMN diagrams.

## Pattern Disadvantages

The index card may not be appropriate way of organizing some rules and excessive use of this pattern may create a proliferation of tables which could be hard to maintain.

## Pattern Example

This example uses routing rules from SWIFT bank interchange. 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 exist to decide where to route a SWIFT message within the international banking network. The routes regularly 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 SWIFT routing table:

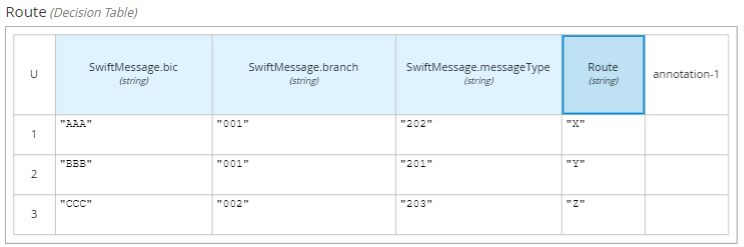


Figure 14: SWIFT Routing Table

Only three rows were shown, but in practice this routing table could have thousands of rows based on variates: *Bic, Branch* and *Message Type*.

So how do we scale this table?

One approach is to step back and consider whether this is a decision table at all. Are the ‘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:

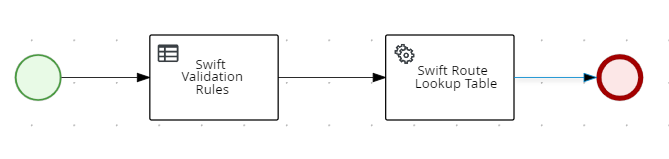


Figure 15: Hybrid BPMN/DMN solution calling rules and a database

If you want to use rules because a database query would be too complex or want to govern changes, 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:

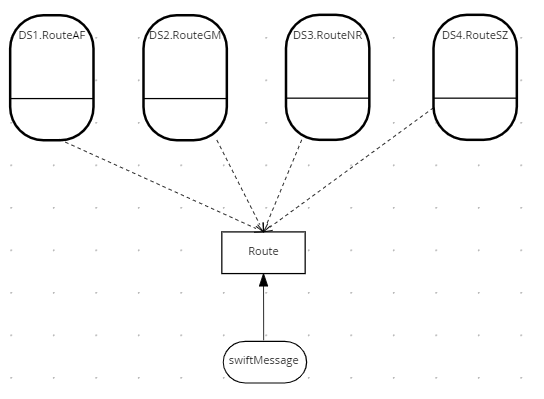


Figure 17: DMN to split Tall thin table

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:

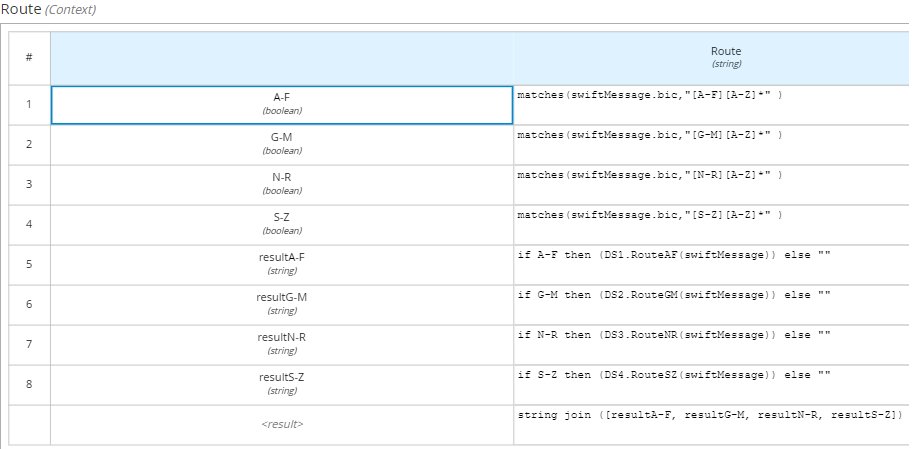


Figure 18: FEEL to invoke alphabetically indexed decision services

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

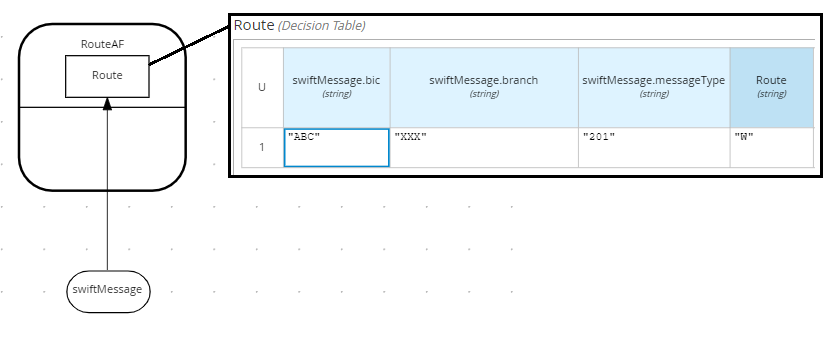


Figure 19: Routing rules for Swift Bic starting A-F

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:

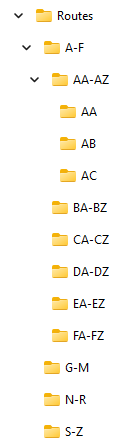


Figure 20: Routing folder structure

# The Service Level Pattern

The *Service Level Pattern* reuses common rules and simplifies conditional logic. 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 1: Service Level Pattern

## Advantages

The Service Level Pattern reduces the amount of repeating business logic and simplifies design.

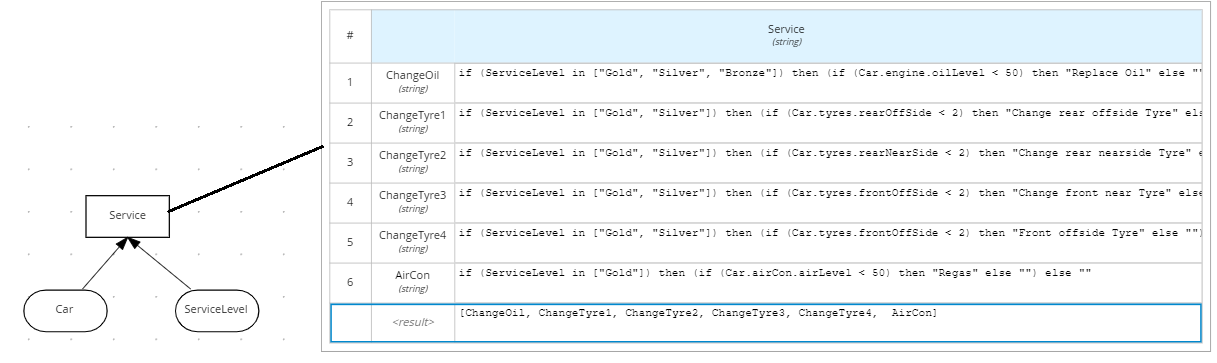
## Disadvantages

Additional logic is required to control execution of services

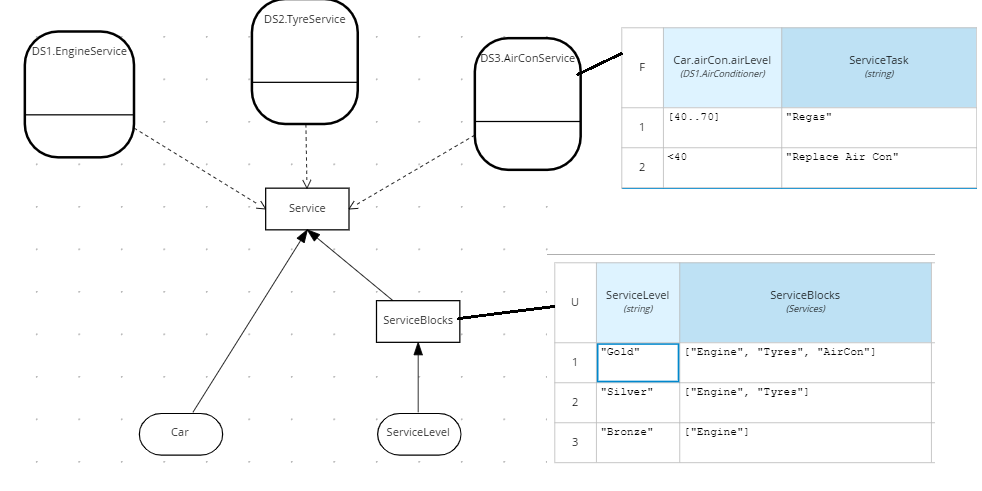
## Pattern Example

Consider a decision service to determine servicing tasks for a car. There are three levels: *Gold, Silver, Bronze*. Each level determines the maintenance work to be performed. For example, *Gold* covers air conditioning whereas Silver and Bronze do not.

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



This implementation is difficult to understand and hard to maintain. 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 block is associated to a decision service.



The Service Level pattern simplifies DMN in this example for two reasons:

**Decoupling**. The service level is decoupled from service tasks

**Reuse**. New decision services could reuse the Service Levels. For example, truck servicing could reuse the engine and aircon Service Levels.

# Conclusion

In this lab we presented 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
* The **Service Level Pattern** to reuse decisions and reduce complexity

Consider applying these patterns to your DMN to help you scale!