TM Forum Technical Report

Utility - Intent Extension Model

TR291I

Maturity Level: **General Availability (GA)**	Team Approved Date: 04-Jul-2024
Release Status: Production	Approval Status: TM Forum Approved
Version 3.6.0	IPR Mode: RAND

Notice

Table of Contents

Executive Summary

Intent management is expected to make operational decisions that lead to optimal use of resources for providing the maximum possible business value. To enable decision-making towards this objective an intent manager needs information that quantifies business value. This intent extension model introduces utility information based on utility functions within intents. This allows intent owners to annotate intent requirements with information regarding business value of potential outcomes. An intent owner can therefore tell an intent handler its notion of business value. An intent handler can use utility information received within multiple intent from different intent owners to apply utility based decision and optimization techniques.

Introduction

Autonomous networks and autonomous domains within get their requirements through intent and are then expected to find solutions, including configurations and resource usage strategies to comply to these requirements. This is primarily a binary decision. The autonomous domain either meets a requirement or not. A typical autonomous domain that manages resources and services for many customers with a great number of use cases to consider. This domain and its intent manager will eventually end up in a situation where not all requirements can be met. This might be caused by many factors, such as errors and outages, unexpected user behavior due to extraordinary events or by communication service providers overusing the network by selling too many services to too many customers and accepting lower levels of user experience.

From the perspective of intent handlers, these situations will lead to partially observed degradation with respect to intent requirements. The intent handler is left with the task to detect conflicts in resource allocation and mitigate them. It might not be able to find solutions that satisfy all requirements, but it should be able to find solutions that leave the managed autonomous domain in the best possible state. This refers to implicit and continuous optimization performed by intent handlers within their intent management control loops. They would continuously look for opportunities to optimize for reaching higher levels of intent compliance.

The central challenge for any algorithm that performs the decision-making in such optimization processes is prioritization between multiple options. It needs to be able to compare potential changes in observed metrics and then decide if a change is overall an improvement or not. Furthermore, it needs to identify the most preferential target configuration out of all available options. This bears a couple of practical challenges. For example, if a different configuration is expected to improve latency, but degrades throughput and also uses more energy, the autonomous system needs to decide if the gain in latency is worth the other degradation. How does the intent handler compare a change in latency with a change in throughput and balance between them? Ideally it would assess the solution with respect to business value and then take the solution it considers leaving the autonomous domain in an overall most valuable state.

This process of optimization for business value is referred to as decision by business utility. In this respect utility is a normalized quantification of business value. It is normalized, because equal changes in utility mean equal impact on business value. This means if we can calculate a utility score for latency and also for throughput and energy consumption, we have mapped these different metrics into comparable values. This allows a detailed and quantifiable assessment if an improvement in latency outweighs a degradation in other metrics or not.

The utility intent extension model specified in this document introduces utility information to be added into an intent. Utility information specifies a function to be used to map the values of a metric into a utility score. This utility score is a measure of business value. Having this information in the intent and associated with requirements expressed with the same metrics allows an intent owner to convey its notion of business value and its preferences to the intent handler. It is doing so in a quantifiable way that is directly enabling calculation within the intent handler for making optimization and conflict resolution decisions.

Furthermore, the utility intent extension model determines how utility scores can be represented in intent reports and how an intent owner can explain the correct interpretation of its utility information using utility profiles.

Scope

This document is part of the TR291 series specifying intent extension models published as optional models within the TM Forum Intent Ontology (TIO).

This document defines general purpose vocabulary extending the intent models for assigning validity to intents and intent elements.

Revision information

This version 3.6.0 of the Utility Intent Extension Model is the first version. The version number indicates that it is aligned with the version 3.6.0 of the TM Forum Intent Ontology (TIO).

1. Notation and namespaces

The proposal of best intent model depends on the following models:

Model	Prefix	Namespaces	Published by	Purpose in the model
Intent Common Model	icm	`http://tio.models.tmforum.org/tio/v3.6.0/IntentCommonModel/`	TM Forum	General ontology model of intent and intent report expression. This document is part of the intent common model specification.
Intent Management Ontology	imo	`http://tio.models.tmforum.org/tio/v3.6.0/IntentManagementOntology/`	TM Forum	Defines basic vocabulary and concepts of intent based operation. This document specifies vocabulary for intent management functions and their roles, as well as the types of intent models within the TM Forum Intent Ontology (TIO).
Utility	ut	`http://tio.models.tmforum.org/tio/v3.6.0/Utility/`	TM Forum	(This model) Defines the vocabulary to define and apply utility information within an intent.
Mathematical Functions Definition and Collection	mf	`http://tio.models.tmforum.org/tio/v3.6.0/MathFunctions/`	TM Forum	Identifies the model specified in this document. Definition of mathematical functions
Metrics and observation	met	`http://tio.models.tmforum.org/tio/v3.6.0/MetricsAndObservations/`	TM Forum	The metrics and observation model introduces metrics as measurable and observable items and defines vocabulary to manage observations about the metric
Set Operators Ontology	set	`http://tio.models.tmforum.org/tio/v3.6.0/SetOperators/`	TM Forum	Specification of set operators (This document)
Function Definition Ontology	fun	`http://tio.models.tmforum.org/tio/v3.6.0/FunctionOntology/`	TM Forum	Basic expression of functions
Quantity Ontology	quan	`http://tio.models.tmforum.org/tio/v3.6.0/QuantityOntology`	TM Forum	Introduction of quantities and related operators

Model	**Prefix**	**Namespaces**	**Published by**	**Purpose in the model**
Conditions and Logical Operators Ontology	log	`http://tio.models.tmforum.org/tio/v3.6.0/LogicalOperators/`	TM Forum	Specifies logical operators to express logical relationships and the evaluation of truth values.
Time Ontology in OWL	t	Specification: https://www.w3.org/TR/owl-time/ Namespace and ontology: `http://www.w3.org/2006/time#`	W3C	Expression of date and time [owltime]
RDF version 1.1	`rdf`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	W3C	Providing fundamental modeling basics [rdf11]
RDF Schema 1.1	`rdfs`	`http://www.w3.org/2000/01/rdf-schema#`	W3C	Providing schema for knowledge modeling [rdfs11]
XML Schema	`xsd`	`http://www.w3.org/2001/XMLSchema#`	W3C	Providing data types for literal objects [xsd-1] [xsd-2]
Examples	ex	http://www..example.org/	IANA	Reserved domain name for examples

Table 1: Model references

The model is based on the Resource Description Framework (RDF) [rdf, rdf_mt, rdf_primer] and the Resource Description Framework Schema (RDFS) [rdfs] published by the World Wide Web Consortium (W3C)

The links in Table 1 are currently not available, but there are plans in place to create the related assets and make them available soon.

2. Utility Information

Utility information is specified within an intent in connection with observable properties such as metrics which are used to define the intent requirements. It defines mapping from observed values into utility scores, which are an expression of business value. Utility information is expressed by instances of class `ut:UtilityInformation`. It is a specialization and subclass of the class `icm:Information` introduced by the intent common model.

The property `ut:utility` assigns utility information to an element of an intent. This establishes knowledge about the perception of business value of potential outcomes of intent handling by the intent owner. An intent handler can use this information in its implementation of decision-making, prioritization and conflict resolution.

The main purpose of utility information is to establish knowledge about how to map an observed outcome of intent handling into some notion of business value according to the intent owner. This mapping is achieved using functions. A function used to map an observation into a utility score if referred to as utility function. A utility score is in this context a number representing business value and relative priority. Therefore, utility information primarily defines what function to be used for a metric to obtain utility scores.

For example, a utility function regarding latency might map measured latency values into utility scores. If a latency value of 10ms is observed, this utility function might map it into the utility score 1.0. A later measurement of 20ms might be mapped into the utility score of 0.5, which means that the business value of achieving 20ms is only half of the business value of achieving 10ms. The business value might degrade further for higher observed latency values. This establishes qualitative knowledge of perceived business value.

The property `ut:function` assigns a function to utility information. This function shall be used as utility function to calculate a utility score. A function suitable as utility function provides a result with a numeric datatype, such as `xsd:decimal` or a quantity without unit.

A useful base for expressing utility functions is the mathematical function ontology [TR292H]. It specifies how mathematical functions are expressed and modeled. It also introduces a set of basic functions. The mathematical function ontology is based on the function definition ontology [TR292C], which introduces a general function notation in the TM Forum Intent Ontology (TIO). This includes concepts such as defining new functions as combination of multiple other functions and how to define functions over distinct input value ranges. These are concepts that are useful for defining custom functions suitable for expressing utility value mapping.

Mathematical functions have input values, and they are characterized by a set of parameters that determine the shape of the function and how it maps the input into a result. Input for the function is part of the function arguments. Parameters of the function are either part of the function definition or also set using additional function arguments. The function definition ontology allows a list notation of function argument, and it allows named function arguments. If a list notation is used, the position of the argument value in the list determines its meaning. If named arguments are used, values are directly assigned to individual arguments. Within utility information the use of an argument list is described in chapter 2.1. and the use of named arguments in chapter 2.2.

2.1. List of function arguments

Utility information specifies the argument list to be used with the function using the property `ut:withArguments`. Its range contains a list populated with values and objects that should match the argument specification of the function specified as utility function.

Input values, which represent values for metrics to be mapped into utility scores are not known when the utility information is specified, because they are dynamically observed. They would be represented by B-nodes as placeholders in the function argument expression. For arguments that set parameters of the function and determine the function shape and its value mapping, the list of arguments should provide concrete values to be used for the utility function.

For a function to be used as utility function, the utility information needs to connect the function to the metric the utility is specified for. The property `ut:forMetric` maps function arguments to metrics. It contains a list representing a value pair. The first entry in this is the B-node used in the argument list provided by `ut:withArguments`. The second entry is the metric represented by this B-node. This specifies that values of this metric shall be used as argument of the function. This therefore also specifies for which metrics this utility function is supposed to be used.

For example:

```
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                   "-2.4"^^xsd:decimal
                   "1.0"^^xsd:decimal
                   "10ms"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:latency )
.
```

This example defines utility information `ex:U1`. It specifies that the function `mf:logistic` shall be used as utility function. This is a pre-defined logistic function specified in the function definition ontology [TR292H]. This example also specifies that it shall be a utility function for the metric `ex:latency`. This is expressed using the `ut:forMetric` property mapping the B-node `_:x` to `ex:latency`. The B-node `_:x` is then used to represent the metric `ex:latency` in the argument list of the function specified with the `ut:withArguments` property. For this utility function values of latency shall be the first argument of the function. The second and third argument of the function are parameters that define the exact shape of the used logistic function. The second argument is the logistic growth parameter that determines the steepness of the logistic function. It is set to the value of "-2.4". The third argument is the supremum and upper boundary of the logistic function. It expresses that the logistic function approaches a maximum value of "1.0". The fourth argument is an offset for the midpoint of the logistic function in the direction of the input dimension (x-Axis). Here, the function is moved to a midpoint at 10ms. With this utility information, an intent handler can map any observed or predicted value for the metric `ex:latency` into a utility score.

2.2. Named function arguments

Utility information specifies the value assigned to an argument for the function using the property `ut:withArgument`. Its range contains a list with two entries. The first entry is a named argument of the function. The second entry is the value to be used.

When using named arguments it is still possible to use B-nodes for representing the input value and then associate it with the utility metric. In this case it would use ut:withArgument to map the named input argument of the function to a B-node and then assign the utility metric to the B-node using `ut:forMetric`. However, with named arguments the use of B-nodes is not necessary and the named input argument can be directly associated with the utility metric using the property `ut:forMetric`.

For example:

```
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArgument ( mf:k "-2.4"^^xsd:decimal ) ,
                  ( mf:x0 "10ms"^^quan:quantity ) ;
  ut:forMetric ( mf:input ex:latency )
.
```

This example defines the same utility information as the example in chapter 2.1, but it is doing so by using named function arguments rather than an argument list. The use of B-nodes for associating the utility metric to the function input is not necessary when named arguments are used. The utility metric ex:latency is directly associated with the identifier of the function argument `mf:input`. The value for supremum `mf:l` is not explicitly stated here, because its default value of "1.0" applies. In the list based notation in chapter 2.1 it needed to be provided, because the order of arguments is significant.

3. Allocation of Utility Information

3.1. Utility associated with intent

Utility information can be assigned to an intent object. If all utility information is assigned directly with the intent object, this defines the utility function to be used throughout the intent and within all branches of the expectation and condition tree within the intent.

For example:

```
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                  "-2.4"^^xsd:decimal
                  "1.0"^^xsd:decimal
                  "10ms"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:latency )
.
ex:U2
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                  "2.0"^^xsd:decimal
                  "1.0"^^xsd:decimal
                  "100Mbps"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:throughput )
.
ex:I1
  a icm:Intent ;
  log:allOf ( ex:E1 ) ;
  ut:utility ex:U1, ex:U2
.
ex:E1
  a icm:PropertyExpectation ;
  icm:target ex:T1 ;
  log:allOf ( ex:C1 ex:C2 )
.
ex:C1
  a log:Condition ;
  quan:smaller ( [ met:lastValue ( ex:latency ) ]
                "10ms"^^quan:quantity )
.
ex:C2
  a log:Condition ;
  quan:greater ( [ met:lastValue ( ex:throughput )
                "200Mbps"^^quan:quantity )
.
```

This example shows an intent with one property expectation and two conditions formulating requirements for latency and throughput. Furthermore, this example specifies the utility information objects `ex:U1` and `ex:U2` with utility functions associated with latency and throughput metrics. These two utility information objects are assigned to the intent using the property `ut:utility`. This specifies that the intent handler can use the utility functions defined in `ex:U1` and `ex:U2` if it needs to map an observation for the used metrics into utility scores.

3.2. Utility associated with intent elements

Utility information can be allocated at and associated with various elements. Intents have a tree structure described in the intent common model [TR290A] based on expectations and conditions associated with each other and the intent object through logical operators. This is the structure used for determining compliance to intent requirements or partial requirements within the intent. Utility Information can be associated with any of these intent elements. The utility information is then applicable to the intent element it is associated with and all subsequent elements in the respective subtree. Utility information associated with an element in the intent tree would be inherited by the entire subtree. However, utility information associated within branches of the intent would override the utility information associated with parent elements if it concerns the same metrics. If an intent handler needs to calculate utility for a metric, it would use the utility function defined within the branch in the intent tree structure closest to the metric.

For example:

```
ex:U2
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                    "2.0"^^xsd:decimal
                    "1.0"^^xsd:decimal
                    "100Mbps"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:throughput )
.
ex:U3
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                  "1.0"^^xsd:decimal
                  "1.0"^^xsd:decimal
                  "50Mbps"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:throughput )
.
ex:I1
  a icm:Intent ;
  log:allOf ( ex:E1 ex:E2 ) ;
  ut:utility ex:U2
.
ex:E1
  a icm:PropertyExpectation ;
  icm:target ex:T1 ;
  log:allOf ( ex:C1 ex:C2 )
.
ex:C1
  a log:Condition ;
  quan:greater ( [ met:lastValue ( ex:throughput ) ]
               "200Mbps"^^quan:quantity )
.
ex:E2
  a icm:PropertyExpectation ;
  icm:target ex:T2 ;
  log:allOf ( ex:C2 ) ;
  ut:utility ex:U3
.
ex:C2
  a log:Condition ;
  quan:greater ( [ met:lastValue ( ex:throughput ) ]
               "100Mbps"^^quan:quantity )
.
```

This example shows an intent with two different utility information specifications for the same throughput metric. The utility information `ex:U2` is assigned to the intent object. This is therefore the default utility information for throughput to be used in the intent. Furthermore, the intent has two property expectations with distinct targets. The expectation `ex:E1` and its condition `ex:C1` do not contain further utility information. If the intent handler calculates the utility for the throughput observed for target `ex:T1`, it would therefore use the utility `ex:U2` associated with the intent. The utility information `ex:U3` is assigned to the expectation `ex:E2`. As it concerns the same throughput metric, this means that within the scope of the expectation `ex:E2` the intent handler shall consider utility according to `ex:U3` rather than `ex:U2`. This means, that in this intent the utility of throughput shall be calculated differently depending on the expectations and their targets. Throughput observed for target `ex:T1` has a business value according to ex:U2 and throughput observed for `ex:T2` has a business value according to `ex:U3`.

4. Default Utility

Utility information can be provided within the intent of intent elements as described in this document. This is, however, optional information. Intent owners do not need to provide this information or might not provide it for all metrics used. The reason might be that the intent owner's implementation does not consider utility, or it does not have the means to generate sensible utility information for all intents. Intent based operation and in particular intent handlers shall not be fully reliant of the availability or completeness of utility information. However, there are multiple ways an intent handler can assume or acquire default utility information if dedicated utility information is missing in the intent.

4.1. Default utility assumed by intent handlers

An intent handler might be configured with a set of default utility information for metrics it supports. This intent handler utility default would be applied if there are no other sources of utility information that can be associated with the intent. Chapters 4.2 and 4.3 discuss additional sources of utility information.

A potential algorithm for setting assumed default utility functions within the intent handler might be contextualized based on the requirements stated within the intent. For example, the maximum utility for a latency metric might be associated with the required threshold value: A possible function can award full utility score if required value is met or exceeded. It might also award zero utility score to 5 times higher resulting values for this KPI with a linearly declining function for values in-between. A sensible shape of a default utility functions is specific to a type of metric and can also depend on use cases and context. A good default would typically require research or a statistical evaluation of typical values. This investigation for finding the default is outside the scope of this specification.

As intent handlers might change their default assumed utility function, it should be good practice that intent owners always provide utility information if it matters and if they support utility. Intent owners should not rely on a particular implementation of utility defaults in the intent handler.

4.2. Default utility per intent owner

Intent managers can state default utility information for metrics in their intent manager capability profile [TR298]. This communicates utility information and utility functions to intent handlers which should be considered as default and baseline for intent received from this intent manager. This means an intent handler would know utility functions associated with a metric within an intent, even if the intent owner did not provide dedicated utility information within the intent.

4.3. Default utility per product and service

Intent specifications [TR299] can also define default utility information. For intent that is derived from an intent specification, this utility information can be used as default if no dedicated utility information is provided within the intent. This would override utility information defaults specified on a per-owner basis through the intent manager capability profile.

5. Utility functions

Utility information relies on functions. These functions describe the mapping of values for observable metrics to utility scores that express business value and relative priority. A useful base for expressing utility functions is the mathematical function ontology [TR292H]. It specifies how mathematical functions are expressed and modeled. It also introduces a set of basic functions. The mathematical function ontology is based on the function definition ontology [TR292C], which introduces a general function notation in the TM Forum Intent Ontology (TIO). This includes concepts such as defining new functions as combination of multiple other functions and how to define functions over distinct input value ranges. These are concepts that are useful for defining custom functions suitable for expressing utility value mapping.

This chapter provides examples of various use cases regarding the expression of utility functions. This is not introducing additional vocabulary to the utility intent extension model. It rather illustrates how the function definition ontology and mathematical function ontology models can be used for common utility expression use cases.

5.1. Custom utility function

Instead of relying on pre-defined mathematical functions, custom functions can be defined within an intent and used for expressing utility information.

For example:

```
ex:myFunction
  a fun:function ;
  fun:argumentNames ( ex:input ) ;
  fun:argumentTypes ( quan:Quantity ) ;
  fun:resultType quan:Quantity ;
  rdf:value [ quan:sum ( [ mf:poly ( ex:input
                                     ( "0.0"^^xsd:decimal
                                       "0.001(Mbps)^-1"^^quan:quantity
                                     )
                                   ) ]
                         [ mf:logistic ( ex:input
                                         "2.0"^^xsd:decimal
                                         "1.0"^^xsd:decimal
                                         "10Mbps"^^quan:quantity
                                       ) ]
                       )
            ] ;
  fun:arityMin 1 ;
  fun:arityMax 1
.
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:myFunction ;
  ut:withArguments ( _:x ) ;
  ut:forMetric ( _:x ex:throughput )
```

```
.
```

This example defines the custom function `ex:myFunction`. It is a function with one quantity argument and a value calculated from the sum of a specific polynomial and logistic functions. This new custom function is then used as utility function in the specification of utility information.

5.2. Composite and contextualized utility functions

Composite utility functions consist of multiple functions that in combination define utility. For example, different function shapes shall be considered within distinct value ranges of the metric. This can be achieved by defining a custom function with partial values contributed by other functions and the use of validity context to define which function shall determine the value utility within distinct metric value ranges.

Next to defining value ranges, validity context can steer the use of functions for utility in many ways. For example, it can distinguish utility based on date and time if temporal validity conditions are used. Furthermore, conditions based on additional metrics can be applied with validity context and capture a broad range of side conditions, context and influencing factors that impact utility scoring.

For example:

```
ex:myFunction
  a fun:function ;
  fun:argumentNames ( ex:input ) ;
  fun:argumentTypes ( quan:Quantity ) ;
  fun:resultType quan:Quantity ;
  rdf:value [ mf:poly ( ex:input
                        ( "0.0"^^xsd:decimal
                          "0.001(Mbps)^-1"^^quan:quantity
                        )
                      ) ;
              iv:validIf [ quan:atMost ( ex:input "50Mbps"^^quan:quantity ) ]
            ] ;
  rdf:value [ mf:logistic ( ex:input
                          "2.0"^^xsd:decimal
                          "1.0"^^xsd:decimal
                          "50Mbps"^^quan:quantity
                        ) ;
              iv:validIf [ quan:greater ( ex:input "50Mbps"^^quan:quantity ) ]
            ] ;
  fun:arityMin 1 ;
  fun:arityMax 1
.
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:myFunction ;
  ut:withArguments ( _:x ) ;
  ut:forMetric ( _:x ex:throughput )
.
```

This example defines a custom utility function with two `rdf:value` statements. The first assigns a value based on a polynomial function. It has a validity for input values up to `50Mbps`. The second `rdf:value` statement assigns a value determined by a logistic function. It has a validity for input values greater than `50Mbps`. This means, for different value ranges of the input value, the utility is determined by different functions.

In the previous example the value ranges do not overlap. However, if the value ranges would overlap the default value combination method for associated values would apply as defined in Intent Management Elements [TR292A]. For quantities and other numerical data types the defined default combination is the sum of values. Furthermore, [TR292] defines a default associated value. It would be assumed as default for invalid objects unless explicitly specified using the property `iv:valueIfInvalid` described in the intent validity extension model [TR291A]. The default associated value for quantities and other numerical data types is "0". In the example above with its mutually exclusive value ranges, the default associated value 0 and the default combination method of a sum of values lead to the correct result. However, if value ranges overlap, a different combination of function values might be wanted.

For example:

```
ex:myFunction
  a fun:function ;
  fun:argumentNames ( ex:input ) ;
  fun:argumentTypes ( quan:Quantity ) ;
  fun:resultType quan:Quantity ;
  rdf:value [ quan:mean ( [ mf:poly ( ex:input
                                   ( "0.0"^^xsd:decimal
                                     "0.001(Mbps)^-1"^^quan:quantity
                                   )
                                 ) ;
                          iv:validIf [ quan:atMost ( ex:input "50Mbps" ) ]
                        ]
                        [ mf:logistic ( ex:input
                                        "2.0"^^xsd:decimal
                                        "1.0"^^xsd:decimal
                                        "10Mbps"^^quan:quantity
                                      )
                          iv:validIf [ quan:atLeast ( ex:input "50Mbps" ) ]
                        ]
                      )
            ] ;
  fun:arityMin 1 ;
  fun:arityMax 1
.
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:myFunction ;
  ut:withArguments ( _:x ) ;
  ut:forMetric ( _:x ex:throughput )
.
```

This example defines a custom utility function. The two partial functions have overlapping validity for input values of `50Mbps`. Here the contributions from both functions are combined by calculating the mean value. As no further default value for not being valid was explicitly defined, the default associated value "0" applies for each function if it is invalid. This means for the value range of input values smaller than `50Mbps` the polynomial function contributes a value according the input and the invalid logistic function contributes the default value "0" into the overall calculation of the utility value.

5.3. Multidimensional utility functions

Providing utility on a per metric basis with one dimensional functions is relatively easy and very modular. An intent handler can combine the partial utility scores as needed and utility functions can be applied in various combinations across use cases. However, metrics often have complex mutual relationships with each other and a combined impact on the notion of business value. Therefore, a collection of independent one dimensional utility functions might not always be sufficient to express utility scoring correctly.

Multidimensional utility functions can be defined to better represent a combined utility calculation. These are custom functions with multiple input values.

For example:

```
ex:myFunction
  a fun:function ;
  fun:argumentNames ( ex:input1 ex:input2 ) ;
  fun:argumentTypes ( quan:Quantity ) ;
  fun:resultType quan:Quantity ;
  rdf:value [ quan:sum ( [ mf:poly ( ex:input1
                                  ( "0.0"^^xsd:decimal
                                    "0.001(Mbps)^-1"^^quan:quantity
                                  )
                                ) ]
                         [ mf:logistic ( ex:input2
                                         "-2.6"^^xsd:decimal
                                         "3.0"^^xsd:decimal
                                         "50ms"^^quan:quantity
                                       ) ]
                       )
            ] ;
  fun:arityMin 2 ;
  fun:arityMax 2
.
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:myFunction ;
  ut:withArguments ( _:x1 _:x2 ) ;
  ut:forMetric ( _:x1 ex:throughput ) ;
  ut:forMetric ( _:x2 ex:latency )
.
```

This example defines the custom two-dimensional function. It defines the function value as sum of two contributing functions. Each of the contributing functions has a different input. The first input of the custom function is associated with the throughput metric in the utility information. The second input is associated with latency. The sum of the polynomial with throughput as input and the logistic function with latency as input determine the overall resulting utility score.

It is also possible to utilize further input dimensions for applying side conditions. For example:

For example:

```
ex:myFunction
  a fun:function ;
  fun:argumentNames ( ex:input1 ex:input2 ) ;
  fun:argumentTypes ( quan:Quantity ) ;
  fun:resultType quan:Quantity ;
  rdf:value [ mf:poly ( ex:input1
                      ( "0.0"^^xsd:decimal
                        "0.001(Mbps)^-1"^^quan:quantity
                      )
                    ) ;
              iv:validIf [ quan:atMost ( ex:input2 "1000"^^quan:quantity ) ]
            ]
  rdf:value [ mf:poly ( ex:input1
                      ( "0.0"^^xsd:decimal
                        "0.05(Mbps)^-1"^^quan:quantity
                      )
                    ) ;
              iv:validIf [ quan:greater ( ex:input2 "1000"^^quan:quantity ) ]
            ]
  fun:arityMin 2 ;
  fun:arityMax 2
.
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:myFunction ;
  ut:withArguments ( _:x1 _:x2 ) ;
  ut:forMetric ( _:x1 ex:throughput ) ;
  ut:forMetric ( _:x2 ex:numberOfUsers )
.
```

This example defines a two-dimensional utility function. The first input is throughput and the second input is the number of users. The custom function to be used as utility function defines two polynomials. The first one shall determine the utility score of an observed throughput if the number of users is 1000 or less and the other polynomial shall determine the utility score for observed throughput for more than 1000 users. This means, in this example, depending on the number of users the perception of business value is different.

5.4. Discrete utility functions

Requirements based on discrete values can also have a utility function. Utility information can use discrete functions such a mapping function to map discrete input values into associated utility scores.

For example:

```
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:mapping ;
  ut:withArguments ( _:x1
                    ( ( "1.0" ex:ServiceDeployed ex:ServiceHighPerformance )
                      ( "0.5" ex:ServiceLowPerformance )
                      ( "0.0" ex:ServiceNotAvailable )
                    )
                  ) ;
  ut:forMetric ( _:x1 ex:serviceState )
.
```

This example shows how a state of a service can be mapped into utility information. A service in state deployed or high performance results in utility score "1.0". If the service state indicates low performance, a lower utility score of 0.5 is awarded. If the service is not available, the utility score drops to "0".

6. Utility profiles

Intent owners can have unique strategies how to use and set utility to convey their view on business value from intent compliance to intent handlers. For example, one intent owner might provide intent with a value range from 0 to 1, with higher utility values signifying higher value of the outcome. Other intent owners might use values between 1 and 100 or other unique value ranges. As intent handlers can receive intent from multiple distinct intent owners, it faces the challenge to interpret the provided utility information correctly. It would need to understand, for example, that the utility of 0.8 calculated from the utility information from one intent owner means the same business value level as 80 calculated based on the utility information coming from another value.

Utility profiles introduced in this chapter provides the intent handler with the needed information and explanation about the utility strategy used by an intent owner.

Instances of class `ut:UtilityProfile` contain information that explains the utility information provided. It is modeled as subclass of `icm:Information`.

The property `ut:utilityProfile` allows assigning utility profiles, for example to intent, other elements within an intent, such as expectations and conditions or utility information objects within an intent. It can also add utility profile information to intent manager capability profiles [TR298] or intent specifications [TR299].

The properties `ut:minUtility` and `ut:maxUtility` are used to define the value range of utility scores provided by a utility function. In this range `ut:minUtility` defines the lower boundary of the value range of utility scores. It therefore marks the lowest possible business value. The property `ut:maxUtility` defines the upper boundary of the value range of utility scores. This value is interpreted as the maximum business value. For the interpretation of elevated utility see chapter 6.1.

For example:

```
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                    "-2.4"^^xsd:decimal
                    "1.0"^^xsd:decimal
                    "10ms"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:latency ) ;
  ut:utilityProfile ex:UP1
.
ex:UP1
  a ut:UtilityProfile ;
  ut:minUtility "0.0"^^xsd:decimal ;
  ut:maxUtility "1.0"^^xsd:decimal
.
ex:U2
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                    "2"^^xsd:decimal
                    "1.0"^^xsd:decimal
                    "100Mbps"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:throughput ) ;
  ut:utilityProfile ex:UP2
.
ex:UP2
  a ut:UtilityProfile ;
  ut:minUtility "1.0"^^xsd:decimal ;
  ut:maxUtility "100.0"^^xsd:decimal
.
```

This example defines two utility profiles ex:UP1 and ex:UP2 and assigns them to utility information. This example states that the utility scoring for latency according to ex:U1 provides utility values between "0" an "1". The utility scoring for ex:U2 would provide utility values between "1" and "100". Based on this information an intent handler would assume that a utility score of "0" for latency means the same as a utility score of "1" for throughput. Also, a utility score of "1" for latency would mean the same business value as a utility score of 100 for throughput. This allows the intent handler to map and normalize the utility scores into comparable value ranges.

6.1. Considering elevated utility

Assuming that maximum possible utility levels mean the same business value is not always correct, because utility functions might also be stretched to express relative priority and elevated business value. For example, the communication service provider (CSP) might consider one customer more important than another one. It would consider this in the intent by assigning utility functions to the requirements of the more important customer, which deliver higher utility scores for comparable outcomes. This means the utility is stretched or elevated.

The property `ut:maxUtility` provides the maximum utility value without elevation. It therefore represents the basic shape and value ranges of utility function used by the intent owner. It can be used for normalizing utility scores to make them comparable even if some scores are elevated. However, with elevated scores, `ut:maxUtility` does not specify the actual maximum utility score possible.

The property `ut:elevatedMaxUtility` specifies the maximum elevated utility score provided by a utility function. This is the actual maximum utility score used including all implicit prioritization and score elevation.

For example:

```
ex:U1
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                  "-2.4"^^xsd:decimal
                  "1.0"^^xsd:decimal
                  "10ms"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:latency ) ;
  ut:utilityProfile ex:UP1
.
ex:UP1
  a ut:UtilityProfile ;
  ut:minUtility "0.0"^^xsd:decimal ;
  ut:maxUtility "1.0"^^xsd:decimal ;
  ut:elevatedMaxUtility "100"^^xsd:decimal
.
ex:U2
  a ut:UtilityInformation ;
  ut:function mf:logistic ;
  ut:withArguments ( _:x
                     "2"^^xsd:decimal
                     "1.0"^^xsd:decimal
                     "100Mbps"^^quan:quantity ) ;
  ut:forMetric ( _:x ex:throughput ) ;
  ut:utilityProfile ex:UP2
.
ex:UP2
  a ut:UtilityProfile ;
  ut:minUtility "0.0"^^xsd:decimal ;
  ut:maxUtility "100.0"^^xsd:decimal
.
```

This example specifies that the utility according to utility profile ex:UP1 has and ordinary value range between "0" and "1" and an elevated range up to a maximum value of "100". This means the respective utility function for latency might be stretched to implicitly express relative priority. In this example the utility score "1" for latency still means the same business value as a utility score of "100" used for throughput. However, the utility for latency can be elevated and reach much higher maximum values up to a value of "100". Although the utility for latency and the utility for throughput can both have the same maximum value, in this setup a "100" utility score for latency marks a much higher relative business value and much higher relative contribution to overall utility. In this example, latency bears a business value that is relatively greater than the one for throughput by a factor of 100.

6.2. Providing utility profiles as intent manager capability

Utility profiles can be provided by intent owners in various ways and through multiple artifacts and documents. Adding them to an intent as described earlier in chapter 6 is one possibility that is limited in scope as the provided utility profile would only be applicable for the intent it is directly associated with. However, intent owners would typically be consistent and not change the way they use utility between intents or between different metrics. Therefore, utility profiles are often an inherent property of an intent owner and can be treated as such.

Utility profiles can be provided in intent manager capability profiles. This means an intent owner can communicate in general what its provided utility information means through its capability profile. This utility profile would be treated as default interpretation for utility information for all intent coming from an intent management function unless there is utility information directly added to the intent. It is therefore possible for an intent owner to support and use multiple utility strategies and override its default communicated strategy on a per intent basis. See the specification of intent manager capability profiles [TR298] for more details.

6.3. Enforcing utility with intent specifications.

Intent specifications [TR299] define rules and constraints regarding allowed content of an intent. They are typically added to service catalogs and associated to service specifications to define the allowed intent content for intents used in association with a particular service. This can be used in various ways to enforce and limit utility and its possible use and interpretation.

Intent specifications can allow or mandate the use of utility information. This might include a selection of approved functions and value ranges of function parameters. This also allows limiting the maximum utility that can be assigned to metrics within a valid intent. It therefore effectively establishes a basic authorization mechanism for using utility and by doing so impacting the preference an intent handler would give to some intent owner's requirements and intents over the intent's and requirements from other intent owners and parties. This can help to preserve a basic degree of fairness in intent handling with requirements from various parties which might be in competition with each other for getting best results for themselves.

Utility profiles can also be used in intent specifications. This would prescribe what utility strategies can and shall be used in intents. It can be used as constraint for intent owners as they would need to follow certain approved utility profiles. It can also serve a utility interpretation aid for intent handlers as they might be able to rely on that intents expressed according to the intent specification use utility in a particular way.

7. Reporting about utility evaluation results

Intent handlers use utility information to score the associated business value of a state of the operated system or network. Typically, they apply utility information from various sources to calculate a utility score for individual observations and then aggregate them into partial or global utility scores. This means utility scores provide insights into the overall decision-making and optimization processes of an intent handler and might explain why the intent handler prefers certain operational states over others. Intent owners might be interested in knowing the utility scoring results of an intent handler as explanation for why their intent requirements might not be met.

In accordance to the principles of reporting expectations, the target of the utility reporting expectation sets the scope of intent elements a utility score shall be provided for. This also determines the reporting object present in the resulting intent report. Intent handling events determine when a utility report shall be created.

7.1. Requesting utility reports for observations

Intent owners can ask for a utility scoring report with a utility reporting expectation in the intent. It is an instance of class `ut:UtilityReportingExpectation` and subclass of `icm:ObservationReportingExpectation` defined in the intent common model [TR290A]. Requesting an intent report with a utility reporting expectation implies that information about utility scoring by the intent handler shall be added to the intent report. As it is a subclass of an observation reporting expectation, it also implies that observation reporting is added to the intent report as described in the intent common model [TR290A] [TR290B].

Utility scores are provided within an intent report using the property `ut:utilityResult`. It is assigned to observation objects used to report observations for metrics. The property `ut:utilityResult` adds a utility score for a metric to the observation object that reports the metric value. This states the utility score for individual observations for metrics. This provided utility result is a utility score aggregation using not only the utility information from the intent that reports about it, but also the utility information associated with other intents present in the intent handler. These other intents may provide additional requirements for the respective observed metric and therefore contribute utility scores as well. The aggregation algorithm used by the intent handler is not specified in detail. However, the reported utility result represents the opinion of the intent handler regarding the overall business value of the achieved value for the metric.

The intent report can also include a reference to the utility profile used for utility results provided within an observation. This can be stated using the property `ut:utilityProfile`. It can be included as part of each individual observation or allocated with other elements of the intent to set defaults applicable to parts of the intent or the entire intent. If no utility profile is provided, an intent handler owner as receiver of the intent profile can assume that the handler uses the utility profile it has provided for this intent or through other means, such as its intent manager capability profile. This can however be ambiguous especially for other receivers of the intent report. It is therefore recommended to always include a reference to the used utility profile at least in more complex reporting scenarios.

For example:

```
ex:Intent1
  a icm:Intent ;
  log:allOf ( ex:E1 ex:RE1 )
.
ex:T1
  a icm:Target ;
  rdfs:member ex:Slice001
.
ex:E1
  a icm:PropertyExpectation ;
  icm:target ex:T1 ;
  log:allOf ( ex:C1 )
.
ex:C1
  a log:Condition ;
  quan:smaller ( [ met:lastValue ( ex:sliceLatency ) ]
                 "10ms"^^quan:quantity )
.
ex:T2
  a icm:Target ;
  rdfs:member ex:Intent1, ex:E1, Ex:C1
.
ex:RE1
  a ut:UtilityReportingExpectation
  icm:target ex:T2 ;
  icm:reportDestination [ a rdfs:Container ;
                          rdfs:member ex:IMF1 ]
  icm:reportTriggers [ a rdfs:Container ;
                       rdfs:member imo:Degrades ,
                                   imo:Complies ]
.
```

This intent contains a requirement for a slice regarding slice latency. The intent owner requests intent reports if the compliance to the intent changes from compliant to degraded or vice versa. As the report was requested with a utility reporting expectation, it shall contain details about the observed value for the slice latency metric as well as the resulting utility score:

```
ex:IR0001
  a icm:IntentReport ;
  icm:about ex:Intent1
  icm:reportGenerated [ t:inXSDDateTimeStamp "2023-02-
06T13:30:10+01:00"^^xsd:dateTime ] ;
  icm:reportNumber 1 ;
  imo:handlingState imo:StateCompliant ;
  imo:updateState icm:StateNoUpdate ;
  icm:result true ;
  icm:resultFrom ex:ER1
.
```

```
ex:TR1
  a icm:TargetReport ;
  rdfs:member ex:Slice001
.
ex:ER1
  a icm:PropertyExpectationReport ;
  icm:targetReport ex:TR1 ;
  icm:result true ;
  icm:resultFrom ( ex:CR1 )
.
ex:C1
  a log:ConditionReport ;
  icm:result true ;
  met:observed [ met:observedMetric ex:sliceLatency ;
                rdf:value "8ms"^^quan:quantity ;
                met:obtainedAt [ t:inXSDDateTimeStamp "2023-07-
05T01:00+01:00"^^xsd:dateTimeStamp ] ;
                ut:utilityResult "0.8"^^xsd:decimal ;
                ut:utilityProfile ex:UP1
              ]
.
```

This intent report contains detailed reports about the intent, the property expectation and the condition. As this intent report is generated for a utility reporting expectation it includes the observation regarding the metric used by the condition in the intent. Furthermore, the reported observation contains the utility result as well as a reference to the utility profile used for this utility value.

In this example the targeted slice instance meets the requirement and therefore from the perspective of this intent only, it might be awarded full utility score of "1.0". However, the reported utility score is only "0.8". This is caused by other intents present in the intent handler setting further requirements for the same metric observed for this slice. The stated utility result represents the overall utility based opinion of the intent handler about the achieved result. This is an indication that the intent handler might still look for changed configurations that further improve latency, although from this intent's perspective the requirements are met, and the intent owner would be satisfied with the result.

It is also possible that the reverse situation is reported. The utility originating from this intent might be low, for example "0.5" but the intent handler reports a high utility result value. This is an indication that this intent does not have sufficiently high priority and business value compared with other intents. The solution found and applied by the intent handler for maximizing overall business value has prioritized other intents due to their higher contribution to overall utility.

8. Administrative Appendix

8.1. Document History

8.1.1. Version History

Version Number	Date Modified	Modified by:	Description of changes
3.5.0	03-May-2024	Alan Pope	Final edits prior to publication
3.6.0	04-Jul-2024	Alan Pope	Final edits prior to publication

8.1.2. Release History

Release Status	Date Modified	Modified by:	Description of changes
Pre-production	03-May-2024	Alan Pope	Initial Release
Pre-production	10-Jun-2024	Adrienne Walcott	Updated to Member Evaluated status
Pre-production	04-Jul-2024	Alan Pope	Updated to v3.6.0
Production	30-Aug-2024	Adrienne Walcott	Updated to reflect TM Forum Approved status

8.2. Acknowledgments

Team Member (@mention)	Company	Role*
Jörg Niemöller	Ericsson	Author, Project Co-Chair
Kevin McDonnell	Huawei	Project Co-Chair
Yuval Stein	Amdocs	Project Co-Chair
Kamal Maghsoudlou	Ericsson	Key Contributor
Leonid Mokrushin	Ericsson	Key Contributor
Marin Orlić	Ericsson	Key Contributor
Aaron Boasman-Patel	TM Forum	Additional Input
Alan Pope	TM Forum	Additional Input
Dave Milham	TM Forum	Additional Input
Xiao Hongmei	Inspur	Reviewer

**Select from: Project Chair, Project Co-Chair, Author, Editor, Key Contributor, Additional Input, Reviewer*

9. Appendix A: Vocabulary Reference

This chapter contains a reference definition of all model vocabulary. It is sorted alphabetically.

9.1. elevatedMaxUtility

The property `ut:elevatedMaxUtility` specifies the maximum elevated utility score provided by a utility function. This is the actual maximum utility score used including all implicit prioritization and score elevation.

Instance of: `rdf:Property`
Sub-property of: `icm:information`Range: `ut:UtilityProfile`

9.2. forMetric

The property `ut:forMetric` maps function arguments to metrics. It contains a list representing a value pair. The first entry in this is the B-node used in the argument list provided by `ut:withArguments`. The second entry is the metric represented by this B-node. This specifies that values of this metric shall be used as argument of the function. This therefore also specifies for which metrics this utility function is supposed to be used.

Instance of: `rdf:Property`
Domain: `ut:UtilityInformation`Range: `rdf:List`

9.3. function

The property `ut:function` assigns a function to utility information. This function shall be used as utility function to calculate a utility score. A function suitable as utility function provides a result with a numeric datatype, such as `xsd:decimal` or a quantity without unit.

Instance of: `rdf:Property`
Sub-property of: `icm:information`Domain: `ut:UtilityInformation`Range: `fun:Function`

9.4. maxUtility

The property `ut:maxUtility` is a property of utility profile information. It defines the upper boundary of the value range of utility scores used. In this range `ut:maxUtility` is interpreted as the maximum business value.

Instance of: `rdf:Property`
Sub-property of: `icm:information`Range: `ut:UtilityProfile`

9.5. minUtility

The property `ut:minUtility` is a property of utility profile information. It defines the lower boundary of the value range of utility scores used. In this range `ut:minUtility` corresponds to the utility score that marks the lowest business value.

Instance of: `rdf:Property`
Sub-property of: `icm:information`Range: `ut:UtilityProfile`

9.6. utility

The property `ut:utility` assigns utility information to an element of an intent.

Instance of: `rdf:Property`
Sub-property of: `icm:information`
Range: `ut:UtilityInformation`

9.7. UtilityInformation

Utility information is expressed by instances of class `ut:UtilityInformation`.

Instance of: `rdfs:Class`
Subclass of: `icm:Information`

9.8. UtilityProfile

Instances of class `ut:UtilityProfile` contain information that explains the utility information provided.

Instance of: `rdfs:Class`
Subclass of: `icm:Information`

9.9. utilityProfile

The property `ut:utilityProfile` allows assigning utility profiles, for example to intent, other elements within an intent, such as expectations and conditions or utility information objects within an intent. It can also add utility profile information to intent manager capability profiles [TR298] or intent specifications [TR299].

Instance of: `rdf:Property`
Sub-property of: `icm:information`Range: `ut:UtilityProfile`

9.10. utilityResult

The property `ut:utilityResult` allows assigning calculated utility scores. It is, for example, used in intent reports to state the utility score associated with a metric observation.

Instance of: `rdf:Property`
Range: `ut:UtilityProfile`

9.11. withArgument

Utility information specifies the value assigned to an argument for the function using the property `ut:withArgument`. Its range contains a list with two entries. The first entry is a named argument of the function. The second entry is the value to be used.

Instance of: `rdf:Property`
Domain: `ut:UtilityInformation`Range: `rdf:List` with two entries

9.12. withArguments

Utility information specifies the argument list to be used with the function using the property `ut:withArguments`. Its range contains a list populated with values and objects that should match the argument specification of the function specified as utility function.

Instance of: `rdf:Property`
Domain: `ut:UtilityInformation`Range: `rdf:List`

9.13. Vocabulary

The object ut:`Vocabulary` is a container of all model elements.

Instance of: `rdfs:Container`