Date: May 2016

Decision Model and Notation (DMN)

V1.1

OMG Document Number: formal/2016-06-01

Standard document URL: http://www.omg.org/spec/DMN/1.1

Normative Machine Consumable File(s):

http://www.omg.org/spec/ DMN/20151101/dmn.xmi

http://www.omg.org/spec/ DMN/20151101/dmn.xsd

Informative Machine Consumable File(s):

http://www.omg.org/spec/ DMN/20151101/ch11example.xml

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope

of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.
For a complete list of trademarks, see: http://www.omg.org/legal/tm_list.htm. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.
Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue.

目次

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture®

(MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language®); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel™); and industry-specific standards for dozens of vertical markets.
More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:
http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications Middleware Specifications

- **CORBA/IIOP • Data Distribution Services • Specialized CORBA**

IDL/Language Mapping Specifications Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI • UML Profile**

Modernization Specifications Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices • CORBAFacilities**

OMG Domain Specifications CORBA Embedded Intelligence Specifications CORBA Security Specifications Signal and Image Processing Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue

Needham, MA 02494

USA

Tel: +1-781-444-0404

Fax: +1-781-444-0320

Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text, table text, bullets

Helvetica/Arial – 9 or **10 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier new/Courier – 10 pt. Bold`: Programming Languages

Helvetica/Arial – 10 pt.: Exceptions

Issues

The reader is encouraged to report any technical or editing issues/problems with this by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under documents, Report a Bug/Issue.

1 スコープ

DMN の主な目的は、全てのビジネスユーザーが容易に理解できる共通の表記法を提供することである。その表記法は、最初に意思決定要求を作成し、より詳細な意思決定モデルを提供するビジネスアナリスト、次に意思決定プロセスの自動化の責任を持つ技術開発者、最後に意思決定を管理およびモニタリングするビジネス・パーソンが容易に理解できるものである。DMN は、ビジネス意思決定デザインと意思決定実装のギャップを埋める標準的な架け橋を作成する。DMN の表記法は標準的な BPMN ビジネスプロセス表記法と一緒に適切に使用できるようにデザインされる。

他の目的は、XML 表現を経ることにより、意思決定モデルが組織を越えて交換可能であることを確実にすることである。

作者は既存の意思決定モデリング・コミュニティから専門知識と経験を生み、これらの様々な表記法から単一の標準の表記法に共通のアイデアを統合することに努めた。

2 適合

2.1 適合レベル

もしソフトウェアが完全に仕様書に規定された該当箇所合致しているのであれば、ソフトウェアは DMN1.1 への準拠や適合を主張しても良い。ソフトウェアが部分的に適合点に合致している場合は仕様書に基づくと主張して良いが、仕様書に適合または準拠と主張することは許されない。

本仕様書は、適合レベル１、適合レベル２、適合レベル３の３つの適合レベルを定義する。

実装が適合レベル１を主張する場合は、適合レベル２または３のサポートは要求されない。実装が適合レベル２を主張する場合は、適合レベル３のサポートは要求されない。

実装がレベル１を主張する場合には、本書の、６章（要求）、７章（意思決定ロジック）、８章（デシジョンテーブル）に記載された仕様のすべてに準拠しなければならない。実装がレベル１準拠に一致しているという主張する場合には、表現(書式で記載されたモデル)をデシジョンモデルに翻訳（インタープリート）ことを要求されることはない。しかしながら、レベル１の規定に準拠していることを説明するために、実装は７章の仕様にある書式の解釈に準拠しなければならない。

実装がレベル２を主張する場合には、本書の、６章（要求）、７章（意思決定ロジック）、８章（デシジョンテーブル）に記載された仕様のすべてに準拠しなければならない。さらに、実装には９章の仕様にあるシンプルな式言語(simple expression language:S-FEEL)の翻訳（インタプリタト）を要求される。

実装がレベル３を主張する場合には、本書の、６章（判断の要求）、７章（判断の論理）、８章（デシジョンテーブル）および第１０章（書式言語）に記載された仕様のすべてに準拠しなければならない。９章の仕様にあるシンプルな表現書式(simple expression language:S-FEEL)が EFFL のサブセットになっていることに注意すること、それ故、実装がレベル３を主張する場合には、レベル２（そしてレベル１も）主張することができる。

さらに加えて、DMN 1.1 の３つのレベルのそれぞれについて、準拠していると主張するためには、第 2.2 章の要求のすべてにも準拠しなければならない。

2.2　一般的な準拠の要求

2.2.1　ビジュアルな外観

DMN のキーとなる要素は、本仕様書で識別されたグラフィカルな要素として使用できる図形やアイコンを選択することである。その目的は、すべてのデシジョンモデルを認識して理解できる標準的なビジュアル言語を作ることにある。デシジョンモデルのダイアグラムを作成または表示するすべての実装は、本仕様書で示されたグラフィカルな要素、図形、マークを使わなければならない。

他の場所で仕様化されていなければ、定義されたグラフィカルな要素に対して、サイズや色、線のスタイル、テキストの場所は変更可能である。

DMN ダイアグラムに対して以下の拡張が許される：

- 新しいマーカーやインジケータをグラフィカルな要素に追加することができる。それらのマーカーやインジケータは、DMN エレメントの上位属性や下位の調和したコンセプトで使用できる。
- 新しい種類の構造のための新しい図形をダイアグラムに追加してもよい、ただし新しい図形は、仕様にある全ての図形と矛盾してはならない。
- グラフィカルな要素は、色を付づけることができる、また色を付けることはこの標準内の仕様としての要素として広められる情報を拡張する意味として明記することができる。
- グラフィカルな要素の線のスタイルは変えることができる、但しその線のスタイルは仕様で要求される全ての線のスタイルと矛盾してはならない。

拡張は、グラフィカルな要素またはマーカーとして定義された図形を変えてはならない（例えば、実線を破線に変更する、四角を三角に変更する、角丸四角を四角に変更する）。

2.2.2 意思決定セマンティックス

この仕様書では、意思決定の定義で使用する多くのセマンティックな概念を定義する、それらをグラフィカルな要素やマーカーやコネクションに関連づける。

実装は、概念に関連づくセマンティックな仕様に基づき、いくつかの DMN ダイアグラム要素の翻訳を行う、ただしその翻訳は、本書で定義された意味と一致しなければならない。

2.2.3 属性とモデルへの関連づけ

本使用は、グラフィカルな要素やマーカーやコネクションについてセマンティックな要

素について属性とプロパティの数を定義する。いくつかの属性は必須として定義され、図形表現を持っていない、あるいはオプションの図形表現しかない(訳注：represent を画像的な表現と解釈しました)。そして、いくつかの属性はオプションとして定義される。

本書で必須と定義した各々の属性やプロパティのために、準拠した実装では、属性またはプロパティを作成でき表示できる、いくつかの仕組みを作り込まなければならないこの仕組みは、属性やプロパティを持つように定義された各 DMN 要素の値の作成や表示をユーザーに許可しなければならない。

属性やプロパティが必要と定義されたグラフィカルな表現の場合には、グラフィカルな表現が使用されなければならない。属性やプロパティがオプションと定義されたグラフィカルな表現の場合には、実装は、グラフィカルな表現またはいくつかの他の仕組みを使用することができる。
もし、グラフィカルな表現を使用するならば、それは定義された表現でなければならない。属性またはプロパティにグラフィカルな表現が定義されていない場合には、実装では一般的なグラフィカルな表現や他のいくつかの仕組みを使うことができる。もし、グラフィカルな表現を使うならば、それは全ての他の DMN 要素のグラフィカルな表現と矛盾してはならない。

3 参照

3.1 基準

BMM - *Business Motivation Model (BMM), Version 1.2*, OMG Document number: formal/2014-05-01, May 2014 http://www.omg.org/spec/BMM/1.2

BPMN 2.0 - *Business Process Model and Notation, version 2.0*, OMG Document Number: formal/2011-01-03, January 2011 http://www.omg.org/spec/BPMN/2.0

IEEE 754 - *IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic*, International Electrical and Electronics Engineering Society, December, 2008 http://www.techstreet.com/ieee/searches/5835853

ISO 8601 - *ISO 8601:2004, Data elements and interchange formats -- Information interchange -- Representation of dates and times*, International Organization for

Standardization, 2004
http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=40874

ISO EBNF - *ISO/IEC 14977:1996, Information technology -- Syntactic metalanguage -- Extended BNF, International Organization for Standardization*, 1996 http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip

Java - *The Java Language Specification, Java SE 7 Edition*, Oracle Corporation, February 2013 http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf

PMML - *Predictive Model Markup Language (PMML)*, Data Mining Group, May, 2014 http://www.dmg.org/v4-2-1/GeneralStructure.html

RFC 3986 - *RFC 3986: Uniform Resource Identifier (URI): Generic Syntax*. Berners-Lee, T., Fielding, R., and Masinter, L, editors. Internet Engineering Task Force, 2005. http://www.ietf.org/rfc/rfc3986.txt

UML - *Unified Modeling Language (UML), v2.4.1*, OMG Document Number formal/2011-08-05, August 2011 http://www.omg.org/spec/UML/2.4.1

XBASE - *XML Base (Second Edition)*. Jonathan Marsh and Richard Tobin, editors. World Wide Web Consortium, 2009. http://www.w3.org/TR/xmlbase/

XML - *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, W3C Recommendation 26 November 2008 http://www.w3.org/TR/xml/

XML Schema - *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation 28 October 2004 http://www.w3.org/TR/xmlschema-2/

XPath Data Model - *XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition)*, W3C Recommendation 14 December 2010 http://www.w3.org/TR/xpath-datamodel/

XQuery and XPath Functions and Operators - *XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)*, W3C Recommendation 14 December 2010 http://www.w3.org/TR/xpath-functions/XQuery

3.2 基準以外

JSON - *ECMA-404 The JSON Data Interchange Standard*, European Computer Manufacturers Association, October, 2013
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

PRR - *Production Rule Representation (PRR), Version 1.0*, December 2009, OMG document number formal/2009-12-01 http://www.omg.org/spec/PRR/1.0/

RIF - *RIF production rule dialect*, Ch. de Sainte Marie et al. (Eds.) , W3C Recommendation, 22 June 2010. http://www.w3.org/TR/rif-prd/

SBVR - *Semantics of Business Vocabulary and Business Rules (SBVR)*, V1.2, OMG document number formal/2013-11-04, November 2013

http://www.omg.org/spec/SBVR/1.2/

SQL - *ISO/IEC 9075-11:2011, Information technology -- Database languages -- SQL -- Part 11: Information and Definition Schemas (SQL/Schemata)*, International Organization for Standardization, 2011
http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=5368

Xpath - *XML Path Language (XPath) Version 1.0*, W3C Recommendation 16 November 1999 http://www.w3.org/TR/xpath

4 追加情報

4.1 謝辞

以下の企業はこの仕様を投稿した：
·Decision Management Solutions
·Escape Velocity
·FICO
·International Business Machines
·Oracle

以下の企業はこの仕様をサポートした:
·KU Leuven
·Knowledge Partners International
·Model Systems
·TIBCO

次の方はこの仕様に貢献したコアチームのメンバーだった：
Martin Chapman, Bob Daniel, Alan Fish, Larry Goldberg, John Hall, Barbara von Halle,

Gary Hallmark, Dave Ings, Christian de Sainte Marie, James Taylor, Jan Vanthienen, Paul Vincent.

さらに次の方はこの仕様のコンテンツの品質向上させる貴重なアイデアやフィードバックで貢献した：
Bas Janssen, Robert Lario, Pete Rivett.

Version 1.1 は次の方または企業によって策定された: Elie Abi-Lahoud, University College Cork; Justin Brunt, TIBCO; Alan Fish, FICO; John Hall, Rule ML Initiative; Denis Gagne, Trisotech; Gary Hallmark, Oracle; Elisa Kendall, Thematix Partners LLC; Manfred Koethe, 88solutions; Falko Menge, Camunda Services GmbH; Zbigniew Misiak, BOC Information Technologies Consulting; Sjir Nijssen, PNA Group; Mihail Popov, MITRE; Pete Rivett, Adaptive; Bruce Silver, Bruce Silver Associates; Bastian Steinert, Signavio GmbH; Tim Stephenson, Omny Link; James Taylor, Decision Management Solutions; Jan Vanthienen, K.U. Leuven; Paul Vincent, Knowledge Partners, Inc.

＊日本語版の謝辞＊

次の団体とコアチームのメンバーは、日本での当専門領域及び関係領域の用語との平仄をとり、この仕様の翻訳品質向上に貢献した：
・IIBA 日本支部、バリューチェーンプロセス協議会(VCPC)、株式会社 KB マネジメント・清水 千博、富士通株式会社・野村 和哉、三菱電機インフォメーションネットワーク株式会社・昆 資之、ニッセイ情報テクノロジー株式会社 押部 幹生、株式会社日立システムズ・藤林 寿、IT コーディネータ・庄司 敏浩、森山 武

4.2 知的所有権と特許

提案者は作業結果を RAND に基づき、RF で、OMG に提供した。

4.3 仕様書のガイド

第 1 章 仕様の目的の概要。
第 2 章 仕様準拠の 3 つのレベルの定義：準拠レベル１、準拠レベル２、準拠レベル３。
第 3 章 仕様のための参照リスト。
第 4 章 仕様の背景と構造を理解するのに役立つ追加の情報の提供。

第 5 章 DMN の範囲を説明する、そして DMN の使用と 2 つのレベルの DMN を含む中心となる概念の導入。
第 6 章 DMN の意思決定の要求レベルの定義：意思決定の要求図（DRG）と意思決定ダイアグラム(DRD)としての表記。
第 7 章 意思決定ロジックは DRG の要素と関係づいても良いという中心概念の導入：つまり、意思決定要求レベルとデシジョン論理レベルはそれぞれどのように関連しているのか。
第 8、9、10 章は DMN の意思決定ロジック・レベルを定義：

・第 8 章 表記とデシジョンテーブルの構文を定義。
・第 9 章 S-FEEL 定義：デシジョンテーブルをサポートする FEEL のサブセット。
・第 10 章 FEEL の全構文と意味を定義：DMN の意思決定ロジックに使われるデフォルトの式言語

第 11 章 シンプルなビジネスプロセスにおける、DMN を用いて人による意思決定と自動化された意思決定をモデル化した例
第 12 章 交換フォーマットと機械可読ファイル(XSD と XMI)への参照を提供（訳注：Addresses exchange format を変換フォーマット訳した）

付録は標準以外の背景情報を提供する：

・付録 A DMN と BPMN 間の関係の説明。
・付録 B 用語集の提供

5 DMNイントロダクション

5.1 コンテキスト

DMNの目的は、意思決定をモデル化するのに必要な構成物を提供することである。それにより図を用いて素早く組織的な意思決定を描くことができ、ビジネスアナリストにより正確に定義でき、そして（オプションで）自動化できる。

意思決定は既存のモデリング標準により次の 2 つの視点から記述される：

・ビジネスプロセス・モデル（例えば BPMN）は、その内部において意思決定が行われる具体的なタスクまたはアクティビティを定義することによりビジネスプロセス内の意思決定のやり方を記述できる。

・意思決定ロジック（例えば、PRR、PMML）は個々の意思決定を行うための具体的なロジックを定義することができる。例えばビジネス・ルール、デシジョンテーブル、または実行可能なアナリティック・モデルなどである。

しかし、本標準の作成者（提出チームのメンバーを含む）は、意思決定が、これらのモデリング視点のいずれにおいても簡単には捕らえられない内部構造を持っているのに気がついた。本書の意図は、DMN がビジネスプロセス・モデルと意思決定ロジック・モデルの間の橋渡しをする 3 番目の視点――意思決定要求ダイアグラム――を提供することである：

・ビジネスプロセス・モデルは、意思決定を要求するビジネスプロセス内のタスクを定義する。

・意思決定要求ダイアグラムは、意思決定ロジックのためのタスク、それらの相互関係、そして意思決定ロジックへの要求を定義する。

・意思決定ロジックは、検証や自動化を可能にするために、要求される意思決定を十分な詳細度で定義する。

意思決定要求ダイアグラムと意思決定ロジックは、共に合わさってプロセス・タスクにおいて実行される意思決定を詳細に指定してビジネスプロセス・モデルを補完する完全な意思決定モデル

を提供できる。
これらの 3 つのモデル間の関係を、図 1 に示す。

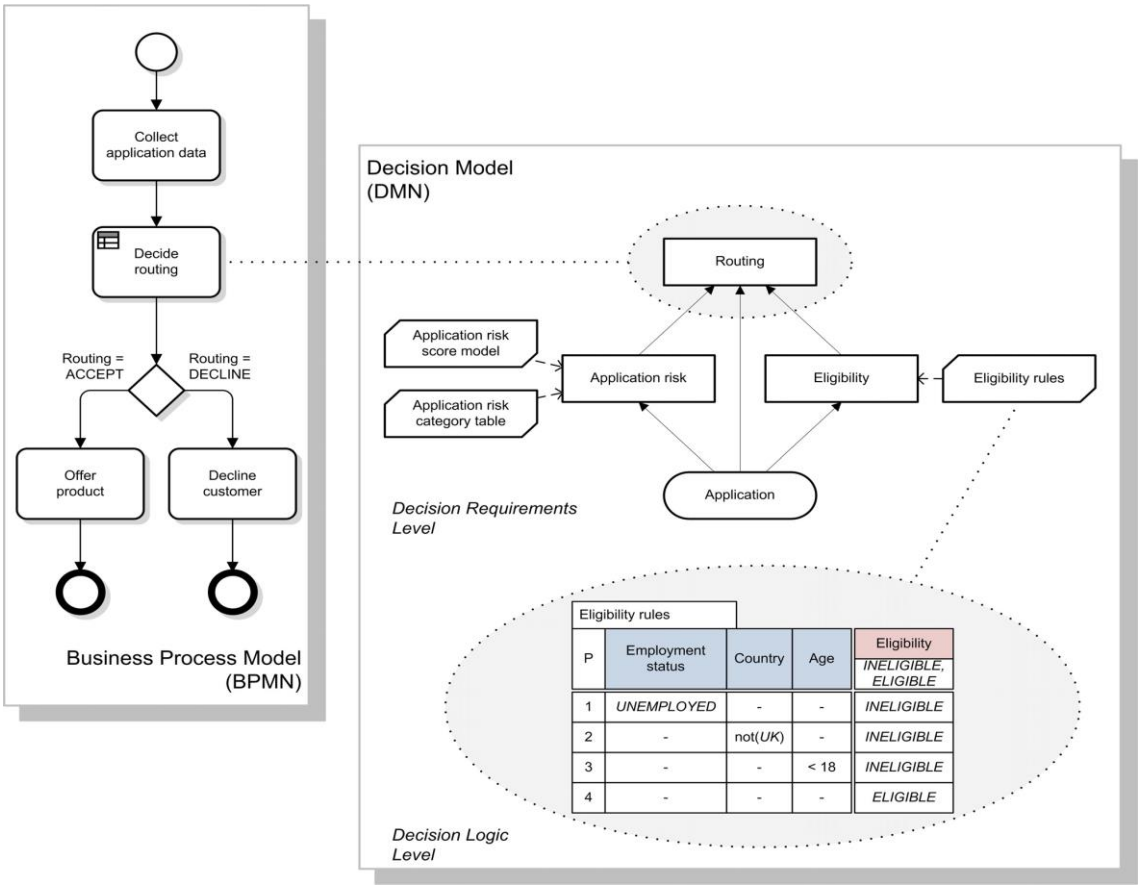


図 1 モデリングの概観

相互に関係するモデルの組合せは、次の詳細なモデリングを可能にする。それはビジネスプロセスにおけるビジネス・ルールの役割とアナリティック・モデル、モデル間の相互検証、トップダウンによるプロセスの設計と自動化、そして意思決定の自動実行（例えば、ビジネス・ルール・マネジメントシステムが提供している意思決定サービスを呼びだしているビジネスプロセス・マネジメント・システムによる自動実行）である。

DMN と他の標準の間の関係を説明するため図 1 はビジネスプロセス・モデルと意思決定モデルの間のリンクを示すが、DMN が BPMN に依存しないことは強調されなければならない。そして、意思決定のドメインをモデル化するため、ビジネスプロセスにリンクせずに 2 つのレベル（意思決定要求と意思決定ロジック）を個別に、または一緒に使用できる（5.2 章を参照）。
DMN は、意思決定要求モデリングと意思決定ロジック・モデリングの両方に関する構成物を提供する。
意思決定要求モデリングでは意思決定要求グラフ（DRG）のコンセプトを定義する。それ

は要素一式と対応する要素間のビジネス・ルールの組み合せによる複数の表記法（意思決定要求ダイアグラム DRD）からなる。

一方、意思決定ロジック・モデリングでは、FEEL と呼ばれる言語を提供する。それはデシジョンテーブルを定義し、計算、if/then/else ロジック、簡単なデータ構造を組み立て、セマンティックを形式的に定義した実行形式に落とせる Java と PMML を使って外部定義されたロジックを提供する。
また、意思決定ロジック（「囲み式」）のための表記法も提供する。それは意思決定ロジック・レベルのコンポーネントをグラフィカルに描画し、意思決定要求ダイアグラムの要素と関連づけることができる。これらの構成物の間の関係を図 2 に示す。

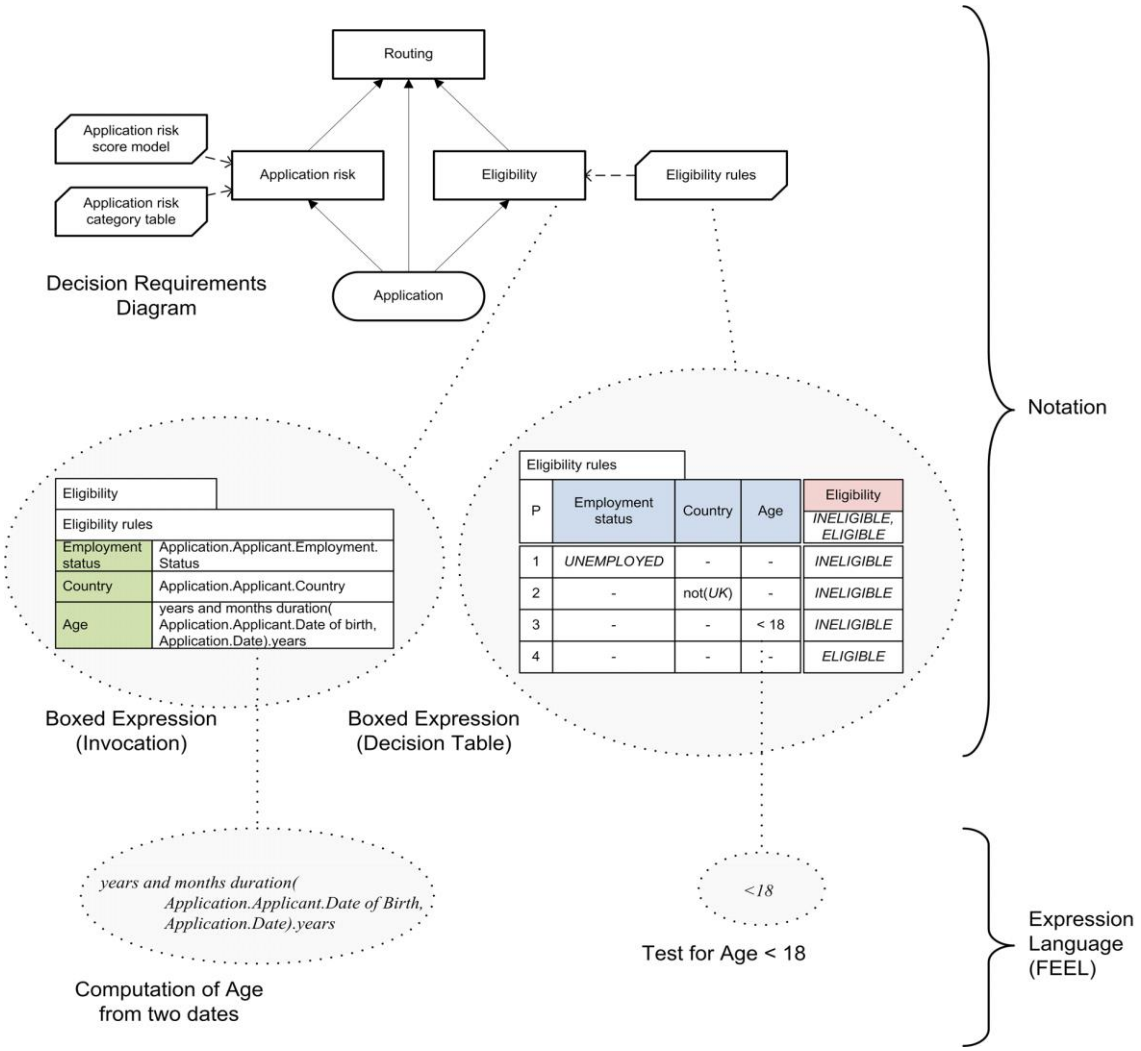


図 2 DMN 構成物

5.2 DMNのスコープと用途

意思決定モデリングは、ビジネスアナリストがビジネスや組織で行われる意思決定を理解し定義するために作成される。このような意思決定は典型的には、日々のビジネスプロセスにおいてなされる業務上の意思決定であり、ルールや表現が少なくて済む戦略的な意思決定ではない。

以上のことから DMN の 3 つの用途が認められる：

1. 人による意思決定のモデリング
2. 自動化された意思決定のための要求のモデリング
3. 意思決定の自動化の実現

5.2.1 人による意思決定のモデリング

DMN は、組織内の人員によりなされる意思決定をモデル化するために使用できる。
人による意思決定は、相互依存する意思決定要素のネットワークに分解され、DRD を使ってモデル化できる。DRD における意思決定はかなり概念的なので、意思決定ロジックというよりも自然言語により記述されるだろう。

知識ソースは意思決定のガバナンスのモデル化を定義することができる。その場合の知識ソースとは、人々（例えばマネジャー）、規制機関（例えばオンブズマン）、文書（例えば、ポリシー冊子）、あるいは立法府（例えば、政府）である。

これらの知識ソースは組み合わされることがある。例えば、その目的は一つの意思決定が（a）規制機関により定義された規制一式による知識ソース、と（b）マネジャーが維持する会社のポリシー文書による知識ソースによって管理されていることを示すためである。

ビジネス知識モデルは、意思決定する際に、使用されるビジネス知識の具体的なエリアを表すことができる。これにより、DMN がナレッジ・マネジメントのための要求のフォーマルな定義のためのツールとして使用できる。ビジネス知識モデルは、知識のエリアの間の相互依存を示すためにリンクできる（既存の「知識構造マッピング」のテクニックにおいて使われるのと同様な方法によって）。
知識ソースは、ビジネス知識の管理方法または維持方法を示すために、ビジネス知識モデルとリンクできる。例えば、全社ポリシー文書（知識ソース）はビジネス・ポリシー・セット（ビジネス知識モデル）を定義できる。

場合によっては、意思決定のための具体的な規則またはアルゴリズムを定義することが可能かもしれない。規則またはアルゴリズムは、DRD を使ったビジネス知識モデルを特定す

るために、意思決定ロジック（例えば、ビジネス・ルールまたはデシジョンテーブル）を使ってモデル化されるかもしれない。それは、叙述的に（どのように現在決定がされるか、またはどのように特定の期間の間にそれらが作られたかを記録するために)、もしくは規範的に（どのように意思決定がされるべきであるか、または、未来に作られるべきかを定義するために)、なされることがある。

DMN で作られた意思決定のモデルが BPMN を使って作成されたビジネスプロセス内のタスクまたはアクティビティにマッピングされることがある。概要レベルでは、協働的な意思決定タスクがグループまたは部門の全体の意思決定アクティビティを表している DRD での意思決定のサブセットにマッピングされることもある。
より詳細レベルでは、BPMN を一緒に使用している多くの個人またはグループによりなされた意思決定の間の相互依存をモデル化できる、すなわち意思決定の各参加者はコラボレーション内の別々のプール、および意思決定モデル内の別々の DRD によって代表される。そしてそれらの DRD での決定がプールのタスクにマッピングされ、DRD への入力データがプール間のメッセージ・パッシングにマッピングされる。

したがって、BPMN と DMN を併用することにより、ビジネスプロセスのアクティビティから意思決定ロジックの詳細な定義まで、組織内の人による複数のレベルの意思決定を説明するためのグラフィカルな言語が提供される。このコンテキスト内で、DMN モデルは協働的な組織的意思決定、そのガバナンス、そのために必要なビジネス知識を記述できる。

5.2.2 自動化された意思決定のための要求のモデリング

自動化された意思決定のための要求をモデル化するために DMN を使用することは、記述的であるというよりも完全に規範的である点を除いては、人による意思決定をモデル化する際の DMN と同様に使用できる。そして詳細な意思決定ロジックがより重視される。

意思決定を完全に自動化するためには、意思決定ロジックは完全でなければならない、すなわち、入力データの値の可能な組合せ全てにおいて、意思決定結果を提供できなければならない。

しかし、部分的な自動化の方がより一般的であり、いくらかの人による意思決定が残る。

人による意思決定と自動化された意思決定の間の相互作用は上記のような併用によりモデル化されることがある。それは人および自動化された意思決定者のためにモデル化されることもあり、人と自動化の意思決定の個別のプールに割り当てることもあり、ま

たはより単純に意思決定をビジネスプロセス・モデルの個別のタスクに割り当てることもある。例えば、ユーザータスクを人の意思決定に割り当て、そしてビジネスルール・タスクを自動化された意思決定に割り当てることもある。それゆえ、自動化されたタスクのための意思決定ロジックは全部指定される必要があるが、レビュアーの意思決定は不特定のままにしておくことが可能である。

DRD での意思決定がいったん BPMN ビジネスプロセスフローのタスクにマッピングされたら、モデルの 2 つのレベルを横断的に確認することが可能である。例えば、ビジネスプロセスにおいて DRD のすべての入力データが先行するタスクによって提供されること、その後続のタスクまたはゲートウェイだけにおいてビジネスプロセスが意思決定の結果を使うことを検証できる。

DMN は意思決定とビジネスプロセスの関係をモデル化し、ビジネスプロセスを完成するために必要な意思決定を特定することができ、また意思決定を指定するか実行する具体的な意思決定タスクを仕様化できる。DMN の ItemDefinition または InputData から BPMN の DataObject へのマッピングは規格上提案されてないが、実装では、そのマッピングができるかのチェックを含めることができる。

したがって、BPMN と DMN を併用すれば、自動化された意思決定のための要求とビジネスプロセス内の人による意思決定との相互作用を仕様化することができる。

これらの要求はどのような詳細度、またはすべての詳細度においても仕様化できる。ビジネスプロセス・モデル、DRD、意思決定ロジックの 3 層マッピングにより、モデル・ベースのコンピュータ支援設計ツールで各層の要求の定義がサポートできるようになる。

5.2.3　自動化された意思決定の実装

もしすべての意思決定とビジネス知識モデルが、意思決定ロジックを使って完全に仕様化できれば、意思決定モデルの実行が可能になる。

例えば、ビジネス・ルール・マネジメント・システム（BRMS）から展開された「意思決定サービス」を使用し、ビジネスプロセス・マネジメント・システム（BPMS）から呼ぶことができる。意思決定サービスは、DRD 内の入力データおよび意思決定のサブセットと一致しているインタフェースを提供して、DRD をサポートしている決定ロジックをカプセル化する。入力データのセットによって呼ばれる時には、意思決定サービ

スは、仕様化された意思決定を評価し、その結果を戻す。すべての意思決定ロジックに副作用がないということは、意思決定サービスが SOA 原則に従うことを意味し、システム設計を簡素化する。

意思決定モデルの構造は、DRD において視覚化されるように、実装を計画するための基礎として使用できるかもしれない。特定のプロジェクト・タスクは、意思決定ロジック（例えば、エキスパートによるルールの発見、または分析的なモデルの作成）の定義、および意思決定モデルのコンポーネントの実装をカバーすることが含まれることがある。

意思決定サービスにおいてカプセル化されたビジネス知識を表しているいくつかの意思決定ロジックは、意思決定の責任者により特別な「知識メンテナンス・インタフェース」を使用して長期的に維持されなければいけない。DMN は知識メンテナンス・インタフェースの効果的なデザインと実装をサポートする。メンテナンスを必要とするいかなるビジネス知識も DRD でモデル化されるべきであり、また信頼できる人員も知識ソースとしてモデル化されるべきである。そして DRD は、必要な知識メンテナンス・インタフェースとそのユーザーの仕様を提供し、意思決定ロジックは、維持されるビジネス知識の初期のコンフィギュレーションを指定する。

他の意思決定ロジックは通常のアナリティカル・モデリングによってリフレッシュされる必要がある。DMN の機能としてのビジネス知識モデルの表現は、非常に簡単な意思決定サービスにおいてアナリティカル・モデルを使用する。すなわち機能として表現が可能ないかなるアナリティカル・モデルも、直接呼ばれるか、または意思決定サービスにインポートできる。

5.2.4　モデリングの適用の組み合わせ

上記に説明された 3 つのコンテキストは、相互に排他的な選択肢ではない。大規模なプロセスオートメーションプロジェクトは DMN のすべてを次の 3 つの方法で使うことができる。

第一に、既存のプロセス内の意思決定は、現在の意思決定の完全な範囲および関係するビジネス知識のエリアを特定するために作成できる。この「AsIs」分析によりプロセス改善のためのベースラインが提供される。

次に、このプロセスは、自動化と人による意思決定の中で最も効果的に使用するために、

しばしば二者間の協働により再設計できる（例えば、人の意思決定者への自動照会の使用、あるいはユーザーに助言や制約を課す自動化された意思決定支援システムの使用）。この再設計は、組織で個々のプロセス・タスクおよび個人またはグループの役割と責任に存在するような意思決定のための要求をモデル化することを含む。このモデルは、必要なプロセスおよびそれが関与する意思決定の「ToBe」仕様を提供する。

「AsIs」および「ToBe」モデルの比較により、自動化技術のために要求が示されるだけではなく、人の役割・責任の変更および新しいまたは修正されたビジネス知識をサポートするトレーニング、すなわちチェンジ・マネジメントのためにも要求が示される。

そして、「ToBe」モデルは実行可能なシステム・ソフトウェアとして実装される。提供される意思決定ロジックは、FEEL や他の外部ロジック（例えば、外面的に定義された Java または PMML モデル）において完全に仕様化され、意思決定モデルのコンポーネントはソフトウェア・コンポーネントとして直接実装できるかもしれない。

DMN は、どのような特定の手法も、上記の活動を実行することも規定しない。DMN はそのために使われるモデルをサポートするだけである。

5.3 基本コンセプト

5.3.1 意思決定要求レベル

「意思決定」という言葉は一般的な使用において、2 つの定義がある。意思決定は複数の可能な選択肢の中から選択する行為を示すことがある。あるいは、選択される選択肢そのものを示すこともある。これら定義の中で、前者の方法を採用する。すなわち、意思決定は、アウトプットがどのようにインプットから決定されるかを定義するロジックを使って、多くのインプット値からアウトプット値（選ばれた選択肢）を決定する行為である。この意思決定ロジックは、ビジネス・ルール、アナリティカル・モデル、または他の方法でビジネス・ノウハウをカプセル化するビジネス知識モデルを含むことがある。すべての意思決定モデルが作成されているこの基本的な構造を図 3 に示す。

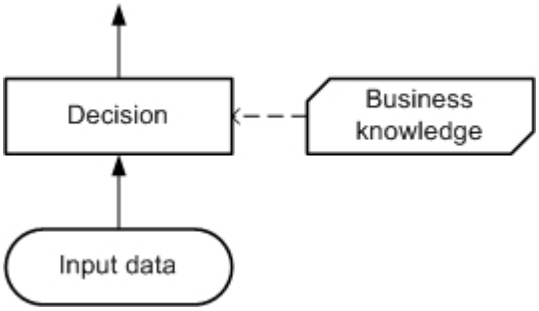


図 3 意思決定モデルの基本要素

シンプルにかつ普遍に表現するために、本仕様書の図の多くは、関連するひとつのビジネス知識モデルをもつものとしての個々の意思決定を示しているが、DMN がそれを要求していないことに注意するべきである。

意思決定ロジックをカプセル化するビジネス知識モデルの使用は、スタイルと方法の問題であり、意思決定は、関連したビジネス知識モデルがあってもなくてもモデル化できる。

意思決定またはビジネス知識モデルのために根拠を定義することがある。根拠はビジネス知識モデルを定義または維持することに責任がある業務領域の専門家であることもあればビジネス知識モデルが導き出される大元の文書のこともある。または、意思決定と整合したテストケースのセットのこともある。これらは知識ソースと呼ばれる（図 4 を参照）。

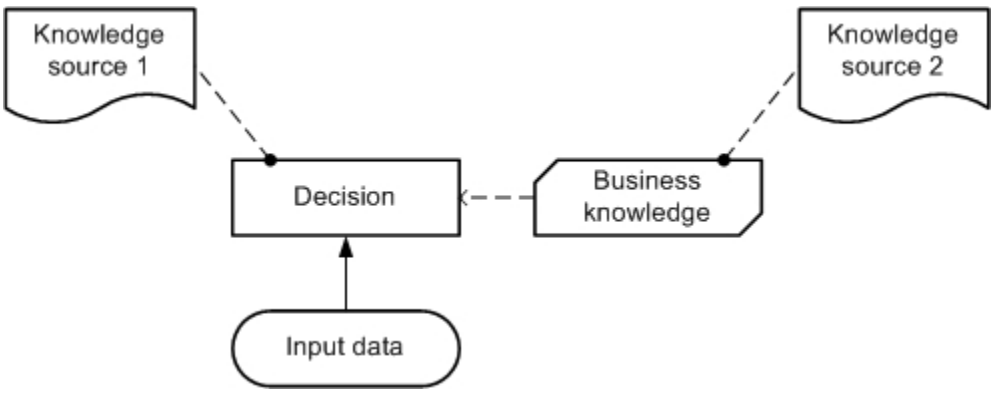


図 4 知識ソース

意思決定は、そのアウトプットを決定するために、そのインプットを「要求する」と言われる。インプットは入力データ、または他の決定のアウトプットであるかもしれない。（どちらのケースでも、インプットは単なる簡単なデータ項目というよりもデータ構造

のこともある。）もし意思決定 Decision1 のインプットが別の意思決定 Decision2 のアウトプットを含むならば Decision1 は Decision2 を「要求する」。

したがって、
意思決定は、意思決定要求グラフ（DRG)」と呼ばれるネットワークのなかで接続されることがある。そして「意思決定要求グラフ（DRG)」は「意思決定要求ダイアグラム（DRD)」として描かれることがある。DRD は、意思決定セットの相互依存性、入力データへの依存性、またビジネス知識モデルへの依存性を示す。二つの意思決定を持つ DRD の簡単な例を図 5 に示す。

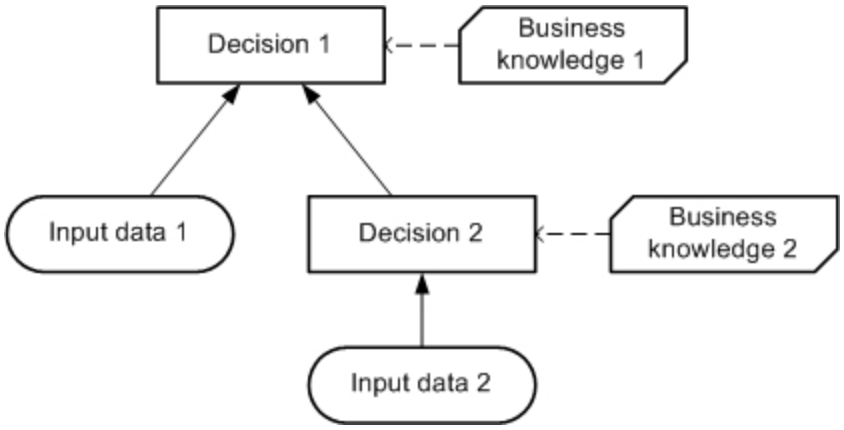


図 5 単純な意思決定要求ダイアグラム（DRD）

図 6 に示すように、ひとつの意思決定は複数のビジネス知識モデルを要求することもあり、またひとつのビジネス知識モデルは他の複数のビジネス知識モデルを要求することもある。これは、（例えば）多様な領域のビジネス知識を結びつけることと、様々な状況で意思決定ロジックを使用するためにいくつかのバージョンの意思決定ロジックを提供することによって、複雑な意思決定ロジックのモデリングを可能とする。

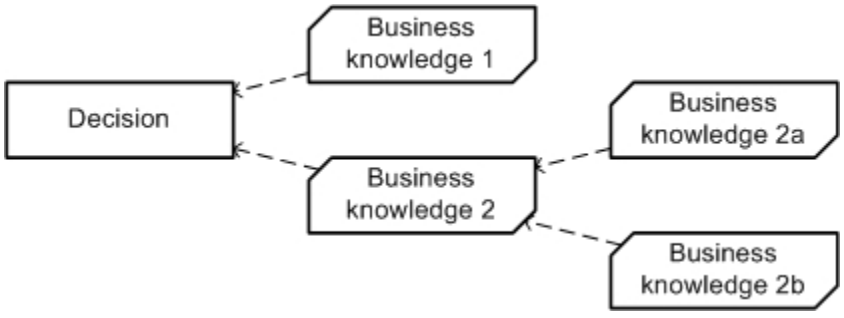


図 6 ビジネス知識モデルの結合

DRD としての DRG およびそれらの表記法は、第 6 章において詳細に仕様化される。

5.3.2 意思決定ロジック・レベル

上記のように、意思決定モデルの意思決定要求レベルのコンポーネントは、ビジネスコンセプトだけを使って説明できる。意思決定領域のビジネスアナリシスではこのレベルの説明で十分なことが多く、関係するビジネス上の意思決定、それに必要な相互関係、ビジネス知識のエリアとデータ、ビジネス知識のソースを識別する。意思決定ロジックを使うことによって、同じコンポーネントは、ビジネス・ルールと計算の完全なセットにするために、より詳しく仕様化できる。（もし望むならば）意思決定を完全に自動化できる。

意思決定ロジックは、どのように意思決定モデルの要素を表示するかについての追加の情報を提供することもある。例えば、デシジョンテーブルのための意思決定ロジックの要素によってルールを行で示すか列で示すかを指定できることがある。計算のための意思決定ロジックの要素によって用語を垂直に並べるか水平に並べるかを指定できることがある。

意思決定要求レベルのコンセプトと意思決定ロジック・レベルのコンセプトとの対応を下記に説明する。図 7、図 8 にあるように灰色の楕円と点線は、種々のレベルの意思決定間の対応を紹介するためだけに描かれていることに注意すること。それらは、6.2、8.2、10.2 で定義されている DMN 表記法の正式な定義に含まれるものではない。実装が、モデリングのレベルの間を移動するための「オープニング」「ドリル・ダウン」または「ズーム・イン」などの便利な機能を提供することは考慮しているが、DMN は、どのように行うべきかまでは指定しない。

意思決定ロジック・レベルでは、DRG でのすべての決定は、どのように決定のアウトプットがそのインプットから決定されるかを指定する「値式」を使って定義される。そのレベルでは、意思決定は、表現の評価であると考えられる。「値式」は図 7 に例示するような「囲み式の表現」を使って表記されることがある。

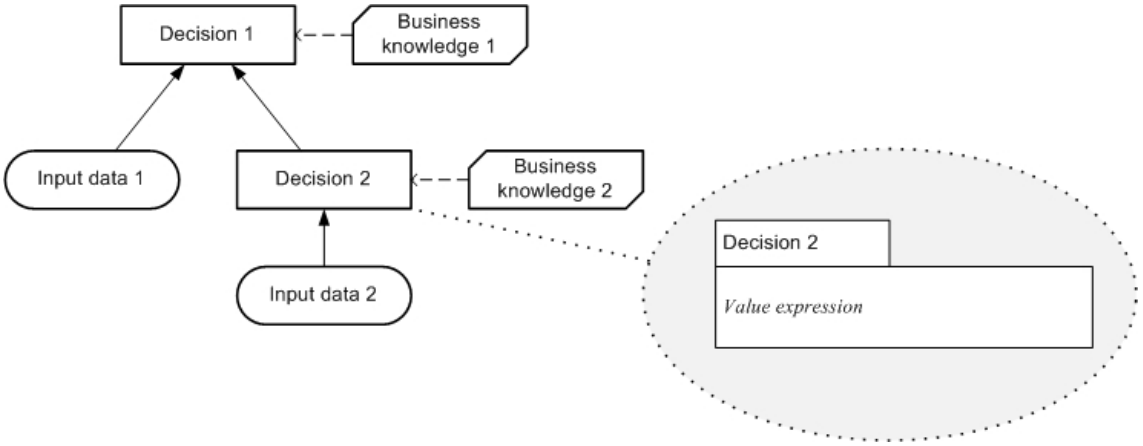


図 7 意思決定と対応する値式

同じように、意思決定ロジック・レベルでは、アウトプットがインプットの集まりからどのように決定されるかを指定する値式を使ってビジネス知識モデルが定義される。ビジネス知識モデルでは、値式は関数定義としてカプセル化される。この関数定義は意思決定の値式から呼び出されることがある。

DMN 内の関数としてのビジネス知識モデルの逐次解釈は、図 6 におけるビジネス知識モデルの結合が関数合成の明確なセマンティクスをもつことを意味する。 ビジネス知識モデルの値式は図 8 に例示するように、「囲み式」を使って表記されることがある。

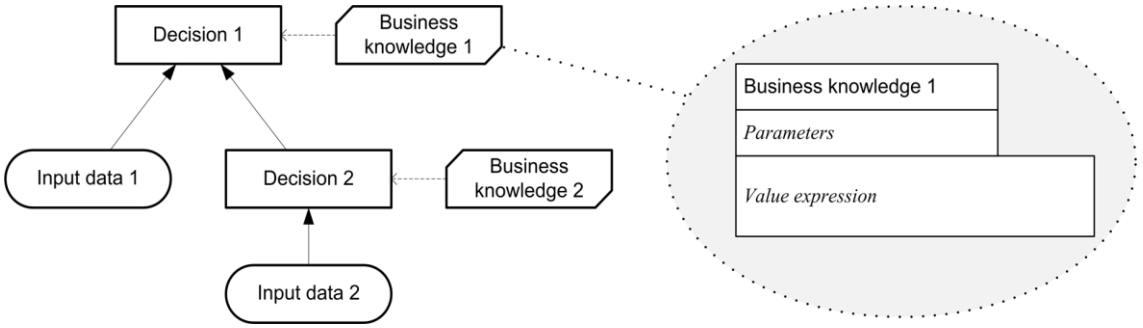


図 8 ビジネス知識モデルと対応する値式

ビジネス知識モデルには、関数として表現できるどのような意思決定ロジックを含めてもよい。これによって、多くの既存の意思決定ロジック・モデリング標準（例えばビジネスルールや分析モデル）から DMN にインポートできる。特に DMN においてサポートされる、ビジネス知識の重要な形式は、デシジョンテーブルである。デシジョンテーブルを使って表記されたビジネス知識モデルを図 9 に示す。

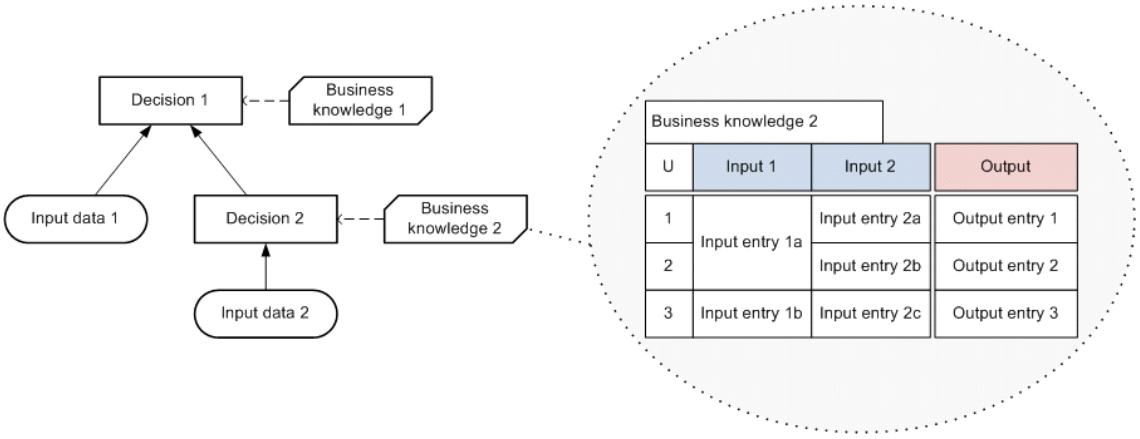


図 9 ビジネス知識モデルと対応するデシジョンテーブル

ほとんどの場合において、意思決定ロジックはビジネス知識モデルにカプセル化される。そして意思決定に関連する値式により、どのようにビジネス知識モデルが起動され、意思決定のアウトプットを計算するために起動の結果がどのように結合されるかが指定される。意思決定の値式は、ビジネス知識モデルを起動することなく、インプットからアウトプットがどのように決定されるかも指定できる。このような場合、ビジネス知識モデルは意思決定と全く関連しない（その場合、意思決定要求レベルでも意思決定ロジック・レベルでも関連しない）。

上記すべての概念を含む DMN の意思決定ロジックを定義するための式言語である FEEL（the Friendly Enough Expression Language）は、10 章において仕様化される。デシジョンテーブルのための表記法は 8 章において詳細に仕様化される。

5.3.3 意思決定サービス

DMN の重要な用途の 1 つは、「意思決定サービス」を使用して意思決定ロジックの自動化を定義することである。意思決定サービスは、意思決定モデルから 1 つ以上の意思決定をサービスとして公開する。 BPMN プロセスモデル・サービスが呼び出されると、必要なインプットデータと意思決定結果とともに、公開された意思決定のアウトプットが返される。DMN 意思決定モデルをカプセル化する意思決定サービスはステートレスで副作用もない。例えばウェブサービスとして実装可能である。 DMN は、そのようなサービスをどのように実装すべきかを規定していないが、意思決定モデルに対してサービスの機能を定義することは可能である。

その前提は、クライアントが一連の意思決定を必要とすることと、サービスがその要件を満たすように作成されていることである。 意思決定サービスの唯一の機能は、その

意思決定セットを評価した結果（「アウトプット意思決定」）を返すことである。サービスでは、サービスの外部で評価された意思決定の結果（「インプット意思決定」）が提供されることがある。
サービスは、アウトプット意思決定をカプセル化しなければならない。のみならずインプット意思決定では提供されないアウトプット意思決定によって（直接的または間接的に）要求される DRG の意思決定もカプセル化しなければならない（「カプセル化意思決定」）。

デシジョン・サービスへのインタフェースは、次のもので構成される。
・インプットデータ：カプセル化された意思決定に必要なすべてのインプットデータのインスタンス。
・インプット意思決定：すべてのインプット意思決定の結果のインスタンス。
・アウトプット意思決定：提供されたインプット意思決定およびインプットデータを使用して、（少なくとも）アウトプット意思決定すべてを評価した結果。

サービスが呼び出され、インプットデータとインプット意思決定を提供すると、アウトプット意思決定が返される。

注意すべきは、意思決定サービスを定義するのに必要なのは、アウトプット意思決定の指定と、インプット意思決定またはカプセル化意思決定のいずれかの指定だけでよいことである。残りの属性（必須のインプットデータ、およびカプセル化またはインプットが指定されていない意思決定）は、サービス定義に使用されている意思決定モデルから推論できる。あるいは、必要以上に多くの属性が定義されている場合は、それらを意思決定モデルに対しての妥当性を確認することができる。

図 10 は、3 つの意思決定を含む意思決定モデルに対して定義された意思決定サービスを示す。このサービスのアウトプット意思決定は{意思決定 1}であり、インプット意思決定は{}である。

意思決定 1 の結果が返され、外部意思決定の結果は提供されない。意思決定 1 はサービスにインプットとして提供されない意思決定 2 を必要とするため、サービスも意思決定 2 をカプセル化する必要がある。意思決定 3 はカプセル化する必要はない。したがって、カプセル化された意思決定は{意思決定 1、意思決定 2}である。このサービスでは、インプットデータ 1 とインプットデータ 2 が必要だが、インプットデータ 3 は必要ではない。

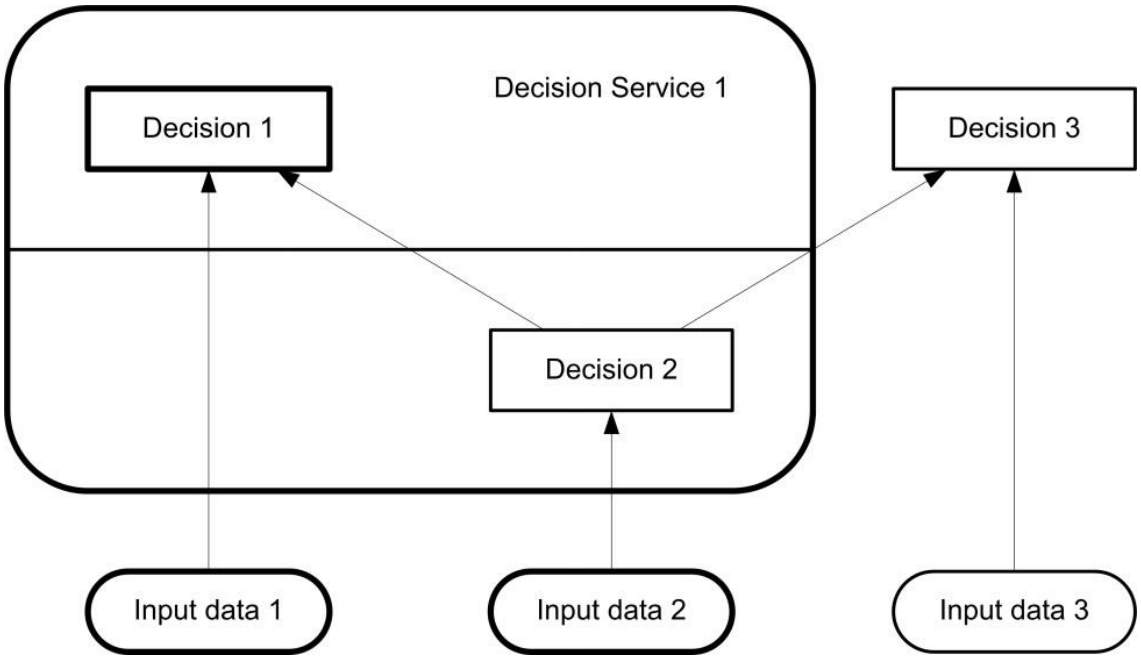


図 10 意思決定サービス

同じ意思決定モデルに対して複数の意思決定サービスを定義することができる。 図 11 は、アウトプット意思決定が{意思決定 1}であり、インプット意思決定が{意思決定 2}である同一の意思決定モデルに対して定義される意思決定サービスを示す。
このサービスのカプセル化された意思決定は{意思決定 1}である。 サービスにはインプットデータ 1 が必要だが、インプットデータ 2 およびインプットデータ 3 は必要ない。

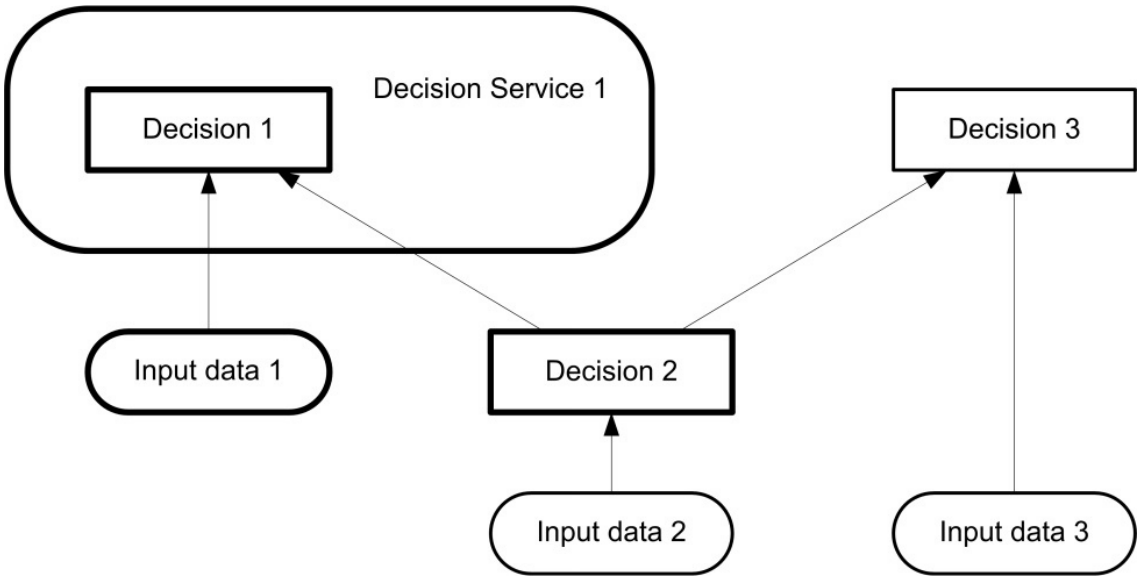


図 11 意志決定をインプットとして受け入れる意思決定サービス
最も単純な形式では、デシジョン・サービスはアウトプット・セット内のすべての意思

決定を常に評価し、すべての結果を返す。
計算効率のために、この基本的な解釈に対する様々な改善が想定され得る。例えば、

- 「要求された意思決定」(最小出力セットのサブセット) のリストを指定するオプションのインプット・パラメータ。要求された意思決定の結果のみがアウトプット・コンテキストで返される。

- 「既知の意思決定」(カプセル化セットのサブセット) のリストとその結果を指定するオプションのインプット・パラメータ。 意思決定サービスはこれらの意思決定を評価せず、提供されたインプット値を直接使用する。

そのような実装の詳細はすべて、ソフトウェア・プロバイダに委ねられている。

意思決定サービスが「完了」するのは、すべての可能なインプットデータ値について、すべてのカプセル化された意思決定を評価するための意思決定ロジックを含む場合である。サービスに対して要求が「有効」であるのは、評価が必要な意思決定によって必要とされる入力意思決定および入力データについてインスタンスが提供される場合である。例えば（単純な場合には）すべてのカプセル化された意思決定、または（上記のオプション・パラメータを仮定して）要求された意思決定およびそれらの要求されたサブ意思決定がまだ判明していないものである。

6 要求（DRGおよびDRD）

6.1 イントロダクション

DMNにおける意思決定モデルの意思決定要求レベルは、意思決定要求ダイアグラム（DRD）によって記述される意思決定要求グラフ（DRG）から構成される。
DRGは、意思決定のドメインをモデル化し、意思決定における最重要な要素と、その要素間の依存関係を可視化する。モデル化された要素とは、意思決定、ビジネス知識、ビジネス知識のソース、インプットデータである。

- **意思決定**の要素は、複数のインプットからのアウトプットを決定する行為を表す。意思決定には、ビジネス知識モデルを参照することのある意思決定ロジックを用いる。
- **ビジネス知識モデル**の要素は、ビジネス知識を内包した機能を表す。ビジネス知識の例としては、ビジネス・ルール、デシジョンテーブル、分析モデルなどがある。
- **インプットデータ**の要素は、意思決定でインプットとして使われる情報を表す。
- **知識ソース**の要素は、ビジネス知識モデルや意思決定に対する根拠を表す。

これらの要素間の依存関係は、３種類の要求として表現される。すなわち、情報、知識、根拠である。

- **情報要求**は、ある意思決定へのインプットとして使用されるインプットデータまたは他の意思決定のアウトプットを表す。
- **知識要求**とは、決定での意思決定ロジックによるビジネス知識モデルを呼び出すものを表す。
- **根拠要求**とは、あるDRG要素が他のDRG要素のガイダンスまたは知識のソースとして振る舞う依存関係を表す。

DRDは、ダイアグラムの注釈を表す成果物を任意の個数持ってもよい。

- 注釈は、説明のためにモデラーが入力するテキストである。
- 関連は、注釈とDRG要素を結ぶものであり、点線で表される。

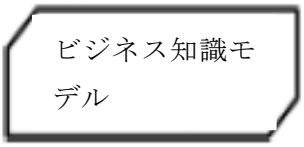
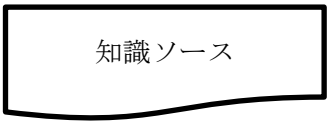
これらの要素を表１にまとめた。また、その詳細については、6.2項で述べる。

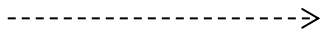
DRG は、要求を表す線によって連結された要素から成る図であり、DRG 内のすべての意思決定に対してモデル化された全要求が同じ DRG 内にある（意思決定の情報、知識、根拠に対する直接のソースがすべてある）という点で自己完結的である。DRG の定義と特定のビューを表現する DRD を区別することが重要である。部分的あるいは取捨選択した表示については、6.2.4 項を参照のこと。

6.2 表記法

DRD のすべての部品の表記法を表 1 にまとめた。以降でさらに詳述する。

表 1 DRD 部品

部品		説明	表記法
要素	意思決定	意思決定の要素は、複数のインプットからのアウトプットを決定する行為を表す。意思決定には、ビジネス知識モデルを参照することのある意思決定ロジックを用いる。	
	ビジネス知識モデル	ビジネス知識モデルは、ビジネス知識（例えば、ビジネス・ルール、デシジョンテーブル、分析モデルなど）を内包した機能を表す	
	インプットデータ	インプットデータの要素は、意思決定でインプットとして使われる情報を表す。知識モデルに埋め込まれた場合は、その知識モデルのパラメータであることを表す。	
	知識ソース	知識ソースは、ビジネス知識モデルや意思決定に対する根拠を表す。	

要求線	情報要求線	情報要求線は、ある意思決定の入力の一つとして用いられるインプットデータまたは他の意思決定のアウトプットを表す。	
	知識要求線	知識要求線は、意思決定での意思決定ロジックによるビジテキスト注釈は角括弧とそれに続くモデラー入力の説明テキストまたはコメントから成ります。ネス知識モデルを呼び出すものを表す。	
	根拠要求線	根拠要求線とは、あるDRD要素に対して別のDRD要素がガイダンスや知識のソースとして振る舞う依存関係を表す。	
成果物	注釈	注釈は角括弧とそれに続くモデラー入力の説明テキストまたはコメントから成る。	
	関連	関連は、注釈を説明またはコメントしているDRG要素にリンクする。	

6.2.1 DRD 要素

6.2.1.1 意思決定表記法

意思決定は、表 1 にあるように、実線の四角形として DRD において表現される。名前を表示することで各意思決定をラベル付けできるように実装しなければならない。また、質問や説明などの他のプロパティを表示することでラベル付けできるように実装してもよい。その場合には、同じ DRD 内の他のすべての DRD 要素のラベルと区別できなければならない。また、ラベルは DRD 要素の図形の内側に明瞭に配置しなければならない。

インプットデータをリスト表示するオプションを用いた場合には（6.2.1.3 インプットデータ表記法 参照)、すべての意思決定の要求線に対するインプットデータは、図 12 に示すように、意思決定のラベルの下に水平線で明確に区切って表示しなければならない。リスト化したインプットデータ名は、DRD 要素の図形の内側に明瞭に配置しなければならない。

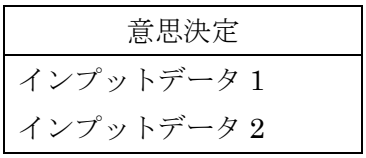


図 12 インプットデータをリスト表示するオプションを使用した意思決定

意思決定のプロパティは、6.3.8「ビジネス・コンテキスト要素メタモデル」で列挙し説明する。

6.2.1.2 ビジネス知識モデル表記法

ビジネス知識モデルは、表 1 にあるように、実線を使って描かれ、二つの角が削られた四角形として DRD において表現される。各ビジネス知識モデルに対してその名前を表示することでラベル付け出来るよう実装しなければならない。また、説明などの他のプロパティを表示することでラベル付けして実装してもよい。その場合には、同じ DRD 内の他のすべての DRD 要素のラベルと区別できなければならない。また、ラベルは DRD 要素の図形の内側に明瞭に配置しなければならない。

ビジネス知識モデルのプロパティは、6.3.8「ビジネス・コンテキスト要素メタモデル」で列挙し説明する。

6.2.1.3 インプットデータ表記法

インプットデータの要素は、表 1 にあるように、実線を使って描かれ、左右両端が半円で上下が平行線である図形として DRD において表現される。各インプットデータの要素に対してその名前を表示することでラベル付け出来るよう実装しなければならない。また、説明などの他のプロパティを表示することでラベル付けして実装してもよい。その場合には、同じ DRD 内の他のすべての DRD 要素のラベルと区別できなければならない。また、ラベルは DRD 要素の図形の内側に明瞭に配置しなければならない。

インプットデータに対する要求を表示するという代替の方法は、DRD が大きかったり複雑であったりしたときに特に有用である。この方法では、インプットデータは DRD において独立した要素としては描かず、そのインプットデータを必要とする意思決定要素の中にリスト表示する。本書ではこの方法を便宜上、「リスト表示インプットデータ」オプションと呼ぶ。
実装では、このオプションを提供してもよい。図 13 は二つの同等の DRD を示している。一方は、インプットデータを独立した要素として描き、他方は、「リスト表示インプットデータ」オプションとして描いている。もしも、インプットデータを要素として表示しない場合は、そのインプットデータを必要とするすべての意思決定にリスト表示しなければならないことに注意せよ（6.2.4「部分ビューおよび隠ぺい情報」で論じるように、それが故意に隠されない限りにおいて）。

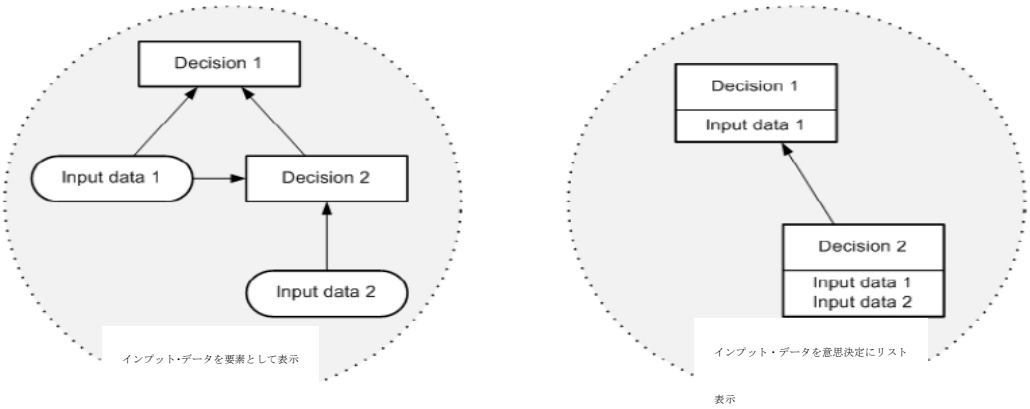


図 13 リスト表示されたインプットデータ オプション

インプットデータのプロパティは、6.3.10 で列挙し説明する。

6.2.1.4 知識ソース表記法

知識ソースは、表 1 にあるように、標準的に実線によって描かれ、三方が直線で下側が波型の図形として DRD において表現される。実装では、各知識ソースをその名前によりラベル付けしなければならない。また、説明などの他のプロパティによりラベル付けしてもよい。その場合には、ラベルは同じ DRD 内の他のすべての DRD 要素のラベルと区別できなければならない。また、DRD 要素の図形の内側に明瞭に配置しなければならない。

知識ソースのプロパティは、6.3.11 で列挙し説明する。

6.2.2 DRD 要求

6.2.2.1 情報要求線表記法

情報要求線は、インプットデータ要素から意思決定へ引くこともあれば、意思決定から別の意思決定に引くこともある。情報要求線は、インプットデータやある意思決定の結果に他の意思決定が依存していることを表している。また、データ・フローとして解釈されることもある。意思決定とインプットデータと情報要求だけを表示している DRD は、意思決定を評価する時点でのこれらの要素間の情報伝達を示すデータ・フロー図と同等である。妥当な DRG の情報要求は、有向非巡回グラフとなる。

情報要求は、表 1 にあるように、先端が「▲」形の実線の矢印(solid arrowhead)として DRD において表現される。矢印は情報の向きで描かれる、すなわち、その情報を必要とする意思決定に向けて描かれる。

6.2.2.2 知識要求線表記法

知識要求線は、ビジネス知識モデルから意思決定へ引くこともあれば、ビジネス知識モデルから別のビジネス知識モデルに引くこともある。知識要求線は、意思決定するときにビジネス知識の起動を表している。また、関数の呼び出し（ファンクションコール）として解釈されることもある。意思決定とビジネス知識モデルと知識要求線だけ表示している DRD は、意思決定を評価に関わる関数呼び出しを示す関数の階層構造と同等である。妥当な DRG の知識要求は、有向非巡回グラフとなる。

知識要求線は、表 1 にあるように、先端が「∧」形(open arrowhead)の破線の矢印として DRD において表現される。矢印は、関数を評価する結果の情報の向きで描かれる。すなわち、そのビジネス知識を必要とする要素に向けて描かれる。

6.2.2.3 根拠要求線表記法

根拠要求線は、二つの方法で使用される。

a. 知識ソースから意思決定やビジネス知識モデル、他の知識ソースに向けて引く。これは、ある DRD 要素が、その知識ソースに依存していることを表す。これは、ひとまとまりのビジネス・ルールが発行済の文書（たとえば、ある法律あるいはビジネス・ポリシー記述書）と一致しなければならないことや、特定の人物や組織グループが意思決定ロジックを定義することに対して責任を持つという事実を記録するために使用されることがある。また、意思決定がある人物やグループによってマネジメントされることを記録するために使用されることもある。知識ソースのこの使用例を図 14 に示す。この使用例では、ビジネス知識モデルが二つの根拠を必要とする。すなわちポリシー文書と法律を必要とし、そのポリシー文書がポリシー・グループを根拠として必要とすることを表している。

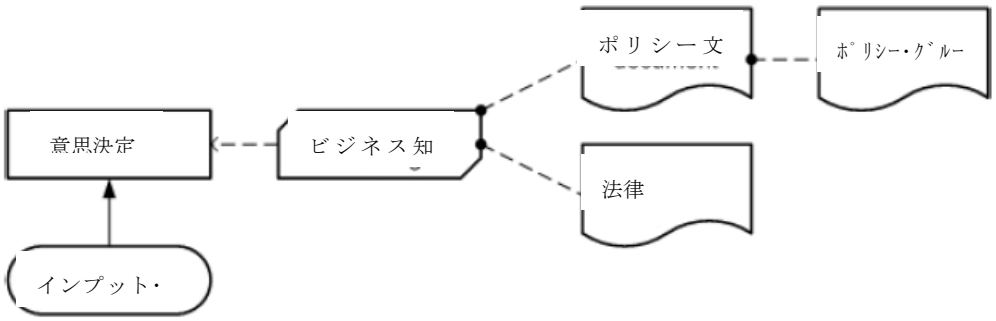


図 14 根拠を表す知識ソース

b. インプットデータや意思決定から知識ソースに向けて引く。これは、a の使用法と併せて、インプットデータや意思決定結果のインスタンスから、アナリティクスを使用してビジネス知識モデルが導出されることを表す。知識ソースは、典型的には分析モデル（あるいはモデリング・プロセス）を表す。すなわち、ビジネス知識モデルは、分析モデルから直接生成される、あるいは分析モデルに基づいて間接的に生成される実行可能なロジックを表す。この知識ソースの使用例を図 15 に示す。このケースでは、ビジネス知識モデルは、ある分析モデルに基づいており、その分析モデルは、インプットデータとそれに基づく意思決定の結果から導きだされることを表している。

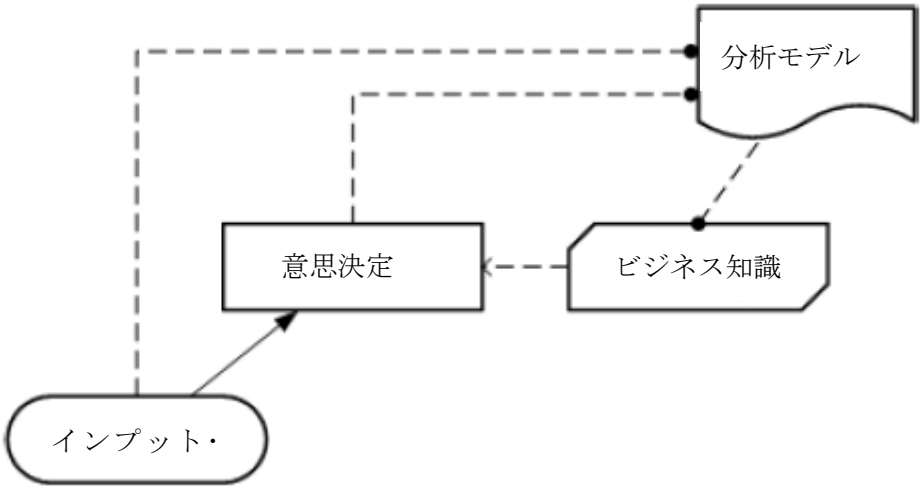


図 15 予測分析を表す知識ソース

ただし、上図はあくまで例である。根拠要求線の使用例は他にも多くありうる（知識ソースと根拠要求線は、実行セマンティックを持たないため、その解釈は、必然的に曖昧になる)。そのため、この仕様では適用の細部は実装者に委ねられている。

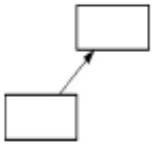
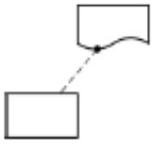
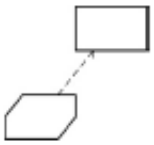
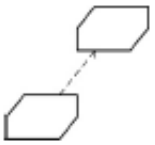
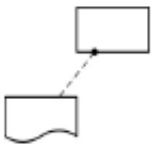
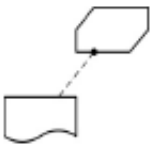
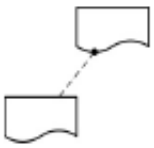
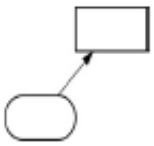
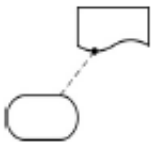
根拠要求線は、表 1 にあるように、先端が「●」形(filled circular head)の破線として DRD において表現される。線は根拠のソースからその根拠によって統制される要素に向けて引かれる。

6.2.3 連結ルール

DRD における要素と要求線の間で許される連結方法の規則は、6.2.2「DRD 要求」に記載されており、表 2 にまとめられている。それを明確にするために、許された各々の連結による単純な DRD が示されている。それぞれの図の中では、上部（連結先）の要素は、下部（連結元）の要素を必要とする。

インプットデータを連結先とする要求線は描かれない、さらにインプットデータには要求線がないことがある。また、要求線のタイプは、二つの要素の連結によって一意に決定されることに注意する。

表 2 要求線連結ルール

		To 連結先			
		意思決定	ビジネス知識モデル	知識ソース	インプット・データ
連結元	意思決定	 情報要求線	許されない	 根拠要求線	許されない
	ビジネス知識モデル	 知識要求線	 知識要求線	許されない	許されない
	知識情報源	 根拠求線	 根拠要求線	 権威要求	許されない
	インプット・データ	 情報要求	許されない	 権威要求	許されない

6.2.4 部分的なビューと隠された情報

メタモデル（「6.3 メタモデル」参照）は、DRG の各要素にプロパティを提供する。このプロパティは、通常、DRD には表示されないが、各要素の特性や機能についての付加情報を提供する。たとえば、ある意思決定に対して、どの **BPMN** プロセスとタスクがその意思決定を使用するか、を規定するプロパティを含んでいる。このようなプロパティを特定したり表示したりするための実装手段を提供しなければならない。

意思決定のどのような重要なドメインに対しても、DRG を完全に表現する DRD は、大きくて複雑な図になることがある。DRG の部分的なビューやフィルター経由のビューによる DRD を表示できる実装手段を提供してもよい。たとえば、要素のカテゴリーカテゴリーを隠すことや、ネットワークの領域を隠したり折りたたんだりすることで。**DMN** は、そのようなビューをどのように表現したらよいかについては規定しない。しかし、情報が隠されている場合は、そのことを明示的に視覚的に表示する実装機能を提供しなければならない。

DRG の部分的なビューを提供する二つの DRD の例を図 16 に表す。DRD1 は、一つの意思決定に直接関係する要求線だけを表示している。DRD2 は、情報要求線と、その情報要求線が直接連結している要素だけを表示している。この例では、説明のために、隠れた要求線を持つ要素だけを破線で示している。

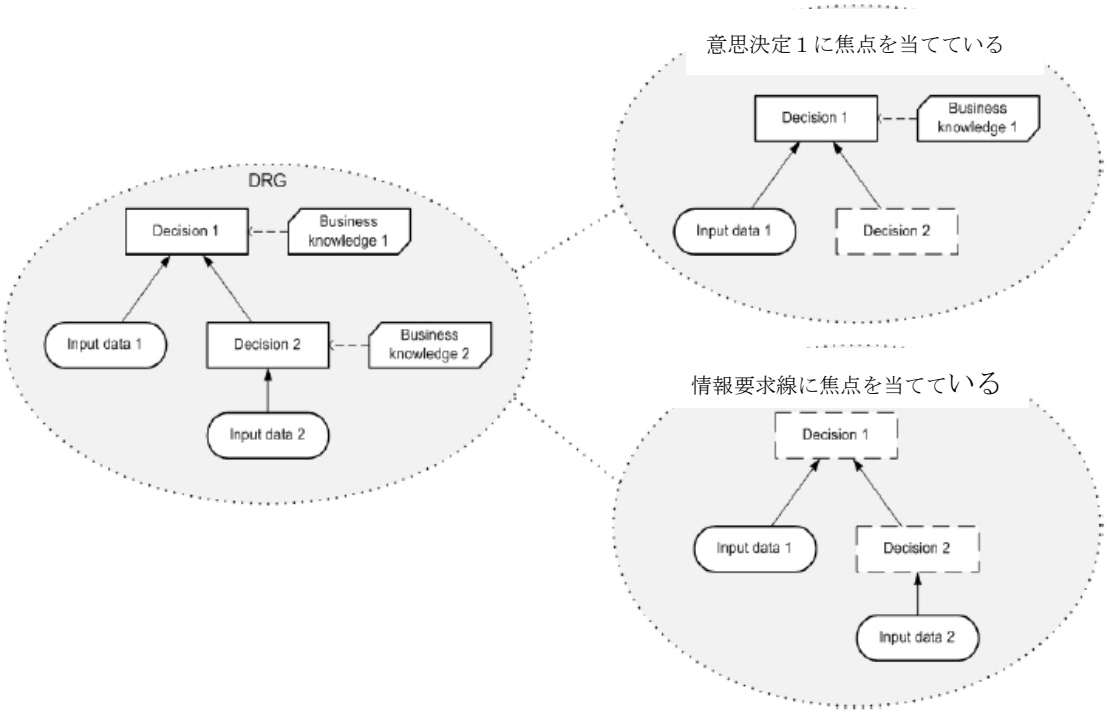


図 16 DRG の部分的なビューとしての DRD

DRD はメタモデルに現れない。したがって、(提供者が異なる様な 2 つの実装間で DRD を交換する場合に) DRD が交換されないことがある。その様な場合には DRG を含む定義群を交換してもよいし、受手側は、受手側の実装によりサポートされる方法で、それらから実装に適した DRD を生成できる。

6.2.5 意思決定サービス

意思決定サービスは、太い実線の枠線で描かれた丸角四角として DRD に表示される。実装は各意思決定サービスにその名前を表示することによってラベルを付けることができるようにしなければいけない。その「記述」のように他のプロパティを表示することによってラベルを付けてもよい。表示される場合、ラベルは同じ DRD 内のすべての DRD 要素のラベルとは異なるものとし、はっきりと四角形の内側にあるものとする。 境界はカプセル化されたすべての意思決定を囲み、他の意思決定や入力データは囲まないものとする。境界は他の DRG 要素を囲んでもよいが、この DRG 要素は意思決定サービスの定義の一部とはならない。

意思決定出力の集合がカプセル化された意思決定の集合よりも小さい場合、意思決定サービスは実線の直線で 2 つの部分に分割される。一方は､意思決定出力とラベルのみを囲み、他方は、意思決定出力の集合に含まれないすべてのカプセル化された意思決定を囲む。 どちらの部分も他の DRG 要素を囲むことができるが、この DRG 要素は意思決定サービスの定義の一部とはならない。

明確化のために、長方形またはその部分は影を付けることができ、境界を構成するすべての要素（意思決定出力、意思決定入力、入力データ）は、境界線と同じ太さと色で描くことができる。 図 17 は、2 つの意思決定出力を持つ意思決定サービスを示す。 他の例（単一の意思決定出力）を図 10 と図 11 に示す。

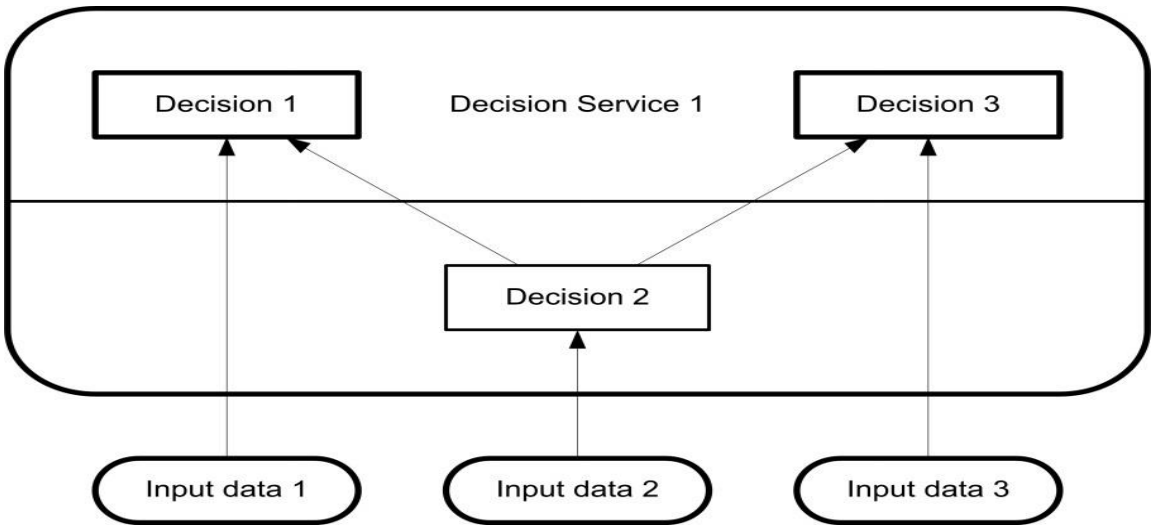


図 17: 意思決定サービスの表記

6.3 メタモデル

6.3.1 DMN 要素メタモデル

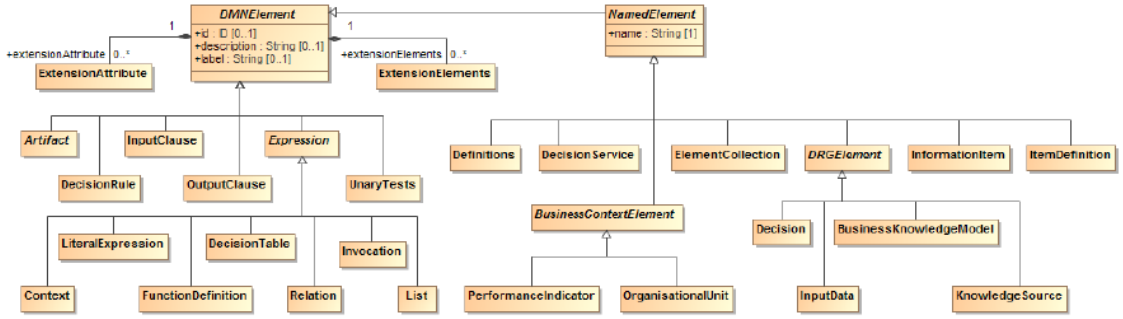


図 18:DMNElement クラス図

DMNElement は、意思決定モデル要素の抽象化されたスーパークラスである。これは、他の要素に継承されるオプション属性である id、String 型の description と label 属性を提供する。DMNElemetnt の id は、XML ID (http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#ID), の構文に、より強く制約を受けるので、意思決定モデルの中においては一意でなければならない。

DMNElements は、抽象的特殊化の NamedElement や Expression と、具象的特殊化の UnaryTest を持っている。NamedElement は、要求された属性名を追加し、抽象的特殊化の BusinessContextElement や DRGElements、同じく具象的特殊化の Definition、ItemDefinition、InformationItem、EementCollection、DecisionService を含んでいる。

表 3 に、DMNElement の属性およびモデル・アソシエーションについて示す

表 3:DMNElement の属性およびモデル・アソシエーション

属性	概要
id: ID [0..1]	この要素のオプション識別子。Definitions 要素の中において一意でなければならない。
description: String [0..1]	この要素の概要。
label: String [0..1]	この要素のラベル。
extensionElements: ExtensionElement [0..1]	追加要素を任意の DMN 要素に付加するためのコンテナとして用いられる。拡張性の追加情報については「6.3.16 拡張性」を参照のこと。

extensionAttributes: ExtensionAttribute [0..*]	命名された拡張属性とモデル・アソシエーションを付加するのに用いられる。 XML スキーマの交換が用いられている場合は、このアソシエーションを適用できない。これは、他の名前空間からの「anyAttribute」をサポートする XSD の仕組みは、既にこの要求を充たしているためである。拡張性の追加情報については「6.3.16 拡張性」を参照のこと。

表 4: NamedElement の属性とモデル・アソシエーション

属性	概要
name: string	Name:この要素の名称。

6.3.2 Definitions メタモデル

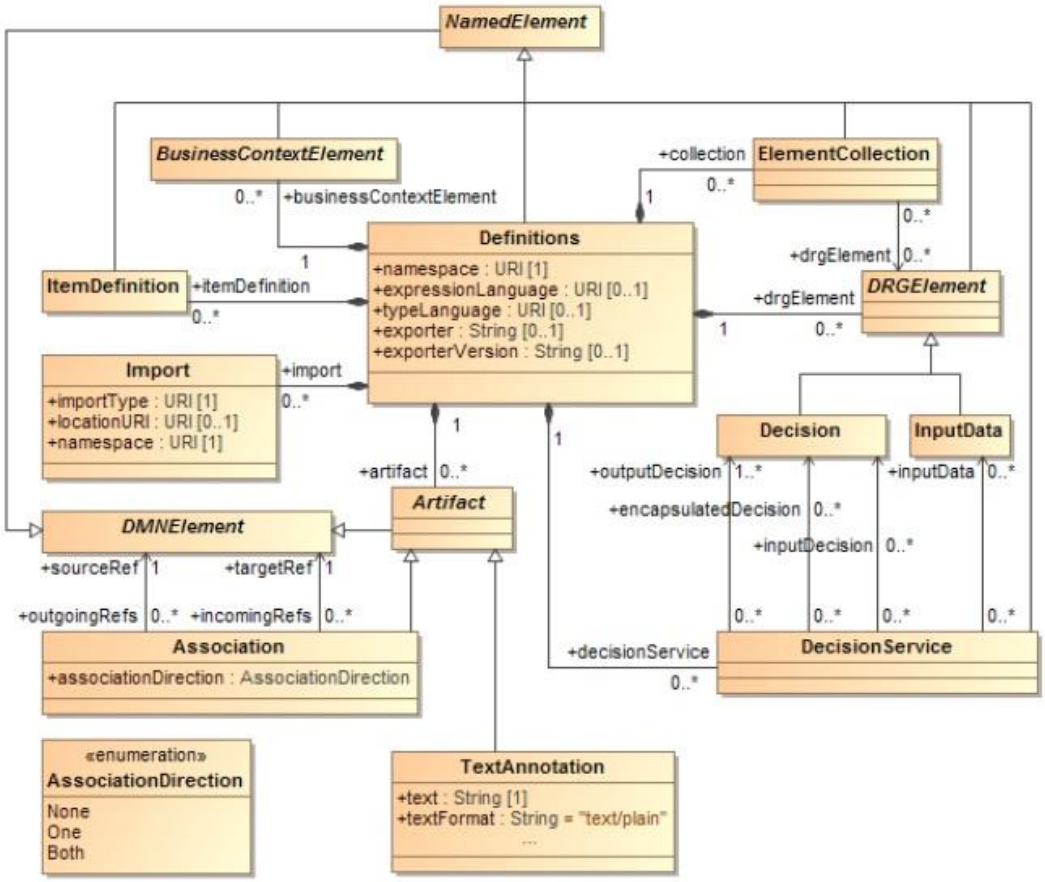


図 19:Definitions クラス図

すべての DMN 決定モデルの要素の中で、Definitions クラスは、オブジェクトを含む最も外側に位置する。これは、可視性の範囲、および含まれる全要素の名前空間を定義する。Definitions のインスタンスに含まれる要素は、それ自身の定義されたライフサイクルを持っており、他の要素と一緒に削除されることはない。DMN ファイルの交換は、かならず Definitions を通して行われる。

Definitions は、NamedElement の一種であり、Definitions のインスタンスは NamedElement からの名前、オプションの id、String 型の description と label 属性を継承する。

Definitions のインスタンスは、String 型である namespace を持つ。namespace は、Definition の要素であるデフィルトのターゲット名前空間を識別し、XML スキーマの書式に従う。

Definitions のインスタンスは、expressionLanguage を指定する。これは、Definitions のスコープ内の要素で用いられるデフォルトの式言語を識別する URI である。この URI は、個々の LiteralExpression に上書きされることもある。この言語は、URI フォーマッ

トによって指定されなければならない。デフォルト式言語は、FEEL(第 10 章)であり、URI"http://www.omg.org/spec/feel/20140401"に示される。FEEL のサブセットでありシンプルな式言語である S-FEEL(第 9 章)も、同じ URI に示されている。なお、DMN は、機械的に解釈できない式言語としても URI を提供している。(例えば FEEL に似ているがそうではない擬似コード) "http://www.omg.org/spec/DMN/uninterpreted/20140801".

Definition のインスタンスは、typeLanguage を指定することができる。これは、このDefinitions のスコープ内の要素の中で用いられるデフォルトの言語タイプを識別する URI である。例えば、"http://www.w3.org/2001/XMLSchema"の typeLanguage 値は、このDefinitions 内で定義されたデータ構造が、デフォルトで XML スキーマ・タイプの形式であることを示す。指定されない場合は、デフォルトの typeLanguage 値は FEEL である。この値は、個々の ItemDefinition によって上書きされることもある。typeLangage 値は URI 形式で指定される必要がある(FEEL の URI は "http://www.omg.org/spec/FEEL/20140401" であり、URI "http://www.omg.org/spec/DMN/uninterpreted/20140801" は、タイプ定義が解釈されることになっていないことを示すときに用いられる)。

Definitions のインスタンスは、exporter や exporterVersion を指定することができる。これは XML シリアライゼーションを行うツールやバージョンを命名するための String 型である。BPMN のような規格では、これによりツール間のモデル交換を支援することもある。

Definitons のインスタンスは、DRGElements のインスタンスである 0 以上の drgElements、ElementCollection のインスタンスである 0 以上の collections、DecisionService のインスタンスである 0 以上の decisionServices、BusinessContextElement のインスタンスである 0 以上の businessContextElements から、構成されている。

これには、Import のインスタンスである、多くの関連した import を含まれる場合がある。Import はこの Defintions 外にて定義された要素(例えば、他の Definitions 要素)をインポートするために用いられ、それらの要素をこの Definitions 内の要素によって使えるようにする。

Defintions は、NamedElement からすべての属性とモデル・アソシエーションを継承する。表 5 に、追加される属性と Definitions 要素のモデル・アソシエーションを表す。

表 5: Definitions の属性とモデル・アソシエーション

属性	説明
namespace: anyURI [1]	Definitions に関連する名前空間を特定し、XML スキーマによって設定された書式に従

	う。
expressionLanguage: anyURI [0..1]	LiteralExpressions で使用される式言語を Definitions.の範囲内で特定する。デフォルトは FEEL（第 10 章）である。 この値は、個々の LiteralExpression で上書きすることができる。 言語は URI 形式で指定するものとする。
typeLanguage: anyURI [0..1]	LiteralExpressions で使用される言語型を Definitions.の範囲内で特定する。デフォルトは FEEL（第 10 章）である。 この値は、個々の ItemDefinition で上書きすることができる。 言語は URI 形式で指定するものとする。
exporter: string [0..1]	XML シルアライゼーションをエクスポートで使われるツールを指定する。
exporterVersion: string [0..1]	XML シルアライゼーションのエクスポートで使われるツールのバージョンを指定する。
itemDefinition: ItemDefinition [*]	Definitions に含まれる ItemDefinition インスタンスをリストする。
drgElement: DRGElement [*]	この Definition に含まれる DRGElements のインスタンスをリストする。
businessContextElement: BusinessContextElement [*]	この Definition に含まれる BusinessContextElement のインスタンスをリストする。
collection: ElementCollection [*]	この Definition に含まれる ElementCollection のインスタンスをリストする。
decisionService: DecisionService [*]	この Definition に含まれる DecisionService のインスタンスをリストする。
import: Import [*]	外部で定義された要素をインポートし、この Definitions に含まれる要素から利用できるようにする。
artifact: Artifact [0..*]	Artifacts は注釈と DMN 要素間の関連を含む。

6.3.3 Import メタモデル

Import クラスは、他の Definitions 要素に含まれる各 DMN DRGElement インスタンスや、XML スキーマや PMML ファイルなどの非 DMN 要素など、外部の要素を参照するときに利用される。Import は、明確に定義される必要がある。

Import のインスタンスには importType がある。これは String 型で、当該要素に関連するインポートの種類を指定する。例えば、http://www.w3.org/2001/XMLSchema"という値は、インポートされた要素は XML スキーマであるということを示す。DMN 名前空間は、インポートされた要素が DMN Definition 要素であることを示す。

インポートされた要素の位置は、オプションの locationURI を Import のインスタンスに関連づけることによって指定することができる。その locationURI は、URI である。

Import のインスタンスには名前空間があり、これはインポートされた要素の名前空間を見分ける URI である。

表 6 に、Import 要素の属性とモデル・アソシエーションを示す。

表 6: Import の属性とモデル・アソシエーション

属性	説明
importType: anyURI	この Import に関連づけられたインポートのスタイルを指定する。
locationURI: anyURI [0..1]	インポートされた要素の位置を特定する。
namespace: anyURI	インポートされた要素の名前空間を特定する。

6.3.4 ElementCollection メタモデル

ElementCollection クラスは、DRGElement インスタンスの名前付きグループを定義するのに利用される。ElementCollection は、実装に関するあらゆる目的のために用いることができる。例えば、

・意思決定の要求サブグラフを識別するため(つまり、全ての要素がそのセットの要求が閉じられた名前空間内にある)

・DRD 上に描かれた要素を識別するため

ElementCollection は、NamedElement の一種であり、ElementCollection のインスタンスから、String 型である name とオプションの id と description と label 属性を継承する。Element Collection の id 要素は、Definitions のインスタンス内では一意である必要がある。

ElementCollection 要素は、関連する多くの drgElement を持っている。その drgElement は、その ElementCollection がグループとして一緒に定義されている DRGElemetnt のイ

ンスタンスである。ここで、ElementCollection 要素は、それが収集する DRGElement のインスタンスを参照しなければならず、含んではいけないことに留意する。つまり Definitions 要素だけに DRGElement のインスタンスを含めることが出来る。

ElementCollection は、NamedElement からすべての属性とモデル・アソシエーションを継承する。表 7 に、ElementCollection 要素に追加された属性とモデル・アソシエーションを表す。

表 7: ElementCollection の属性とモデル・アソシエーション

属性	説明
drgElement: DRGElement [*]	ElementCollection グループの DRGElement のインスタンスをリストする

6.3.5 DRGElement メタモデル

DRGElement は、Definitions に含まれると共に、DRD によるグラフィック表現を持つ全ての DMN 要素の抽象化されたスーパークラスである。直接的には、Definition 要素には含まれない DMN 意思決定モデルの全要素(注釈：具体的には、三種類の要求線の全て、バインディング、句と意思決定ルール、インポート、目的変数)は、DRGElement のインスタンスに含まれているか、DRGElement のインスタンスに含まれるモデル要素に含まれている必要がある。

DRGElement の具象的特殊化は、Decision、InputData、BuisinessKnowledgeModel、KnowledgeSource である。

DRGElemetnt は、NamedElement の特殊化であり、NamedElement から name、id、description、label を継承している。DRGElement 要素の id は、Definition のインスタンス中で一意である必要がある。

Decision Requirements Diagram(DRD)は、DRGElemetns のインスタンスと、その情報や知識や根拠の要求関係を図式的に表現したものである。DRGElement のインスタンスは、図の一番上に表記され、末端には InformationRequirement、KnowledgeRequirement、または AuthorityRequirement のインスタンスを表す(6.3.12 Information Requirement メタモデル，6.3.13Knowledge Requirement メタモデル，そして 6.3.14 Authority Requirement メタモデルを参照)。接続のルールは 6.3.12 接続ルールで規定される)。

DRGElements は、NamedElement のすべての属性とモデル・アソシエーションを継承する。但し、DRGElement の要素としては、属性やモデル・アソシエーションは追加されない。

6.3.6 Artifact メタモデル

Artifact は、意思決定モデルについての追加情報を提供するために用いられる。DMN は、2 つの Artifact 標準を規定する：Association と Text Annotation である。Association は、どんな DMNElement でも Artifact とリンクさせるのに用いる事ができる。

6.3.6.1 Association

Association は、情報と成果物を DMN グラフィカルな要素にリンクさせるときに用いられる。Text Annotation とほかの Artifact は、グラフィカルな要素に関連付けることが出来る。Association の矢印は必要に応じてフローの方向を示す(例えばデータ)。

Association 要素は、DMNElement の属性とモデル・アソシエーションを継承する(表 3 参照)。表 8 に、Association に追加される属性とモデル・アソシエーションを示す。

表 8: Association の属性とモデル・アソシエーション

属性	説明
associationDirection: AssociationDirection = None {None | One | Both}	associationDirection は、Association に矢印が付いているかどうかを定義する属性である。デフォルトは、None(矢印なし)。One は、矢印が接続先側にあることを意味する。Both は、矢印が Association の両端にあることを意味する。
sourceRef: DMNElement[1]	Association の接続元である DMNElement。
targetRef: DMNElement[1]	Association の接続先である DMNElement。

6.3.6.2 Text Annotation

Text Annotation は、モデル作成者向けの仕組みであり、DMN ダイアグラムの読者への追加のテキスト情報を提供する。

TextAnnotation 要素は、DMNElement の属性とモデル・アソシエーションを継承する(表 3 参照)。表 9 に、TextAnnotation に追加される属性とモデル・アソシエーションを示す。

表 9:TextAnnotation 属性

属性	説明
text: string	text は、モデル作成者がダイアグラムの読者に伝えたいテキストの属性である。
textFormat: string = “text/plain”	テキスト形式を識別する。これは mime-type 形式に従わなければならない。デフォルトは”text/plain”である。

6.3.7 Decision メタモデル

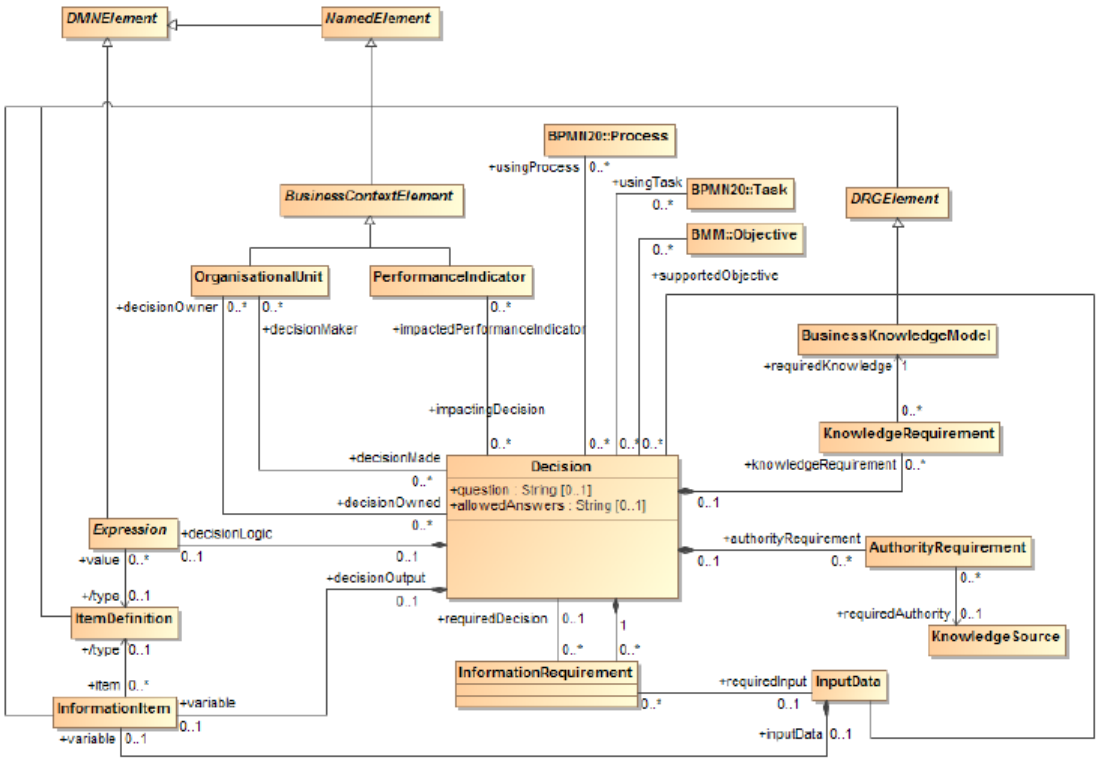


図 20: Decision クラス図

Decision クラスは、意思決定をモデル化するのに用いられる。

Decision は DRGElemetns の具象的特殊化であり、NamedElement から name とオプションの id、description、label 属性を継承する。

さらに、String 型である question と allowedAnswers を持つことができる。オプションの description 属性には、Decision 内で具体化される意思決定の簡単な説明を含める。オプションの question 属性は自然言語による質問を含める。これは、Decision のアウトプットが質問への回答となっているような Decision の特徴を明らかにする。オプションの allowedAnswers 属性は、Yes/No、回答候補リスト、数値の範囲など、質問に対する回答内容についての説明を含める。

DRD において、Decision 内のインスタンスは、意思決定ダイアグラムの要素として表される。

Decision 要素は、Expression のインスタンスである decisionLogic と、0 以上の informationRequirement、knowledgeRequirement、authorityRequirement から構成される。informationRequirement、knowledgeRequirement、authorityRequirement は、

それぞれ InformationRequirement、KnowledgeRequirement、AuthorityRequirement のインスタンスである。

さらに、Decision はそのアウトプットを表す InformationItem を定義する。この InformationItem は、ItemDefinition や Decision の起こりうる結果のデータ型を指示している他の型定義を参照するオプションの typeRef を含むことができる。

Decision 要素の要求線サブグラフは、Decision 要素にある、informationRequirements、knowledgeRequirements、および各 requiredDecision 要素または requiredKnowledge 要素の要求線サブグラフの和集合から構成される有向グラフである。つまり、Decision の要素線サブグラフは、その Decision 要素に向かう informationRequirement、requiredInput、requiredDecision、knowledgeRequirement、requiredKnowledge との関連のみから構成される。

Decision のインスタンス、つまり意思決定のモデルは、すべての informationRequirement、knowledgerequirement 要素が整形式の場合のみ、整形式だといわれる。

特に、Decision 要素の要求線サブグラフは、非巡回でなければならないという制約がある、つまり、直接間接を問わず Decision 要素は自分自身を要求してはならない。

情報要求線や意思決定ロジックなどの論理部品の他に、意思決定モデルは意思決定のためのビジネス・コンテキストも文書化することができる（6.3.8 Business Context 要素のメタモデルと下記図 21 を参照）

Decision のインスタンスの為のビジネス・コンテキストは、OMG BMM で定義された Objective のインスタンスであるいくつかの supportedObjectives と、PerformanceIndicator のインスタンスであるいくつかの impactedPerformanceIndicators と、いくつかの decisionMaker, そして OrganisationalUnit のインスタンスであるいくつかの decisionOwner,との関連で定義される。

それに加えて、Decision のインスタンスは OMG BPMN 2.0 で定義された Process のインスタンスであるいくつかの usingProcess、OMG BPMN 2.0 で定義された Task のインスタンスであるいくつかの usingTask、Decision 要素を使用する Processes と Tasks を参照することができる。

Decision は、DRGElement の全ての属性とモデル・アソシエーションを継承する。表 10 に、継承以外の Decision クラスの属性とモデル・アソシエーションを示す。

表 10:Decision 属性とモデル・アソシエーション

属性	説明
question: string [0..1]	自然言語の質問で、Decision のアウトプットが質問の回答となるような Decision の特徴を明らかにする。
allowedAnswers: string [0..1]	Yes/No や回答候補リスト、数値の範囲など、質問が求める回答内容。
variable: InformationItem	InformationItem のインスタンスで、この Decision の結果を保持する。
typeRef: String[0..1]	この Decision が取りうる出力値のデータ型の仕様へのポインタ、定義された expressionLanguage 基本型である ItemDefinition かインポート型となる。
decisionLogic: Expression [0..1]	この Decision の意思決定ロジックを示す Expression のインスタンス。
informationRequirement: InformationRequirement [*]	この Decision を構成する InformationRequirement のインスタンスのリスト。
knowledgeRequirement: KnowledgeRequirement [*]	この Decision を構成する KnowledgeRequirement のインスタンスのリスト。
authorityRequirement: AuthorityRequirement [*]	この Decision を構成する AuthorityRequirement のインスタンスのリスト。
supportedObjective: BMM::Objective [*]	この Decision をサポートする BMM::Objective のインスタンスのリスト。
impactedPerformanceIndicator: PerformanceIndicator [*]	この Decision により影響される PerformanceIndicator のインスタンスのリスト。
decisionMaker: OrganisationalUnit [*]	この Decision を行う OrganisationalUnit のインスタンス。
decisionOwner: OrganisationalUnit [*]	この Decision を所有する OrganisationalUnit のインスタンス。
usingProcesses: BPMN::process [*]	Decision を要求する BPMN::process のインスタンスのリスト。

usingTasks: BPMN::task [*]	この Decision を行う BPMN::task のインスタンスをリストする。

6.3.8 Business Context 要素メタモデル

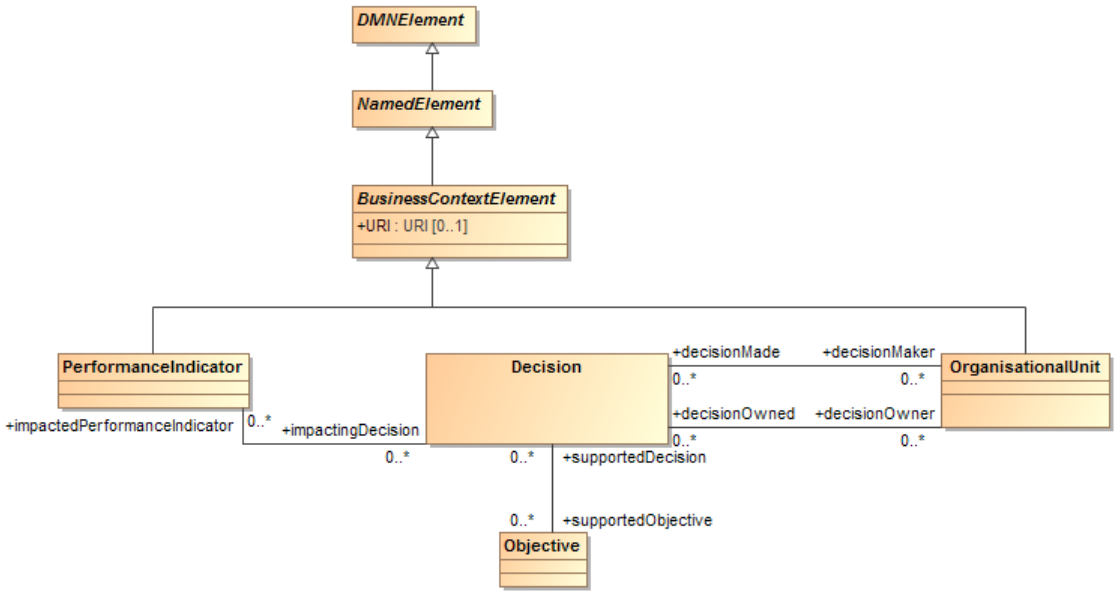


図 21: BusinessContextElement クラス図

抽象クラス BusinessContextElement、そしてその具象化である PerformanceIdicator と OrganizationUnit は、プレースホルダー（暫定的呼称）である。これは将来開発される OMG OSM のような他の OMG メタモデルからの定義を適用させるときのために用意された。BusinessContetElements は、NamedElement の特殊化であり、name、id、description、label といった属性を継承している。

さらに、BusinessContextElements のインスタンスに URI（クラス）をもつことができる。次の特徴を持つ。

- PerformanceIndicator のインスタンスは、それに影響を与える Decision 要素 であるいくつかの impactingDecision を参照する。
- OrganisationalUnit のインスタンスは、組織単位が作成または所有する意思決定をモデル化する Decision 要素 である decisionMade と decisionOwned をいくつか参照する。

BusinessContextElement は、NamedElement から全ての属性とモデル・アソシエーションを継承する。表 11 に、継承以外の BusinessContextElement クラスの属性とモデル・アソシエーションを示す。

表 11:BusinessContextElement 属性とモデル・アソシエーション

属性	説明
URI: anyURI [0..1]	この BusinessContextElement の URI。

PerformanceIndicator は、BusinessContextElement から全ての属性とモデル・アソシエーションを継承する。表 12 に、継承する以外の PerformanceIndicator クラスの属性とモデル・アソシエーションを示す。.

表 12:PerformanceIndicator の属性とモデル・アソシエーション

属性	説明
impactingDecision: Decision [*]	PerformanceIndicator に影響を与える Decision のインスタンスのリスト。

OrganisationalUnit は、BusinessContextElement から全ての属性とモデル・アソシエーションを継承する。表 13 に、継承以外の OrganisationalUnit クラスの属性とモデル・アソシエーションを示す。.

表 13:OrganisationalUnit の属性とモデル・アソシエーション

属性	説明
decisionMade: Decision [*]	この OrganisationalUnit によって作成される Decision のインスタンスのリスト。
decisionOwned: Decision [*]	この OrganisationalUnit によって所有される Decision のインスタンスのリスト。

6.3.9 Business Knowledge Model メタモデル

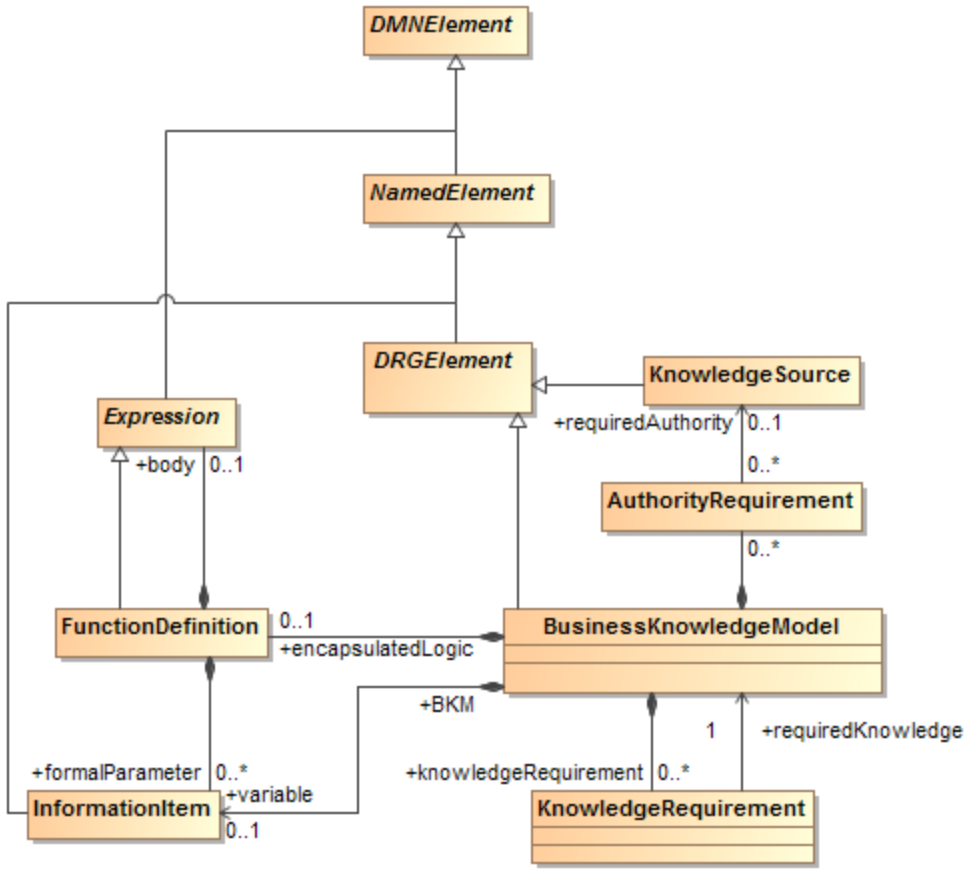


図 22:BusinessKnowledgeModel クラス図

意思決定に関係するビジネス知識モデルは、意思決定ロジックの全部または一部を表現している再利用可能なモジュール式である

BusinessKnowledgeModel クラスは、ビジネス知識モデルを作成するのに用いられる。

BusinessKnowledgeModel は、DRGElement の具象的特殊化であり、name、オプション属性である id、description、label 属性を NamedElement から継承する。

DRD において、BusinessKnowledgeModel のインスタンスは、ビジネス知識モデル図の要素によって表現される。

一つの BusinessKnowledgeModel 要素は、KnowledgeRequirement のインスタンスである knowledgeRequirement と、AuthorityRequirement のインスタンスである authorityRequirement を持てる。

BusinessKnowledgeModel 要素の要求線サブグラフは、BusinessKnowledgeModel 要素自身とその knowledgeRequirement 要素、knowledgeRequirements によって参照される. requiredKnowledge 要素の要求線サブグラフの和集合全てを含む有向グラフである。

BusinessKnowledgeModel インスタンスは、knowledgeRequirement を全く持っていないか、全ての knowledgeRequirement 要素が整形式である場合に限り、整形式と言われる。とりわけ、BusinessKnowledgeModel 要素の要求線サブグラフは非巡回でなければならない。つまり、直接間接を問わず BusinessKnowledgeModel 要素は 自分自身を要求してはならない。

意思決定ロジック・レベルにおいて、BusinessKnowledgeModel 要素は、FunctionDefinition を含む。これはパラメータを含むこともある Expression のインスタンスであり InformationItem のインスタンスでもある。BusinessKnowledgeModel 要素に含まれる FunctionDefinition は、BusinessKnowledgeModel 要素によって表現される意思決定ロジックの再利用可能なモジュールである。 BusinessKnowledgeModel 要素は、InformationItem を含み、名前によって Decision を呼び出す FunctionDefinition を保持する。その InformationItem の 名前は、BusinessKnowledgeModel 要素の名前と同じでなければならない。BusinessKnowledgeModel は、DRGElement から全ての属性とモデル・アソシエーションを継承する。表 14 に、BusinessKnowledgeModel クラスで追加される属性とモデル・アソシエーションを示す。

.

表 14: BusinessKnowledgeModel 属性とモデル・アソシエーション

属性	説明
encapsulatedLogic: FunctionDefinition [0..1]	この BusinessKnowledgeModel によりカプセル化されたロジックをカプセル化する関数。
variable: InformationItem	関数に紐づけられた InformationItem のインスタンス。 この変数を名前によって参照できる。
knowledgeRequirement: KnowledgeRequirement[*]	BusinessKnowledgeModel に含まれる KnowledgeRequirement のインスタンスをリストする。
authorityRequirement: AuthorityRequirement [*]	BusinessKnowledgeModel に含まれる AuthorityRequirement のインスタンスをリストする。
variable: InformationItem	FunctionDefinition を保持する変数を定義する。そのことで名前によって Decision を呼び出すことができるようになる。

6.3.10 Input Data メタモデル

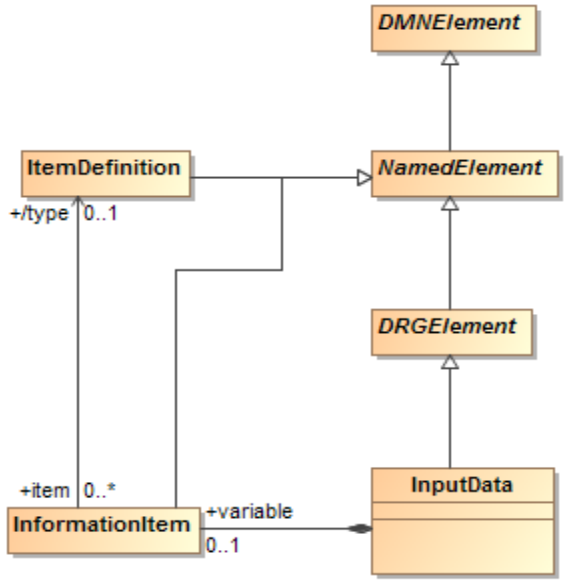


図 23: InputData クラス図

DMM1.1 は、意思決定モデルの外部で定義された値をこの意思決定の入力としてモデル化するのに InputData クラスを使う。

InputData は DRGElement の具象的特殊化である。そして NamedElement から name、オプション属性である id、description、label 属性を継承する。

InputData のインスタンスは、その値を保持する InformationItem を定義する。この InformationItem には、データのタイプを指定する typeRef が含まれることがある。すなわちこの InputData は ItemDefinition または expressionLanguage で指定される基本型、もしくはインポートされた型を表す

DRD において InputData のインスタンスは、入力データ図の要素として表わされる。InputData 要素は、要求線サブグラフを持たない。そして常に整形式である。

InputData は、DRGElement から全ての属性とモデル・アソシエーションを継承する。表 15 に、InputData クラスに追加される属性とモデル・アソシエーションを示す。

表 15: InputData の属性とモデル・アソシエーション

属性	説明
variable: InformationItem	InformationItem のインスタンスで、この InputData の結果を保持する。
typeRef: String[0..1]	この InputData が取りうる値のデータ型をしめす。ItemDefinition または expressionLanguage で指定される基本型、もしくはインポートされた型となる。

6.3.11Knowledge Source メタモデル

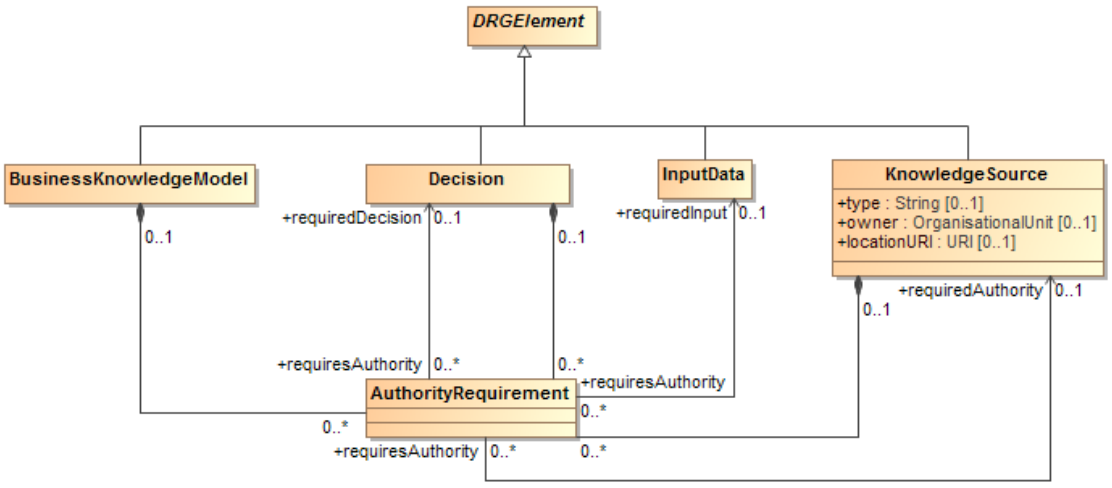


図 24: KnowledgeSource クラス図

KnowledgeSource クラスは、意思決定モデルの中で根拠となる知識ソース をモデル化するのに使用される。

DRD において KnowledgeSource のインスタンスは知識ソース図の要素として表される。

KnowledgeSource は、DRGElement の具象的特殊化であり、したがって NamedElement の具象的特殊化である。KnowledgeSource は、NamedElement から name 属性を継承され、option 属性としての id、 description、label を継承される。加えて、 KnowledgeSource は URI である locationURI を持つ。KnowledgeSource は、文字列型の type と OrganisationalUnit のインスタンスである owner を持つ。その型は、根拠のソースの種類例えばポリシー文書(Policy Document)、規定(Regulation)、 分析的洞察(Analytic Insight)を識別することを意図する。

KnowledgeSource 要素はまた、AuthorityRequirement のインスタンスである authorityRequirement 要素からできている。

KnowledgeSource は、DRGElement の全ての属性とモデル・アソシエーションを継承する。表 16 に、KnowledgeSource クラスの属性とモデル・アソシエーションを示す。

表 16: KnowledgeSource 属性とモデル・アソシエーション

属性	説明
locationURI: anyURI [0..1]	この URI は、どこにこの KnowledgeSource が位置するか示す。locationURI は URI 書式に従っていなければならない。
type: string [0..1]	この KnowledgeSource の型。
owner: OrganisationalUnit [0..1]	この KnowledgeSource の所有者。

authorityRequirement: AuthorityRequirement [*]	この KnowledgeSource に提供する AuthorityRequirement のインスタンスをリストする。

6.3.12 Information Requirement メタモデル

InformationRequirement クラスは、情報要求線をモデル化するのに用いられ、DRD 内で単純矢印として表される。

InformationRequirement 要素は、Decision 要素の構成物であり、Decision 要素を要求する Decision 要素を、Decision のインスタンスである requiredDecision 要素や InputData のインスタンスである requiredInput 要素に関連付ける。

InformationRequirement 要素は、変数を定義する Decision または InputData のインスタンスを参照する。InformationItem のインスタンスである変数は、意思決定ロジック・レベルの InformationRequirement 要素を表す。

InformationRequirement 要素は、要求された Decision 要素と関連づけられる Decision または InputData のインスタンスを参照しなければならないことに注意すること。
含んではいけない。つまり Decision または InputData インスタンス は、Definitions 要素にのみ含めることができる。

InformationRequirement のインスタンスは、下記が全て真である場合に限り、整形式と言われる：

- それは requiredDecision または requiredInputelement を参照するが、両方は参照しない。
- 参照された requiredDecision または requiredInput の要素は整形式である。
- もしこの InformationRequirement 要素が何かを参照している時に、InformationRequirement のインスタンスに含まれる Decision 要素は、requiredDecision 要素に参照された要求線サブグラフの中に無いこと。

表 17 に、InformationRequirement 要素の属性とモデル・アソシエーションを示す。

表 17: InformationRequirement 要素の属性とモデル・アソシエーション

属性	説明
requiredDecision: Decision [0..1]	この InformationRequirement が含む Decision 要素に関連づけられている Decision のインスタンス。

requiredInput: InputData [0..1]	この InformationRequirement が含む Decision 要素に関連付けられている InputData のインスタンス。

6.3.13　Knowledge Requirement メタモデル

KnowledgeRequirement クラスは、知識要求線をモデル化するのに用いられ、DRD では破線矢印で表される。

KnowledgeRequirement 要素は、Decision 要素または BusinessKnowledgeModel 要素のコンポーネントであり、BusinessKnowledgemodel のインスタンスである requiredKnowledge 要素とともに、要求された Decision または BusinessKnowledgeModel に関連づけられる。

KnowledgeRequirement 要素は、BusinessKnowledgeModel のインスタンス（BusinessKnowledgeModel の要素が要求された Decision か BusinessKnowledgeModel 要素と関連させる）を参照しなければならないということに注意すること。BusinessKnowledgeModel のインスタンスを含んではいけない。BusinessKnowledgeModel のインスタンス は要素だけを含めることができる。

KnowledgeRequirement のインスタンスは、以下の項目が全て真の場合にだけ整形式と言われる:

- requiredKnowledge 要素を参照している
- 参照している requiredKnowledge 要素は整形式である.
- もし、KnowledgeRequirement 要素が BusinessKnowledgeModel のインスタンスを含む場合、その BusinessKnowledgeModel 要素は requiredKnowledge 要素の要求線サブグラフではない

表 18 に、KnowledgeRequirementelement の属性とモデル・アソシエーションを示す。

表 18: KnowledgeRequirementelement の属性とモデル・アソシエーション

属性	説明
requiredKnowledge: BusinessKnowledgeModel	KnowledgeRequirement が Decision または BusinessKnowledgeModel 要素に関連づける BusinessKnowledgeModel のインスタンス。

6.3.14 Authority Requirement メタモデル

AuthorityRequirement クラスは、根拠要求のモデル化に使用され、DRD では先頭が円形の破線矢印で表現される。

AuthorityRequirement 要素は 、Decision 、BusinessKnowledgeModel または KnowledgeSource 要素のコンポーネントである。そして、要求された Decision、BusinessKnowledgeModel または KnowledgeSource 要素と requiredAuthority 要素と関連付ける。requiredAuthority 要素は、KnowledgeSource、requiredDecision（Decision のインスタンス）または requiredInput （InputData のインスタンス）要素のインスタンスである。 AuthorityRequirement 要素は、自分自身を含めない要求された要素と関連づける KnowledgeSource、Decision または InputData のインスタンスを参照しなければならなず、含んではいけないことに注意すること。

つまり KnowledgeSource、Decision または InputData のインスタンスは、Definitions 要素にのみ含めることができる。

表 19 に、AuthorityRequirement 要素の属性とモデル・アソシエーションを示す

表 19: AuthorityRequirement 属性とモデル・アソシエーション

属性	説明
requiredAuthority: KnowledgeSource [0..1]	AuthorityRequirement が、それが含まれる KnowledgeSource 、 Decision または BusinessKnowledgeModel と関連づけた KnowledgeSource のインスタンス。
requiredDecision: Decision [0..1]	AuthorityRequirement が、それが含まれる KnowledgeSource 要素と関連づけた Decision のインスタンス。
requiredInput: InputData [0..1]	AuthorityRequirement が、それが含まれる KnowledgeSource の要素と関連づけた InputData のインスタンス。

6.3.15 Decision service メタモデル

DecisionService クラスは、Definitions のインスタンスが意思決定モデルに含まれるのに対して、名前付き意思決定サービスを定義するのに用いられる。

DecisionService は、NamedElement の一種で、このインスタンスは NamedElement より name とオプション属性である id と String 型である description と label 属性を継承する。DecisionService 要素の id は、Definitions のインスタンスの中で一意でなければならない。

DecisionService の要素は、関連する outputDecisions を持ち、outputDecisions はこの DecisionService によって出力される必要がある Decision のインスタンスである。つまり、その意思決定は意思決定サービスが呼び出された時に結果を返さなければならない。

DecisionService の要素は、encapsulatedDecision を持つ（省略可）。それらはこの DecisionService からカプセル化が要求された Decision のインスタンスである。つまりこの意思決定は意思決定サービスが呼び出された時に評価される。

DecisionService の要素は、inputDecision を持つ（省略可）。inputDecision はこの DecisionService によって入力として要求された Decision の要素である。つまり、この意思決定の結果は意思決定サービスが呼び出された時に提供される。

DecisionService の要素は、inputData を持つ（省略可）。inputData はこの DecisionService から入力として要求された InputData のインスタンスである。つまり、その入力データは意思決定サービスが呼び出された時に提供される。

上記 encapsulatedDecisions、inputDecisions、inputData の属性は、オプションである。ただし、encapsulatedDecisions と inputDecisions 属性の少なくとも 1 つの属性は指定されなければならない。

DecisionService は、NamedElement の全ての属性とモデル・アソシエーションを継承する。表 20 に、 DecisionService の要素で追加される属性とモデル・アソシエーションを示す。

表 20: DecisionService の属性とモデル・アソシエーション

属性	説明
outputDecisions: Decision [1..*]	DecisionService から出力を要求された Decision のインスタンスをリストする。
encapsulatedDecisions: Decision [0..*]	もし存在するなら、DecisionService 内にカプセル化された Decision のインスタンスをリストする。
inputDecisions: Decision [0..*]	もし存在するなら、DecisionService 内で入力として要求された Decision のインスタンスをリストする。
inputData: InputData [0..*]	もし存在するなら、サービスで入力として要求された InputData のインスタンスをリストする。

6.3.16 Extensibility

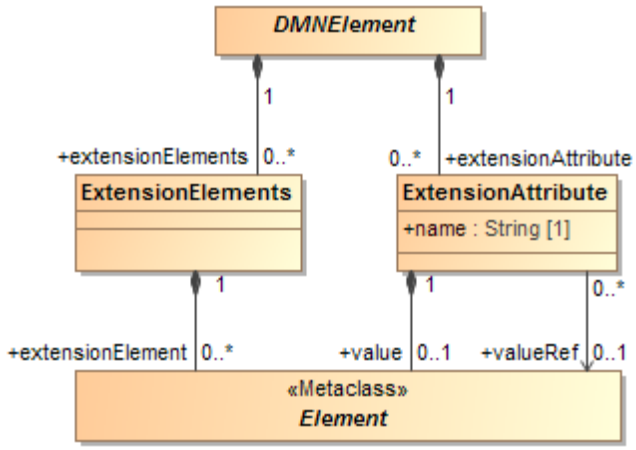


図 25: Extensibility クラス図

DMN メタモデルは拡張性を目指している。そのため、 DMN の採用者は、 拡張のために指定されたメタモデルを DMN に準拠した方法で拡張することができる。拡張要素のセットを提供することで、DMN の採用者は、標準と既存の DMN 要素に、属性と要素を追加することができる。このアプローチにより交換可能なモデル数は多くなる。なぜなら、標準要素はなんら変わらないため、他の DMN 採用者に理解できるからである。ただし、追加された属性と要素だけは交換中に失われるかもしれない。

2 つの異なる要素を用いて DMN を拡張することができる。

1. ExtensionElements
2. ExtensionAttribute

ExtensionElements は、他のメタモデルの任意の要素を DMN 要素に接続するためのコンテナである。ExtensionAttribute は、これらの添付物に名前を持たせる。このため DMN 採用者は、任意のメタモデルでも DMN メタモデルに統合させることと、既存のモデル要素を再利用することができる。

6.3.16.1 ExtensionElements

上記 ExtensionElements の要素は、DMNElements 内にあるほかのメタモデルからの要素を集めるためのコンテナである。表 21 に、ExtensionElements のエレメントの属性とモデル・アソシエーションを示す。

表 21: ExtensionElements の属性とモデル・アソシエーション

属性	説明
extensionElement: Element [0..*]	コンテナ要素である。これは XML スキーマ交換が使われるときには関連づけられない。なぜなら、他の名前空間の全て要素をサポートする XSD メカニズムが、既にこの要件を満たしているからである。

6.3.16.2 ExtensionAttribute

上記 ExtensionAttribute の要素は、Element または他のメタモデルからの Element の参照をコンテナ化している。ExtensionAttribute は、関係する要素の役割や目的を定義するための名前も持つ。この型は、XML スキーマによる交換が使われるときは適用できない。なぜなら他の名前空間の任意の属性をサポートする XSD メカニズムが、すでにこの要件を満たしているからである。表 22 に、ExtensionAttribute 要素のモデル・アソシエーションを示す。

表 22: ExtensionAttribute の属性とモデル・アソシエーション

属性	説明
name: string	拡張された属性の名前。
value: Element [0..1]	コンテナ Element。 この属性は、valueRef と一緒には使用できない。
valueRef: Element [0..1]	参照する関連 Element この属性は value と一緒には使用できない。

6.4 例

DRD の例は、11.3 意思決定要求レベルに記載されている。

7 意思決定ロジックと意思決定要求の関連付け

7.1 イントロダクション

第 6 章は、意思決定領域の構造をモデル化するために、意思決定モデル（一つ以上の DRD で表現された DRG）の意思決定要求レベルがどのように用いられることがあるのかを説明した。しかし、それぞれの意思決定の結果がどのようにしてそのインプットから導き出されるのかについての詳細は、意思決定ロジック・レベルでモデル化しなければならない。本章は、どの意思決定ロジックが DRG の要素と関連することがあるのかを示すことで、この原則を紹介する。そして意思決定ロジックの具体的な表現（デシジョンテーブルと FEEL 式）は、第 8 章、第 9 章、第 10 章で定義される。

DMNにおける意思決定モデルの意思決定ロジック・レベルは値式からなる。値式としてモデル化された意思決定ロジック要素には、デシジョンテーブルや起動などの表形式の式、および年齢＞30のような文字（テキスト）式などがある。

- 「文字式」は入力値から出力値がどのように導き出されるかを説明する文章で意思決定ロジックを表現する。表現する言語は形式的なものであったり実行可能なものであったりするが、必ずしもそうである必要はない。文字式の例は、意思決定のロジックを分かり易い英語で記述したもの、論理命題、Java コンピュータ・プログラム、および PMML 文書である。第 10 章は FEEL と呼ばれる実行可能な式言語について記述する。第 9 章は、DMN デシジョンテーブル（第 8 章）の文字式のためのデフォルト言語である FEEL のサブセット（S-FEEL）について記述する。
- 「デシジョンテーブル」は、意思決定ロジックを表形式で表現したものである。それは、意思決定の入力値として取り得る値の離散化をベースとしており、離散化された入力値を離散化された出力値に対応するルールで構成されている（第 8 章参照）。
- 「起動」は、あるビジネス知識モデルによって表現された意思決定ロジックが、ある意思決定または他のビジネス知識モデルによって、どのように呼び出されるかを表形式で表現したものである。起動は文字式によって表すこともできる。しかし、多くの場合、表形式で表現する方がより分かり易い。

本仕様書では、以降、意思決定ロジックの表形式の式を*囲み式*と呼ぶ。

三つの「DMN」適合レベルはすべて、上記の式すべてを含む。DMN 適合レベル 1 では文字式は

解釈されないので、自由である。「DMN」適合レベル 2 では、文字式は S-FEEL に限定されている。第 10 章では「DMN」適合レベル 3 でさらに利用可能な囲み式について記述する。

DRG においては、いくつかの意思決定モデル要素の中の値式コンポーネントを含めることで、意思決定ロジックが意思決定モデルに追加される。

・意思決定ロジックの観点からは、意思決定は、入力データに基づき与えられた問いにどのように答えられるかを定義する一つのロジックである。結果として、ある意思決定モデルの中のそれぞれの「意思決定」要素は、ある意思決定の結果が、おそらくはビジネス知識モデルを呼び出すことにより、求められるインプットからどのようにして導き出されるかを表す値式を含むことがある。
・意思決定ロジックの観点からは、ビジネス知識モデルは、複数の意思決定において再利用できる関数として定義された一つの意思決定ロジックである。結果として、それぞれの「ビジネス知識モデル」要素は、その関数の値式を含むことがある。

意思決定ロジック・レベルにおける次の重要なコンポーネントは「変数」である。変数は値式において「入力値」を表すために用いられる。入力値は変数に割り当てられ、値式は変数を参照する。変数は DRG における情報要求を、意思決定ロジック・レベルでの値式に結び付ける。

・意思決定ロジックの観点からは、情報要求は、ある意思決定を評価できるように、その意思決定ロジックの中のフリーな変数に対して割り当てるべき、外部から提供される値に対する要求である。結果として、ある意思決定モデルにおけるそれぞれの「情報要求」は、関連するデータ入力を表す変数を定義する意思決定または入力値を示す。
・DRG においてビジネス知識モデル要素によって定義された関数の中で使用される変数は、必要な意思決定それぞれの情報源と結びつけられなければならない。結果として、それぞれの「ビジネス知識モデル」は、関数のパラメータである変数を含むことがある。

意思決定ロジック・レベルの第三の重要な要素は、意思決定モデルのデータ項目のタイプと構造を明らかにする「項目定義」である。DRG の中の「入力データ」要素、および意思決定ロジック・レベルの「変数」と「値式」は、変数に割り当てられるか、または式を評価した結果として入力されることが期待される、データのタイプと構造を記述した関連項目定義を参照することがある。
「知識ソース」は意思決定ロジック・レベルでは表現されないことに注意して欲しい。知識ソースは意思決定ロジックの文書化の一部であり、意思決定ロジック自体ではない。

ある DRG の中の情報と知識要求によって表されているように、意思決定、求められる情報源、

そしてビジネス知識モデルの間の依存関係は、これらの要素に関連した値式がお互いにどのように関連し合うのかを制約する。

上記で説明されているとおり、DRG レベルでのすべての情報要求は、意思決定ロジック・レベルでの変数と式の組合せと関連する。ある意思決定の式によって参照されるそれぞれの入力変数は、その意思決定の一つの情報要求における変数もしくは意思決定要求サブグラフにおける情報要求でなければならない。さらに、ある意思決定の情報要求における変数は、その意思決定の式において参照される変数である。

- 意思決定が他の意思決定を必要とするのであれば、必要とされる意思決定の情報要求の値式は、必要とする意思決定を評価するのに用いられる変数の値に割り当てる。これが DMN において、意思決定ロジック・レベルでの意思決定を構成する一般的なメカニズムである。

- 意思決定がある入力データを必要とするのであれば、その変数の値は実行時における入力データに付属するデータソースの値が割り当てられる。これが DMN において、意思決定におけるデータ要求のインスタンス化の一般的なメカニズムである。必要とされている入力データにとって、FEEL は、テスト・データを含めることができるようにしていることに注意してほしい。

意思決定の意思決定ロジックの入力変数は、その値式またはそのコンポーネントの値式以外で使用してはならない。意思決定要素は、その意思決定ロジックのための入力変数として静的スコープを定義する。名前の衝突と曖昧さを避けるために、変数名は、そのスコープの中でユニークでなければならない。DRG 要素が FEEL に対応付けられるとき、ある変数の名前は、それに関連する入力データまたは意思決定の名前（修飾子が付いていることもある）と同じである。そのことは、名前が一意であることを保証する。

DRG 要素が FEEL に対応付けられるとき、DRG の中のすべての意思決定と入力データは、*コンテキスト*を定義する。それは、その意思決定要素と関連するロジックを表す文字式であり、そのスコープを表す。意思決定の中の情報要求要素は、関連するコンテキストにおける *コンテキスト・エントリー*である。そこでのキーは、その情報要求を定義する変数名である。その意思決定ロジックとして意思決定に関連する値式は、コンテキストの結果を特定するコンテキスト・エントリーの中の式である。

同様に、ビジネス知識モデル要素は、そのパラメータの静的スコープの定義であり、すなわち入力変数の体系を定義する。

FEEL では、あるビジネス知識モデルに関連するロジックを表現する文字式とスコーピングの構成概念は、*関数定義*である（10.3.2.11 関数セマンティックスを参照のこと）。その関数の*仮パラメータ*はビジネス知識モデル要素中のパラメータの名前であり、*式*はビジネス知識モデル要素体系の値式である。

知識モデル要素が他のビジネス知識モデルを必要とするならば、明示的な値式をもたなければならない。それは必要とされるビジネス・モデルがどのように呼び出され、その結果がどのように結び付けられるかを明らかにする、さもなければどのように詳細化されるかを明らかにする。

意思決定ロジック・レベルでは、それ自身の入力値と結びついたパラメータをもつ、必要とされるビジネス知識モデルの値式を評価することによって、意思決定がビジネス知識モデルを呼び出す。このことがどのようにして達成できるかは、その意思決定ロジックが、意思決定とビジネス知識モデルとをどのように分けているのかに依存する。

- 意思決定要素が複数のビジネス知識要素を必要とするのなら、値式は、ビジネス知識モデル要素がどのように呼び出され、その結果が意思決定の結果とどのように結びつくのかを明らかにする文字式でなければならない。
- 意思決定が、ビジネス知識モデルを一切必要としないなら、値式は、入力から出力へ派生する全体の意思決定ロジックを明らかにする文字式またはデシジョンテーブルでなければならない。
- 同様に、意思決定要素がただ一つのビジネス知識モデル要素を必要とするが、その意思決定ロジックは必要とされたビジネス知識モデルのロジック上で詳細になるのであれば、その意思決定要素は文字式を持たなければならない。それはビジネス知識モデルの値式がどのように呼び出されるのかを明らかにするため、また意思決定の結果をもたらすために、その結果がどのように詳細化されるのかを明らかにするためである。
- その他すべてのケースにおいて（すなわち意思決定がただ一つのビジネス知識モデルを必要とし、そのロジックを詳細化することはないとき）、意思決定要素の値式はタイプ起動の値式であることもある。タイプ起動の値式においては、ビジネス知識モデルのパラメータと意思決定の入力データとのバインディングだけが明らかにされる必要がある。その意思決定の結果は、そのパラメータに渡された値のためのビジネス知識モデルの値式によって返された結果である。

ビジネス知識モデルのパラメータのバインディングは、ビジネス知識モデルを呼び出している意思決定の入力変数の値から、パラメータに渡される値がどのように導き出されるのかを明らかにする値式である。

7.2 表記法

7.2.1 式

囲み式と呼ばれる意思決定ロジックのためのグラフィカル表記法を定義する。この表記法により、意思決定ロジック・モデルを DRG 成果物と関連付けることのできる小さな単位へと分解することができる。DRD に囲み式を用いると意思決定モデルを完全に明らかにし、ほぼ完全なグラフィカル言語を形成する。

囲み式の一般的な概念に加えて、後続の二つの節で次の二つの囲み式について詳述する。
・囲み文字式
・囲み起動
デシジョンテーブルのための囲み式については、第 8 章で明らかにする。囲み式のさらなるタイプについては、第 10 章で FEEL のために明らかにする。

囲み式は再帰的に定義される。すなわち、囲み式は他の囲み式を含めることができる。最上位の囲み式は単一の DRG 成果物の意思決定ロジックに対応する。この囲み式は、DRG 成果物の名前を含む名前の枠を持たなければならない。図 26 に示されているように、名前の枠は最上段に単一の枠として付けることができる。

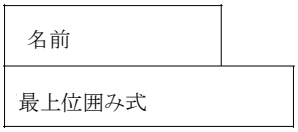


図 26：囲み式

もう一つの方法として、図 27 に示されるように、名前の枠と式の枠の間を空白で分離し、左端を線で結ぶこともできる。

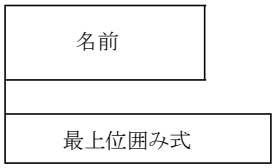


図 27：名前と式の枠を分離した囲み式

名前は、DRD 要素と囲み式の間で定義される唯一の視覚的なつながりである。グラフィカルなツールは、そのつながりを適切に図示することに役立つことが期待されている。例え

ば、意思決定を示す図形をクリックするとデシジョンテーブルが開く、などである。囲み式が DRD 要素と視覚的にどのように関連するかは、実装に委ねられている。

7.2.2　囲み文字式

囲み式では、文字式は文章で表現される。しかしながら、囲み文字式の可読性を向上させるために、二つの表記法が提供されている。それは、引用符で囲まれた文字列と、引用符で囲まれた日時表記である。

7.2.2.1　引用符で囲まれた文字列

”謝絶”のように引用符で囲まれた文字列は、引用符の代わりにイタリック体で*謝絶*と表現することもできる。例えば、図 28 は図 29 と同等のものである。

クレジットにおけるコンティンジェンシー係数表

U	リスク・カテゴリー	クレジットにおけるコンティンジェンシー係数
1	*高い，謝絶*	0.6
2	*中程度*	0.7
3	*低い，非常に低い*	0.8

図 28：イタリック体を用いたデシジョンテーブル

クレジットにおけるコンティンジェンシー係数表

U	リスク・カテゴリー	クレジットにおけるコンティンジェンシー係数
1	”高い”,”謝絶”	0.6
2	”中程度”	0.7
3	”低い”,”非常に低い”	0.8

図 29：引用符を用いたデシジョンテーブル

「高い，謝絶」が「*高い，謝絶*」なのか「”高い，謝絶”」なのかを識別する必要がないように、引用符で囲まれた文字列ではカンマ(”,”)は使用してはならない。引用符で囲まれた文字列 FEEL は、文法規則 27（名前）に適合する。

7.2.2.2　引用符で囲まれた日時表記

日付（"2013-08-09"）のような日付、時間、および日時、または期間は、*2013-08-09* と太字のイタリック体で表すこともできる。この文字は第 10 章の 10.3.2.3.4 時間、10.3.2.3.5 日付，および 10.3.2.3.7 期間 に記述されている構文に従うべきである。

7.2.3　囲み起動

起動は、ビジネス知識モデル体系を評価するためのコンテキストを提供する、パラメータのバインディングのためのコンテナである。

起動は､明示的にリストされたパラメータのバインディングを伴うビジネス知識モデルの名前で表される。

囲み式として、起動は呼び出されるビジネス知識モデルの名前を含む枠と、バインディングをリストした枠によって表現される。それぞれのバインディングは、一つの行に二つの囲み式を記述することで表される。左側の枠にはパラメータ名が入り、右側にはバインディング式が入る。そのバインディング式は､呼び出されたビジネス知識モデルを評価目的のパラメータに割り与えられる式である。（図 30 参照）

図 30：囲み起動

呼び出されたビジネス知識モデルは、そのビジネス知識モデルの名前で表される。その他の視覚的なつながりは、すべて実装に委ねられる。

7.3 メタモデル

DMN における意思決定とビジネス知識モデルの重要な特性の一つは、それらが、あるモデル化された意思決定が行われるときに従うべきロジック、またはロジックの一部を明らかにする式を含むことがある

Expression というクラスは、DMN モデルの意思決定ロジックの一部またはすべてを記述するために用いられる、すべての式を抽象化したスーパークラスであり、インタープリートされたとき（7.3.1 Expression メタモデル 参照）に、単一の値を返す。ここの「単一の値」は複数のアウトプット行をもつデシジョンテーブルのような構造化データであってもよい。

DMN は、Expression を具象化した 3 つのクラス LiteralExpression、DecisionTable（8 章 参照)、Invocatoin を定義する。

インタープリートされたとき､その式の値が参照された変数に割り当てられた値に依存するなど、式は変数を参照することもある。InformationItem クラスは、式の中で変数をモデル化するために使用される。

変数に割り当てられた値のように、式の値は構造と許容範囲をもつことがある。ItemDefinition クラスは、データの構造と範囲をモデル化するために使用される。

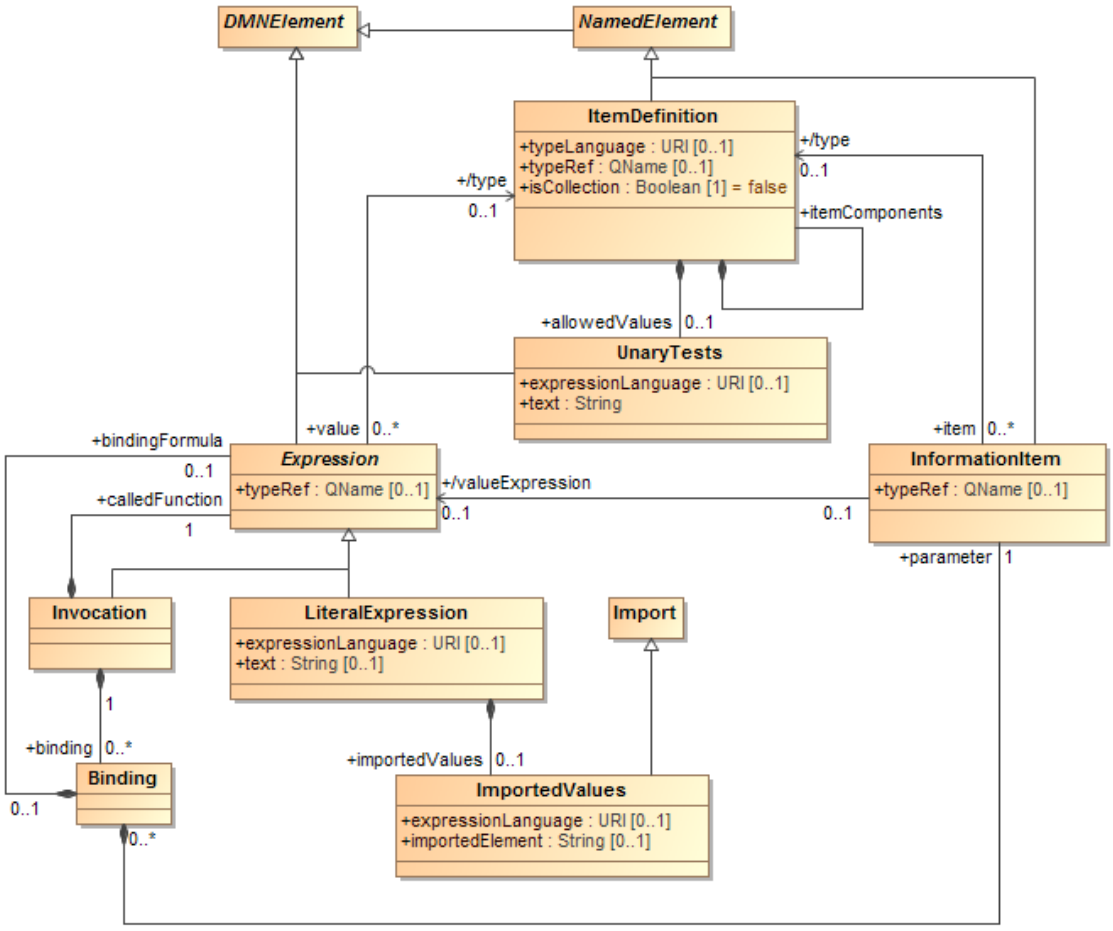


図 31-Expression クラス図

7.3.1 Expressionメタモデル

意思決定とビジネス知識モデルの重要な特性の一つは、それらが、あるモデル化された意思決定が行われるときに従うべきロジック、またはロジックの一部を明らかにする式を含むことがある。

Expression は DMNElement の特化であり、そこから name、およびオプショナルな id と description と label の各属性を継承する。

Expression のインスタンスは直接的または間接的に使用される、Decision 要素、BusinessKnowledgeModel 要素、または ItemDefinition 要素のコンポーネント、あるいは Expression の他のインスタンスのコンポーネントである。

Expression のテキストに名前を使用して暗黙的に変数を参照できる。InformationItem のインスタンスであるこれらの変数は、Expression 型ごとの静的スコープになっている。

Expression が Decision のロジックである場合、Decision の要求がスコープに含まれる。Expression が BusinessKnowledgeModel の encapsulatedLogic の主体である場合、FunctionDefinition のパラメータと BusinessKnowledgeModel 要求がスコープに含まれる。Expression が ContextEntry の値である場合、Context の中に、事前エントリーをスコープに含める。Expression のインスタンスはオプションの typeRef を参照する。この typeRef, は、デフォルトの typeLanguage の基本型、ItemDefinition で指定されたカスタム型、またはインポート型のいずれかを指す。参照された型は、Expression の取りうる値の範囲を指定する。Decision 要素の出力を定義する Expression のインスタンスに typeRef を含む場合、参照された型は、Decision 要素の型と同じでなければならない。

Expression のインスタンスは、その変数へ割当られた値から単一の値を導き出すために、翻訳される。Expression 要素の値がどのようにその変数へ割り当てられた値から導かれるかは、Expression の具体的な型に依存する。

Expression は DMNElement の属性とモデル・アソシエーションを継承する。

7.3.2 ItemDefinitionメタモデル

意思決定の入力と出力は、意思決定ロジック・レベルにおいて、その値が変数に割り当てられるか、または値式によって表されるデータ項目である。

意思決定モデルのデータ項目の重要な特性は、それらの構造である。DMN はデータ構造のための特定の形式を要しないが、デフォルトとして FEEL のサブセットを指定する。

ItemDefinition クラスは、構造および意思決定の入力値と出力値の範囲をモデル化するために用いられる。

NamedElement を特化した具象クラスとして、ItemDefinition のインスタンスには name とオプションの id と description がある。ItemDefinition 要素の name は、Definitions のインスタンスと名前空間内で一意でなければならない。

すべての要素のデフォルト・タイプの言語は、typeLanguage 属性を用いて Definitions 要素の中で指定することができる。例えば”http://www.w3.org/2001/XMLSchema”という typeLanguage の値は、その Definitions 内で、要素によって用いられるデータ構造は XML スキーマ・タイプの形式であることを示す。何も指定されていなければ、デフォルトは FEEL である。

Definition のインスタンスに関連する typeLanguage に組み込まれているデータタイプは、その Definitions 要素に含まれる ItemDefinitions 要素によって再定義される必要はない。それらはインポートされたとみなされ、Definitions 要素内の DMN 要素の中で参照することができる。

言語タイプは、ItemDefinition 要素内の typeLanguage 属性を用いてローカルに上書きできる。

Definitions のインスタンスに関連する Import 要素を用いてインポートされたデータ・モデルの最上位で定義されたデータタイプとデータ構造は、その Definitions 要素に含まれる ItemDefinition 要素によって再定義する必要がないことに留意すること。それらは、インポートされたとみなされ、Definitions 要素内の DMN 要素の中で参照することができる。

ItemDefinition 要素は、typeRef を持ち、それは名前空間の接頭語とローカル名によって定義される QName であり、現行インスタンスの ItemDefinition またはインポートされた DMN,XSD,他の文書内で定義された組み込み型のいずれかが参照される。後者の場合には、その外部文書は ItemDefinition のインスタンスを含む Definitions の中で Import 要素を用いてインポートされなければならない。例えば XML スキーマによって与えられたデータ構造の場合は、Import は、そのスキーマのファイルの存在場所を指定するために使用される。そして typeRef 属性はそのインポートスキーマにおけるタイプまたは要素定義を参照する。タイプ言語が FEEL であれば、組み込まれたタイプは FEEL に組み込まれたデータタイプ、すなわち、*数値*、*文字列*、*論理型*、*日付*、*年月*、*時間*、*日時*である。

ItemDefinition 要素は、allowedValue 属性を用いて typeRef から許可された値を制約することができる。allowedValue は typeRef のドメイン内で許可された値または許可された値の範囲を指定する unaryTests のインスタンスである。許容される値の型は、ItemDefinition 要素と矛盾してはならない。ItemDefinition 要素が allowedValues を含む場合、allowedValues は、ItemDefinition が表す値の明確な範囲を指定する。ItemDefinition 要素に allowedValues が含まない場合、許容される値の範囲は、参照される typeRef 全体の範囲である。ItemDefinition 要素の表す値が許容範囲内の値の集合である場合、多重度は isCollection 属性に投影される。 この属性のデフォルト値は*偽*である。

ItemDefinition のインスタンスを定義する別の方法は、ItemComponent 要素を複合したものとして定義することである。ItemDefinition のインスタンスは、itemComponent を含むことがあり、itemComponent 自体は ItemDefinitions である。各 itemComponent については、typeRef と allowedValues または入れ子の itemComponent によって定義される。 このよ

うにして、複合した型を DMN 内で定義することができる。itemComponent（入れ子の ItemDefinition）の名前は、ItemDefinition 内で一意でなければならない。ItemDefinition 要素は複数の代替的な方法のうち、いずれか一つを用いて定義されなければならない。

- 組み込みまたはインポートされた typeRef への参照。allowedValues で制限されることもある。
- ItemDefinition 要素の組合せ。

temDefinition 要素は NamedElement を特化し、その属性およびモデルのアソシエーションを継承する。表 23 は、ItemDefinition 要素の追加の属性およびモデル・アソシエーションを示す。

表 23：ItemDefinition の属性とモデル・アソシエーション

Attribute 属性	Description 説明
typeRef: String [1]	名前空間接頭語で ItemDefinition の基本型を特定する。
typeLanguage: String [0..1]	ItemDefinition の基本タイプを指定するために使用される言語タイプを特定する。この値は、Definitions 要素で指定された言語タイプを上書きする。URI 形式で指定されなければならない。
allowedValues: UnaryTests [0..1]	ItemDefinition で許可された基本タイプの値または値の範囲をリストする。
itemComponent: ItemDefinition[*]	ItemDefinition を構成する入れ子された ItemDefinitions があれば定義する。
IsCollection: Boolean	このフラグが真ならば、ItemDefinition によって定義された実際の値は、許可された値の集合である。デフォルトは偽。

7.3.3 InformationItemメタモデル

InformationItem クラスは、意思決定モデルの意思決定ロジック・レベルで変数をモデル化するために用いられる。

InformationItem は、NamedElement を具象化したサブクラスであり、id、オプションの

name、description、label 属性を継承する。ただし InformationItem 要素に他の Expression 要素で表現するために使用される name 属性は除く。InformationItem 要素の name は、スコープ内で一意でなければならない。

変数は、意思決定の入力となる値、外部データソースからの入力データに割り当てられた値、または関数として定義されている意思決定ロジックのモジュールに渡される（そして、ビジネス知識モデル要素によって表された）値を示す。第 1 または第 2 のケースでは、変数は、情報要求線を用いた他の従属した意思決定によって参照される。第 3 のケースでは、変数は、意思決定ロジック・レベルにおいて、ビジネス知識モデル要素を実現した結果である関数のパラメータの一つである。

InformationRequirement によって参照される Decision または InputData のインスタンスを表す変数は、InformationRequirement 要素を含む Decision 要素内の意思決定ロジックの値式によって参照される。 BusinessKnowledgeModel のインスタンスのパラメータは、その BusinessKnowledgeModel 要素の値式の変数である。

Decision に含まれる InformationItem 要素には、Decision の値式の値が代入される。

- FunctionDefinition のパラメータである InformationItem 要素には、Invocation のインスタンスの一部として Binding 要素の値が代入される。
- InputData に含まれる InformationItem 要素には、実行時に添付される外部データソースの値が代入される。
- ContextEntry に含まれる InformationItem 要素には、ContextEntry の値式の値が代入される。

いずれにせよ、InformationItem のインスタンスに関連付けられた typeRef によって示されるデータ型は、その値を取る DMN モデル要素に関連付けられたデータ型と互換であること。InformationItem は、NamedElement のすべての属性とモデル・アソシエーションを継承する。表 24 に、InformationItem 要素の追加の属性とモデル・アソシエーションを示す。

表 24：InformationItem の属性とモデル・アソシエーション

属性	説明
/valueExpression: Expression [0..1]	値がこの InformationItem に代入される式。これは誘導属性である。
typeRef: QName [0..1]	InformationItem の型の修飾名。

7.3.4 Literal expressionメタモデル

LiteralExpression クラスは、特定の式言語においてテキストによる値が指定される値式をモデル化するために用いられる。

LiteralExpression は、Expressoion を具象化したサブクラスであり、そこから id と typeRef 属性を継承する。

LiteralExpression のインスタンスは、String であるオプショナルな text および、text の式言語を特定する String であるオプショナルな expressionLanguage をもつ。expressionLanguage が指定されない場合、text の式言語は、Definitions に含まれるインスタンスに関連する expressionLanguage である。expressionLanguage は、URI 形式で指定しなければならない。デフォルトの式言語は FEEL である。

Expression のサブクラスとして、LiteralExpression のそれぞれのインスタンスは値をもつ。LiteralExpression のインスタンスの中の text は、LiteralExpression の expressionLanguage のセマンティクスにしたがって、その値を決定する。本仕様書で記述されている DMN1.1 の意思決定モデルのセマンティクスは、そのモデルの中の LiteralExpression のすべてのインスタンスの text が、それらに関連する式言語の中で有効な式である場合にのみ適用される。

LiteralExpression のインスタンスには、literalExpression のテキストの位置を識別する Import サブクラスのインスタンスである importedValues が含まれる。importedValues は、インポートされたドキュメントからテキストを選択する式である。LiteralExpression のインスタンスは、テキストと importedValues の両方を持ってはならない。 importedValues の importType は、インポートされたテキストを含むドキュメントの型を識別し、LiteralExpression 要素の expressionLanguage と整合しなければならない。importedValues 要素の expressionLanguage は、インポートするテキストからテキストを抽出する方法を識別する。たとえば、importType が XML 文書を示す場合、importedValues の expressionLanguage は XPATH 2.0 になりうる。

LiteralExpression は Expression のすべての属性とモデル･アソシエーションを継承する。表 25 は LiteralExpression 要素の追加となる属性とモデル・アソシエーションを示す。

表 25－LiteralExpression の属性とモデル・アソシエーション

属性	説明
text: string [0..1]	LiteralExpression のテキスト。expressionLanguage の妥当な式でなければならない。
expressionLanguage: anyURI [0..1]	LiteralExpression で用いられる式言語を特定する。この値は、DecisionRequirementDiagram の containing インスタンスのために指定された式言語を上書きする。この言語は URI 形式で指定しなければならない。
importedValues: ImportedValues [0..1]	LiteralExpression のテキストが存在する場所を指定する ImportedValues のインスタンス。

7.3.5 Invocationメタモデル

起動は、他の値式（呼び出した値式）の中で、呼び出された式の入力変数と呼び出している式の中の値をローカルに結び付けることにより、一つの値式（呼び出された式）の評価を可能にするメカニズムである。起動では、呼び出された式の入力変数は、通常、parameters と呼ばれる。起動は、同じ値式を、使用している全ての式の中で副次式として重複することなく複数の式で再利用することを可能にする。

Invocation クラスは、Expression の一種として起動をモデル化するために使用される。Invocation は、Expression の特化である。

Invocation のインスタンスは、0 以上の binding、つまり Binding のインスタンスであり、bindingFormula が呼び出された関数の formalParameter にどのようにバインドされるかをモデル化する。 FunctionDefinition の formalParameter は、InformationItem であり、Binding のパラメータは、InformationItem である。InformationItem の名前を照合することによってバインディングされる。

Invocation には、必ず関数に評価される Expression である calledFunction が含まれる。

最も一般的には、BusinessKnowledgeModel という名前の LiteralExpression である。

Invocation のインスタンスの値は、関連付けられた calledFunction 本体の値であり、Invocation の中で実行中にバインディングされるごとに、formalParameters に値が代入される。

Invocation は意思決定モデルの起動モデルに使われるのは次の場合である。Decision 要素がひとつの knowledgeRequirement 要素だけを持ち、Decision 要素の decisionLogic が,requiredKnowledge が参照する BusinessKnowledgeModel 要素を呼び出すことのみで構成され、かつより複雑な値式が存在しない場合。

Decision 要素の decisionLogic として Invocation インスタンスを使用すると、BusinessKnowledgeModel を必要とする Decision の任意のインスタンスのロジックとして BusinessKnowledgeModel の encapsulatedLogic を再利用することができる。各 Decision 要素は encapsulatedLogic のパラメータに対して独自のバインディングを指定する。

Invocation 要素に関連付けられている calledFunction は、Invocation を含む Decision 要素が必要とする BusinessKnowledgeModel 要素の encapsulatedLogic である。Invocation 要素は、BusinessKnowledgeModel の encapsulatedLogic の各パラメータに対してひとつだけのバインディングを持つ。

Invocation は、Expression のすべての属性とモデル・アソシエーションを継承する。 表 26 は、Invocation 要素の追加の属性とモデル・アソシエーションを示す。

表 26：Invocation 属性とモデル・アソシエーション

属性	説明
calledFunction: Expression [1]	値が関数である式。
binding: Binding [*]	この属性は、この Invocation の calledFunction の formalParameters をバインドするために使用される Binding のインスタンスをリストする。

7.3.6 Bindingメタモデル

Binding クラスは、Invocation 要素で、calledFunction の formalParameters を値にバインドするために使用される。

Binding は、ひとつの bindingFormula（これは Expression である)と、ひとつのパラメータ（これは InformationItem である）からできている。

Binding 要素のパラメータ名は､calledFunction の formalParameters のサブセットでなければならない。

Invocation 要素が実行されると、Invocation 要素の binding によって parameter として参照される各 InformationItem 要素に、実行時に、bindingFormula の値が代入される。

表 27 に、Binding 要素の属性とモデル・アソシエーションを示す。

表 27：Binding 属性とモデル・アソシエーション

属性	説明
parameter: InformationItem	Invocation が所有しているインスタンスの calledFunction が依存する InformationItem 。その Invocation は Binding によってバインドされている。
bindingFormula: Expression [0..1]	所有しているインスタンスが評価されるときの Expression のインスタンス（バインドされる Bind のパラメータ)。

8 デシジョンテーブル

8.1 イントロダクション

DRD 意思決定成果物に対応する意思決定ロジックを説明する一つの方法はデシジョンテーブルである。デシジョンテーブルは関係する一式の入力と出力式を表形式で表したもので、特定の入力エントリーを出力エントリーに対応させることで、ルールを構成する。デシジョンテーブルには全て（必要十分）の出力を決定するのに必要なすべての入力を含む、(含むのはそれだけでもある)。さらに、完全な表には、すべての可能性のある組み合わせの入力値が定義される(全てのルール).

デシジョンテーブルとデシジョンテーブル体系は意思決定ロジック表現の分野で実績がある。相異なるデシジョンテーブルのフォームと型を DMN の標準として定義することも本章の目的のひとつである。

デシジョンテーブルは、以下のものから構成される：

- 情報項目名:デシジョンテーブルの値式である InformationItem の名前。これは通常、デシジョンテーブルが意思決定ロジックを提供する意思決定またはビジネス知識モデルの名前になる。
- 出力ラベル、出力ラベルはデシジョンテーブルの出力を説明するどんなテキストでも良い。他の式において、デシジョンテーブルの結果は、出力ラベルではなく、情報項目名を使用して参照しなければならない。
- 一式の入力(0 以上)。各入力はひとつの入力式と多数の入力エントリーから構成される。入力式とすべての入力エントリーの詳細は入力節として言及される。
- 一式の出力。単一の出力には名前はなく、値のみである。複数の出力は出力コンポーネントと呼ぶ。各出力コンポーネントには名前を付けなければならない。各出力(コンポーネント) は各ルールの出力エントリーを明確にしなければならない。出力コンポーネントの名前(複数の出力がある場合)と全ての出力の詳細は出力節において言及される。
- 表の行または列のルールのリスト (方向によって異なる)。各ルールは表の行 (または列) における具体的な入力エントリーと出力エントリーから構成される。ルールが行として表わされる場合、列は節であり、その逆も同様。

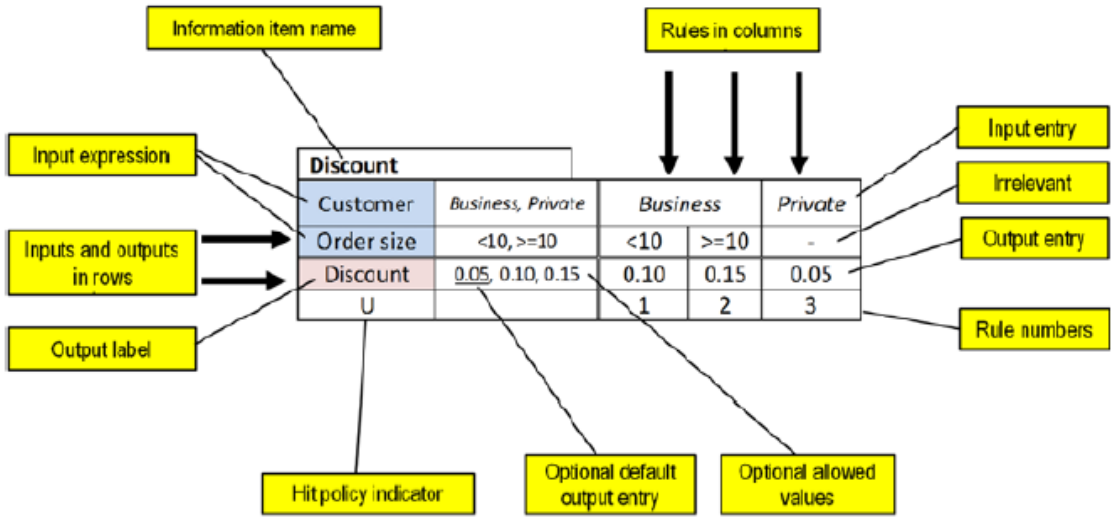


図 32：デシジョンテーブルの例（垂直型テーブル：列のルール）

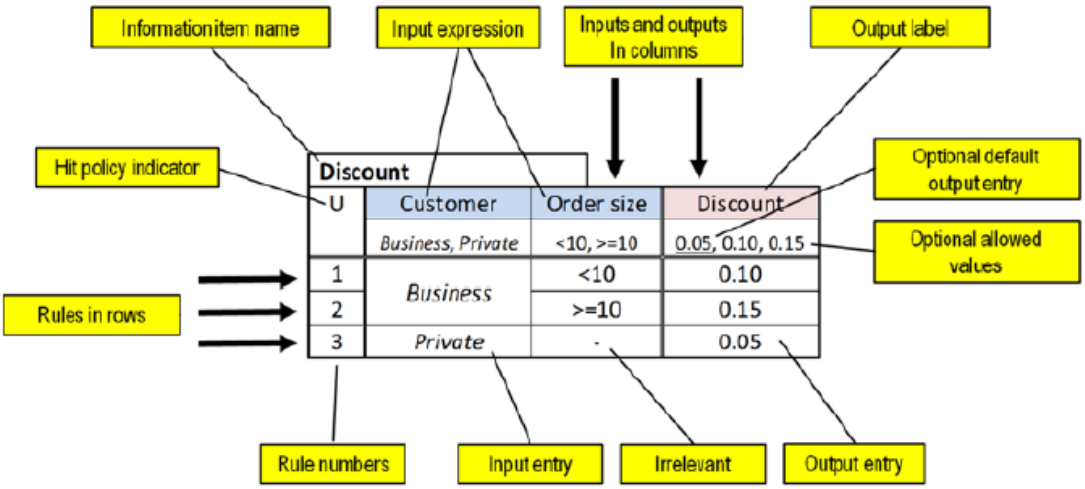


図 33：デシジョンテーブルの例（水平型テーブル：行のルール）

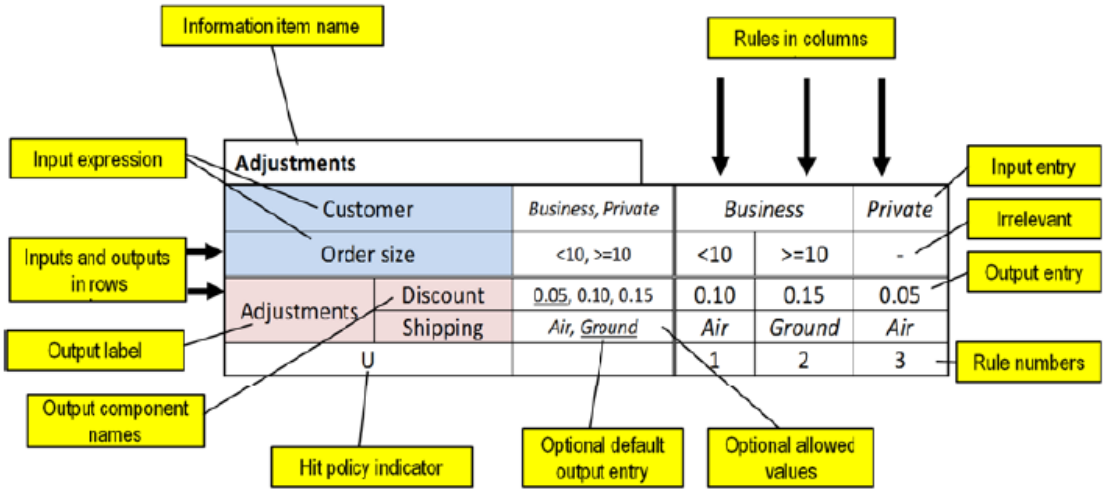


図 34：デシジョンテーブルの例（垂直型テーブル：複数の出力コンポーネント）

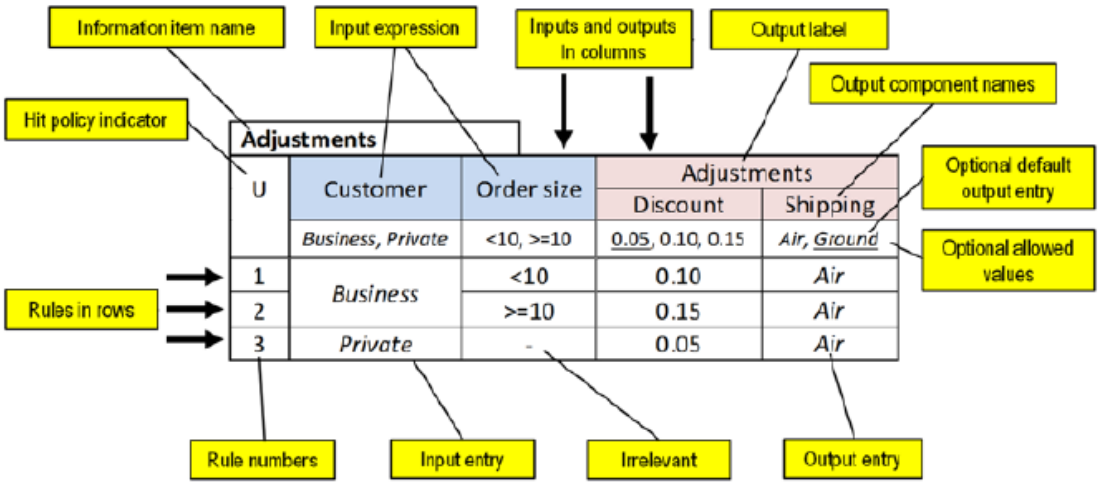


図 35：デシジョンテーブルの例（水平型テーブル：複数の出力コンポーネント）

デシジョンテーブルは、表のセルにエントリーを配置するという簡易な表記方法で、ルールを表示している。この簡易な表記方法は各ルールの全ての入力を同じ順序で表示する、この方法の利点は読みやすく検証が容易なことである。
例えば：

Customer	OrderSize	Discount
Business	<10	0.10

以下の様に読解する：
If Customer = “Business” and OrderSize < 10 then Discount = 0.10

これを一般化すると、つぎのように表す。

input expression 1	input expression 2	Output label
input entry a	input entry b	output entry c

上のデシジョンテーブル（部分）にある 3 つの強調表示されたセルは、次のルールを示している：

expression1 の入力値が入力エントリーa を満たし
かつ、expression2 の入力値は入力エントリーb を満たすならば、
ルールと合致し、デシジョンテーブルの結果は出力エントリーc となる。

値が等しいまたは入力エントリーまたは 入力エントリーで示された値リスト（例えばリストや範囲）に含まれる場合には入力式の値は入力エントリーを満足する。入力値

が‘-’(関係なしと)の場合、入力式のすべての値は入力エントリーの要求を満たすが、その特定の入力は設定されたルールとは無関係である。

各入力式の値がそれぞれ対応する入力エントリーを満たす場合、ルールは合致する。入力がない場合も、ルールは合致する。

ルールのリストは、意思決定のロジックを表す。与えられた入力値のセットがある場合、合致ルール（または複数の合致ルール）は出力名のための結果の値を示す。ルールが重複している場合は、複数のルールが合致するため、hit policy が複数の合致をどのように扱えば良いか、その方法を示す。

同じ入力式の２つの入力エントリーが値を共有しない場合、そのエントリー（セル）は分離(*disjoint*)と呼ばれる。共通部分がある場合、そのエントリーは重複（または引き分け)と呼ばれる。関係なし（‘-’）は、入力式の入力エントリーと重複する。

すべての対応する入力エントリーが重複している場合、2つのルールは重複している。入力データの具体的な設定は、2つのルールに合致する。

すくなくとも１組の対応する入力式が分離している場合、2つのルールは分離している（重複していない)となる。入力データの具体的な設定で 2 つのルールに合致するものは無い。

表がルールの重複を許される場合、不整合を避けるために、テーブルヒット・ポリシーは、重複したルールと出力名のための結果値の取扱い方法を示す。

8.2　表記法

この節は、7.2 節の表記法で定義された意思決定と囲み式の一般的な表記法を基づいている。デシジョンテーブルの表現の標準化は以下の通りである：

- テーブルとして示される並べ方（行また列もしくはクロス・テーブルなどのルール)。
- 標準化されたセルのグリッド上に、入力と出力と(オプションの)許容値を配置。各入力式は、オプションで単項式テストに関連付けられて入力値が制限される。本規定では、許容値を持つオプションの出力値は反転される。各出力（コンポーネント）は、オプションで許容値に関連付けられる。本規定では、許容値を持つオプションの出力値は反転される。
- 線のスタイルとオプションの色の使用。
- 具体的なルールと入力および出力エントリーのセルの内容。
- ヒット・ポリシー。重複する入力の組み合わせを解釈する方法を示す。

- 図 36、図 38、および図 40 に示すように、情報項目名、ヒット・ポリシー（H)、ルール番号の配置。ルール番号は連続した自然数で 1 から始まる。
  ヒット・インジケータ F（最初）または R（ルール順）が付いたテーブルにはルール順序に依存するため、ルール番号が必要である。クロス・テーブルにはルール番号はない。ルールの番号付けは、他のテーブル・タイプではオプションである。

入力式、入力値、出力値、入力エントリー、出力エントリーは、任意のテキスト（例えば、自然言語、形式言語、擬似コード）で記載することができる。レベル 2 または 3 に準拠することを主張する実装は、(S-) FEEL 構文をサポートしなければならない。レベル 1 に準拠すると主張する実装は、式を解釈する必要はない。誤った解釈を避けるために（例えば、式が有効でない（S-) FEEL では、(S-) FEEL 構文の見た目＆操作性と異なる場合)、入力式が URI： ″http://www.omg.org/spec/DMN/uninterpreted/20140801″を使用して解釈出来ない場合には、準拠する実装はそのことを示すべきである。

8.2.1 線の種類と色

線のスタイルには標準がある。入力セクションと出力セクションの間には二重線があり、入力式/出力式とルールエントリーのセルの間にも二重線がある。他のセルは一重線で区切られる。

色は提言されているが、その意味には影響しない。入力式セクションと出力名セクションには異なる色を使用し、ルールエントリーには別の色（または無色）を使用することが良いと認める。

8.2.2 テーブルの型

サイズに応じて、デシジョンテーブルは水平型テーブル（行としてのルール)、または垂直型テーブル（列としてのルール)、もしくはクロス表（2 つの入力次元から構成されるルール）として表示できる。クロス表にはデフォルトのヒット・ポリシーしか設定できない（後述)。

デシジョンテーブルの入力と出力を混在させてはならない。水平型テーブルでは、すべての入力列がすべての出力列の左に表示されなければならない。 垂直型テーブルでは、すべての入力行がすべての出力行の上に表示されなければならない。 クロス表では、すべての出力セルが表の右下部分に表示されなければならない。

表は次のいずれかの方法で配置されなければならない（図 36、図 38、図 40 を参照）。反転で表示されたセルはオプションである。

入力セルの入力値 ’－’は「関係なし」を意味する。 HC（ハイフン文字）は、ヒット・ポリシーを示すインジケータ（たとえば、U、A、F、...）のプレースホルダである。

information item name			
H	input expression 1	input expression 2	Output label
	value 1a, value 1b	value 2a, value 2b	value 1a, value 1b
1	input entry 1.1	input entry 2.1	output entry 1.1
2		input entry 2.2	output entry 1.2
3	input entry 1.2	-	output entry 1.3

図 36：行としてのルール － 図レイアウト

Discount				
U	Customer	OrderSize	Delivery	Discount
	Business, Private, Government	<10, >=10	*sameday, slow*	0, 0.05, 0.10, 0.15
1	*Business*	<10	-	0.05
2		>=10	-	0.10
3	*Private*	-	*sameday*	0
4			*slow*	0.05
5	*Government*	-	-	0.15

図 37：行としてのルール － 例

information item name				
input expression 1	value 1a, value 1b	input entry 1.1		input entry 1.2
input expression 2	value 2a, value 2b	input entry 2.1	input entry 2.2	-
Output label	value 1a, value 1b	output entry 1.1	output entry 1.2	output entry 1.3
H		1	2	3

図 38：列としてのルール － 図レイアウト

Discount						
Customer	*Business, Private, Government*	*Business*		*Private*		*Government*
Ordersize	<10, >=10	<10	>=10	-		-
Delivery	*sameday, slow*	-	-	*sameday*	*slow*	-
Discount	0, 0.05, 0.10, 0.15	0.05	0.10	0	0.05	0.15
U		1	2	3	4	5

図 39：列としてのルール － 例

information item name			
Output label		input expression 1	
		input entry 1.1	input entry 1.2
input expression 2	input entry 2.1	output entry 1.1	output entry 1.3
	input entry 2.2	output entry 1.2	output entry 1.4

図 40：クロス表としてのルール － 図レイアウト（オプションの入力および出力値は表示されない）

Discount				
Discount		Customer		
		Business	*Private*	*Government*
Ordersize	<10	0.05	0	0.15
	>=10	0.10	0	0.15

図 41：クロス表としてのルール － 2 つの入力のみを持つ単純化された例

Discount					
Discount		Customer, Delivery			
		Business	*Private*		*Government*
		-	*sameday*	*slow*	-
Ordersize	<10	0.05	0	0.05	0.15
	>=10	0.10	0	0.05	0.15

図 42：クロス表としてのルール － 3 つの入力を持つ例

2 以上の入力を持つクロス表もあり得る（図 42 を参照)。

8.2.3 入力式

入力式は通常単純である、例は、名前（CustomerStatus など）やテスト（Age <25 など)。

入力式の順序は、実装時の実行順序とは関係ない。

8.2.4 入力値

入力式は、数の制限または値の範囲制限を設定できる。デシジョンテーブルを完全と見なすためには、入力期待値をモデル化することは重要である、なぜならそのルールがすべての入力式について入力期待値の組み合わせをカバーしている必要があるからである。

入力期待値をどのようにモデル化されているかにかかわらず、入力値は相互に排他的で完全であるべきである。

相互に排他的とは入力値が互いに素であるという意味である。完全とはすべての関連する入力値が当該領域に存在することを意味する。

たとえば、次の 2 つの入力値の範囲、<5 と<10 は、重なる。次の 2 つの範囲、<5 と> 5 は不完全である。

入力値のリストはオプションである。入力値のリストが作成される場合は、対応する入力は満たすかという単項式テストのリストとなる。

8.2.5 情報項目名、出力ラベル、出力コンポーネント名

複数の出力コンポーネントを含むデシジョンテーブルは、各出力コンポーネントに名前を設定する必要がある。

InformationItem の値を表現しているデシジョンテーブル（例えば、意思決定ロジッ

クや囲み式起動バインド式）は、InformationItem として情報項目名を設定しなければならない。他の囲み式の無いデシジョンテーブルでは、すぐ上に隣接する名前ボックスに情報項目名をレイアウトしなければならない。

他の囲み式があるデシジョンテーブルでは、情報項目名に含まれている式を使用できる。たとえば、関数パラメータにバインドされたデシジョンテーブルの情報項目名は、関数パラメータ名である。また、スペースを節約するために、 情報項目名ボックスを無くし、出力ラベルで代用することができる。

8.2.6 出力値

デシジョンテーブルの出力エントリーは、しばしば、出力値のリストを基に描かれる。

出力値のリストはオプションである。出力値のリストが 作成される場合には、それは所与の出力値に対する出力エントリーの制限リストである。

ヒット・ポリシーが P（優先）の場合、複数のルールが合致する可能性がある、しかしただ 1 回のヒットだけが返されるべきである、出力値のリストの順序（降順）が優先度の指定に使用される。

出力値のリストの順序は、ヒット・ポリシーが出力順である場合にも使用される。

8.2.7 複数の出力

デシジョンテーブルは合成出力を表示できる（図 43、図 44、図 45 を参照）。

information item name				
H	input expression 1	input expression 2	output label	
			output component 1	output component 2
	input value 1a, input value 1b	input value 2a, input value 2b	output value 1a, output value 1b	output value 2a, output value 2b
1	input entry 1a	input entry 2a	output entry 1.1	output entry 2.1
2		input entry 2b	output entry 1.2	output entry 2.2
3	input entry 1b	-	output entry 1.3	output entry 2.3

図 43：複数の出力コンポーネントを含む水平型テーブル

<table>
<tr><td colspan="5">information item name</td></tr>
<tr><td colspan="2">input expression 1</td><td>input value 1a,
input value 1b</td><td colspan="2">input entry 1a</td><td>input entry 1b</td></tr>
<tr><td colspan="2">input expression 2</td><td>input value 2a,
input value 2b</td><td>input entry 2a</td><td>input entry 2b</td><td>-</td></tr>
<tr><td rowspan="2">output
label</td><td>output
component
1</td><td>output value 1a,
output value 1b</td><td>output entry 1.1</td><td>output entry 1.2</td><td>output entry 1.3</td></tr>
<tr><td>output
component
2</td><td>output value 2a,
output value 2b</td><td>output entry 2.1</td><td>output entry 2.2</td><td>output entry 2.3</td></tr>
<tr><td colspan="2">H</td><td></td><td>1</td><td>2</td><td>3</td></tr>
</table>

図 44：複数の出力コンポーネントを含む垂直型テーブル

<table>
<tr><td colspan="4">information item name</td></tr>
<tr><td colspan="2">output label</td><td colspan="2">input expression 1</td></tr>
<tr><td colspan="2">output component 1,
output component 2</td><td>input entry 1a</td><td>input entry 1b</td></tr>
<tr><td rowspan="2">input expression
2</td><td>input entry
2a</td><td>output entry 1.1,
output entry 2.1</td><td>output entry 1.3
output entry 2.3</td></tr>
<tr><td>input entry
2b</td><td>output entry 1.2,
output entry 2.2</td><td>output entry 1.4,
output entry 2.4</td></tr>
</table>

図 45：複数の出力コンポーネントを含むクロス表

8.2.8　入力エントリー

ルールの入力エントリーは式である。

ダッシュ記号（’－’）は、任意の入力値を意味するために使用する、つまり、入力は含まれているルールとは無関係となる。

単項式テストの入力エントリーは、’－’または入力値制限のサブセットでなければならない。たとえば、入力 ’Age’の入力値が[0..120]と制限されている場合、マイナスとなる入力エントリーは無効と判定される。

’-’である入力エントリーが少なくとも１つある表は、縮小表と呼ばれる。 それ以外は拡大表と呼ばれる。

各入力エントリーが真または偽もしくは’-’である表は、歴史的には制限付エントリー表と呼ばれていたが、この規定を維持する義務はない。

デシジョンテーブルにおける入力式の評価は、他の入力式の評価に影響を与えるよう

な副作用を生じない。この意味は、式の評価やルールの実行は同じ表の他の式やルールの評価を変えてはならないということである。これは、ルールが事前に定義された順序で評価される最初のヒット・テーブルにおいて特に重要である。すなわち、ルールの評価または実行は他のルールに影響を与えてはならない。

8.2.9 マージ化入力エントリーセル

図 46 と図 47 に示す様に、同じ内容で同じ先行するセルを持つ場合は（先行するセルが無いこともある）、異なるルールの隣接した入力エントリーはマージできる。ルール出力セルをマージすることはできない（クロス表を除く）。

<table>
<tr><th colspan="4">information item name</th></tr>
<tr><td rowspan="2">H</td><td>input expression 1</td><td>input expression 2</td><td>Output label</td></tr>
<tr><td>input value 1a,
input value 1b</td><td>input value 2a,
input value 2b</td><td>output value 1a,
output value 1b</td></tr>
<tr><td>1</td><td rowspan="2">input entry 1a</td><td>input entry 2a</td><td>output entry 1.1</td></tr>
<tr><td>2</td><td>input entry 2b</td><td>output entry 1.2</td></tr>
<tr><td>3</td><td>input entry 1b</td><td>-</td><td>output entry 1.3</td></tr>
</table>

図 46：ルール入力セルをマージできる場合

<table>
<tr><th colspan="4">information item name</th></tr>
<tr><td rowspan="2">H</td><td>input expression 1</td><td>input expression 2</td><td>Output label</td></tr>
<tr><td>input value 1a,
input value 1b</td><td>input value 2a,
input value 2b</td><td>output value 1a,
output value 1b</td></tr>
<tr><td>1</td><td rowspan="2">input entry 1a</td><td>input entry 2a</td><td>output entry 1.1</td></tr>
<tr><td>2</td><td>input entry 2b</td><td>output entry 1.2</td></tr>
<tr><td>3</td><td rowspan="2">input entry 1b</td><td>input entry 2b</td><td>output entry 1.3</td></tr>
<tr><td>4</td><td>input entry 2a</td><td>output entry 1.4</td></tr>
</table>

図 47：ルール入力セルをマージすることは出来ない場合

8.2.10　出力エントリー

ルールの出力エントリーは式である。

ルールの出力セルはマージできない（クロス表は除く、同じ内容の隣接する出力セルをマージできる）。

簡易表記法

出力名がひとつである（出力名が情報項目名に等しい）垂直型テーブル（ルールは列で表現）では、簡易表記法を使用できる、つまり出力値がある場合には（'X'）、無い場合には（'-'）とすることができる、これはデシジョンテーブルでは一般的な方法である。

出力名はただひとつの出力エントリーしかないため、各々のルールは「X」のみしか表示できない。 他の出力エントリーには'-'となる。

図 48 の表は、図 49 の表の簡易表記法である。ルールの数が増えるにつれて幅が広がる傾向のある縦型テーブルにおいて、出力エントリーをすべての列に書き込む必要はなく、1 文字表記（'X'または'-'）で示すことで垂直型テーブルのスペースを節約するため、簡易表記法と呼ばれる。 出力値は、出力式パートのルールの前に、一度だけ書き込まれる。

情報項目名が提供され出力名が 1 つしかない場合（出力名が情報アイテム名と同じ場合）、出力名は省略可能である。

Applicant Risk Rating					
Applicant Age	< 25		[25..60]	> 60	
Medical History	*good*	*bad*	-	*good*	*bad*
Low	X	-	-	-	-
Medium	-	X	X	X	-
High	-	-	-	-	X
U	1	2	3	4	5

図 48：垂直型テーブルの簡易表記法（ルールは列で表現）

Applicant Risk Rating					
Applicant Age	< 25		[25..60]	> 60	
Medical History	*good*	*bad*	-	*good*	*bad*
Applicant Risk Rating	*Low*	*Medium*	*Medium*	*Medium*	*High*
U	1	2	3	4	5

図 49：垂直型テーブルの完全な表記法（ルールは列で表現）

8.2.11 ヒット・ポリシー

デシジョンテーブルには通常ルールを複数定義する。デフォルトでは、ルールは重複しない。ルールが重複するのは、複数のルールが与えられた入力値のセットと合致していることを意味し、この時テーブル型を認識すると共に意思決定ロジックを明確に理解するために、ヒット・ポリシーのインジケータが必要である。ヒット・ポリシーは設計時に正確性をチェックするために使用できる。

ヒット・ポリシーは、ルールが重複している場合のデシジョンテーブルの結果、すなわち複数のルールが入力データと合致する場合に結果が何になるのかを特定する。明確にするため、ヒット・ポリシーはデシジョンテーブル内の特定セルに、一文字を使って要約される。水平型テーブルでは左上のセル(図 33）で、縦型テーブルでは左下のセル（図 32）となる。 文字は、定義されたヒット・ポリシーの最初の 1 文字（Unique(一意)，Any(任意)，Priority(優先)，First(最初)，Collect(収集)，Output order(出力順),Rule order(ルール順）)である。 クロス表は常に一意であるため、インジケータは必要ない。

ヒット・ポリシーが一意となるようなデフォルトのケースでは、ヒット・インジケータはオプションとなる。 一意となるヒット・ポリシーを持つデシジョンテーブルは、ルールが重複してはならない。

ヒット・ポリシーが空集合でないサブセットの場合のみツールはサポートすることができる、但しテーブルの型を明確にするためにヒット・ポリシーの表示が強要されるデフォルトで一意なテーブルを除く。一意なテーブルは常にサポートされるべきである。

単一および複数ヒット表

単一ヒット表は、ひとつのルールの出力だけを返す。一方、複数ヒット表では複数のルールの出力（または値の合計のような関数の出力）を返すことができる。 重複したルールが許される場合、ヒット・ポリシーは、どのように重複するルールを読み解くかについて示す。

ヒット・ポリシーの最初の文字は、表が単一ヒットか複数ヒットかを示す。

単一ヒット表は、重複するルールがあってもなくても、ひとつのルールの出力のみが返される。ルールが重複しているケースでは、ヒット・ポリシーは複数の合致したルールからどれを選択するか示す。いくつかの制限が合成出力を持つテーブルに適用される。

単一の出力を持つデシジョンテーブルのための単一**ヒット・ポリシー**は以下の通り：

1. Unique(一意)：重複は無くすべてのルールは互いに素であること。ただひとつのルールのみ合致する。これがデフォルトとなる。
2. Any(任意)：合致するすべてのルールが、各出力に対して等しい出力エントリーを示している場合には、合致するものはすべて使用できる。しかし重複する可能性も残されている。出力エントリーが等しくない場合、ヒット・ポリシーは正しいとは言えず、結果は定義されない。
3. Priority(優先)：異なる出力エントリーを持つ複数のルールが合致できる。このポリシーは、出力優先度が最も高い合致したルールを返す。優先度の高い順に出力値を並べたリストで出力優先度は定義される。ただし優先順位はルールの順序に影響されないという点に留意すること。
4. First(最初)：異なる出力エントリーのある複数の（重複する）ルールが合致できる。ルールを順に評価し最初にヒットしたルールが返される（そして評価は停止できる)。最初のヒットを強制することによって矛盾を解決するため、この方法は依然として一般的な使用方法となっている。しかしながら、明確な意思決定ロジックが示されていないため最初にヒットしたテーブルが適切とはみなせない。選択結果の意味はルールの順序に影響されるため、このタイプのポリシーは特別視することが重要である。最後のルールは通常は余り物となる。順序性があるため、ポリシーを机上で検証するのが難しく、使用には細心の注意が必要である。

複数ヒット表は複数のルールから出力エントリーを返すことができる。
結果は、ルール出力のリストまたは単純な出力関数となる。

単一出力のデシジョンテーブルに対する複数**ヒット・ポリシー**は次のようになる：

5. Output order(出力順)：すべてのヒットを出力優先順位の高い順（降順）で返す。

優先度の高い順に出力値を並べたリストで出力優先度は定義される。

6. Rule order(ルール順)：すべてのヒットをルール順で返す。注：意味はルールの順序によって変わる可能性がある。
7. Collect(収集)：すべてのヒットを任意の順序で返す。 演算子（'+'、'<'、'>'、'#'）を追加して、単純な関数を出力に適用することができる。演算子を追加しない場合、結果は全出力エントリーのリストとなる。

   収集（Collect）演算子は次の通り：

   a）+（sum）：デシジョンテーブルの結果は、すべての識別された出力の合計である。

   b）<（min）：デシジョンテーブルの結果は、すべての出力の最小値である。

   c）>（max）：デシジョンテーブルの結果は、すべての出力の最大値である。

   d）#（count）：デシジョンテーブルの結果は、「識別された出力」の数である。

   上記以外のより複雑な出力操作といったポリシーは、その出力リストを後処理することで実現できる（デシジョンテーブル外で）。

合成出力のあるデシジョンテーブルは、収集（collect）演算子が複数の出力に対して定義できないため、Unique(一意)，Any(任意)，Priority(優先)，First(最初)，Output order(出力順),Rule order(ルール順），演算子を除く Collect(収集)のみをサポートする。

優先（Priority）および出力順（Output order）ヒット・ポリシーでは、出力値が提供される全ての出力について、合成出力テーブルで優先順位を決める。 各出力の優先順位は、優先度の高い順に並んだ出力値の順序リストで設定する。全体的な優先順位は水平型テーブルの場合は左から右への順序付き出力を考慮して設定する（つまり、左側の列が右側の列より優先される)、または垂直型テーブルの場合は上から下へとなる。出力値が提供されない出力は、順序付けられた合成出力テーブルにエントリーが含まれているが、順序付け時には考慮されない。

上記より、たとえば、Age = 17、Risk カテゴリー= "HIGH"、Debt review = true で呼び出された場合、図 50 の Routing Rules テーブルは、2、4、3、1 の順に 4 つのルールすべての出力を返す 。

Routing rules						
O	Age	Risk category	Debt review	Routing	Review level	Reason
		LOW, MEDIUM, HIGH		*DECLINE, REFER, ACCEPT*	*LEVEL 2, LEVEL 1, NONE*	
1	-	-	-	*ACCEPT*	*NONE*	*Acceptable*
2	< 18	-	-	*DECLINE*	*NONE*	*Applicant too young*
3	-	*HIGH*	-	*REFER*	*LEVEL 1*	*High risk application*
4	-	-	true	*REFER*	*LEVEL 2*	*Applicant under debt review*

図 50：複合出力による出力順序

注 1

クロス表は一意でありクロス表の定義から完全であるといえるため、ヒット・ポリシーは必要ない。

注 2

最初（First）（単一ヒット）およびルール順（Rule）（複数ヒット）を除き、デシジョンテーブル内のルールの順序は、意味に影響しない。これらの表は慎重に使用する必要がある。

8.2.12　デフォルト出力値

テーブルはデフォルト出力を指定することができる。 デフォルト値は、出力値のリストに下線が引かれる。

8.3　メタモデル

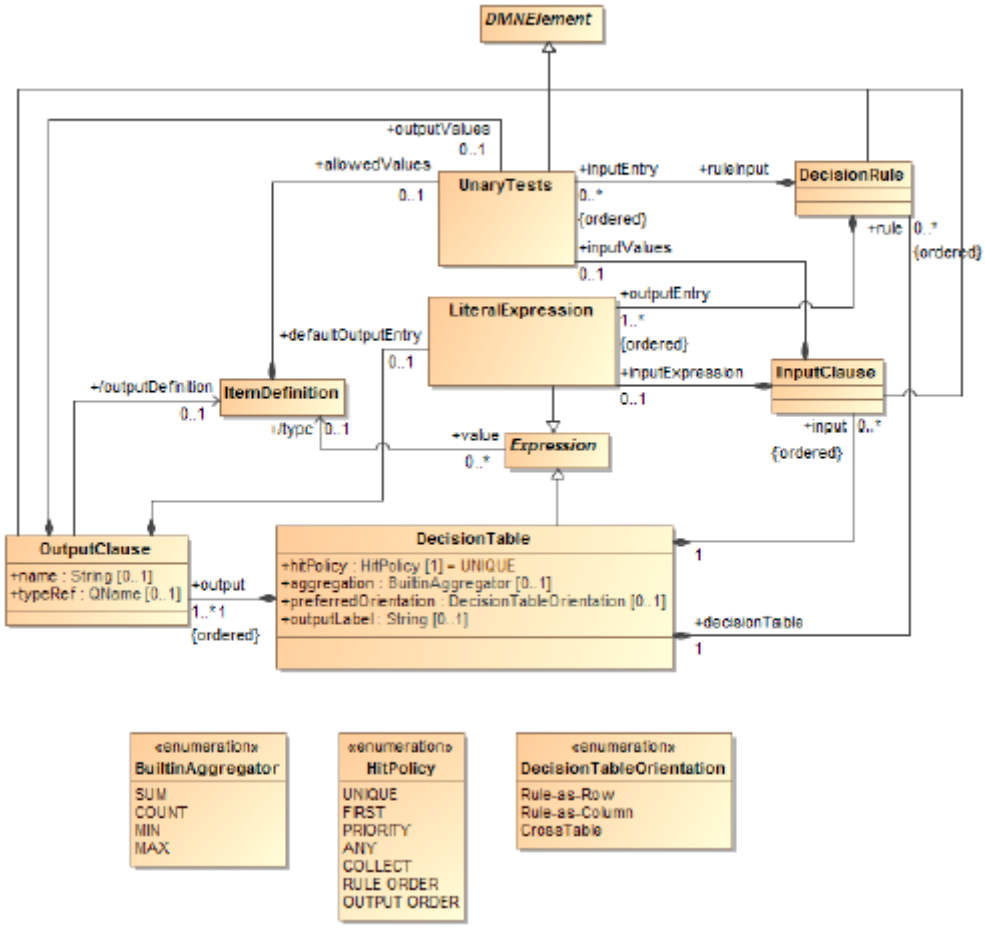


図51：DecisionTableのクラス図

8.3.1　Decision Tableメタモデル

DecisionTable クラスは、デシジョンテーブルをモデル化するために使用される。

DecisionTable は Expression の具象化である。

DecisionTable のインスタンスには、DecisionRule のインスタンスであるルールの集まり、InputClause のインスタンスである入力の集まり、OutputClause のインスタンスである出力の集まりが含まれる。

DecisionTable は、列挙された DecisionTableOrientation （Rule-as-Row、Rule-as-Column、 CrossTable）の内のひとつを保持する preferredOrientation を持つ。 DecisionTable のインスタンスは、8.2.2 テーブル型で定義されているように、preferredOrientation で指定されたとおりに表示されなければならない。

DecisionTable のインスタンスは、列挙されたヒット・ポリシー（UNIQUE、FIRST、PRIORITY、ANY、COLLECT、RULE ORDER、OUTPUT ORDER）の内のひとつを保持する関連する hitPolicy を持つ。hitPolicy 属性のデフォルト値は UNIQUE である。DecisionTable のインスタンスの図式表現では、hitPolicy は 8.2.11 ヒット・ポリシーで指定されているように表される。

DecisionTable のインスタンスに関連付けられているセマンティックスは、後述および 8.2.11 ヒット・ポリシーに記載されている様に、それに関連づけられたヒット・ポリシーに依存する。DecisionTable のインスタンスの hitPolicy 属性は、8.2.11 ヒット・ポリシーに指定されているとおりに表される。

DecisionTable のインスタンスに関連付けられた hitPolicy が FIRST または RULE ORDER なら、DecisionTable に関連付けられた rules は ordered でなければならない。DecisionTable の図式表現をつかって、ルールを明示的に番号付けし、順序付けを表示する。

DecisionTable のインスタンスに関連付けられた hitPolicy が PRIORITY または OUTPUT ORDER の場合、outputValues は 8.2.11 ヒット・ポリシーで明記された結果を判定する。

DecisionTable のインスタンスに関連付けられている hitPolicy が COLLECT である場合、DecisionTable は列挙された BuiltinAggregator（8.2.11 ヒット・ポリシーを参照）のひとつである関連する aggregation を持つことができる。

Expression の一種として、DecisionTable のインスタンスは値を持つ、この値は、関連付けられたルールの出力と、関連付けられた hitPolicy、および関連付けられた aggregration に依存する（存在する場合）。 DecisionTable のインスタンスの値は、次の仕様に従って決定される。

- 関連付けられた hitPolicy が UNIQUE の場合、DecisionTable のインスタンスの値は適用可能な唯一のルールの conclusion の値となる（ルール合致可能性の定義については、8.3.3 Decision Rule メタモデルを参照すること）。
- 関連付けられた hitPolicy が FIRST の場合、DecisionTable のインスタンスの値は、

ルールの順序に従って評価し最初に合致したルールの conclusion の値である。

- 関連付けられた hitPolicy が PRIORITY である場合、DecisionTable のインスタンスの値は、出力値のリストで示された outputEntry の順序に従って評価し最初に合致したルールの conclusion の値である。
- 関連付けられた hitPolicy が ANY の場合、DecisionTable のインスタンスの値は合致可能ないずれかのルールの値である。
- 関連付けられた hitPolicy が COLLECT で aggregation が指定されている場合、DecisionTable のインスタンスの値は DecisionTable の aggregation 属性によって設定された集計関数を、全ての合致した rules の conclusions の値に順序付けられていない値の集合に適用した結果である; aggregation 属性が設定されていない場合、デシジョンテーブルの値は順序つけられていない値の集合そのものとなる;
- 関連付けられた hitPolicy が RULE ORDER の場合、DecisionTable のインスタンスの値は、ルールの順序付けに従って順序付けされた、合致可能なすべての rules の conclusions の値のリストである。
- 関連付けられた hitPolicy が OUTPUT ORDER の場合、DecisionTable のインスタンスの値は、すべての合致可能な rules の conclusions の値のリストであり、conclusion の outputEntry の順序に従って並べられる。

DecisionTable は、すべての属性とモデルの関連付けを Expression から継承する。 表 28 に、DecisionTable 要素に追加される属性とモデル・アソシエーションを示す。

表 28 : DecisionTable 属性とモデル・アソシエーション

属性	説明
input: InputClause [*]	この属性は、この DecisionTable を構成する InputClause のインスタンスをリストする。
output: OutputClause [*]	この属性は、この DecisionTable を構成する OutputClause のインスタンスをリストする
rule: DecisionRule [*]	この属性は、この DecisionTable を構成する DecisionRule のインスタンスをリストする。
hitPolicy: HitPolicy	この DecisionTable のセマンティクスを決定するヒット・ポリシー。 デフォルトは UNIQUE である。
aggregation: BuiltinAggregator	存在する場合、DecisionTable の hitPolicy

	が COLLECT であれば、適用される集計関数をこの属性が設定する。この集計関数は、適応可能なルールによって順序付されていない値の集合に適用される。
preferredOrientation: DecisionTableOrientation [0..1]	この DecisionTable の図式表現について推奨されるテーブル型。この DecisionTable は、この属性で指定されたとおりに表現されるべきである。
outputLabel: string[0..1]	この属性は、デシジョンテーブル出力の説明を提供する。そしてしばしばデシジョンテーブルが値式となる InformationItem の名前と同じ名前となる

8.3.2 Decision Table Input and Outputメタモデル

DecisionTable では、input は inputExpression（件名）と inputEntries の数を指定する。output は、出力値の名前と出力値の定義の領域、outputEntries の数を指定する。

InputClause クラスはデシジョンテーブルの入力のモデル化に使用され、OutputClause クラスはデシジョンテーブルの出力のモデル化に使用される。

InputClause のインスタンスは、オプションの inputExpression と inputEntry の順序付けられたリストで構成され、これらは UnaryTests のインスタンスである。OutputClause のインスタンスはデータ型を設定する typeRef をオプションで参照し、LiteralExpression のインスタンスである outputEntry の順序付きリストと、LiteralExpression のインスタンスでもあるオプションの defaultOutputEntry から構成される。

DecisionTable に複数の OutputClause が含まれている場合、各 OutputClause は名前を持つ必要がある。

DecisionTable にひとつしか OutputClause がない場合は、OutputClause は名前を持ってはいけない。

表 29a と表 29b に、InputClause と OutputClause の属性とモデル・アソシエーションを示す

表 29a：InputClause 属性とモデル・アソシエーション

属性	説明
inputExpression: Expression [0.1]	この InputClause の件名
inputEntry: Expression [*]	この属性は、この InputClause を構成する Expression のインスタンスをリストする。

表 29b：OutputClause 属性とモデル・アソシエーション

属性	説明
typeRef: QName [0..1]	単一の出力しか持たないデシジョンテーブルの OutputClause は、typeRef を設定してはならない。 複数の出力を持つデシジョンテーブルの OutputClauses は、typeRef を設定することができる。 typeRef は、出力のデータ型の名前であり、設定された expressionLanguage の基本型である ItemDefinition、またはインポートされた型のいずれかである。
name: string [0.1]	単一の出力しか持たないデシジョンテーブルの OutputClause は名前を設定してはならない。複数の出力を持つデシジョンテーブルの OutputClauses は名前を設定しなければならない。
outputEntry: Expression [*]	この属性は、この OutputClause を構成する Expression のインスタンスをリストする。
defaultOutputEntry: Expression [0.1]	未完成な表において、この属性は Expression のインスタンスをリストする。この Expression はデシジョンテーブルに合致するルールがない場合に選択される。

8.3.3 Decision Ruleメタモデル

DecisionRule クラスは、デシジョンテーブルのルールをモデル化するために使用される（8.2 注記を参照）。

DecisionRule のインスタンスは、UnaryTests のインスタンスである inputEntry インスタンスの順序付きリストと、LiteralExpression のインスタンスである outputEntry インスタンスの順序付きリストを持っている。

DecisionTable のインスタンスを含む表形式では、DecisionRule のインスタンスの表現は、デシジョンテーブルの型に依存する。 例えば、デシジョンテーブルが水平型の場合（8.2.2 テーブル型参照)、DecisionRule のインスタンスは行として表され、すべての inputEntry がすべての outputEntrys の左側に表示される。

定義により、inputEntrys を持たない DecisionRule 要素は常に適用できる。それ以外の場合、ルールの inputEntrys の少なくともひとつが対応する inputExpression 値に適用できる場合にのみ、DecisionRule のインスタンスが合致可能と言われる。inputEntrys は任意の順序で合致する。

inputEntry 要素は、その要素を包含する DecisionTable の入力と同じ順序でなければならない。

i 番目の inputExpression は、8.1 節イントロダクションで定義されているように、すべての inputEntrys と DecisionRule が合致するためには、i 番目の inputEntry を満たさなければならない。

outputEntry 要素は、その要素を包含する DecisionTable の出力と同じ順序でなければならない。

i 番目の outputEntry は、i 番目の OutputClause の typeRef と整合していなければならない。

表 30 に、DecisionRule 要素の属性とモデル・アソシエーションを示す。

表 30：DecisionRule 属性とモデル・アソシエーション

属性	説明
inputEntry: UnaryTests[0..*]	この DecisionRule が（索引によって）対応する inputExpression と合致しなければならない入力条件を設定するための UnaryTests のインスタンス。
outputEntry: LiteralExpression [1..*]	この DecisionRule の出力コンポーネントを構成する LiteralExpression のインスタンスのリスト。

8.4　例

表31に、この章で説明したさまざまな型のデシジョンテーブルの例を示す。DMN意思決定モデルの完全な例とのからみで、11.3「意思決定要求レベル」に一層多くの例がある。

表 31：デシジョンテーブルの例

Single Hit
Unique

Applicant Risk Rating

U	Applicant Age	Medical History	Applicant Risk Rating
1	> 60	good	Medium
2		bad	High
3	[25..60]	-	Medium
4	< 25	good	Low
5		bad	Medium

Applicant Risk Rating

Applicant Age	< 25		[25..60]	> 60	
Medical History	good	bad	-	good	bad
Applicant Risk Rating	Low	Medium	Medium	Medium	High
U	1	2	3	4	5

Applicant Risk Rating

Applicant Age	< 25		[25..60]	> 60	
Medical History	good	bad	-	good	bad
Low	X	-	-	-	-
Medium		X	X	X	
High					X
U	1	2	3	4	5

Single Hit
Any

Person Loan Compliance

A	Persons Credit Rating from Bureau	Person Credit Card Balance	Person Education Loan Balance	Person Loan Compliance
1	A	< 10000	< 50000	Compliant
2	Not(A)	-	-	Not Compliant
3	-	>= 10000	-	Not Compliant
4	-	-	>= 50000	Not Compliant

Example case: not A, >= $10K, >= 50K -> Not Compliant (rules 2,3,4)

Single Hit
Priority

Applicant Risk Rating

P	Applicant Age	Medical History	Applicant Risk Rating
			High, Medium, Low
1	>= 25	*good*	*Medium*
2	> 60	*bad*	*High*
3	-	*bad*	*Medium*
4	< 25	*good*	*Low*

Single Hit
First

Special Discount

F	Type of Order	Customer Location	Type of Customer	Special Discount %
1	*Web*	*US*	*Wholesaler*	10
2	*Phone*	-	-	0
3	-	*Non-US*	-	0
4	-	-	*Retailer*	5

Special Discount

Type of Order	*Web*		-
Customer Location	*US*		-
Type of Customer	*Wholesaler*	*Retailer*	-
Special Discount %	10	5	0
F	1	2	3

Example case: **Web, non-US, Retailer -> 0 (rule 3)**

Multiple Hit
No order

Holidays

Age	-	<18	>=60	-	(18..60)	>=60	-
Years of Service	-	-	-	>=30	(15..30)	-	>=30
Holidays	22	5	5	5	2	3	3
C+	1	2	3	4	5	6	7

Example case: **Age=58, Service=31 -> Result=sum(22, 5, 3)=30**

Multiple Hit

Output order

Holidays

O	Age	Years of Service	Holidays
			22, 5, 3, 2
1	-	-	22
2	>= 60	-	3
3	-	>= 30	3
4	< 18	-	5
5	>= 60	-	5
6	-	>= 30	5
7	[18..60)	[15..30)	2
8	[45..60)	< 30	2

Example case: Age=58, Service=31 -> Result=(22, 5, 3)

Multiple Hit

Rule order

Student Financial Package Eligibility

R	Student GPA	Student Extra-Curricular Activities Count	Student National Honor Society Membership	Student Financial Package Eligibility List
1	> 3.5	>= 4	*Yes*	20% Scholarship
2	> 3.0	-	*Yes*	30% Loan
3	> 3.0	>= 2	*No*	20% Work-On-Campus
4	<= 3.0	-	-	5% Work-On-Campus

Example case: For GPA=3.6, EC Activities=4, NHS Membership -> result = (20% scholarship, 30% loan)

9 簡易式言語（S-FEEL）

9.1　イントロダクション

DMN 1.1 は、意思決定モデル（第 10 章参照）において多くの種類の式に標準実行可能セマンティックを提供する目的で、フレンドリーな式言語（Friendly enough expression language：FEEL）を定義している。

この章では、シンプルな式のみを使用する意思決定モデルに標準の実行可能なセマンティクスを与える目的で FEEL の簡易なサブセット S-FEEL（Simple subset of FEEL）を定義する。特に、意思決定ロジックがデシジョンテーブルでモデル化されている場合には有効である。

9.2　S-FEEL シンタクス

このセクションで使用されている S-FEEL 式の構文は、以下の EBNF で指定される。FEEL 構文のサブセットであり、製造番号 FEEL EBNF（拡張バッカスナウア記法）の 10.3.1.2 文法規則に従う。

文法規則：

```
1.    expression = simple expression ;
4     arithmetic expression =
4.a   addition | subtraction |
4.b    multiplication | division |
4.c    exponentiation |
4.d    arithmetic negation ;
5      simple expression = arithmetic expression | simple value | comparison ;
6      simple expressions = simple expression , { ",", simple expression } ;
7      simple positive unary test =
7.a    [ "<" | "<=" | ">" | ">=" ] , endpoint |
7.b    interval ;
8      interval = ( open interval start | closed interval start ) , endpoint , "..",
       endpoint , ( open interval end | closedinterval end ) ;
9      open interval start = "(" | "]" ;
10     closed interval start = "[" ;
```

```
11    open interval end = ")" | "[" ;
12    closed interval end = "]" ;
13    simple positive unary tests = simple positive unary test , { "," , simple
      positive unary test } ;
14    simple unary tests =
14.a  simple positive unary tests |
14.b  "not", "(", simple positive unary tests, ")" |
14.c  "-";
18    endpoint = simple value ;
19    simple value = qualified name | simple literal ;
20    qualified name = name , { "." , name } ;
21    addition = expression , "+" , expression ;
22    subtraction = expression , "-" , expression ;
23    multiplication = expression , "*" , expression ;
24    division = expression , "/" , expression ;
25    exponentiation = expression, "**", expression ;
26    arithmetic negation = "-" , expression ;
27    name = name start , { name part | additional name symbols } ;
28    name start = name start char, { name part char } ;
29    name part = name part char , { name part char } ;
30    name start char = "?" | [A-Z] | "_" | [a-z] | [¥uC0-¥uD6] | [¥uD8-¥uF6] |
      [¥uF8-¥u2FF]  |  [¥u370-¥u37D]  |[¥u37F-¥u1FFF]  |  [¥u200C-¥u200D]  |
      [¥u2070-¥u218F] | [¥u2C00-¥u2FEF] | [¥u3001-¥uD7FF] | [¥uF900-¥uFDCF] |
      [¥uFDF0-¥uFFFD] | [¥u10000-¥uEFFFF] ;
31    name part char = name start char | digit | ¥uB7 | [¥u0300-¥u036F] |
      [¥u203F-¥u2040] ;
32    additional name symbols = "." | "/" | "-" | "' " | "+" | "*" ;
33    simple literal = numeric literal | string literal | Boolean literal | date
      time literal ;
34    string literal = '"' , { character - ('"' | vertical space) }, '"' ;
35    Boolean literal = "true" | "false" ;
36    numeric literal = [ "-" ] , ( digits , [ ".", digits ] | "." , digits ) ;
37    digit = [0-9] ;
38    digits = digit , {digit} ;
39    date time literal = ("date" | "time" | "duration") , "(" , string literal ,
       ")" ;
```

51　comparison =
51.a　expression , ("=" | "!=" | "<" | "<=" | ">" | ">=") , expression ;

9.3　S-FEELデータタイプ

S-FEEL は number, string, boolean, days and time duration, years and months duration, time などの FEEL データ型をすべてサポートするが、その中の一部の定義は簡略化されている。

S-FEEL number 型は、XML スキーマ 10 進データ型と同じ文字および数値の空間を持つ。実装では、精度を 10 進数の 34 桁に制限し、偶数の隣接値を優先する隣接値に丸めることができる。 「精度はこの値空間に反映されない。つまり、数値 2.0 は数値「2.00」とは区別されない[XML Schema」 ことに注意。この値空間は完全に順序付けられていることにも注意すること。 S-FEEL number 型の定義は、FEEL number 型の定義より簡略化されている。

S-FEEL は FEEL string 型と FEEL Boolean 型をサポートしている。FEEL string 型は、Java 文字列データ型と XML スキーマ文字列データ型と同じ文字と数値の空間を持つ。FEEL Boolean 型 は、Java ブーリアンデータ型および XML スキーマブーリアンデータ型と同じ文字と数値の空間を持つ。

S-FEEL は FEEL time データ型をサポートしている。 FEEL time の単語と数値の空間は、XML スキーマの time データ型の文字と数値の空間である。 要注意： 「単語表現ではオプションのタイムゾーン・インジケータが使用可能なため、2 つの値のうち 1 つがタイムゾーンを持ち、もう一方がタイムゾーンを持たないという 2 つの値については順序が設定することができないことがあるので、この場合は時刻の順序が部分的にしか判断できない。 尚、時間値のペアが両方ともタイムゾーン・インジケータを持つ場合と持たない場合には、完全に順序付けされる。」[XSD]。

S-FEEL は FEEL の日付と時刻をサポートしていない。 ただし、FEEL の date and time から時、分、秒を抜かした date and time をサポートしている。FEEL date 型、単語と数値の空間は、XML スキーマの日付データ型の文字と数値の空間である。

S-FEEL は、FEEL の days and time duration、年月のデータ型をサポートしている。FEEL days and timeduration、および years and month duration は、それぞれ Xpath データ・モデルの dayTimeDuration および yearMonthDuration データ型と同じ文字と数値の空間を持つ。 つまり、FEEL の dayTimeDuration は、XML スキーマの持続時間データ型から、その単語表現に日、時間、分、秒のコンポーネントのみを含むように制限することによって導出され、FEEL の年月の期間は、XML スキーマの期間データ型 年と月のコンポーネントのみを含む単語表現である。

FEEL のデータ型は、項目 10.3.2.2「等価性、同一性、等価性」で詳細に規定されている。

9.4 S-FEEL セマンティクス

S-FEEL には、ほとんどの式およびプログラミング言語に共通する限られた基本機能と、ほとんどの式およびプログラミング言語に一致するセマンティクスが含まれている。

S-FEEL 式のセマンティクスは、XML スキーマデータ型と XQuery 1.0 と XPath 2.0 データ・モデルのデータ型のセマンティクスの観点、および XQuery 1.0 と XPath 2.0 で定義されている対応する関数と演算子（前に ″op：″が付く）の観点から、本節で定義される。セマンティクスの完全なスタンドアロン仕様は、FEEL の定義の一部として 10.3.2 セマンティクスにおいて定義される。 S-FEEL のスコープ内で、2 つの定義は同等であり、同じ規範である。

算術加算と減算（文法規則 4a）は、以下の同じセマンティクスを持つ。

・op：numeric-add と op：numeric-subtract（2 つのオペランドが数値の場合）。
・op：add-yearMonthDurations と op：subtract-yearMonthDurations（2 つのオペランドが年月期間の場合）。
・op：add-dayTimeDuration と op:subtract-dayTimeDurations（2 つのオペランドが日時期間の場合）。
・op:add-yearMonthDuration-to-date と op:subtract-yearMonthDuration-from-date、第 1 オペランドが年月期間（終了日）、第 2 オペランドが年月期間（開始日）の場合。
・op：add-dayTimeDuration-to-date と op：subtract-dayTimeDuration-from-date、第 1 オペランドが日時期間（終了日）、第 2 オペランドは日時期間（開始日）の場合。
・op：add-dayTimeDuration-to-time と op：subtract-dayTimeDuration-from-time、第 1 オペランドが日時期間（終了時刻）、第 2 オペランドは日時期間（開始時刻）の場合。

さらに、算術減算は、2 つのオペランドがそれぞれ日付または時刻であるときに、op：subtract-dates または op：subtract-times のセマンティクスを持つ。

算術加減算は他の場合には定義されない。

算術乗算と除算（文法規則 4b）は、2 つのオペランドが数字であるとき、それぞれ op：

numeric-multiply と op：numeric-divide で定義されたセマンティクスと同じ意味を持つ。 それらは別途定義されない。 算術べき乗（文法規則 4c）は、2 つのオペランドが数値である場合に、第 1 オペランドを第 2 オペランドのべき乗の結果として定義される。他の場合には定義されない。
算術否定（文法規則 4d）は、そのオペランドが数字の場合にのみ定義される。その場合、意味は op：numeric-unary-minus の仕様に従う。

数値間の比較演算子（文法規則 7.a）は、op：numericequal、op-numeric-less-than、op:numeric-greater-than の指定に従って定義され、日付間の比較は op: date-equal、op：date-less-than、op：date-greater-than; 時間の比較は、op：time-equal、op：time-less-than、op：time-greater-than の指定に従って定義される。 年月期間の比較は、op：duration-equal、op：yearMonthDuration-less-than、op：year-MonthDuration-greater-than の指定に従って定義される。 日時期間の比較は、op：duration-equal、op：dayTimeDuration-less-than、op：dayTimeDuration-greater-than の指定に従って定義される。

文字列と論理型は等式のみで比較される。文字列と論理型の等式のセマンティクスは、各々、fn：codepoint-equal と op：Boolean-equal の等式の仕様で定義される。

比較演算子は、2 つのオペランドが同じ型を持つ場合にのみ定義される。ただし、年月期間と日時期間の長さは等式で比較できる。 しかし、ゼロ長期間を除いて「xs：dayTimeDuration のインスタンスは xs：yearMonthDuration のインスタンスに等しいことはない」″[XFO] ことに注意。

テストされる式 o が与えられると、2 つのエンドポイント e1 と e2 は下記の式で定義される。

- is in the interval (e1..e2), also notated]e1..e2[, if and only if o > e1 and o < e1
- is in the interval (e1..e2], also notated]e1..e2], if and only if o > e1 and o ≤ e2
- is in the interval [e1..e2] if and only if o ≥ e1 and o ≤ e2
- is in the interval [e1..e2), also notated [e1..e2[, if and only if o ≥ e1 and o < e2

テストされる式は、式がリストで少なくともひとつの 1 行単体テストを満たすか、または 1 行単体テストが「 - 」(ハイフォン) である場合にのみ 1 行単項式テストのインスタンスを満たす (文法規則 14)。

9.5 S-FEEL 式の用途

本節では、式言語が S-FEEL の場合、どの役割でどの種類の S-FEEL 式が許可されるかを要約する。

9.5.1 項目定義

許容値を定義する式は、simple unary tests (文法規則 14) のインスタンスでなければならない。ここでは、テストを満たす定義型または参照型の値のみが許容値である。

9.5.2 起動

起動のバインディングでは、バインディング式は simple expression (文法ルール 5) でなければならない。

9.5.3 デシジョンテーブル

各入力式は simple expression (文法規則 5) でなければならない。
入力値の各リストは、simple unary tests (文法規則 14) のインスタンスでなければならない。
出力値の各リストは、simple unary tests (文法規則 14) のインスタンスでなければならない。
各入力項目は、simple unary tests (文法規則 14) のインスタンスでなければならない。
各出力項目は、simple expression (文法規則 5) でなければならない。

10 式言語(FEEL)

10.1 イントロダクション

DMN では、すべての意思決定ロジックは囲み式として表される。7.2 章において、囲み式のコンセプトを紹介し、囲みリテラル式と囲み起動という、2 つの基本種別を定義した。第 8 章では、デシジョンテーブルを定義した。デシジョンテーブルは囲み式において重要な特性である。
本章では、それ以外の囲み式を記載する特性を定義し、意思決定ロジックのグラフィカルな表記方法の説明を完結させる。なお「囲み式」という表現は FEEL の式のことである。FEEL は Friendly Enough Expression Language の略で、次の特性がある：
・副作用なし
・番号づけられた簡潔なデータ・モデル, 日付,文字列,リスト, コンテキスト
・幅広い利用者のためのシンプルな構文
・SQL と PMML に基づく true、false、null の３値ロジック
本章では、FEEL の構文とセマンティクスについても完全に明確にする。 構文は文法（10.3.1 構文）として明確にする。 囲み式としてグラフィカル表示を意図した構文のサブセットも、メタモデルとして明確にする（10.5 メタモデル)。
FEEL は次の２つの役割を持つ：

1. デシジョンテーブルなどの囲み式のボックス内に記載するテキストの表記法。
2. 式の論理を表現するための言語、および DRG を使用してシンプルで統一的な方法で構成することを主目的としたセマンティクス。

10.2 表記法

10.2.1 囲み式

本節は、7.2 章表記法で定義された意思決定ロジックと囲み式の一般的な表記法を基に記述する。
囲み式と呼ばれる意思決定ロジックのためのグラフィカルな表記法を定義する。この表記法は、意思決定ロジック・モデルを小さな断片に分解するのに使用し、DRG の成果物に関連付けることができる。DRG と囲み式は、意思決定モデルを完全に仕様化できるグラフィカル言語である。
以下に囲み式の種別を示す：
・デシジョンテーブル

・囲み FEEL 式
・囲み起動
・囲みコンテキスト
・囲みリスト
・関連
・囲み関数

囲み式は再帰的に定義される、つまり、囲み式は他の囲み式を含むことができる。最上位の囲み式は、ひとつの DRG における成果物の意思決定ロジックとなる。この囲み式には、DRG 成果物の名前が含まれた名前ボックスを付けられる。図 52 に示すように、名前ボックスは、単一ボックスの一番上に付けることができる。

図 52:囲み式

その代わりに、図 53 に示す様に、名前ボックスと式ボックスを白い空間と左側だけ接続された線で、分けても良い。

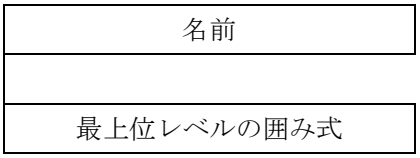


図 53:名前と式ボックスが離された囲み式

グラフィカルツールは、例えば、意思決定ダイアグラムをクリックすると、デシジョンテーブルが開くといった、適切なグラフィカル・リンクをサポートすることが期待される。

10.2.1.1 デシジョンテーブル

ここで定義する実行可能なデシジョンテーブルは、第 8 章で定義されたデシジョンテーブルと同じ表記法を用いる。また実行セマンティクスは、10.3.2.8 デシジョンテーブルで定義される。

10.2.1.2 囲み FEEL 式

囲み FEEL 式は、図 54 における任意の FEEL 式 *e* である。FEEL 式 *e* は、FEEL 文法（10.3.1

構文）で定義される。

図 54:囲み FEEL 式

e を含む囲み式は、FEEL（*e*、**s**）である。ここで **s** はスコープである。スコープには、デシジョン・サービスの 10.4 実行セマンティクスで説明されるように、含まれている DRD から派生したコンテキストと、*e* を含む囲みコンテキストが含まれる。

通常は、*e* を比較的単純にして、囲み式の中に小さな囲み式を描く方法が良い方法となる。

10.2.1.3 囲み起動

囲み起動の構文は、7.2.3 囲み起動で説明している。 この構文は、任意の関数を起動するために使用する（例えば、ビジネス知識モデル、FEEL 組込み関数、囲み関数定義）。「起動されたビジネス知識モデル」とラベル付けされたボックスは、図 55 に示すように、その値が関数である任意の囲み式となる。

名前	
値を返す関数式	
パラメータ 1	バインディング式 1
パラメータ 2	バインディング式 2
パラメータ n	バインディング式 n

図 55:囲み起動

囲み式の構文は、文法規則 40, 41, 42, 43 によって定義されたテキスト構文に位置づけられる。囲み起動は名前付きパラメータを使用する。位置に基づく起動は、パラメータの順番に基づく起動を含む囲み式を使用できる。

囲み式の構文にはパラメータが必要である。パラメータの無い関数は、図 56 に示すような、テキスト構文を使用して起動しなければならない。

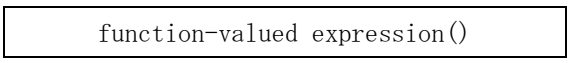


図 56:パラメータの無い関数

正式には、囲み起動は、等価なテキスト呼び出しのセマンティクスによって記述される。

例： **function-valued expression**(parameter $_1$: **binding expression** $_1$, parameter $_2$: **binding expression** $_2$, …)

10.2.1.4　　囲みコンテキスト

囲みコンテキストは、任意の名前と戻り値を持つペア n（名前、値）の集合である。各ペアはコンテキスト・エントリーと呼ばれる。コンテキスト・エントリーは、空白で区切り、左（または上）の行に接続することができる。つまり、 コンテキストのすべてのエントリーは、垂直コンテキストの先頭列または水平コンテキストの先頭行を視ることによって容易に識別できる。セルは、次のいずれかの方法で配置される（図 57、図 58 参照)。

名前 1	値 1
名前 2	値 2
名前 n	値 n
結果	

図 57:垂直型

名前 1	名前 2	名前 n		結果
値 1	値 2	値 n		

図 58:水平型

コンテキスト内のコンテキスト・エントリーは、それぞれが名前を持つ単純な式に複雑な式を分解するために使用される。これらのコンテキスト・エントリーは、中間結果と考えることができる。例えば、最終結果ボックスのないコンテキストは、ケースデータ（10.6 の例のケース）を表す(図 59 を参照)。

申込みデータ		
年令	51	
婚姻状況	"M"	
雇用状況	"EMPLOYED"	
既顧客	false	
月次	収入	10000.00
	返済	2500.00
	支出	3000.00

図 59:コンテキスト・エントリーの利用

最終結果ボックスのコンテキストは、計算を表す（図 60 を参照)。

適格性	
年令	Applicant. Age
月収	Applicant. Monthly. Income
信用調査前のリスクカテゴリー	Affordability. Pre-Bureau Risk Category
Installment Affordable 支払い額の容易性	Affordability. Installment Affordable
if Pre-Bureau Risk Category = "DECLINE" or Installment Affordable = false or Age < 18 or Monthly Income < 100 then "INELIGIBLE" else "ELIGIBLE"	

図 60：最終結果ボックスの利用

デシジョンテーブルが（結果ではない）コンテキスト・エントリーである場合は、出力セルを使用してエントリーに名前を付けることができ、スペースを節約できる。任意のフォーマットのデシジョンテーブルを、垂直コンテキストで使用することができる。ギザギザの右端が許されている。コンテキスト・エントリー間の空白が助けになる。 図 61 を参照。

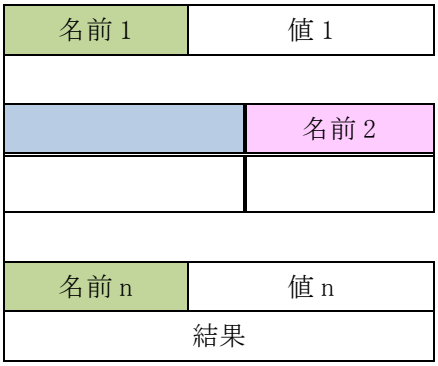


図 61:デシジョンテーブルエントリーのある垂直コンテキスト

色は推奨される。

名前は正当な FEEL の名前でなければならない。 値とオプションの結果は囲み式である。

囲みコンテキストは結果としてデシジョンテーブルを持つことができ、名前付きコンテキスト・エントリーを計算し、名前を付ける。たとえば、次のようになる（図 62 を参

照)。

信用調査後の リスクカテゴリー	
既顧客	Applicant. ExistingCustomer
与信スコア	Report. CreditScore
申込み者のリスク・スコア	Affordability Model(Applicant, Product). Application Risk Score

U	既顧客	申込み者のリスクスコア	与信スコア	信用調査後リスクカテゴリー
1	true	<=120	<590	“HIGH”
2			[590..610]	“MEDIUM”
3			>610	“LOW”
4		>120	<600	“HIGH”
5			[600..625]	“MEDIUM”
6			>625	“LOW”
7	false	<=100	<580	“HIGH”
8			[580..600]	“MEDIUM”
9			>600	“LOW”
10		>100	<590	“HIGH”
11			[590..615]	“MEDIUM”
12			>615	“LOW”

図 62:デシジョンテーブルのある囲み式

正式には，戻り値なしの場合の囲みコンテキストは｛“Name 1”：Value 1，“Name 2”：Value 2，…，“Name n”：Value n｝。戻り値ありの場合は｛“Name 1”：Value 1，“Name 2”：Value 2，…，“Name n”：Value n，“result”：Result｝.result。太字は FEEL セマンティクス領域の要素を示している。 スコープには、DRG を含むコンテキストから派生したコンテキストが含まれる。なお DRG を含むコンテキストは、デシジョン・サービスの 10.4 実行セマンティクスで説明される。

10.2.1.5 囲みリスト

囲みリストは n 項目のリストである。以下の方法（図 63、図 64）で表示できる。

項目 1
項目 2
項目 n

図 63:縦型

項目 1, 項目 2, 項目 n

図 64:水平型

線は標準スタイルである。 項目は囲み式である。 正式には、囲みリストはリストの手段に過ぎない、例えば、［ **Item 1**, **Item 2**, …, **Item n** ］である。 スコープには、デシジョン・サービスの 10.4 実行セマンティクスで説明される DRG を含むコンテキストが含まれる。

10.2.1.6 関係

同じ種類の項目からなる水平コンテキスト（結果セルなし）の垂直リストは、リストの一行目に一度だけ名前を表示することができ、 図 65 に示すような関係表となる。

名前 1	名前 2	名前 3
値 1a	値 2a	値 na
値 1b	値 2b	値 nb
値 1m	値 2m	値 nm

図 65 関係

10.2.1.7 囲み関数

囲み関数の定義はパラメータ付き囲み式の書式である ．

ビジネス知識モデルと関係のある囲み式は、囲み関数かデシジョンテーブルでなければならない。このとき、デシジョンテーブルの入力式はパラメータ名と見なされる。
囲み関数は３種類のセルを持つ：

1. Kind ボックスには次のいずれかの頭文字が入る：
    - FEEL

- PMML
- Java

デシジョンテーブルを含むFeel関数の場合は、Kindボックスを省略することができる。

2. Parameters: カンマ区切りの括弧で囲まれた名前
3. Body:囲み式

３個のセルは図66のように表示しなければならない

K	(パラメータ1, パラメータ2, …)
Body	

図66:囲み関数の定義

FEEL関数の場合は、 Kind FEELと表示するかまたはKindは省略できる。Bodyは、パラメータを参照するFEEL式でなければならない。Kind Javaと表示される外部定義関数場合、Bodyは″10.3.2.11.2外部定義された関数″で説明されるコンテキストでなければならず、またマッピング情報のフォームもjavaフォームでなければならない。Kind PMMLと表示される外部定義関数の場合、Bodyは10.3.2.11.2で説明されるコンテキストでなければならず、またマッピング情報のフォームも*pmml*フォームでなければならない。

正式には、囲み関数は単に関数である、すなわち、kindがFEELなら*FEEL(function(Parameter1、Parameter2、...)Body)*となり、それ以外は *FEEL(function(Parameter1、Parameter2、...) external Body)* となる。スコープには、デシジョン・サービスの10.4実行セマンティクスで説明されるDRGを含むコンテキストからのコンテキストが含まれる。

10.2.2 FEEL

次の章で定義されるFEELのサブセットは、囲み式の″ボックス内の表記″としてサービスする。FEEL オブジェクトは、数値、文字列、日付、時間、期間、関数、コンテキスト、またはFEELオブジェクトのリスト（入れ子リストを含む）である。
注：JSONオブジェクトは、数値、文字列、コンテキスト（JSONではマップと呼ぶ）またはJSONオブジェクトのリストである。そのため、この点ではFEELはJSONの拡張版である。さらに、FEEL はリテラル値のために分かり易い構文を提供し、コンテキストのKeyを引用する必要はない。
ここでは、言語を ″感じ″てもらうため、簡単な例から始める。

10.2.2.1　　範囲の比較

範囲と範囲リストは、デシジョンテーブルの入力項目、入力値、および出力値セルに表示される。 表 32 の例では、該当部分に下線が引かれている。 文字列、日付、時間、および期間には 7.2.2.1 で定義された引用符で囲まれた文字列を用いる。

表 32:FEEL 範囲比較

FEEL 式	値
5 in (<=5)	true
5 in ((5..10])	false
5 in ([5..10])	true
5 in (4, 5, 6)	true
5 in (<5, >5)	false
2012-12-31 in ((2012-12-25..2013-02-14))	true

10.2.2.2 数

FEEL の数と計算結果値は表 33 で例示する。

表 33:FEEL の数と計算値

FEEL 式	値
decimal(1, 2)	1.00
.25 + .2	0.45
.10 * 30.00	3.0000
1 + 3/2*2 - 2**3	-4.0
1/3	0.3333333333333333333333333333333333
decimal(1/3, 2)	0.33
1 = 1.000	true
1.01/2	0.505
decimal(0.505, 2)	0.50
decimal(0.515, 2)	0.52
1.0*10**3	1000.0

10.3 全 FEEL 構文とセマンティクス

第 9 章では、適合レベル 2（第 2 章参照）のデシジョンテーブルをサポートするのに十分な FEEL のサブセットを紹介した。適合レベル 3 に必要な完全な DMN friendly-enough expression language (FEEL) は本章で仕様化される。FEEL は、Java、JavaScript、XPath、SQL、PMML、Lisp や他の多くのものから着想を得たシンプルな言語である。
構文は文法規則を使用して定義され、複雑な式が単純な式から構成されることが分かる。同様に、シマンテッィス規則は、複雑な表現の意味が単純な表現から構成されていることが分かる。
DMN は、外部定義関数を起動しない場合における完全な FEEL 式を定義する。なお、実装定義のセマンティクスは無い。FEEL 式（外部定義関数を起動しない）は副作用がなく、すべての適合した実装で同じ解釈をする。また、外部定義関数は、処理が確定的で副作用のないものであるべきである。

10.3.1 構文

FEEL 構文は本章で文法として定義され、メタモデル(10.5 メタモデル)において、UML クラス図としても定義される。

10.3.1.1 文法表記

文法規則は、ISO EBNF 表記を使用する。 各ルールは終端を示さないシンボルとして S を定義し、S には S_1、S_2、...などのシンボルも含む。なお、シンボルにはスペースを含めることができる。 次の表は、EBNF 表記法をまとめたものである。

表 34:EBNF 表記法

例	意味
$S = S_1$;	シンボル S はシンボル S_1 の語彙をつかって定義された
S_1 / S_2	S_1 か S_2 のどちらか
S_1, S_2	S_1 に続いて S_2
[S_1]	S1 が 0 回か 1 回発生する
{S_1}	S1 が 0 回以上繰り返す
$k * S_1$	S_1 が k 回繰り返す
"and"	終端文字シンボル

簡潔にするため、以下のように ISO 表記の文字範囲を拡張する。

文字範囲は以下の EBNF 構文とする：

```
character range = "[", low character, "-", high character, "]" ;
low character = unicode character ;
high character = unicode character ;
unicode character = simple character | code point ;
code point = "¥u", hexadecimal digit, 4 * [hexadecimal digit] ;
hexadecimal digit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|
                    "a"|"A"|"b"|"B"|"c"|"C"|"d"|"D"|"e"|"E"|"f"|"F";
```

文字は、例えば、 a、1、$などの様な、Unicode 文字である。文字を指定する別の方法として、文字を ¥ u という接頭辞が付いた 16 進コードで指定することもできる。
すべての Unicode 文字には文字コード番号（code point）がある。範囲内の下位側の文字（low character）は、上位側の文字（high character）の数値よりも小さい数値でなければならない。
たとえば、16 進数字は、次のように文字範囲を使用して簡潔に記述できる。

```
hexadecimal digit = [0-9] | [a-f] | [A-F] ;
```

注記：すべての Unicode 文字を含む文字範囲は[¥u0- ¥u10FFF]である。

10.3.1.2 文法ルール

完全な FEEL 文法を以下に示す。文法規則には番号が付けられている。また、後で参照するために代替文字が追記されているものがある。囲み式の構文（ルール 55）は、ボックス化された式に実行セマンティクスを記述するために使用する。

```
1. expression =
1.a textual expression |
1.b boxed expression ;
2. textual expression =
2.a function definition | for expression | if expression | quantified expression
|
2.b disjunction |
2.c conjunction |
2.d comparison |
2.e arithmetic expression |
2.f instance of |
2.g path expression |
```

2.h filter expression | function invocation |
2.i literal | simple positive unary test | name | "(" , textual expression , ")" ;
3. textual expressions = textual expression , { "," , textual expression } ;
4. arithmetic expression =
4.a addition | subtraction |
4.b multiplication | division |
4.c exponentiation |
4.d arithmetic negation ;
5. simple expression = arithmetic expression | simple value ;
6. simple expressions = simple expression , { "," , simple expression } ;
7. simple positive unary test =
7.a ["<" | "<=" | ">" | ">="] , endpoint |
7.b interval ;
8. interval = (open interval start | closed interval start) , endpoint , ".." , endpoint , (open interval end | closed
interval end) ;
9. open interval start = "(" | "]" ;
10. closed interval start = "[" ;
11. open interval end = ")" | "[" ;
12. closed interval end = "]" ;
13. simple positive unary tests = simple positive unary test , { "," , simple positive unary test } ;
14. simple unary tests =
14.a simple positive unary tests |
14.b "not", "(", simple positive unary tests, ")" |
14.c "-";
15. positive unary test = simple positive unary test | "null" ;
16. positive unary tests = positive unary test , { "," , positive unary test } ;
17. unary tests =
17.a positive unary tests |
17.b "not", " (", positive unary tests, ")" |
17.c "-"
18. endpoint = simple value ;
19. simple value = qualified name | simple literal ;
20. qualified name = name , { "." , name } ;
21. addition = expression , "+" , expression ;

22. subtraction = expression , "-" , expression ;
23. multiplication = expression , "*" , expression ;
24. division = expression , "/" , expression ;
25. exponentiation = expression, "**", expression ;
26. arithmetic negation = "-" , expression ;
27. name = name start , { name part | additional name symbols } ;
28. name start = name start char, { name part char } ;
29. name part = name part char , { name part char } ;
30. name start char = "?" | [A-Z] | "_" | [a-z] | [¥uC0-¥uD6] | [¥uD8-¥uF6] | [¥uF8-¥u2FF] | [¥u370-¥u37D] | [¥u37F-¥u1FFF] | [¥u200C-¥u200D] | [¥u2070-¥u218F] | [¥u2C00-¥u2FEF] | [¥u3001-¥uD7FF] | [¥uF900-¥uFDCF] | [¥uFDF0-¥uFFFD] | [¥u10000-¥uEFFFF] ;
31. name part char = name start char | digit | ¥uB7 | [¥u0300-¥u036F] | [¥u203F-¥u2040] ;
32. additional name symbols = "." | "/" | "-" | "' " | "+" | "*" ;
33. literal = simple literal | "null" ;
34. simple literal = numeric literal | string literal | Boolean literal | date time literal ;
35. string literal = '"' , { character - ('"' | vertical space) }, '"' ;
36. Boolean literal = "true" | "false" ;
37. numeric literal = ["-"] , (digits , [".", digits] | "." , digits) ;
38. digit = [0-9] ;
39. digits = digit , {digit} ;
40. function invocation = expression , parameters ;
41. parameters = "(" , (named parameters | positional parameters) , ")" ;
42. named parameters = parameter name , ":" , expression , { "," , parameter name , ":" , expression } ;
43. parameter name = name ;
44. positional parameters = [expression , { "," , expression }] ;
45. path expression = expression , "." , name ;
46. for expression = "for" , name , "in" , expression { "," , name , "in" , expression } , "return" , expression ;
47. if expression = "if" , expression , "then" , expression , "else" expression ;
48. quantified expression = ("some" | "every") , name , "in" , expression , { name , "in" , expression } , "satisfies" ,

expression ;
49. disjunction = expression , "or" , expression ;
50. conjunction = expression , "and" , expression ;
51. comparison =
51.a expression , ("=" | "!=" | "<" | "<=" | ">" | ">=") , expression |
51.b expression , "between" , expression , "and" , expression |
51.c expression , "in" , positive unary test ;
51.d expression , "in" , " (", positive unary tests, ")" ;
52. filter expression = expression , "[" , expression , "]" ;
53. instance of = expression , "instance" , "of" , type ;
54. type = qualified name ;
55. boxed expression = list | function definition | context ;
56. list = "[" [expression , { "," , expression }] , "]" ;
57. function definition = "function" , "(" , [formal parameter { "," , formal parameter }] , ")" ,
["external"] , expression ;
58. formal parameter = parameter name ;
59. context = "{" , [context entry , { "," , context entry }] , "}" ;
60. context entry = key , ":" , expression ;
61. key = name | string literal ;
62. date time literal = ("date" | "time" | "date and time" | "duration") , "(" , string literal , ")" ;
追加の構文ルール：

- 演算子の優先順位は、文法規則 1,2、および 4 の選択肢を、低いものから高いものへと順に並べることで決定する。例えば、(ボックス化された) 起動は乗算よりも優先順位が高く、乗算は加算よりも優先順位が高く、加算は比較よりも優先順位が高い。加算と減算の優先順位は同じだが、すべての FEEL の中間バイナリ演算子と同様に、結合律に従う。
- name には空白を含めることができる、しかし連続する 2 つ以上の空白は許容されない。
- name start (文法規則 28) は言語のキーワードではない。 (言語のキーワードは、 "and"、 "or"、 "true"、 "false"などの文法規則において二重引用符で囲まれている)。
- name part (文法規則 29) は言語のキーワードである。
- Java スタイルのコメントが使用できる、例えば、 '//' から行の終端までと、 /* … */。

10.3.1.3　　リテラル、データ型、組込み関数

FEEL は、数値、文字列、ブール値、および null のリテラル構文をサポートしている。（10.3.1.2 章文法規則を参照)。 リテラルは、FEEL セマンティクス領域（10.3.2.1 セマンティクス領域）に直接マップされる。
FEEL は次のデータ型をサポートしている：

- 数値型(number)
- 文字列型(string)
- 論理型(boolean)
- 期間型(days and time duration)
- 年月期間型(years and months duration)
- 時刻型(time)
- 日時型(date and time)

時間と日時のデータ型にはリテラル構文はない。組み込み関数（10.3.4.1 変換関数）を使用して文字列表現から変換する必要がある。

10.3.1.4　　コンテキスト、リスト、修飾名、コンテキストリスト

コンテキストとは、コンテキスト・エントリーと呼ばれる key と値が一組となったマップで、コンテキストを区切るための中括弧、エントリーを区切るためのカンマ、key と値を区切るためのコロン（文法ルール 59）を使用して記述される。key には文字列または名前を使用できる。 また、値は式である。
リストは、リストを区切るための角括弧と、リスト項目を区切るためのコンマ（文法規則 56）を使用して記述される。 1 行のリストは、単一の項目に等しい、すなわち、FEEL 式 e に対して[e] = e となる。
コンテキストおよびリストは、他のコンテキストおよびリストを参照することができ、有向非循環グラフを作る。パスを使って名前を書く。コンテキスト・エントリーの修飾名（QN）
は $N_1.N_2$ … N_n という形式で、N_1 はスコープ内コンテキストの名前である。
N1.N2 ... Nn の解釈で入れ子されたリストは保存される。たとえば、

[{a: [{b: 1}, {b: [2.1, 2.2]}]}, {a: [{b: 3}, {b: 4}, {b: 5}]}].a.b =
[[{b: 1}, {b: [2.1, 2.2]}], [{b: 3}, {b: 4}, {b: 5}]].b =
[[1, [2.1, 2.1]],[3, 4, 5]]

入れ子リストは、flatten () 組み込み関数（10.3.4 組み込み関数）を使用してフラット化できる。

10.3.1.5 曖昧性

コンテキスト・エントリーと関数パラメータの名前には、スペースやアポストロフィなどの一般的な文字が使用できる（ただし、コロンやカンマ（ ':'または ','）は使用できない)。この命名の自由さはFEELの構文を曖昧にする。曖昧さはスコープを使用して解決する。名前は、スコープ内の名前と、左から右に比較し、もっとも長く一致したものが選択される。それが望ましくない場合、FEEL 式の曖昧さを解消するために括弧やその他の句読点を使用することができる(括弧やその他の句読点は名前には使用できない)。
たとえば、aからbを減算する時に、 'a-b'がスコープ内のコンテキスト・エントリーの名前である場合、(a)-(b)と書くことができる。

10.3.2 セマンティクス

FEEL セマンティクスは、構文断片をマッピングすることで、FEEL セマンティクス領域の値に変換する。リテラル（10.3.1.3 リテラル、データ型、組み込み関数）は直接マッピングできる。リテラルで構成される式はセマンティクス領域の値にマップされ、単純な論理と、マッピングされたリテラル値に対する算術演算を使用して変換される。一般に、FEEL式のセマンティクスは、その下位の式のセマンティクスから構成される。

10.3.2.1 セマンティクス領域

FEELセマンティクス領域のDは、次の無限個の**kinds**の値からなるが、それはリスト、コンテキスト、範囲、データ型、および**null**である。　各種類は互いに素である（例えば、値は数字とリストの両方になることはできない)。

関数は字句解析を含むラムダ式、またはJavaまたはPMMLによって外部定義される。リストは、領域の要素の順序付けられた集合であり、コンテキストは、コンテキスト・エントリーと呼ばれる部分的に順序付けられた（文字列と値）の組の集合である。

インタクス要素を表すために*斜体*を使用し、セマンティクス要素を表すために**太**字を使用する。 例えば、
FEEL(*[1+1, 2+2]*) は **[2, 4]**である。
注記。太字の []は**FEEL**セマンティクス領域のリストを示すために使用し、太字の2,4は**FEEL**セマンティクス領域の10進数値を示すために使用している。

10.3.2.2　　等式、恒等式、等価

セマンティクスが等式であることは、項目 10.3.2.12「セマンティック・マッピング」のセマンティック・マッピングで判定される。一般に、比較される値は同じ種類でなければならない。 例えば、両方の数値を使用して、null 以外の結果を得る。恒等式は意味空間内で 2 つのオブジェクトが同じオブジェクトであるのか、単に比較する。恒等式は、infix **is**、否定の場合は、infix is **not** と表す。 例えば、FEEL(″1″ = 1) is null と書き表す。注記、結果が null になることは決してない。
すべての FEEL 式のスコープ内の e は、FEEL 意味空間内の要素 e にマッピングできる。このマッピングは、s 内に e を定義する。マッピングは、FEEL（e、s）と書くことができる。 2 つの FEEL 式 e_1 と e_2 が、FEEL（e_1、s）が FEEL（e_2、s）の場合にのみスコープ s 内で等価である。コンテキストから（または重要ではない）s が理解されるとき、e_1 が e_2 であるため、等価であることを省略することができる。また、e1 と e2 が同じで、null でないことを確認したら、次の様に書くことができる。
$e_1 = e_2$

10.3.2.3　　文字とデータ型のセマンティクス

FEEL データ型については、後続の節で説明する。 データ型の意味には次を含む：
1. リテラルフォーム（場合によっては文字列）から意味空間内の値へのマッピング。
2. データ型に属する意味空間の値の集合と、それらに対する操作の正確な定義。

各データ型は、(おろらく無限に）値の集合を記述できる。 以下に定義されているデータ型の集合は、互いに素である。
斜体はリテラルを示す、太字は意味空間を示す。

10.3.2.3.1　数値

FEEL 数値は IEEE 754-2008 Decimal128 形式に基づいており、10 進数で 34 桁の精度で五捨五入する、中間値の場合は偶数側に五捨五入する。数値には、XML スキーマ型 precisionDecimal の制限があり、MathContext DECIMAL128 を使用する Java BigDecimal と同じである。

文法規則 37 は、数値に使うリテラルを定義する。リテラルは、基本 10 桁と小数点（オプション）から構成される。 -INF、+ INF、および NaN リテラルはサポートされていない。また、-0 と 0 の区別はない。number(from、grouping separator、decimal separator)

組み込み関数は、よりリッチなリテラル形式をサポートしている。たとえば、FEEL（数値（ ″1.000.000,01″、 ″。″、 ″、″））= 1000000.01 である。
FEEL は科学記法をサポートしていない。たとえば、1.2e3 は有効な FEEL 構文ではない。これは 1.2 * 10 ** 3 と表記する。

FEEL 数値は、意味空間内で整数の組（p、s）として表され、p は精度情報を保持する符号付き 34 桁の整数であり、s はスケールで、[-6111..6176]の範囲である。このような組は、p /10s の数を表す。 数値を示すには、value（p, s）と書く。たとえば、value（100, 2）= 1 である。 精度に問題がなければ、その値を単純に 1 と書くことができる。多くの異なる組が同じ値を持つことに注意すること。たとえば、value（1, 0）=value（10, 1）=value（100, 2）。

notANumber、positiveInfinity、または negativeInfinity には値は無い。 代わりに null を使用する。

10.3.2.3.2 文字列

文法規則 35 は、文字列を二重引用符で囲まれた文字列（例えば、「abc」）として定義する。

文字列 ″abc″は、 ″abc″と書かれた 3 つの Unicode 文字 a、b、c のシーケンスとしてセマンティックドメインにマップされます。

10.3.2.3.3 論理型

論理型は、文法規則 36 によって定義される。意味空間の値は、true と false である。

10.3.2.3.4 時刻

FEEL には時刻リテラルは無いが、代わりに意味空間の値を表すために時刻を太字で表示する。時刻は文字列と time () 組込み関数で使用できる。(10.3.4.1 変換関数を参照)

意味空間の時刻は、XML Schema の時刻データ型の値である。時刻は、一連の数字で表すことができ、 時、分、秒、および UTC（Universal Coordinated Time）からの時差（オプション）が含まれる。時差が指定されている場合、time offset= 00:00、時刻値は UTC 形式を持ち、UTC フォームを持つすべての時刻値と同じである。時差が指定され

ていない場合、時間オフセットが指定されていない場合、時刻はその場所でのローカル時刻として解釈され、そのUTC時刻との関係は当該場所のタイムゾーンルールに依存し、日によって異なる場合がある。XML Schema パート 2 のデータ型で説明されているように、ローカル時刻値はUTC時刻値と同じである。

時刻 t は、午前0時からの秒数で表すこともできる。 これを $value_t(t)$ と書く。例えば $value_t(01:01:01) = 3661$ と書く。

$value_t$ 関数は 1 対 1 であり、その範囲は[0..86400]に制限されている。したがって、逆関数 $value_t^{-1}(x)$ は、x が[0..86400]にある場合には対応する x の時間値を、x が[0..86400]にない場合は y = x - floor (x / 86400) * 86400 の $value_t^{-1}(y)$ を、返す。

注：つまり、$value_t^{-1}(x)$ は、実際には常に 86400 剰余 x となる。たとえば、$value_t^{-1}$ (3600) は "T01：00：00"の時刻を返し、$value_t^{-1}$ (90000) も "T01：00：00"を返し、$value_t^{-1}$ (-3600) は "T23：00：00"の時刻を返す。また、午前0時の1時間前に-3600秒を処理する。

10.3.2.3.5 日付

FEELには日付リテラルは無いが、代わりに意味空間の値を表すために日付を太字で表示する。日付は文字列と date()組み込み関数が使える (10.3.4.1 変換関数を参照)。意味空間上の date は、年、月、日という数値列である。年は[-999,999,999..999,999,999]の範囲でなければならない。

必要に応じて、$value_{dt}$ 関数(10.3.2.x.6 を参照)を含む日付値は、UTC 午前0時(00:00:00)である日時値と同じであるとみなされる。

10.3.2.3.6 日時

FEEL には日時リテラルは無いが、代わりに意味空間の値を表すために日時を太字で表示する。日時は文字列と time () 組込み関数で使用できる。(10.3.4.1 変換関数を参照)

セマンティクス領域の日付と時刻は、年、月、日、時、分、秒の時間オフセットで、UTC (Universal Coordinated Time) との差である。 年は[-999,999,999..999,999,999]の範囲でなければならない。時間オフセットが 00:00 の場合、日付 - 時刻値は UTC 形式を持ち、UTC 形式を持つ他のすべての日付 - 時刻値と同等である。時間オフセット

が関連づけられていない場合、当該場所のタイムゾーン規則に従い、ローカル時刻となる。タイムゾーンが指定されている場合（例えば、10.3.4.1 変換関数を参照）、IANA tz フォームを使用して、適用可能であれば、タイムゾーン規則を使用して日時値を UTC フォームに変換することができる。

注：想定外のタイムゾーンルール変更（将来の）により、日時値の変更が必要となるかもしれない。

UTC 形式の日時 d は、基準日時（エポックと呼ばれる）からの秒数で表すことができる。d とエポックとの間の秒数は $value_{dt}$ (d) と書く。$value_{dt}$ 関数は 1 対 1 であるため、逆関数 $value_{dt}^{-1}$ がある。例えば、$value_{dt}^{-1}$(valuedt(d)) = d. $value_{dt}^{-1}$ は、有効範囲外の年月日ではなく null を返す。

10.3.2.3.7　期間

FEEL には期間リテラルは無いが、代わりに意味空間の値を表すために期間を太字で表示する。期間は文字列と duration () 組込み関数で使用できる。文字列の引用符内のフォーマットは、Xpath データ・モデルの dayTimeDuration データ型の字句スペースによって定義される。意味空間の期間は、日数、時間、分、秒の一連の数値であり、これらの数値の合計が最小になるように正規化される。たとえば、
FEEL (duration ("P0DT25H")) = **P1DT1H**
また、日と時の値は秒数で表すこともできる。 例えば、$value_{dtd}$(P1DT1H) = 90000。 さらに、$value_{dtd}$ 関数は 1 対 1 であるため、逆関数 $value_{dtd}^{-1}$ がある。
たとえば、$value_{dtd}^{-1}$(90000) = P1DT1H である。

10.3.2.3.8　年月期間

FEEL には年月期間リテラルは無いが、代わりに意味空間の値を表すために年月期間を太字で表示する。年月期間は文字列と duration () 組込み関数で使用できる。文字列の引用符内のフォーマットは、Xpath データ・モデル yearMonthDuration データ型の字句スペースによって定義される。意味空間の年と月の期間は、年数と月数の 2 つの数値の和である。これらの数値の和が最小になるように正規化される。 例えば、
FEEL(duration("P0Y13M")) = **P1Y1M**
年と月の期間は、月数累計で表すこともできる.例えば、 $value_{ymd}$(P1Y1M) = 13。 さらに $value_{ymd}$ 関数は 1 対 1 であるため逆関数 $value_{ymd}^{-1}$ がある。例えば、
$value_{ymd}^{-1}$(13) = P1Y1M.

10.3.2.4　　三値論理

FEEL は、SQL や PMML のように、真理値に三値論理を使用する。これにより、D×D→D の and および or の機能が完結する。 3 値論理は Predictive Modeling Markup Language で欠損データ値をモデル化するために使用される。

10.3.2.5　　リストとフィルター

リストは不変であり、ネストすることができる。
リスト L の *first* 要素は *L [1]*でアクセスでき、*last* 要素は *L [-1]*でアクセスできる。最初から n 番目の要素は *L [n]*でアクセスでき、最後から n 番目の要素は *L [-n]*でアクセスできる。

FEEL (L) = L が FEEL 意味空間のリストである場合、最初の要素は FEEL (L [1]) = L [1] である。なお、L が n 個の項目を含まない場合、L [n]は null となる。

L は角括弧内の論理式でフィルターリングできる。角括弧で囲まれた式は、リスト要素が key ″item″を含むコンテキストでない限り、*item* 名を使用してリスト要素を参照することができる。リスト要素がコンテキストの場合、そのコンテキスト・エントリーは、item 接頭辞なしでフィルター式内で参照できる。例えば：

[1, 2, 3, 4][item > 2] = [3, 4]

[{x:1, y:2}, {x:2, y:3}][x=1] = {x:1, y:2}

フィルター式はリスト内の各項目について評価し、フィルター式が真である項目のみを含むリストが返される。

シングルトンのリストは、単一項目と同じである。 したがって、リストを入力としリスト以外のセマンティクス領域の要素 e を受け取る任意の関数または演算子は、[e]を入力として受け取ったかのように動作する。

便宜上、″.″演算子を使用した選択では、その左側にコンテキストのリストが表示され、選択肢のリストが返される。すなわち FEEL(e.f, c) = [FEEL(f, c'), FEEL(f, c''), …]ここで FEEL(e) = [e', e'', …]と c' は c であり、c は c 'e'のコンテキスト・エントリーの文脈項目で拡張され、c "は、e "などのコンテキスト・エントリーで拡張される。例えば、

[{x:1, y:2}, {x:2, y:3}].y = [2,3]

10.3.2.6 コンテキスト

FEEL コンテキストは、コンテキスト・エントリーと呼ばれ、部分的に順序付けられた（key、式）の組の集合である。構文では、key は名前または文字列となる。key は意味空間の文字列にマップされる。これらの文字列は、コンテキスト内では区別される。空間内のコンテキストは、文字列キーを有する太字の FEEL 構文で表される、例えば、{ **"key$_1$" : expr$_1$, "key$_2$" : expr$_2$, …** }.

コンテキスト値式 *m* から *key$_1$* という名前のエントリーの値を選択するシンタクスは、m.*key$_1$* である。

key$_1$ が登録名でないか、何らかの理由で key を文字列として扱いたい場合は、次の構文が使用できる：get value(m, *"key$_1$"*)。意味空間のコンテキスト m から key により値を選択する場合は、m.key1 または get value(m, *"key$_1$"*) が使用できる。

コンテキスト m から key と値の組のリストを取得するには、*get entries(m)*組込み関数を使用する。

たとえば、次のことが当てはまる。
get entries({key$_1$: "value$_1$"})[key="key$_1$"].value = "value$_1$"

コンテキスト・エントリー内の式は、同じコンテキスト・エントリーの key を参照することはできないが、同じコンテキスト内の他のコンテキスト・エントリーから keys (as QNs) として参照することができる。 これらの参照は非循環であり、部分的な順序を形成しなければならない。また、コンテキスト内の式は、この部分的な順序とともに、一貫して評価されなければならない。

10.3.2.7 範囲

FEEL は、範囲に対してコンパクトなシンタクスをサポートし、デシジョンテーブルのテスト・セルなどで役立つ。 範囲は意味空間にマッピングされる。マッピングされる範囲は、比較可能な値（数値、日付/時間/継続時間、または文字列）、または、エンドポイントがその範囲に含まれているかどうかを示すエンドポイントを包含するコードと、エントリーポイント値との対である。

範囲シンタクスは、リテラルとシンボリックエンドポイントをサポートしているが、任意の表現ではない。なぜなら日付/時刻/継続時間の値にはリテラルによるシンタクスがないためである、そのため、これらのタイプの範囲にシンボリックエンドポイントを使用する必要がある。たとえば、次のコンテキストでは「すぐに(soon)」を 1 分～2 分という形で定義している。

```
{
one min: duration("PT1M"),
two min: duration("PT2M"),
soon: [one min..two min]
}
```

10.3.2.8　　デシジョンテーブル

デシジョンテーブルの表記規則は第 8 章で規定されている。各入力式は、入力変数（修飾名）を参照するテキスト式（文法規則 2）である。入力値の各リストは単項式検定（文法規則 17）のインスタンスである。テストされる値は、条項の入力式の値である。出力値の各リストは単項式検定（文法規則 17）のインスタンスである。テストされる値は、条項で選択された出力項目の値である。各入力項目は単項式検定（文法規則 17）のインスタンスである。デシジョンテーブル組み込み関数の呼び出しを使用したテキスト表現は、FEEL の残りの部分と同じようにシンタクスをセマンティクスに結びつけるために提供されている。 単項式検定（文法規則 17）は、セマンティクス領域に孤立してマッピングすることはできず、単一引用符で囲んだ構文形式のままとなる。例えば、表 26 の最初のデシジョンテーブルをテキスト形式で表すと以下の様になる。

```
decision table(
        outputs: "Applicant Risk Rating",
        input expression list: [Applicant Age, Medical History],
        rule list: [
                ['>60', "good", '"Medium"'],
                ['>60', '"bad"', '"High"'],
                ['[25..60]', '-', '"Medium"'],
                ['<25', '"good"', '"Low"'],
                ['<25', '"bad"', '"Medium"']],
        hit policy: "Unique")
```

10.3.4.6 デシジョンテーブルに記載された組み込みデシジョンテーブルは、単項式テストのシンタクスを FEEL 意味空間にマッピングできる式に構成する。

10.3.2.9　　スコープとコンテキスト・スタック

FEEL 式 e は、常に明確に定義された名前バインディングのセットで評価される。名前バインディングは e の QN を解決するために使用される。この名前バインディングのセットは、e のスコープと呼ばれる。スコープは、コンテキストのリストとしてモデル化されている。なお、スコープ s は、e のスコープ内にあるエントリーを持つコンテキストを含む。 s の最後のコンテキストはビルトイン・コンテキストである。s の最後の前にはグローバル・コンテキストがある。 s の最初のコンテキストは、e（存在する場合）を直接含むコンテキストである。 次は、e（存在する場合）のコンテキストを囲む。 e の QN は、最初のコンテキストの QN で、N が付加されている。ここで、N は、e を含む s の最初のコンテキスト内のエントリーの名前である。e の QN は、s のコンテキストを最初から最後まで調べることによって解決される。

10.3.2.9.1　　ローカル・コンテキスト

e がコンテキスト m のコンテキスト・エントリーの値を表す場合、m は e のローカル・コンテキストであり、m は s の最初の要素である。 それ以外の場合、e にはローカル・コンテキストがなく、s の最初の要素はグローバル・コンテキストであり、いくつかのケースを後で説明するが、s の最初の要素は特別なコンテキストである。

m のすべてのエントリーは e のスコープ内にあり、グラフに依存するものは非巡回であるべきである。上記の紛らわしい問題に対する簡単な解決法を提供する： m が e を含むコンテキスト表現 m を評価した結果である場合、どのように e を評価するためにそれを知ることができるだろうか？シンプルな評価方法はコンテキスト・エントリーを*順番に評価する*ことである。

10.3.2.9.2　　グローバル・コンテキスト

グローバル・コンテキストは、利便性と「事前コンパイル」のために提供されるコンテキストである。任意の数の式を FEEL コンテキスト m 内に名前をつけて表現することができる。シンタクス記述 m は、一度に評価することができる。すなわち、まず m として FEEL ドメインにマッピングする。その後は、複数の式を評価するために再使用される。

10.3.2.9.3　組込みコンテキスト

組込みコンテキストには、すべての組込み関数が含まれている。

10.3.2.9.4　特殊コンテキスト

FEEL 式では、s の前に押し出される特別なコンテキストが評価されることがある。例えば、特別な最初のコンテキストで 'item' という名前を含む要素を、連続するリスト要素にバインドするフィルター式を繰り返し実行することが挙げられる。関数は、name->value のマッピングを含む特殊な最初のコンテキストで実行される。

FEEL 式の正規の名前 (QNs) は、s に関連して解釈される。スコープ s 内の FEEL 式 e は、FEEL（e、s）で表される。 e はスコープ s 内で e と評価する、または e = FEEL（e、s）という事ができる。注意、e と s は FEEL 領域の要素である。 s はコンテキストのリストである。

10.3.2.10　FEEL と他の領域間のマッピング

FEEL 式 e は、セマンティクス領域内の値 e を示す。いくつかの種類の値は、FEEL と外部 Java メソッドの間、FEEL と外部 PMML モデルの間、FEEL と XML の間で値を渡すことができる。表 35 に要約を示す。

表 35：FEEL と他のドメイン間のマッピング

FEEL 値	*Java*	*XML*	*PMML*
number	java.math.BigDecimal	decimal	decimal, PROB-NUMBER, PERCENTAGE-NUMBER
		integer	integer , INT-NUMBER
		double	double, REAL-NUMBER
string	java.lang.String	string	string, FIELD-NAME
date and time, time	javax.xml.datatype. XMLGregorianCalendar	date, dateTime, time, dateTimestamp	date, dateTime, time conversion required for dateDaysSince, et. al.
duration	javax.xml.datatype. Duration	yearMonthDuration, dayTimeDuration	X
boolean	java.lang.Boolean	boolean	boolean
range	TBD	TBD	TBD

function	X	X	X
list	java.util.List	contain multiple child elements	array (homogeneous)
context	java.util.Map	contain attributes and child elements	X

e の型を知りたいだけで、FEEL 式 e を評価したくない場合もある。注記、e に QN がある場合、型を推定するためにコンテキストが必要になることがある。その場合、領域の要素を **FEEL**(e, c)の型として **type**(e)と書く。

10.3.2.11 関数のセマンティクス

FEEL 関数は以下の事が出来る：

- 組込みなど
  decision table (see 10.3.4.6 Decision Table), or
- ユーザー定義など
  function(age) age < 21, or
- 外部定義など
  function(angle) external {
  java: {
  class: "java.lang.Math",
  method signature: "cos(double)"
  }}

FEEL の組込みは第 10.3.4 項で仕様化される。

10.3.2.11.1 ユーザー定義関数

ユーザー定義関数にはフォームがある
function(X_1, … X_n) e
項 X_1, … X_n はパラメータ名である。そして、関数本体は e である。FEEL(function(X_1, … X_n) e, s)は FEEL セマンティクス領域の中で、関数 **function(argument list**: [X_1, … X_n], **body**: e, **scope**: s)（以下 f と記載）と表示する。FEEL 関数は語彙に関して閉じており、本体の式は正規のパラメータとスコープ s の内の他の名前を参照する。FEEL ユーザー定義関数 f の呼び出しは、**f**(Y_1, … Y_n)として示される。FEEL(**f**(Y_1, … Y_n), **S**)は、ここで f はすでに解釈されており、次のように計算される。

1. パラメータ名 X_1, … X_nおよび該当する値 Y_1, … Yn は、コンテキスト c = {X_1 : Y_1, … X_n : Y_n}において対になる。
   Y_i = FEEL(Y_i, S).
2. s' = **insert before**(s, 1, c)において e は s 'で解釈されている（項目 10.3.4.4 リスト関数を参照)。

10.3.2.11.2　外部定義関数

FEEL 外部定義関数は以下のフォームを持つ
function(X_1, … X_n) external mapping-information
マッピング情報は以下のフォームのうち 1 つを持つコンテキストである。

```
{
    java: {class: class-name, method signature: method-signature}
}
```

or

```
{
    pmml: {document: IRI, model: model-name}
}
```

意味流域の要素である外部定義関数は以下の様に表示される
function(argument list: [X_1, … X_n], external: mapping-information).
マッピング情報の java 形式は、外部関数として Java のクラス・メソッドがアクセスされることを示す。クラス名は、クラスパス上の Java クラスの文字列名である必要がある。クラスパスの規約は実装で定義される。*メソッドシグネチャ*は、名前付きクラスのパブリック静的メソッドの名前と、Java 引数名リストからなる文字列である必要がある。オーバーロードされたメソッドを解決するために引数型情報を使用し、それは実行前に領域外エラーを検出するために使用される。

マッピング情報の pmml 形式は、外部関数として PMML モデルがアクセスされることを示す。IRI は、PMML ドキュメントの識別子である必要がある。なお、*モデル名*はオプションである。*モデル名*が指定されている場合は、IRI が参照する文書内のモデルの名前とする。モデル名が指定されていない場合、外部関数がドキュメント内の最初のモデルとなる。

外部定義関数が起動されると、実際の引数値と結果値は、Java または PMML のためのタイプマッピングテーブル(型と推論の表を参照)を使用して、可能な範囲で変換される。

変換できない場合、null が代入される。結果が取得できない場合（例外がスローされた場合など）も、起動の結果は null になる。

パラメータ値を外部メソッドまたはモデルに渡すには、パラメータ型を知る必要がある。Java の場合はリフレクションを用いて得られる。PMML の場合は、選択されたモデルの独立変数に関連付けられたマイニング・スキーマおよびデータ辞書の要素から得られる。

注記：DMN は、外部定義関数を使用する意思決定モデルのセマンティクスを完全には定義していない。外部定義された関数は副作用がなく、確定的なロジックとすること。

10.3.2.11.3　関数名

関数に名前を付けるために、関数をコンテキスト・エントリーとして定義する。 例えば、

```
{
    isPositive : function(x) x > 0,
    isNotNegative : function(x) isPositive(x+1),
    result: isNotNegative(0)
}
```

10.3.2.11.4 位置パラメータまたは名前付きパラメータ

FEEL 関数（組み込み、ユーザー定義、外部定義）の起動は、位置パラメータまたは名前付きパラメータが使用できる。位置があれば、すべてのパラメータが提供される。名前が付いている場合、提供できないパラメータは null となる。

10.3.2.12　　セマンティック・マッピング

各実質的な文法規則は、シンタクスをセマンティクス領域の値にマッピングすることによって下の表のように与えられる。値は、セマンティクス領域にマッピングされている特定の入力値に依存する場合がある。また、入力値は、追加の制約に従わなければならない場合がある。なお、入力領域は、セマンティクス領域の一部であってもよい。また、領域外の入力は NULL 値になる。

表 36:FEEL 関数のセマンティック

文法規則	FEEL 構文	領域へのマッピング
57	*function*(n_1,…n_N) *e*	function(argument list: [n_1, … n_N], body: e, scope: s)
57	*function*(n_1, …n_N) *external e*	function(argument list: [n_1, … n_N], external: e)

項目 10.3.2.7 領域を参照

表 37:それ以外の FEEL 式のセマンティック

文法規則	FEEL 構文	領域へのマッピング
46	for n_1 in e_1, n_2 in e_2, … return e	[FEEL(e, s'), FEEL(e, s''), …]
47	if e_1 then e_2 else e_3	if FEEL(e_1) is true then FEEL(e_2) else FEEL(e_3)
48	some n_1 in e_1, n_2 in e_2, … satisfies e	FEEL(e, s') or FEEL(e, s'') or …
48	every n_1 in e_1, n_2 in e_2, … satisfies e	FEEL(e, s') and FEEL(e, s'') and …
49	e_1 or e_2 or …	FEEL(e_1) or FEEL(e_2) or …
50	e_1 and e2 and …	FEEL(e_1) and FEEL(e_2) and …
51.a	e = null	FEEL(e) is null
51.a	null = e	FEEL(e) is null
51.a	e != null	FEEL(e) is not null
51.a	null != e	FEEL(e) is not null

注意：FEEL 領域では、太字の構文を使用して、コンテキスト、リスト、接続詞、論理和、条件式、真、偽、および null を表示する。

論理積 **a and b** と論理和 **a or b** は、3 値論理によって定義される。これらは全域関数であるため、入力は真、偽、またはその他（真または偽以外の **D** の要素）である可能性がある。

表 38:論理積と論理和のセマンティクス

a	b	a and b	a or b
true	true	true	true
true	false	false	true
true	otherwise	null	true
false	true	false	true
false	false	false	false
false	otherwise	false	null
otherwise	true	null	true
otherwise	false	false	null
otherwise	otherwise	null	null

否定は組込み関数 not を使用して行われる。 3 値論理は表 39 に示すとおりである。

表 39: 否定のセマンティクス

a	not(a)
true	false
false	true
otherwise	null

条件「 a が真であれば、b else c は b と等しく、そうでなければ c と等しい」。

s'は、key である n_1, n_2, などを含む先頭行の特別なコンテキストを持つスコープである。FEEL(e_1) x FEEL(e_2) x …のデカルト積の最初の要素に結合され、s''は、デカルト積の第 2 の要素にバインドされた key を含む先頭行の特別なコンテキストを有する s である。

等号と不等号を、表 40、表 41、表 42 に示されたいくつかの「種類別およびデータ型固有のテスト」にマッピングした。定義上、FEEL(e_1 != e_2)は FEEL(not(e_1=e_2))と同じである。 他の比較演算子は、表 42 にリストされているデータ型に対してのみ定義されている。 注意： 表 42 は '<' のみを定義しているが、'>'は '<' によく似ており、簡潔にするために省略されている。また、e_1<=e_2 は e_1<e_2 または e_1=e_2 と定義される。

表 40:等号と不等号の一般的セマンティクス

文法規則	FEEL 構文	入力領域	結果
51.a	e1 = e2	e1 と e2 は、両方とも同じ種類/データ型であること - 両方とも数字と両方とも文字列など	下記参照
51.a	e1 < e2	e1 と e2 は、両方とも同じ種類/データ型であること - 両方とも数字と両方とも文字列など	下記参照

表 41: 等号に特有のセマンティクス

種類/データ型	$e_1 = e_2$
list	リストは同じ長さ N であること、そして、1≦i≦N に対して e_1[i] = e_2 [i]であること。
context	コンテキストは同じ Key 項目である K を持つこと、そして、e_1.k = e_2.k の各 k は K に含まれること。
range	範囲は、エンドポイントと エンドポイント包括コードを同じにすること。
function	内部関数は、同じパラメーター、ボディー、およびスコープを持つこと。 外部定義関数は、同じパラメーター、そして外部マッピング情報を持つこと。
number	value(e_1) = value(e_2). 値は 10.3.2.3.1 章「数値」で定義。なお、精度は考慮されない。
string	e_1 は e_2 としての文字列と同じである。
date	3 つの要素（項目 10.3.2.3.5「日付」）は全て等しい必要がある。
date and time	指定されていない要素を null として処理した上で、7 つの要素（項目 10.3.2.3.5「日付」）は全て等しい必要がある。
time	指定されていない要素を null として処理した上で、4 つの要素（項目 10.3.2.3.5「日付」）は全て等しい必要がある。
days and time duration	value(e_1) = value(e_2). 値は項目 10.3.2.3.7「期間」で定義。
years and months duration	value(e_1) = value(e_2).値は 10.3.2.3.8 章「年月期間」で定義。
boolean	e_1 と e_2 は共に真、または共に偽、である必要がある。

表 42:不等号に特有なセマンティクス

データ型	$e_1 < e_2$
number	value(e_1) = value(e_2). 値は項目 10.3.2.3.1 「数値」で定義。なお、精度は考慮されない。
string	文字列 e_1 は文字列 e_2 よりも辞書的に小さい。すなわち、必要に応じて¥*u0* 文字を詰めて同じ長さにし、共通の接頭文字を削除し、各文字列の最初の文字が比較される。
date	e_1 < e_2 を判定する時、「 e_1 の年の値」 < 「e_2 の年の値」を判定する。年の値が等しい時は、「e_1 の月の値」 <「 e_2 の月の値」を判定する。年と月の値が等しい時は、「e_1 の日の値」 <「 e_2 の日の値」を判定する。
date and time	valuedt(e_1) < valuedt(e_2) 値は項目 10.3.2.3.5 「データ」で定義。 一方の入力にタイムゾーンオフセットがない場合、その入力には他方の入力のタイムゾーンオフセットを使用する。
time	valuedt(e_1) < valuedt(e_2) 値は項目 10.3.2.3.4 「時刻」で定義。 一方の入力にタイムゾーンオフセットがない場合、その入力には他方の入力のタイムゾーンオフセットを使用する。
days and time duration	valuedtd(e_1) < valuedtd(e_2) 値は項目 10.3.2.3.7 「期間」に定義
years and months duration	valueymd(e_1) < valueymd(e_2) 値は項目 10.3.2.3.8 「年月期間」に定義。

ちなみに、FEEL は、糖衣構文をサポートする。文法規則（項目 10.3.1.2「文法規則」）は、デシジョンテーブルの条件セルに適用されることに注意すること。文法規則 51c では、正式な名前はモデリング時に同等の定数値で評価されなければならない。すなわち、終点はリテラルまたは名前付き定数でなければならない。これらのデシジョンテーブルの構文は、表 43 で定義されている。

表 43:デシジョンテーブル構文のセマンティクス

文法規則	FEEL 構文	同等の FEEL 構文	適用性
51.b	e_1 between e_2 and e_3	e_1 >= e_2 and e_1 <= e_3	
51.c	e_1 in [e_2, e_3, …]	e_1 = e_2 or e_1 = e_3 or…	e_2 and e_3 are endpoints e_2 と e_3 はエンドポイント
51.c	e_1 in [e_2, e_3, …]	e_1 in e_2 or e_1 in e_3 or …	e_2 and e_3 are ranges e_2 と e_3 は範囲
51.c	e_1 in <=e_2	e_1 <= e_2	
51.c	e_1 in <e_2	e_1 < e_2	
51.c	e_1 in >=e_2	e_1 >= e_2	
51.c	e_1 in <e_2	e_1 < e_2	
51.c	e_1 in (e_2..e_3)	e_1 > e_2 and e_1<e_3	
51.c	e_1 in (e_2..e_3]	e_1 > e_2 and e_1<=e_3	
51.c	e_1 in [e_2..e_3)	e_1 >= e_2 and e_1<e_3	
51.c	e1 in [e_2..e_3]	e_1 >= e_2 and e_1<=e_3	

加算と減算は表 44 と表 45 に定義。注意、入力値がリストにない場合、結果は null となる。

表 44:加減算の一般的なセマンティクス

文法規則		入力領域と結果
21	e_1 + e_2	下記参照
22	e_1 - e_2	下記参照

表 45:加減算に特有なセマンティクス

type(e_1)	type(e_2)	e_1 + e_2, e_1 - e_2	結果の型
number	number	e_1=(p_1, s_1) と e_2=(p_2, s_2) は 10.3.2.3.1 章「数値」で定義された。もし、value(p_1, s_1) +/- value(p_2, s_2) が有効なスケールの範囲外のスケールが必要な場合、結果は null となる。そうでない場合 (p, s) は以下の様になる。 ・ value(p, s) = value(p_1, s_1) +/- value(p_2, s_2) + ε	number

		・ $s \leq \max(s_1, s_2)$ s は、p が 34 桁以下であるという制限下で最大化される ε は許容可能な丸め誤差である。	
date and time	date and time	加算は定義されていない。 減算は、$value_{dtd}^{-1}(value_{dt}(e_1)-value_{dt}(e_2))$の様に定義されており、$value_{dt}$ は 10.3.2.3.5 章「日付」で定義されている。また、$value_{dtd}^{-1}$ は 10.3.2.3.7 章「期間」で定義されている。	days and time duration
time	time	加算は定義されていない。減算は $value_{dtd}^{-1}(value_t(e_1)-value_t(e_2))$ の様に定義されており、valuet は 10.3.2.3.4 章に、$value_{dtd}^{-1}$ は 10.3.2.3.7 章「期間」に定義されている。	days and time duration
years and months duration	years and months duration	$value_{ymd}^{-1}(value_{ymd}(e_1)$ +/- $value_{ymd}(e_2))$ valueymd と $value_{ymd}^{-1}$ は項目 10.3.2.3.8「年月期間」に定義。	years and months duration
days and time duration	days and time duration	$value_{dtd}^{-1}(value_{dtd}(e_1)$ +/- $value_{dtd}(e_2))$ $value_{dtd}$ と $value_{dtd}^{-1}$ は項目 10.3.2.3.7「期間」に定義。	days and time duration
date and time	years and months duration	date and time (date(e_1.year +/- e_2.years + floor((e_1.month +/- e_2.months)/12), e_1.month +/- e_2.months - floor((e_1.month +/- e_2.months)/12) * 12, e_1.day), time(e_1)), 名前付きパラメータは表 53 で定義。日付、期間、時刻、床関数は 10.3.4 章で、$value_{dt}$ と $value_{dt}^{-1}$ は項目 10.3.2.3.5「日付」で、$value_{ymd}$ は項目 10.3.2.3.8「年月期間」で定義。	date and time
years and months duration	date and time	減算は定義されていない。 加算は可換であり、前述の規則で定義されている。	date and time

date and time	days and time duration	$value_{dt}^{-1}$(valuedt(e_1) +/- valuedtd(e_2)) valuedt と $value_{dt}^{-1}$ は 10.3.2.3.5 章に、and $valued_{td}$ は項目 10.3.2.3.7「期間」に定義。	date and time
days and time duration	date and time	減算は定義されていない。 加算は可換であり、前述の規則で定義されている。	date and time
time	days and time duration	$value_t^{-1}$(valuet(e_1) +/- valuedtd(e_2)) valuet と $value_t^{-1}$ は項目 10.3.2.3. 「時刻」に、$value_{dtd}$ は項目 10.3.2.3.7 「期間」に定義。	time
days and time duration	time	減算は定義されていない。 加算は可換であり、前述の規則で定義されている。	time
string	string	減算は定義されていない。加算は文字列を連結する。 結果は、e_1 の文字列とそれに続く e_2 の文字列を含む文字列である。	string

乗算と除算は表 46 と表 47 で定義。入力値がリストにない場合、結果は null。

表 46:剰余算の一般的なセマンティクス

文法規則	FEEL	入力領域と結果
23	e1 * e2	下記参照
24	e1 / e2	下記参照

表 47:剰余算に特有なセマンティクス

type(e_1)	type(e_2)	e_1 * e_2	e_1 / e_2	結果の型
number $e_1=(p_1, s_1)$	number $e_2=(p_2, s_2)$	value(p_1, s_1) * value(p_2, s_2) が有効なスケールの範囲外のスケールが必要な場合、結果は null となる。 そうでない場合 (p, s) は以下の様になる	value(p_2, s_2)=0 または value(p_1, s_1) / value(p_2, s_2) がが有効なスケールの範囲外のスケールが必要な場合、結果は null となる。 そうでない場合 (p, s) は以下の様になる	number

		・ value(p, s) = value(p_1, s_1) * value(p_2, s_2) + ε ・ s ≤ s1+s2 ・s は、p が 34 桁以下であるという制限下で最大化される ε は許容な丸め誤差である。	・ value(p, s) = value(p_1, s_1) / value(p_2, s_2) + ε ・ s ≤ s1−s2 ・s は、p が 34 桁以下であるという制限下で最大化される ・ε は許容な丸め誤差である。	
years and months duration	number	$value_{ymd}^{-1}(value_{ymd}(e_1) * value(e_2))$ $value_{ymd}$ と $value_{ymd}^{-1}$ は項目 10.3.2.3.7「期間」に定義。	value(e_2)=0 の場合、結果は null である．そうでない場合、結果は $value_{ymd}^{-1}(value_{ymd}(e1) / value(e_2))$である。 $value_{ymd}$ と $value_{ymd}^{-1}$ は項目 10.3.2.3.8 「年月期間」に定義。	years and months duration
number	years and months duration	上を参照、 e_1 と e_2 を逆にする。		
days and time duration	number	$valued_{td}^{-1}(value_{dtd}(e_1) * value(e_2))$ $value_{dtd}$ と $value_{dtd}^{-1}$ は項目 10.3.2.3.7 「期間」に定義。	value(e_2)=0 ならば、結果は null となる。それ以外の場合の結果は、$value_{dtd}^{-1}(value_{dtd}(e_1) * value(e_2)$となる。 $value_{dtd}$ と $value_{dtd}^{-1}$ は項目 10.3.2.3.7 「期間」に定義。	days and time duration
number	days and time duration	上を参照、 e_1 と e_2 を逆にする。		

表 48: 累乗法のセマンティクス

文法規則	FEEL 構文	入力領域	結果
25	e_1 ** e_2	type(e_1) は数値、value(e_2) は範囲が [-999,999,999..999,999,999] の整数.	value(e_1)$^{value(e2)}$) に範囲外のスケールが必要な場合、結果は**null** となる。そうでなければ、(p, s)は以下の様になる · value(p, s)= value(e_1)value(e_2) + ε · p は 34 桁以下である。 · ε は丸め誤差である

型チェックは表 49 で定義。 注意、型は領域にマッピングされていない、また null は型の名前ではない、そして null はどの型のインスタンスでもない。

表 49:型チェックのセマンティクス

文法規則	FEEL 構文	領域へのマッピング
53	*e instance of type*	true iff type(e) is type

負数は表 50 で定義される。

表 50: 負数のセマンティクス

文法規則	FEEL 構文	同等の FEEL 構文
26	*-e*	0-*e*

起動は表 51 で定義。起動には、位置引数または名前付き引数を使用する。位置引数の場合は、すべての引数を指定する必要がある。名前付き引数の場合は、記載されていない引数は null にバインドされる。注意、e は、ユーザー定義関数、ユーザー定義の外部関数、または組込み関数である。

表 51: 起動のセマンティクス

文法規則	FEEL	領域へのマッピング	適用性
40, 41, 44	*e*(e_1,..)	*e*(e_1, …)	*e* はマッチングすべき引数がある関数である
40, 41, 42, 43	*e*(n_1:e_1, …)	*e*(n_1:e_1, …)	*e* はマッチングすべきパラメータ名がある関数である

プロパティは、表 52 および表 53 で定義される。**type(e)**が日付と時刻、時間または期間で、**name** がプロパティ名の場合の意味は表 53 に示される。たとえば、FEEL(date and time("03-07-2012Z").year) = **2012**。

表 52:属性の一般的なセマンティクス

文法規則	**FEEL**	**領域へのマッピング**	**適用性**
20	*e.name*	*e."name"*	type(e)はコンテキスト
20	*e.name*	下記参照	type(e)は日付/時刻/期間

表 53: 日付、時刻、期間に特定なセマンティクス

type(*e*)	***e . name***	**name =**
date	結果は日付オブジェクト e の名前付き要素である。有効な名前が右側に表示される。	year, month, day
date and time	結果は時間オブジェクト e の名前付き要素である。有効な名前が右側に表示される。タイムオフセットとタイムゾーンが null となる場合がある	year, month, day, hour, minute, second, time offset, timezone
time	結果は時刻オブジェクト e の名前付き要素である。有効な名前が右側に表示される。タイムオフセットとタイムゾーンが null となる場合がある	hour, minute, second, time offset, timezone
years and months duration	結果は年月期間オブジェクト e の名前付き要素である。有効な名前が右側に表示される。	years, months
days and time duration	結果は時間オブジェクト e の名前付き要素である。有効な名前が右側に表示される。	days, hours, minutes, seconds

リストは表 54 で定義。

表 54:リストのセマンティクス

文法規則	FEEL 構文	領域へのマッピング(スコープ s)	適用性
56	$e_1[e_2]$	$e_1[e_2]$	e_1 はリストであり、e_2 は整数（0 の位数)
56	$e_1[e_2]$	e_1	e_1 はリストではなく、null でもない。そして value(e_2) = 1 である
56	$e_1[e_2]$	i が e1 の中にあって、FEEL(e_2,s')が真である場合、i が中に存在する項目 e のリストである。s'はスコープ s であり、コンテキスト・エントリー（「item」、i）を含む特別な第 1 のコンテキストを用いて、そして i がコンテキストの場合、特別なコンテキストは、i のすべてのコンテキスト・エントリーも含む。	e_1 はリストであり type(FEEL(e2, s'))は論理型である
56	$e_1[e_2]$	［e1］ FEEL（e2、s'）が真であれば、s'はコンテキスト・エントリー（ ″item″、e1）を含む特別な先頭コンテキストを持つスコープ s である、そして、e1 がコンテキストの場合、特別なコンテキストには e1 のすべてのコンテキスト・エントリーも含まれと。 そうでない場合は、[]。	e_1 はリストではない、また null でもない、そして type(FEEL(e_2, s'))は論理型である

表 55：コンテキストのセマンティクス

文法規則	FEEL 構文	領域へのマッピング(スコープ s)
59	$\{ n_1 : e_1,\ n_2 : e_2,\ \cdots\}$	すべての si が、特別な先頭コンテキスト ci を持つ s である様な{ $"n_1"$: FEEL(e_1, s_1), $"n_2"$: FEEL(e_2, s_2), …} は、この結果コンテキストのエントリーのサブセットを含む。c_i に n_j のエントリーが含まれている場合、c_j には n_i のエントリーは含まれない。
	$\{ "n_1" : e_1,\ "n_2" : e_2,\ \cdots\}$	
56	$[e_1,\ e_2,\ \cdots]$	[FEEL(e_1), FEEL(e_2), …]

10.3.2.13 エラー・ハンドリング

組込み関数が、定義された領域外のデータにアクセスしようとした時、関数は適切であれば診断情報を報告または記録し、null を返す。

10.3.3 XML データ

FEEL は、XML データを FEEL セマンティクス領域にマッピングすることにより、FEEL コンテキストで XML データをサポートする。XE(e, p)を XML 要素 e にマッピングする関数となるように、また FEEL コンテキストに対する親 FEEL コンテキスト p になるようにする。その関係は以下の表に定義される。また、XE は別のマッピング関数 XV (v) を使用するが、 XML 値 v を FEEL セマンティクス領域にマップする。

XML 名前空間のセマンティクスはマッピングではサポートされない。例えば、名前空間接頭辞宣言である xmlns：p1 = "http://example.org/foobar"と xmlns：p2 = "http://example.org/foobar"では、XML 名前空間のセマンティクスを使用すると、タグ p1：myElement とタグ p2：myElement は同じ要素となる。しかし、名前空間セマンティクスを持たない XML では、異なる。

10.3.3.1 XML 要素(XE)のためのセマンティック・マッピング

表 56 では、*e* は XML 要素の名前、*a* はその属性の名前、*c* は子要素、*v* は値である。 親コンテキスト p の初期値は空である。

表 56:XML 項目のセマンティクス

XML	p のコンテキスト・エントリー	備考
<e />	"e" : null	空要素→p の null 値のエントリー
<q:e />	"q$e" : null	名前空間は無視される。コロン付きの名前は、正式な識別子に変更される。
<e>v</e>	"e":XV(v)	繰り返しがない「属性のない要素」
<e>v_1</e> <e>v_2</e>	"e": [XV(v_1), XV(v_2)]	繰り返しがある「属性のない要素」
<e a="v"/> <c_1>v_1</c_1> <c_n>v_2</c_n><c_n>v_3</c_n> </e>	"e": { "@a": XV(v), "c_1": XV(v_1), "c_n": [XV(v_2), XV(v_3)] }	属性名の先頭に@が付き、属性または子要素を含む要素→コンテキスト
<e a="v_1">v_2</e>	"e": { "@a": XV(v1), "$content": XV($v_2$) }	v_2は生成された$ content エントリーに含まれる

"e"など p 列のコンテキスト・エントリー内のエントリーにおいて、 null は、文字列キー "e"を持つコンテキスト・エントリーを示し、その値は null となる。コンテキスト・エントリーは、コンテキスト p となり、 p は、XML 要素を含むか、XML 文書自体に対応する。

このマッピングは、名前空間接頭辞を名前空間 IRI に置き換えるものではない。 FEEL では、コンテキスト内の key は明確に識別されるため、名前空間接頭辞で十分である。

10.3.3.2 XML 値（XV）のためのセマンティック・マッピング

XML 文書がスキーマで構文解析された場合、いくつかのアトミック値は文字列型以外のデータ型を持つことがある。 表 57 は、型付き XML 値 v が FEEL にどのようにマッピングされるかを定義している。

表 57:XML 値のセマンティクス

v の型	FEEL セマンティクス領域
number	FEEL(v)
string	FEEL("v")
date	"@a": FEEL(date("v"))
dateTime	"@a": FEEL(date and time("v"))
time	"@a": FEEL(time("v"))
duration	"@a": FEEL(duration("v"))
list, e.g. "v_1 v_2"	[XV(v_1), XV(v_2)]
element	XE(v)

10.3.3.3 XML 例

次のスキーマおよびインスタンスは、次の FEEL と等価である。

10.3.3.3.1 スキーマ

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.example.org"
      targetNamespace="http://www.example.org"
      elementFormDefault="qualified">
  <xsd:element name="Context">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Employee">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="salary" type="xsd:decimal"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Customer" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="loyalty_level" type="xsd:string"/>
              <xsd:element name="credit_limit" type="xsd:decimal"/>
```

```
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

10.3.3.3.2　インスタンス

```
<Context xmlns:tns="http://www.example.org"
      xmlns="http://www.example.org">
  <tns:Employee>
    <tns:salary>13000</tns:salary>
  </tns:Employee>
  <Customer>
    <loyalty_level>gold</loyalty_level>
    <credit_limit>10000</credit_limit>
  </Customer>
  <Customer>
    <loyalty_level>gold</loyalty_level>
    <credit_limit>20000</credit_limit>
  </Customer>
  <Customer>
    <loyalty_level>silver</loyalty_level>
    <credit_limit>5000</credit_limit>
  </Customer>
</Context>
```

10.3.3.3.3 同等の FEEL 囲みコンテキスト

Context		
tns$Employee	tns$salary	13000
Customer	loyalty_level	credit_limit
	gold	10000
	gold	20000
	silver	5000

意思決定モデルが評価されるとき、XML スキーマ要素（項 7.3.2 ItemDefinition メタモデル）といった項目定義によって記述された入力データは、FEEL 領域にマッピングされたケースデータにバインドされる。ケースデータは、XML のように、様々なフォーマットにすることができる。上記のように、ケースデータを同等の囲みコンテキストとして入力することもできる。意思決定ロジックは、*Context. tns $ Employee. tns $ salary* の様な式を使用して、コンテキストのエントリーを参照でき、この式の値は 13000 となる。

10.3.4 組込み関数

相互運用性を高めるために、FEEL には組込み関数ライブラリを含む。組込みの構文とセマンティクスは FEEL に準拠した実装に必須である。

この項の表において、上付き文字は、表の脚注に記載された領域制約を参照している。パラメータが領域外にあるときは常に組込みの結果は null となる。

10.3.4.1 変換関数

FEEL は、多くの型間変換をサポートしている。特に重要なのは、文字列から日付、時間、期間への変換である。日付、時間、または期間にはリテラル表現がない。また、1,000.00 などの書式化された数値は、グループ化記号と小数点記号を指定して文字列から変換する必要がある。

組込み関数は、表 58 に要約されている。最初の列には名前とパラメータが表示される。疑問符（？）はオプションのパラメータを示す。2 番目の列は、パラメータの領域を指定する。パラメータの領域は、下記のひとつが指定される。

- 型、たとえば数値型、文字列型
- any - セマンティクス領域の任意の要素（null も含む）
- null 以外 -セマンティクス領域の任意の要素、null を除く
- データ文字列 - XML Schema Part 2 Datatypes で指定された日付型の語彙空間の文字列値
- 時間文字列 - 以下のいずれか
  XML Schema Part 2 Datatypes で指定された時間型の語彙空間の文字列値。または ISO 8601 で指定されているローカル時刻表現の拡張形式で、その後に文字「@」が続き、その後に IANA Time Zones Database (http://www.iana.org/time-zones)のタイムゾーン識別子である文字列値が続く文字列値。
- 日付時刻文字列 - 上記のような日付文字列で構成された文字列値。オプションとして、"T"とそれに続く時間文字列値。
- 期間文字列 - xs：dayTimeDuration または xs：yearMonthDuration の語彙空間の文字列値。データ型は、XQuery 1.0 および XPath 2.0 データ・モデルによって仕様化されている。

表 58:変換関数のセマンティクス

名前（パラメータ）	パラメータ領域	説明	例
date(from)	date string 日付文字列	日付から変換する	date("2012-12-25") - date("2012-12-24") = duration("P1D")
date(from)	date and time 日付時刻	日付から日付に変換する（時刻要素は null に設定する）	date(date and time("2012-12-25T11:00:00Z")) = date("2012-12-25")
date(year, month, day)	year, month, day are numbers 年、月、日の数値	年、月、日の値から日付を作成する	date(2012, 12, 25) = date("2012-12-25")
date and time(date, time)	date is a date or date time; time is a time 日付は日付か日付時刻。時刻は時刻。	与えられた日付（時間成分は無視する）と与えられた時間から日付時刻を作成する	date and time ("2012-12-24T23:59:00") = date and time (date("2012-12-24"), time("T23:59:00"))

date and time(from)	date time string 日付時刻文字列	日付時刻から日付時刻へ変換する	date and time("2012-12-24T23:59:00") + duration("PT1M") = date and time("2012-12-25T00:00:00")
time(from)	time string 時刻文字列	時刻から時刻へ変換する	time("23:59:00z") + duration("PT2M") = time("00:01:00@Etc/UTC")
time(from)	time, date and time 時刻、日付時刻	時刻から時刻へ変換する(データ要素は無視する)	time(date and time("2012-12-25T11:00:00Z")) = time("11:00:00Z")
time(hour, minute, second, offset)	hour, minute, second, are numbers, offset is a days and time duration, or null 時、分、秒は数値。オフセットは日付時刻期間または null	与えられた要素値から時刻を作成する	time(“T23:59:00z") = time(23, 59, 0, duration(“PT0H"))
number(from, grouping separator, decimal separator)	string1, string, string 文字列 1、文字列、文字列	数値から変換する	number("1 000,0", " ", ",") = number("1,000.0", ",", ".")
string(from)	non-null null 以外	文字列から文字へ変換する	string(1.1) = "1.1" string(null) = null
duration(from)	duration string 期間文字列	期間または年月期間から期間または年月期間へ変換する	date and time("2012-12-24T23:59:00") - date and time("2012-12-22T03:45:00")

			= duration("P2DT20H14M") duration("P2Y2M") = duration("P26M")
years and months duration(from, to)	both are date and time 両者とも日付時刻	from と to の間の年月を返す	years and months duration(date("2011-12-22"), date("2013-08-24")) = duration("P1Y8M")

1. グループ化記号はスペース（' '）、カンマ（','）、ピリオド（'.'）、または null のいずれかでなければならない。
   小数点記号はピリオド、コンマ、または null のいずれかでなければならない、但し両方が null でない限りグループ化記号と同じであってはならない。
   from は文法規則 37 に合致しなければならない、 グループ化セパレータのすべての出現を除去した後、もしあれば小数点記号を変更した後、存在する場合、存在する場合にはピリオドに変更する。

10.3.4.2 論理型関数

表 59 に論理型関数を定義する。

表 59:論理型関数のセマンティクス

名前（パラメータ）	パラメータ領域	説明	例
not(negand)	論理型	論理否定	not(true) = false not(null) = null

10.3.4.3 文字列関数

表 60 に文字列関数を定義する。

表 60:文字列関数のセマンティクス

名前（パラメータ）	パラメータ領域	説明	例
substring(string, start position,	文字列、数値 1	文字列中の開始位置から始まる長さ	substring("foobar",3) = "obar"

length?)		分の（またはすべての）文字を返す。 最初 1、最後は-1	substring("foobar",3,3) = "oba" substring("foobar", -2, 1) = "a"
string length(string)	文字列リターン 1	文字列の長さ	string length("foo") = 3
upper case(string)	文字列	大文字の文字列を返す	upper case("aBc4") = "ABC4"
lower case(string)	文字列	小文字の文字列を返す	lower case("aBc4") = "abc4"
substring before (string, match)	文字列、文字列	文字列中の一致文字の前にある部分文字列を返す	substring before("foobar", "bar") = "foo" substring before("foobar", "xyz") = ""
substring after (string, match)	文字列、文字列	文字列中の一致文字列の後の部分文字列を返す	substring after("foobar", "ob") = "ar" substring after("", "a") = ""
replace(input, pattern, replacement, flags?)	文字列 2	正規表現のパターンマッチングと置換	replace("abcd", "(ab)|(a)", "[1=$1][2=$2]") = "[1=ab][2=]cd"
contains(string, match)	文字列	文字列に一致する部分が含まれているか？	contains("foobar", "of") = false
starts with(string, match)	文字列	文字列は一致する文字で始まるか？	starts with("foobar", "fo") = true
ends with(string, match)	文字列	文字列は一致する文字で終わるか？	ends with("foobar", "r") = true
matches(input, pattern,flags?)	文字列 2	入力は正規表現パターンと一致するか？	matches("foobar", "^fo*b") = true

1. *start position*は[-L..L]の範囲内の 0 以外の整数（小数点以下は 0 桁）でなければならない。L は文字列の長さである。長さは[1..E]の範囲内でなければならない。E は L - *start position* が正の場合は開始位置、それ以外は -start 位置となる。
2. *pattern*、*replacement*、*flags*は、XQuery 1.0 と XPath 2.0 の 7.6 節で「関数と演算子」で規定されている構文に適合しなければならない。XPath がエラーとなる場合、FEEL の結果は null となることに注意する。

10.3.4.4 リスト関数

表 61 にリスト関数を定義する。

表 61:リスト関数のセマンティクス

名前（パラメータ）	**パラメータ領域**	**説明**	**例**
list contains(list, element)	リスト、**null** を含む任意のセマンティクス領域の要素	リストに要素が含まれているか？	list contains([1,2,3], 2) = true
count(list)	リスト	リストのサイズが返る	count([1,2,3]) = 3
min(list) min(c1,…, cN), N >1 max(list) max(c1,…, cN), N >1	比較可能なアイテム（のリスト）	最小値（最大）を返す	min([1,2,3]) = 1 min(1,2,3) = 1 max([1,2,3]) = 3 max(1,2,3) = 3
sum(list) sum(n1,…, nN), N >1	数値（のリスト）	数値の合計を返す	sum([1,2,3]) = 6 sum(1,2,3) = 6
mean(list) mean(n1,…, nN), N >1	数値（のリスト）	数値の算術平均（平均）を返す	mean([1,2,3]) = 2 mean(1,2,3) = 2
and(list) and(b1,…, bN), N >1	論理型項目（のリスト）	項目が false の場合は false を返す、そうでない場合は true を返す	and([false,null,true]) = false and(false,null,true) =

			false and([]) = true and(0) = null
or(list) or(b1,…, bN), N >1	論理型項目（のリスト）	項目が true の場合は true を返す、そうでない場合は false を返す	or([false,null,true]) = true or(false,null,true) = true or([]) = false or(0) = null
sublist(list, start position, length?)	リスト、数値[1]、数値[2]	*list [start position]* から始まる *length*（またはすべて）分の要素のリストを返す。最初は 1、最後は-1	sublist([1,2,3], 1, 2) = [2]
append(list, item…)	リスト、null を含む任意の要素	項目を追加して新しいリストを返す	append([1], 2, 3) = [1,2,3]
concatenate(list…)	リスト	引数を連結した新しいリストを返す	concatenate([1,2],[3]) = [1,2,3]
insert before(list, position, newItem)	リスト、数値[1]、null を含む任意の要素	*position* に *newItem* を挿入して新しいリストを返す	insert before([1,3],1,2) = [1,2,3]
remove(list, position)	リスト、数値[1]	*position* が削除されたアイテムのリスト	remove([1,2,3], 2) = [1,3]
reverse(list)	リスト	リストを逆にする	reverse([1,2,3]) = [3,2,1]
index of(list, match)	リスト、null を含む任意の要素	一致を含むリスト位置の昇順リストを返す	index of([1,2,3,2],2) = [2,4]
union(list…)	リスト	重複を削除し連結する	union([1,2],[2,3]) = [1,2,3]

distinct values(list)	リスト	重複を削除する	distinct values([1,2,3,2,1] = [1,2,3]
flatten(list)	リスト	ネストされたリストをフラットにする	flatten([[1,2],[[3]], 4]) = [1,2,3,4]

1. *position* は、[-L..L]の範囲内の 0 以外の整数（小数点以下は 0 桁）でなければならない。L はリストの長さである。
2. *length* は[1..E]の範囲内になければならない。ここで E は L である。開始位置が正の場合は開始位置、それ以外の場合は-*start position* である。

10.3.4.5 数字関数

表 62 に数字関数を定義する。

表 62:数字関数のセマンティクス

名前（パラメータ）	パラメータ領域	説明	例
decimal(n, scale)	数値、数値 1	与えられたスケールで n を返す	decimal(1/3, 2) = .33 decimal(1.5, 0) = 2 decimal(2.5, 0) = 2
floor(n)	数値	最大の整数<= n を返す	floor(1.5) = 1 floor(-1.5) = -2
ceiling(n)	数値	最小の整数<= n を返す	ceiling(1.5) = 2 ceiling(-1.5) = -1

1. スケールは[-6111..6176]の範囲内である。

10.3.4.6 デシジョンテーブル

デシジョンテーブル関数のパラメータは、8.2「表記法」で定義された図に対する名前付きセルに対応する（8.3 メタモデルのメタモデルにも対応する）。また、10.3.2.8 デシジョンテーブルで説明したように、いくつかのパラメータにはシングルクォート(リテラル)単項式テストが含まれる。デシジョンテーブルのセマンティクスは、これらの

単項式テストを単純な式で構成する、その単純な式は意味空間にマッピングされ、ルール・マッチ、ルール・ヒットの後にデシジョンテーブル出力が生成される。これらの生成物を表 63 に定義する。

表 63:デシジョンテーブルのセマンティクス

パラメータ名(*はオプションを意味する)	**領域**
input expression list	N> = 0 の入力式のリスト(表示順)
input values list*	入力式に対応する N 個の入力値のリスト。 各リスト要素は単項式テストリテラルである(下記参照)。
outputs*	名前(文法規則 27 に一致する文字列)または M> 0 の名前のリスト
output values*	出力がリストの場合、出力値はリストの値のリストで、出力毎に 1 つのリストとなる。それ以外の場合は、出力値は出力の値のリストとなる。 それぞれの値は文字列である。
rule list	R> 0 のルールリスト。N 個の入力エントリーの後に M 個の出力エントリーが続くリストがルールである。入力エントリーは単項式テストリテラルである。出力エントリーは式である。
hit policy*	"U", "A", "P", "F", "R", "O", "C", "C+", "C#", "C<", "C>"の内のひとつ(デフォルトは "U")
default output value*	出力がリストの場合、デフォルトの出力値は「出力と出力値」で構成されるエントリーを持つコンテキストである。 リストでない場合、デフォルト出力値は「出力値の内のひとつ」となる

単項式テスト(文法規則 17)は、入力値と入力項目の両方を表すために使用される。 入力式 *e* は、入力エントリー *t*(オプションの入力値 v を有する)を満たすと言われ、*t* の構文に応じて、次のようになる:

- 文法規則 17.a: FEEL(*e in (t)*)=**true**
- 文法規則 17.b: FEEL(*e in (t)*)=**false**
- 文法規則 17.c when v is not provided: **e != null**
- 文法規則 17.c when v is provided: FEEL(*e in (v)*)=**true**

入力項目 $t_1, t_2, \cdots, t_N$ を持つルールは、e_i が 1..N のすべての i に対して t_i（オプションの入力値 v_i）を満たす場合、入力式リスト[e_1、e_2、...、e_N]と一致すると言われる（オプションの入力値は list [v_1、v_2、... v_N]）。

マッチングするとルールがヒットする、そして、ヒット・ポリシーに従って、一致したルールの出力値をデシジョンテーブルの結果に含める。ヒットごとに 1 つの出力値が得られる（複数の出力は単一のコンテキスト値に集められる）。したがって、複数のヒットには集合が必要となる。
ヒット・ポリシーは、次の太字のポリシー名の 1 文字目を使用して指定する。
シングルヒット・ポリシー：

- **Unique** - 1 つのルールだけがマッチングできる
- **Any** - 複数のルールがマッチングできる、但しそれらはすべて同じ出力となる
- **Priority** - 優先度を考慮し、複数のルールを異なる出力でマッチングできる。指定された出力値リストの中から、最初に見つかった出力が返される
- **First** -ルールの順序で最初にマッチングしたものを返す

複数ヒット・ポリシー

- **Collect** - 出力のリストを任意の順序で返す
- **Rule order** -出力のリストをルールの順序で返す
- **Output order** - 出力のリストを出力値の順序で返す

収集ポリシーでは、オプションで次のような集計を指定することもできる：

- **C+** - 出力の合計を返す
- **C#** - 出力数を返す
- **C<** - 有効な値の内で最小を返す
- **C>** - 有効な値の内で最大を返す

集計は、10.3.4.4 リスト関数で指定された以下の組込み関数を使用する：sum, count, minimum, maximum。 複雑さを軽減するために、複合出力を含むデシジョンテーブルでは集計をサポートせず、次のヒット・ポリシーのみをサポートする：*Unique*、 *Any*、*Priority*、 *First*、 *Collect without operator*、および *Rule order*。

デシジョンテーブルは、入力値に対してヒットしないこともある。この場合、結果はデフォルトの出力値で与えられる、但しデフォルトの出力値が指定されていない場合は **null** となる。なお、完全なデシジョンテーブルとするには、デフォルトの出力値を指定してはならない。

デシジョンテーブル呼び出し DTI のセマンティクスを以下に示す：

1. ルールリストの各ルールは入力式リストとマッチされる。マッチ順は不定。
2. ルールがマッチしない場合。
   a. デフォルト出力 d を指定されている場合 DTI=FEEL(d)
   b. そうでなければ DTI=null
3. そうでなければ、入力式とマッチしたルールのサブリスト m を評価する。もし、ヒット・ポリシーが"First"か"Rule order"の場合、ルールで定義した順番で m をソートする。
   a. o を出力式のリストとする、なお、インデックス i の式は、ルール m[i]の出力式である。単一出力となるデシジョンテーブルのルールの出力式は、単にルールの出力項目である。複数出力となるデシジョンテーブルの出力式は、出力名とルールに応じた出力エントリーから構成されたエントリーを含むコンテキストである。ヒットしたポリシーが「Output order」の場合、デシジョンテーブルは単一出力でなければならず、o は出力値の順序と一致した並びとなる s。
   b. 複数ヒット・ポリシーが指定されている場合は DTI=FEEL(aggregation(o))となる、なお、集計は、10.3.4.4 リスト関数で明記された組込み関数 sum、count、minimum から 1 個選択される。
   c. そうでなければ、DTI=FEEL(o[1])となる。

10.3.4.7 ソート

リストのソートには順序付け関数を用いる。例えば、

sort(list: [3,1,4,5,2], precedes: function(x,y) x < y) = [1,2,3,4,5]

表 64：ソート関数のセマンティクス

パラメータ名(*はオプション)	領域
list	任意の項目リスト。Null に注意。
precedes	リスト項目の各組で定義された 2 個の引数の論理型関数

10.4 意思決定サービスの実行セマンティクス

意思決定サービスの実行セマンティクスは FEEL で提供される、その意思決定サービスは

FEEL が式言語である意思決定モデルで定義される。意思決定サービスは **FEEL** 関数と同等である。パラメータは意思決定サービスの入力であり、ロジックは意思決定サービスの意思決定と知識要求線から構成されるコンテキストである。
意思決定モデルの各 FEEL 式には実行セマンティクスがある。LiteralExpression(FEEL テキスト) セマンティクスは 10.3 で定義されている。 10.2.2 FEEL に記述された囲み式は、FEEL テキストにマップすることができ、実行セマンティクスも持つ。
DecisionService が次の４つのリストによって定義されている点を思い出すこと。inputData、inputDecisions、outputDecisions、 encapsulatedDecisions。このリストは独立している訳では無いので、 全てを指定する必要はない。たとえば、各要求された outputDecisions の意思決定(直接と間接) は、 encapsulatedDecision となるか、inputDecision となるか、あるいは inputDecision によって要求されるかのいずれかが必要である。以下の記述を簡単にするために、4 つのリストすべてが正しく完全に指定されているものとみなす。

DecisionService に FEEL 関数 F にマッピングすることにより、実行セマンティクスが提供される。D を入力データ $id_1, id_2, \ldots$、入力意思決定 $do_1, do_2, \ldots.$、カプセル化された意思決定 $de_1, de_2, \ldots$, 出力意思決定 do1, do2, ...、を持つ DecisionService とする。　各入力データ id_i は、$id_i.n.$ という名前を有する。意思決定には BusinessKnowledgeModels bkm1、bkm2、....が必要となることがある。BKM (Business Knowledge Model：ビジネス知識モデル) の名前は bkmi.n と encapsulatedLogic bkmi.f である。FEEL 関数 F の構文は $function(id_1.n, id_2.n, \ldots, di_1.n, di_2.n, \ldots)$である。C.result、C はコンテキストである。

$\{$
$bkm_1.n : bkm_1.f,\ bkm_2.n : bkm_2.f, \ldots,$
$de_1.n : de_1.e,\ de_2.n : de_2.e, \ldots,$
$result: \{do_1.n : do_1.e,\ do_2.n : do_2{:}e, \ldots\}$
$\},$

上記の、 bkm_i, de_i　do_i は部分的に要求順となっている(たとえば、要求された意思決定のためのコンテキスト・エントリーは、それを要求した意思決定の前に置く)。

D の実行セマンティクスは FEEL (F) である。FEEL 意味領域にある値付きで起動された場合、関数入力データと入力意思決定は、すべての出力意思決定の出力値から構成されたコンテキストを返す。

XML 要素を、10.3.3 章「XML データ」で指定された FEEL 意味領域にマッピングしなけ

ればならない。そうでなければ、入出力データ値の構文と FEEL へのマッピングの詳細は未定義となる。

10.5　メタモデル

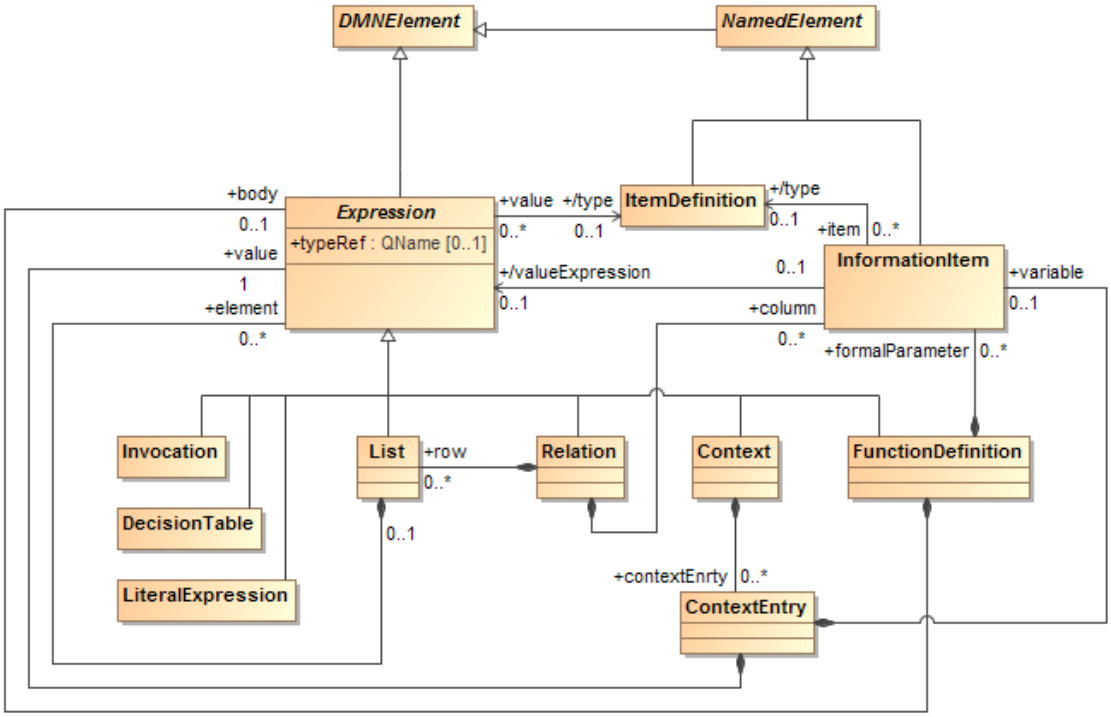


図 67: Expression クラス図

Expression クラスは FEEL で導入された、名前が Context FunctionDefinition Relation List である４種類の新しい囲み式をサポートするため拡張される。

囲み式は、標準的な図形表現[AF1]（7.2.1 章「式」および 10.2.1 章「囲み式」参照）を持つ式である。FEEL のコンテキスト、関数定義、関係およびリストは、おのおの Context、FunctionDefinition、 Relation 、List 要素としてモデル化され、可能な限り囲み式として表現されることが望ましい。それは、これらがトップレベルの式である場合、LiteralExpression のインスタンスに他の Expression 要素を含めることができないためである。

10.5.1　　Context メタモデル

Context は、ContextEntry のインスタンスである contextEntrys の任意の数からでき

ている。

Context 要素は、**囲みコンテキスト**（10.2.1.4 章「囲みコンテキスト」）として図形表現される。FEEL セマンティクス（文法規則 59 と 10.3.2.6 章「コンテキスト」）は可能な限り Context 要素としてモデル化されることが望ましい。Context は、すべての属性とモデル・アソシエーションを Expression から継承する。
表 65 に、Context 要素で追加される属性とモデル・アソシエーションを示す。

表 65: Context の属性とモデル・アソシエーション

属性	**説明**
contextEntry: ContextEntry [*]	この属性は Context を構成する ContextEntry のインスタンス

10.5.2 ContextEntry メタモデル

コンテキストが Context 要素としてモデル化される場合、FEEL *context entries* をモデル化するために、ContextEntry クラスが使われる。

ContextEntry のインスタンスは、オプションの変数で構成される。その変数は、InformationItem 要素である。その要素名は *context entry* では、value の、key である。value は、*context entry* 内の *expression* をモデル化する Expression のインスタンスである。
表 66 に ContextEntry 要素の属性とモデル・アソシエーションを示す。

表 66: ContextEntry の属性とモデル・アソシエーション

属性	**説明**
variable: InformationItem [0..1]	ContextEntry に含まれる InformationItem のインスタンスで、その name はモデル化されたコンテキスト・エントリーの key である。
value: Expression	ContextEntry 含まれる式である Expression のインスタンス

10.5.3 FunctionDefinition メタモデル

FunctionDefinition は formalParameters と body を持つ。FunctionDefinition 要素

は項目 10.2.1.7「囲み関数」で述べられた**囲み関数**として図形表現される。FEEL *function definition*（文法規則 57 と項目 10.3.2.12「セマンティック・マッピング」）は可能な限り FunctionDefinition 要素としてモデル化されるのが望ましい。

FunctionDefinition は全ての属性とモデル･アソシエーションを Expression から継承する。表 67 に、FunctionDefinition 要素で追加される属性とモデル・アソシエーションを示す。

表 67: FunctionDefinition の属性とモデル・アソシエーション

属性	**説明**
FormalParameter: InformationItem [*]	この属性は Context のパラメータである InformationItem のインスタンスの属性をリストする。
body: Expression [0..1]	本体が FunctionDefinition にある Expression のインスタンス

10.5.4 List メタモデル

List は単に Expressions のインスタンスのリストである。 List 要素は囲みリストとして項目 10.2.1.5「囲みリスト」で述べられた**囲みリスト**として図形表示される。 A FEEL *リスト*（文法規則 56 と 項目 10.3.2.12「セマンティクス・マッピング」）は、可能な限り List 要素としてモデル化されることが望ましい。

List は全ての属性とモデル･アソシエーションを Expression から継承する。表 68 に、List 要素で追加される属性とモデル・アソシエーションを示す。

属性	**説明**
element: Expression [*]	この属性はこの List の要素である Expression のインスタンスをリストする

10.5.5 Relation メタモデル

Relation は、類似コンテキストのリストのための便利な簡略表記である。Relation は ContextEntrys を繰り返す代わりに column を持ち、List は各 row に使用される。各行(row)は各列(column)の値に対して List の expression をひとつ持つ。

Relation は Expression の全ての属性とモデル・アソシエーションを継承する。表 69 に Relation の要素で追加される属性とモデル・アソシエーションを示す。

表 69: Relation の属性とモデル・アソシエーション

属性	説明
row: List [*]	この属性はこの Relation の行を構成する List のインスタンスをリストする
column: InformationItem [*]	この属性はこの Relation の列を定義した InformationItem のインスタンスをリストする

10.6 例

FEEL を早く概説する良い方法としては例を示す。
FEEL 式は他の FEEL 式を名前で参照できる。名前の付いた式はコンテキストに含まれる。式はスコープ内で評価される。スコープとは、名前を解決するためのコンテキストのリストである。評価の結果は、FEEL 意味領域の要素である。

10.6.1 コンテキスト

図 68 は、例に使用される囲みコンテキストを示している。このようなコンテキストは、いくつかの方法で現れる。これは、単一または複合意思決定における意思決定ロジックの一部として現れる。または、10.4 章「実行セマンティクス」で定義されている DRG の一部と同等のコンテキストとして現れる。*申込み者*、*要求された商品*、および*信用履歴*は入力データのインスタンスである、*月収*と*月次の費用*は副次的な意思決定であり、*PMT* はビジネス知識モデルである。

申込み者	年齢	51	
	婚姻状況	"M"	
	既顧客	false	
	月次	月収	10000
		返済	2500
		費用	3000
要求された商品	商品タイプ	"STANDARD LOAN"	
	レート	0.25	
	期間	36	
	合計	100000.00	
月収	applicant.monthly.income		
月次支出	applicant.monthly.repayments, applicant.monthly.expenses		
信用履歴	記録日	事象	負担
	date("2008-03-12")	"home mortgage"	100
	date("2011-04-01")	"foreclosure warning"	150
PMT	(rate, term, amount)		
	(amount *rate/12) / (1 - (1 + rate/12)**-term)		

図 68:コンテキスト例

注意：6 つのトップレベルのコンテキスト・エントリーがあり、6 行で表現されている。「申込み者」と名付けられたコンテキスト・エントリーの値は、それ自体がコンテキストであり、また「月次」と名付けられたコンテキスト・エントリーの値もそれ自体がコンテキストである。
「月次支出」と名付けられたコンテキスト・エントリーの値はリストであり、「信用履歴」と名付けられたコンテキスト・エントリーの値は関係（relation）となる。 すなわち、コンテキストのリストであり、1 行にひとつのコンテキストが入る。また、「PMT」と名付けられたコンテキスト・エントリーの値は、パラメータ 「rate」、 「term」、および 「amount」を持つ関数である。

下記の例では、上記のコンテキストを使用している。それぞれの例には、等価な FEEL 式の組が水平線で区切られている。どちらも、意味領域の同じ要素を示している。2 番目の式、「answer」はリテラル値である。

10.6.2　　計算

```
monthly income * 12
```

120000

コンテキストでは、月収を applicationant.monthly.income と定義している。また、10,000 であるコンテキストでも定義している。 月収の 12 倍は 120,000 である。

10.6.3　If, In

```
if applicant.maritalStatus in ("M","S") then "valid" else "not valid"
```

"valid"

in テストでは、左辺の式が、右辺の値または範囲のリストを満たす場合、if 式は then 式の値を返す。さもなければ、else 式の値が返される。

10.6.4　リストのエントリー合計

```
sum(monthly outgoings)
```

5500

月次支出は、[applicant.monthly.repayments、applicant.monthly.expenses]のリスト、または[2500,3000]のリストとして計算される。ただし、角括弧は、囲みコンテキストに書かなくても良い。

10.6.5 ユーザー定義 PMT 関数の起動

コンテキストで定義された PMT 関数は、所与の金利、月数、融資額から月次支払額を計算する。

```
PMT(requested product . rate,
requested product . term,
requested product . amount)
```

3975.982590125562

関数は、関数名の後ろにカッコで囲まれた引数リストのあるテキストを用いて起動される。引数はコンテキストに登録され、0.25、36、100,000 である。

10.6.6 クレジット利用履歴の重みの合計

```
sum(credit history[record date > date("2011-01-01")].weight)
```

150

これは複雑な「一行」で、これを使って部分式を構成するのに役立つ。

- built-in: sum
  - ○path expression ending in .weight
    - filter: [record date > date("2011-01-01")]
      - name resolved in context: credit history

リスト式に続く角括弧で囲まれた式は、リストをフィルターリングする。信用履歴は、コンテキストにおいて relation として登録されている。relation とは類似したコンテキストのリストである。relation 内の最後の要素のみがフィルターを満足する。最初の要素は古すぎる。.weight で終わるパス式は、フィルターが満足したコンテキストまたはコンテキストのリストからエントリー値を抜き出す。与信履歴の最後の要素の重みは 150 である。これがフィルターを満足する唯一の要素なので、合計も 150 となる。

10.6.7 信用履歴から破産状態を判定する

```
some ch in credit history satisfies ch.event = "bankruptcy"
```

false

ある式は、リストまたは relation の中のからテストを満たす要素が最低１つはあるのかどうかを判定する。このコンテキストでは、信用履歴に破産状態はない。

11 DMN 事例

11.1 イントロダクション

この章では、**BPMN**でモデル化された簡単なビジネスプロセスにおける意思決定のモデル化と実行についての**DMN**の使用例を示す。それはビジネスプロセス・マネジメントシステムから意思決定サービスと呼ばれる自動化される意思決定を含む。

11.2 ビジネスプロセス・モデル

図69は、**BPMN2.0**でモデル化される、ローン起案向けの簡単なプロセスを示す。このローン申込みとして処理を行うプロセスは、事案で必要とされる場合に限り、信用調査会社からデータを取得する。そして申込みを受諾すべきか、謝絶すべきか、または人的調査を参照するかを自動的に判定する。参照することになれば申込み者から詳細文書を収集し、クレジット対応係が事案を裁定する。このプロセスは下記の要素から成る。

●**Collect application data（申込みデータを収集する）**タスクは、Requested product（要求された商品）とApplicant（申込み者）を記述するデータを収集する（例えば、オンライン申込み様式から）。

● Decide **bureau Strategy（信用調査戦略**を判定する）タスクは、判定サービスを呼び出し、Requested product（要求された商品）とApplicant data（申込み者データ）を渡す。そのサービスは、Strategy（戦略）とBureau call type（信用調査会社呼び出し型）のふたつの判定を返す。

● **gateway（ゲートウェイ）**は、Strategy（戦略）の値を活用して、Decline application（申込みを謝絶する）、Collect bureau data（信用調査結果を収集する）、またはDecide routing（判定ルーティング）の分岐に送る。

●**Collect bureau data（信用データを収集する）**タスクは、Bureau call type（信用調査会社呼び出し型）の判定に従って信用調査会社からデータを収集し、それから事案をDecide routing（判定ルーティング）に渡す。

●**Decide routing（判定ルーティング）**タスクは、判定サービスを呼び出し、Requested product（要求された商品）、Applicant data（申込み者データ）、そしてBureau data（信用調査データ）（Collect bureau data（信用調査データを収集する）タスクが実施されない場合はBureau data（信用調査データ）をNULLを設定する。このサービスは単一な判定、すなわちRouting（ルーティング）を返す。

●**gateway（ゲートウェイ）**は、Routing（ルーティング）の値を活用して、Accept application（申込みを受諾する）、Review application（申込みを見直す）、またはDecline application（申

込みを謝絶する）というそれぞれのルーティングの分岐に送る。

●**Collect documents（文書を収集する）**タスクは、申込みの詳細文書を申込み者に要請し、それをアップロードする。

●**Review application（申込みを見直す）**タスクは、クレジット対応係に事案を見直し、受諾すべきか謝絶すべきかを判定することを割り当てる。

●**gateway（ゲートウェイ）**は、クレジット対応係のAdjudication（裁定結果）を活用して、Accept application（申込みを受諾する）、またはDecline application（申込みを謝絶する）というそれぞれへのルーティングの分岐に渡す。

●**Accept application（申込みを受諾する）**タスクは、申込み者に、自分の申込みが受諾され、要求された商品の成立を通知する。

●**Decline application（申込みを謝絶する）**タスクは、申込み者に、自分の申込みが謝絶されたことを通知する。

この事例は、最初のふたつの判定ポイント（意思決定サービスへの呼び出しによって自動化される）が**BPMN2.0**でビジネスルール・タスクとして表現されること、そして、3番目の判定ポイント（人的な意思決定）がユーザータスクとして表現されることに留意のこと。

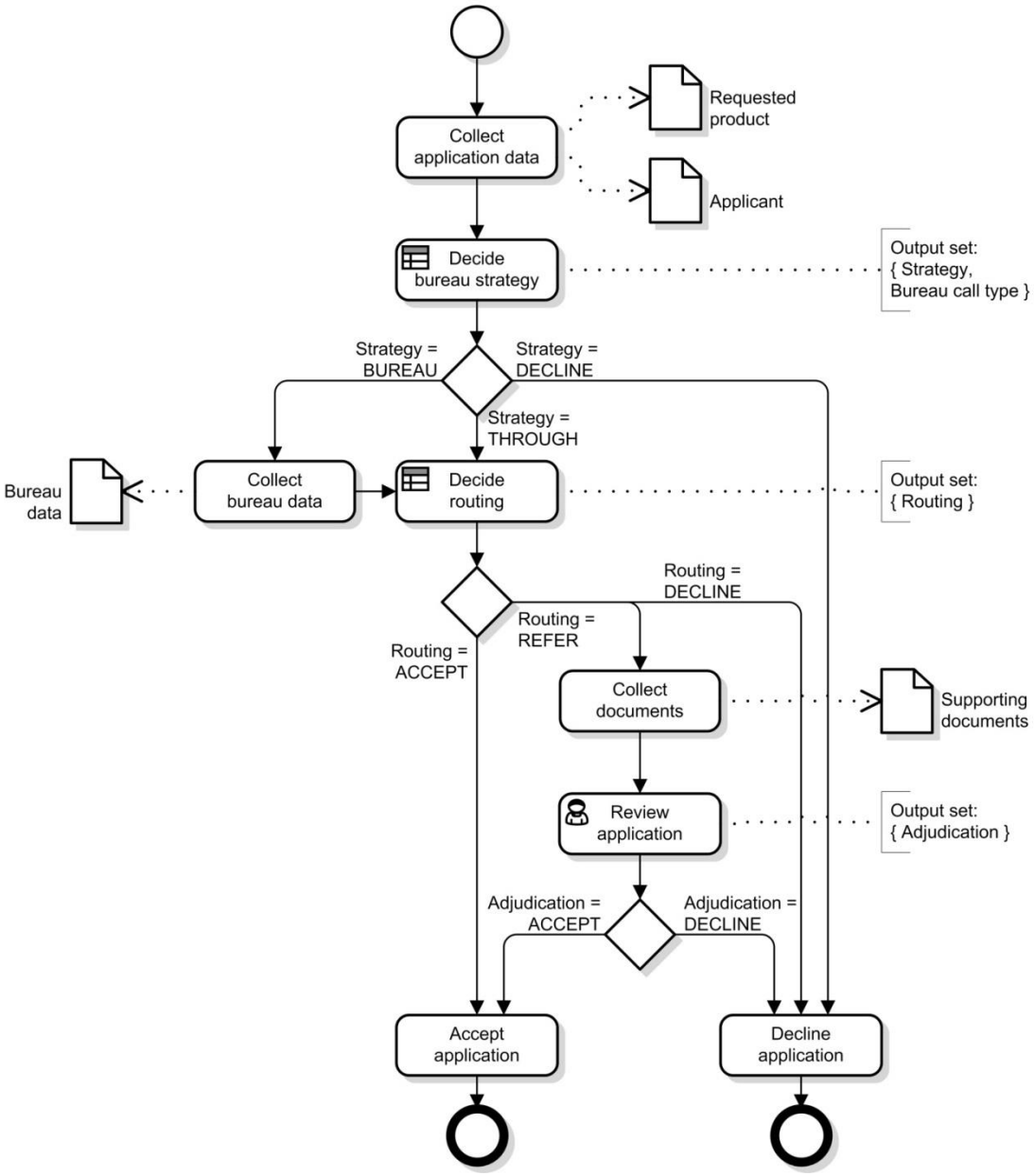


図69:ビジネスプロセスの例

11.3 意思決定要求レベル

図70は、このビジネスプロセスにおける意思決定のすべてのDRDを示す。意思決定のためのインプットデータには、4つのソース（Requested product、Applicant data、Bureau data、Supporting documents）と、意思決定の結果がこのビジネスプロセスで使用される判定には、4

種類（Strategy、Bureau call type、Routing、Adjudication）がある。これらのふたつのカテゴリーカテゴリー（インプットデータと意思決定）の間には、リスクの評価、購入容易性、適格性からなる途中段階の意思決定がある。このDRDの記法上の特徴は下記を含む。

●自動化された意思決定と人的意思決定の両方をカバーする。
●ある種の判定（例えば、Pre-bureau risk category）とインプットデータ（例えば、Applicant data）は、複数の判定を必要とする。つまり情報要求線はツリー状でなくネットワーク状になる。
●ビジネス知識モデル（Affordability calculationを参照のこと）は複数の判定によって起動されることがある。
●ビジネス知識モデル（Credit contingency factorを参照のこと）は他のビジネス知識モデルに起動されることがある。
●ある種の意思決定は関連するビジネス知識モデルをもたない。
●知識ソースは複数の意思決定やビジネス知識モデルに根拠を提供できる。

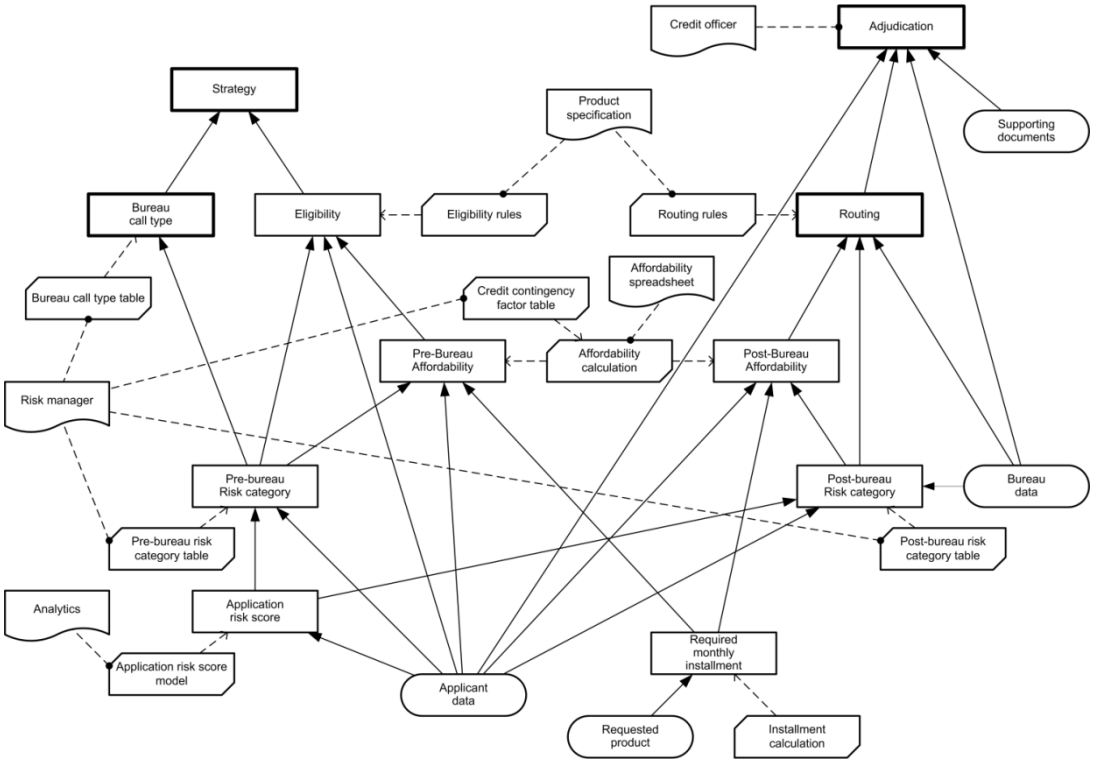


図70;すべて自動化された意思決定によるDRD

3つの意思決定ポイントに対する、分離した（しかし、重複しない）DRDを描くことはより都合が良いと考えられる。
●図71は、Strategyの判定とBureau call typeの判定のサブグラフの要求にあるような、Decide

bureau strategyタスクの意思決定ポイントに必要なDRDを示す。これらの意思決定は、この時点で呼び出される意思決定サービスでのカプセル化を通して自動化される意思決定であり、そのため独自に完結されるべきロジックを必要とする。

●図72は、判定ルーティングの意思決定のサブグラフ要求にあるような、ルーティングを判定するタスクの意思決定ポイントのためのDRDを示す。これらの意思決定も、意思決定サービスによって自動化される意思決定であり、そのため独自に完結されるべきロジックを必要とする。ある種の要素が図71と図72に表れていることに留意する。

●図73は、Adjudication decisionのサブグラフの要求にあるような、申込みを見直すタスクの意思決定ポイントのためのDRDを示す。この意思決定は、人的裁定であり、意思決定ロジックについての関連する仕様をもたないが、DRDが示す通り、クレジット対応係は、申込みの詳細文書を含む事案データとともに自動化されたRouting判定の結果を考慮する。（Routing判定の要求サブグラフはこのDRDでは隠れている。）

図70、図71、図72、そして図73の4つのDRDはすべて同じDRGのビューである。

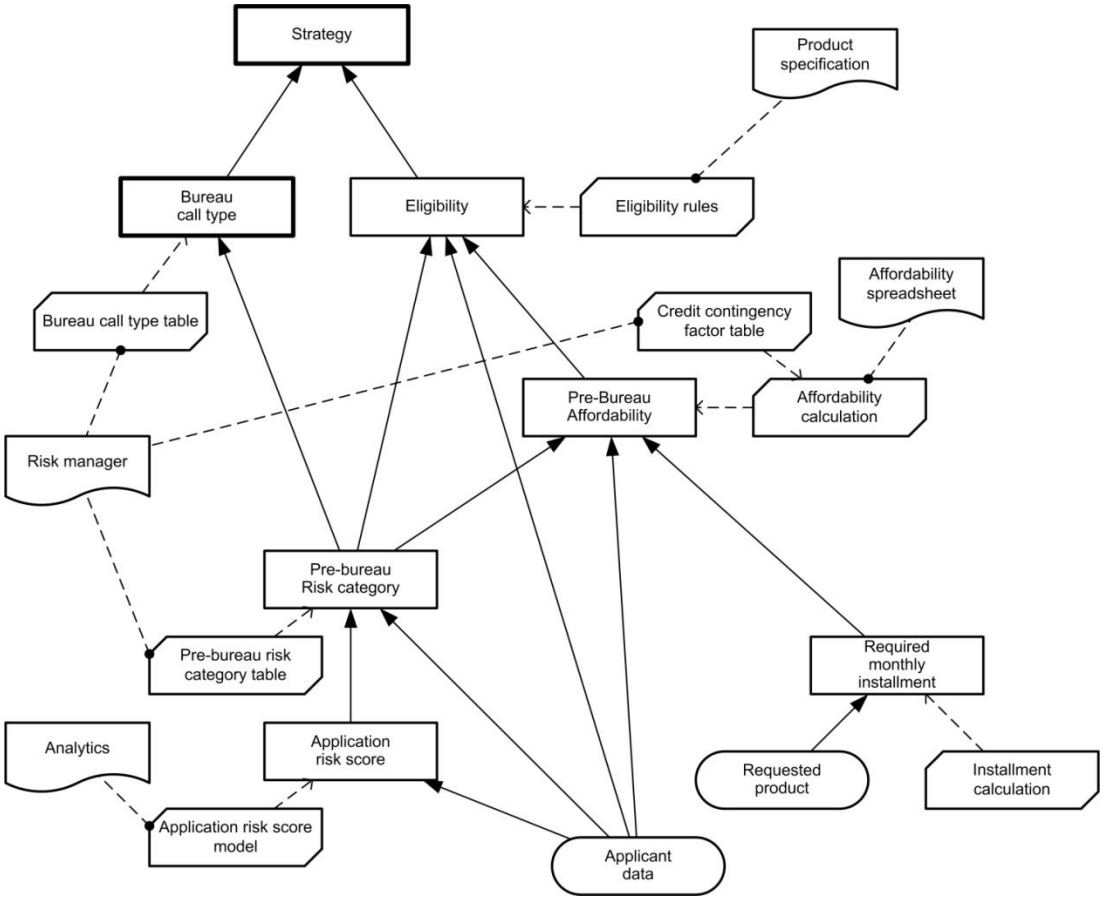


図71：信用調査戦略の意思決定ポイントを判定するDRD

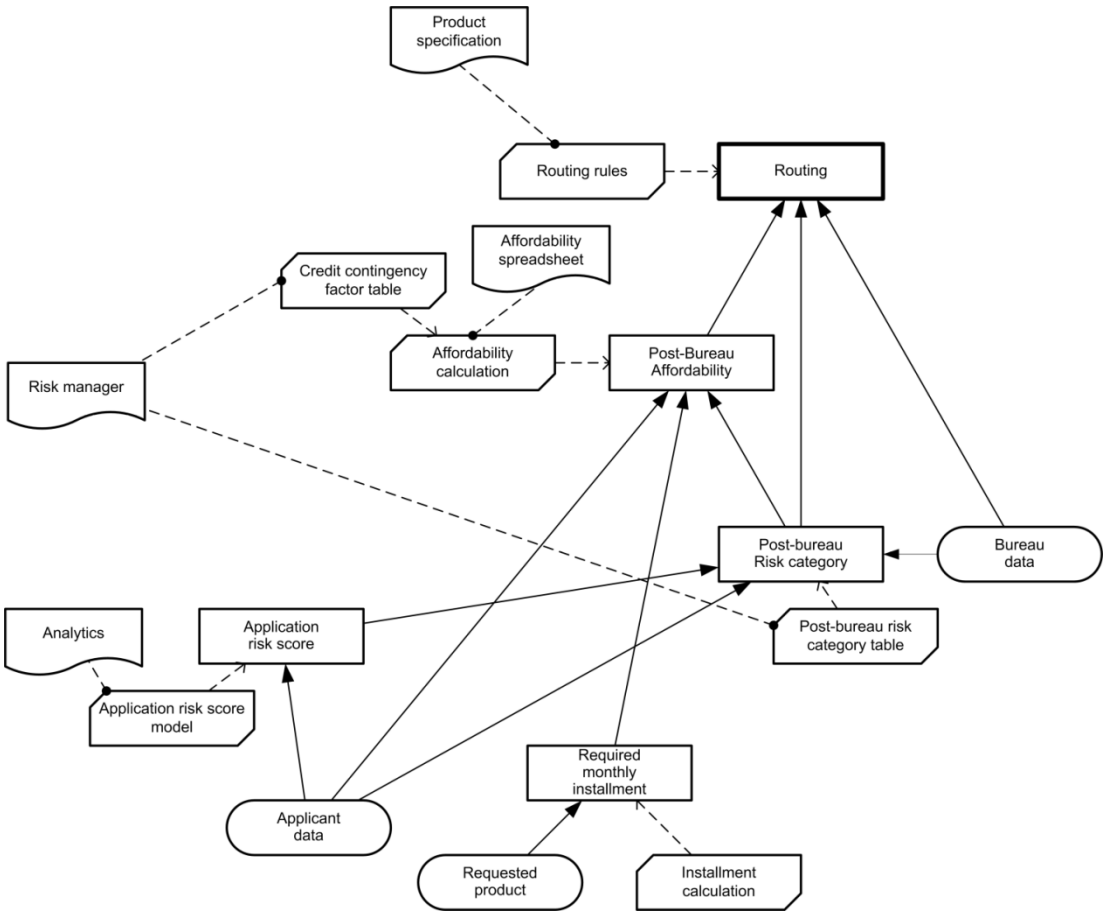


図72:ルーティングの意思決定ポイントを判定するDRD

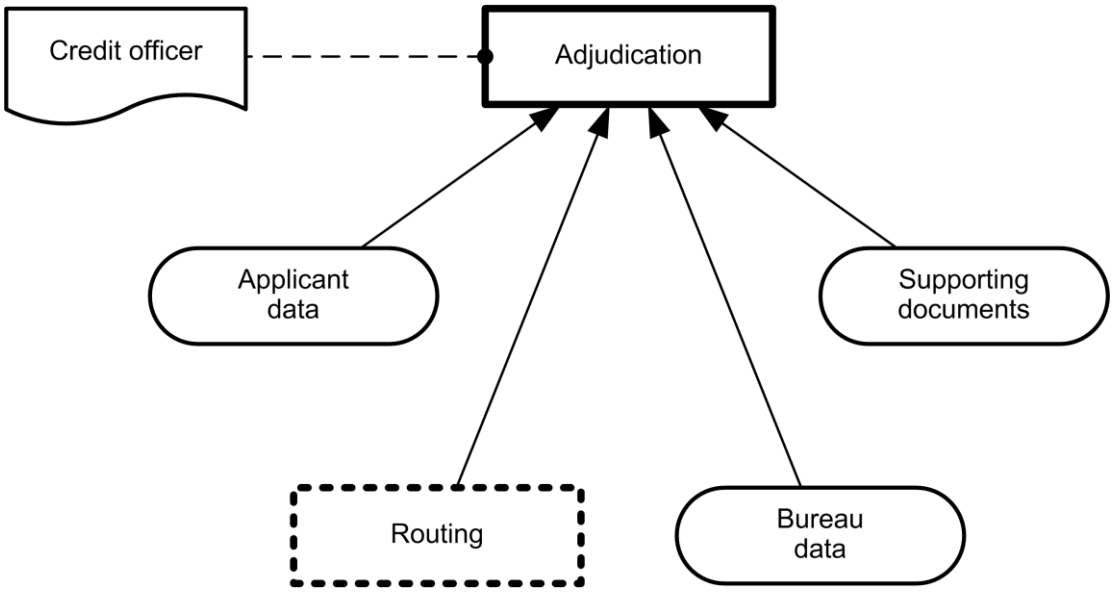


図73:申込みの意思決定ポイントをレビューするDRD

これらのDRDで図示されるDRGは、次の意思決定の間の依存関係を示す。

●**Strategy**判定は、Bureau call typeの判定結果とPre-bureau eligibilityの判定結果を必要とし、図76（ビジネス知識モデルでカプセル化されている表を除く）で示されるStrategy テーブルを起動する。

●**Bureau call type**の意思決定は、Pre-bureau risk categoryの意思決定結果を必要とし、図78に示されるBureau call type テーブルを起動する。

●**Eligibility**の意思決定は、Applicant dataとPre-bureau risk categoryの意思決定結果とPre-bureau affordabilityの意思決定結果を必要とし、図80に示されるEligibility rulesを起動する。

●**Pre-bureau affordability**の意思決定は、Applicant dataとre-bureau risk categoryの意思決定結果とRequired monthly installmentの計算結果を必要とし、図91に示される囲み式のAffordability calculationを起動し、図91は自分の起動の代わりに図92に示されるCredit contingency factorテーブルを起動する。

●**Pre-bureau risk categoryの意思決定**は、Applicant dataとApplication risk scoreの意思決定結果を必要とし、図82に示されるPre-bureau risk category テーブルを起動する。

●**Application risk score**の意思決定は、Applicant dataを必要とし、図84に示されるscore modelを起動する。

●**Routing意思決定**はBureau dataと、Post-bureau affordabilityの意思決定結果と、Post-bureau risk categoryの意思決定結果を必要とし、図86に示されるRouting rulesを起動する。

●**Post-bureau affordability**の意思決定はApplicant dataと、Post-bureau risk scoreの意思決定結果と、Required monthly installmentの計算結果を必要とし、図91に示される囲み式のAffordability calculationを起動し、図91は自分の起動の代わりに図92に示されるCredit contingency factor テーブルを起動する。

●**Post-bureau risk category**の意思決定は、ApplicantとBureau dataとApplication risk scoreの意思決定結果を必要とし、図88に示されるPost-bureau risk category テーブルを起動する。

●**Required monthly installment**の計算は、Requested product dataを必要とし、図94に示される囲み式のInstallment calculationを起動する。

●**Adjudication decisionの意思決定**は、Applicant data、Bureau data、Supporting documents、Routing意思決定結果を必要とするが、関連する意思決定ロジックをもたない。

ビジネスプロセス・モデルによって必要とされるふたつの意思決定サービスは、その意思決定モデルに対してあらためて定義され得る。**Decide bureau strategy**のタスクによって呼称を与えられている、**Bureau Strategy Decision Service**は、アウトプットの意思決定結果（Bureau call type, Strategy）をもち、図74に示される。**Decide routingの**タスクによって呼称が与えられている**Routing Decision Service**は、アウトプットの意思決定結果（Routing）をもち、図75に示される。

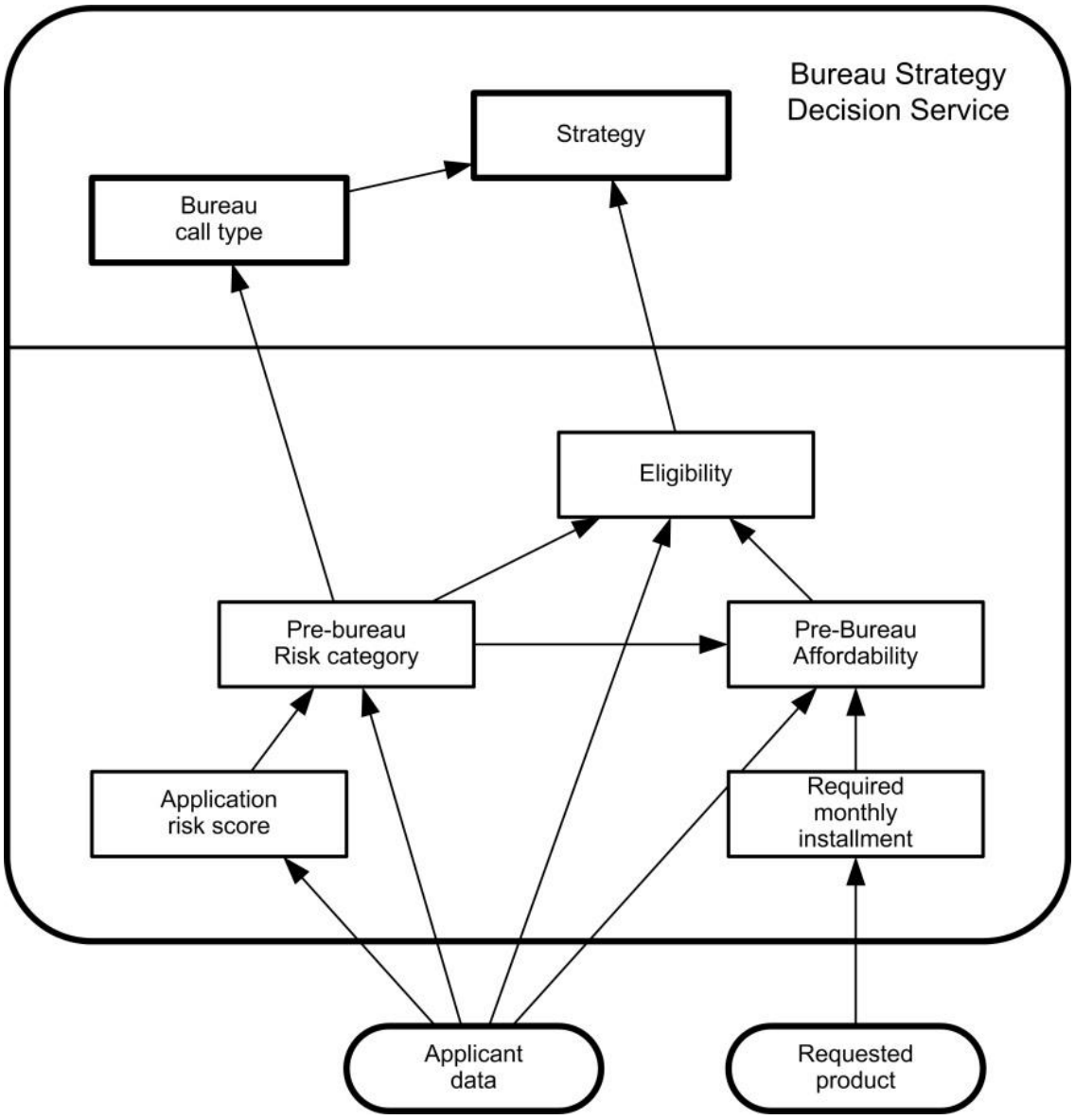


図74：信用調査会社戦略の決定サービス

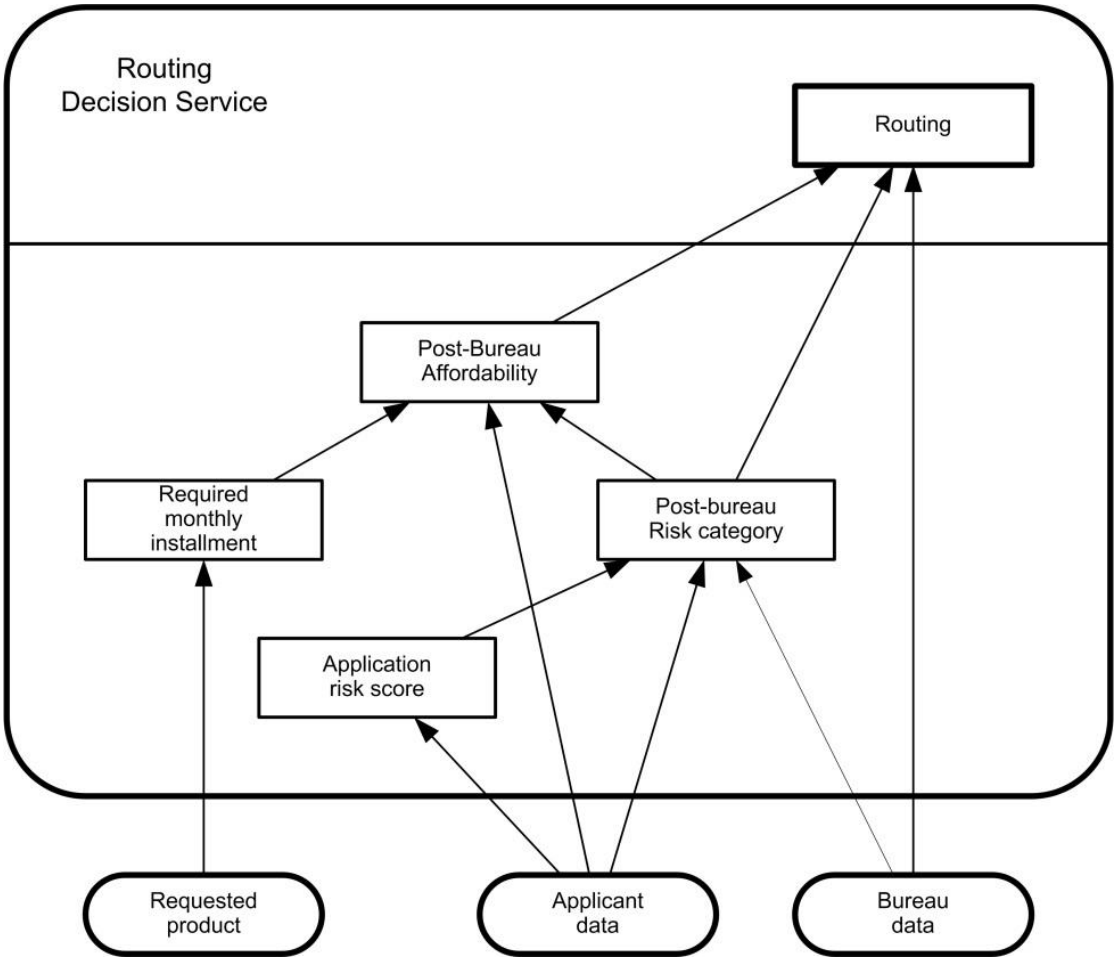


図75:ルーティングの意思決定サービス

11.4 意思決定ロジック・レベル

図70のDRGは、意思決定とビジネス知識モデルに関連する値式の下記仕様で、より詳細に定義される。

●**Strategy**の意思決定ロジック（図76）は、EligibilityとBureau Call TypeからStrategyを導く、完全で、ユニークヒットのデシジョンテーブルを定義する。

●**Bureau Call Type**の意思決定ロジック（図77で囲み起動として示される）は、Bureau call typeのテーブルを起動し、Pre-Bureau Risk Categoryのパラメータとして Pre-bureau risk categoryのアウトプットを渡す。

●**Bureau call type table**の意思決定ロジック（図78）は、Pre-Bureau Risk CategoryからBureau Call Typeを導く、完全で、ユニークヒットのデシジョンテーブルを定義する。

●**Eligibility**の意思決定ロジック（図79で囲み起動として示される）は、Eligibility rulesのビジ

ネス知識モデルを起動し、Applicant dataを渡す。AgeはAge parameterに、Pre-bureau risk categoryの意思決定のアウトプットはPre-Bureau Risk Categoryパラメータに、そしてPre-bureau affordabilityの意思決定のアウトプットはPre-Bureau Affordability parameterパラメータに，それぞれ割り振られる。

●**Eligibility Rules**の意思決定ロジック（図80）は、Pre-Bureau Risk Category、Pre-Bureau Affordability、AgeからEligibilityを導く、完全で、優先順位付きで、シングルヒットのデシジョンテーブルを定義する。

●**Pre-Bureau Risk Category**の意思決定ロジック（図81の囲み起動として示される）は、Pre-bureau risk category tableのビジネス知識モデルを起動し、Applicant dataを渡す。ExistingCustomerはExisting Customer parameterに、Application risk scoreの意思決定アウトプットはApplication Risk Score parameterに、それぞれ割り振られる。

●**Pre-Bureau Risk Category Table**の意思決定ロジック（図82）は、Existing CustomerとApplication Risk ScoreからPre-Bureau Risk Categoryを導く、完全で、シングルヒットのデシジョンテーブルを定義する。

●**Application Risk Score**の意思決定ロジック（図83の囲み起動として示される）は、Application risk scoreのビジネス知識モデルを起動し、Applicant dataを渡す。AgeはApplicant dataのうちのAge parameterに、MaritalStatusはApplicant dataのうちのMarital Status parameterに、EmploymentStatusはEmployment Status parameterに、それぞれ割り振られる。

●**Application Risk Score Model**の意思決定ロジック（図84）は、Age、Marital Status、そしてEmployment Statusから、合致するすべての列のPartial scoresの総計（したがって、これはデシジョンテーブルとして表現される予測スコアカードとなる）として、Application risk scoreを導く、完全で、適用順序に無関係な、何度も適用できるデシジョンテーブルの集合を定義する。

●**Routing**の意思決定ロジック（図85の囲み起動として示される）は、Routing rules のビジネス知識モデルを起動し、Bureau dataを渡す。BankruptはBureau dataのうちのBankrupt parameterに、CreditScoreはCredit Score parameterに、Post-bureau risk categoryの意思決定アウトプットはPost-Bureau Risk Category parameterに、Post-bureau affordabilityの意思決定アウトプットはPost-Bureau Affordability parameterとして、それぞれ割り振られる。Bureau dataがnullの場合（Collect bureau dataタスクを考慮しないTHROUGH strategyに基づくため）、Bankrupt and Credit Score parametersはnullになることに留意のこと。

●**Routing Rules**の意思決定ロジック（図86）は、Post-Bureau Risk Category、Post-Bureau Affordability、Bankrupt、Credit ScoreからRoutingを導く、完全で、優先順位付きで、シングルヒットのデシジョンテーブルを決定する。

●**Post-Bureau Risk Category**の意思決定ロジック（図87の囲み起動として示される）は、Postbureau risk catego ryのビジネス知識モデルを起動し、Applicant dataを渡す。ExistingCustomerはBureau dataのうちのExisting Customer parameterに、CreditScoreはCredit Score parameter,に、Application risk scoreの意思決定アウトプットはApplication Risk

Score parameterに、それぞれ割り振られる。Bureau dataがnullの場合（Collect bureau data タスクを考慮しないTHROUGH strategyに基づくため）、Credit Score parameterはnullになることに留意のこと。

●**Post-bureau risk category table**の意思決定ロジック（図88）は、Existing Customer、Application Risk Score、Credit ScoreからPost-Bureau Risk Categoryを導く、完全で、ユニークヒットのデシジョンテーブルを定義する。

●**Pre-bureau Affordability**の意思決定ロジック（図89の囲み起動として示される）は、Affordability calculationのビジネス知識モデルを起動し、Applicant dataを渡す。IncomeはApplicant dataのうちのMonthly Income parameterに、RepaymentsはApplicant dataのうちのMonthly Repayments parameterに、ExpensesはMonthly Expenses parameterに、Pre-bureau risk categoryの意思決定アウトプットはRisk Category parameterに、Required monthly installmentの計算アウトプットはRequired Monthly Installment parameterに、それぞれ割り振られる。

●**Post-bureau affordability**の意思決定ロジック（図90の囲み起動として示される）は、Affordability calculationのビジネス知識モデルを起動し、Applicant dataを渡す。IncomeはApplicant dataのうちのMonthly Income parameterに、RepaymentsはApplicant dataのうちのMonthly Repayments parameterに、ExpensesはMonthly Expenses parameterに、Post-bureau risk categoryの決定のアウトプットはRisk Category parameterに、Required monthly installmentのアウトプットはRequired Monthly Installment parameterに、それぞれ割り振られる。

●**Affordability calculation**の意思決定ロジック（図91）は、Monthly Income、Monthly Repayments、Monthly Expenses、Required Monthly Installmentから、Affordabilityを導き出す囲み式の機能を定義する。この計算の第一ステップは、Credit contingency factor tableのビジネス知識モデルを起動し、Risk Category parameterとしてRisk categoryの意思決定アウトプットを渡すことにより、Credit contingency factorを導き出す。

●**Credit contingency factor table**の意思決定ロジック（図92）は、Risk CategoryからCredit contingency factorを導く、完全で、シングルヒットのデシジョンテーブルを定義する。

●**Required monthly installment**の意思決定ロジック（図93の囲み起動として示される）は、Installment calculationのビジネス知識モデルを起動し、Requested productを渡す。ProductTypeはRequested productのProduct Type parameterに、RateはRequested productのRate parameterに、TermはRequested product.のTerm parameterに、AmountはAmount parameterに、それぞれ割り振られる。

●**Installment calculation**の意思決定ロジック（図94）は、Product Type、Rate、Term、Amountから月賦額を導き出す囲み式の機能を定義する。この計算の第一ステップは、図68で定義されているPMT calculationと等価の外部機能であるPMTを起動する。

Strategy			
U	Eligibility	Bureau Call Type	Strategy
1	*INELIGIBLE*	-	*DECLINE*
2	*ELIGIBLE*	*FULL, MINI*	*BUREAU*
3		*NONE*	*THROUGH*

戦略

U　適格性　信用調査会社呼び出し型　戦略

1　*INELIGIBLE*　-　*DECLINE*

2　*ELIGIBLE*　*FULL, MINI*　*BUREAU*

3　*ELIGIBLE*　*NONE*　*THROUGH*

図76:戦略決定ロジック

Bureau call type	
Bureau call type table	
Pre-Bureau Risk Category	Pre-Bureau Risk Category

Bureau call type

Bureau call type table

信用調査前のリスクカテゴリーカテゴリー Pre-Bureau Risk Category

図77:信用調査会社呼び出し型の決定ロジック

Bureau call type table		
U	Pre-Bureau Risk Category	Bureau Call Type
1	*HIGH, MEDIUM*	*FULL*
2	*LOW*	*MINI*
3	*VERY LOW, DECLINE*	*NONE*

Bureau call type table

U　信用調査前のリスクカテゴリーカテゴリー 信用調査会社呼び出し型

1　　*HIGH, MEDIUM*　　*FULL*

2　　*LOW*　　*MINI*

3　　*VERY LOW, DECLINE*　　*NONE*

図78:信用調査会社呼び出し型の表型の決定ロジック

Eligibility	
Eligibility rules	
Age	Applicant data . Age
Pre-Bureau Risk Category	Pre-bureau risk category
Pre-Bureau Affordability	Pre-bureau affordability

Eligibility

Eligibility rules

年齢　　Applicant data. Age

信用調査前のリスクカテゴリーカテゴリー　Pre-bureau risk category

信用調査前の購入容易性　Pre-bureau affordability

図79:適格性の意思決定ロジック

Eligibility rules				
P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
				INELIGIBLE, ELIGIBLE
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	< 18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

Eligibility rules

P　信用調査前の　　信用調査前の　　年齢　　適格性

　　リスクカテゴリーカテゴリー　　購入容易性　　INEGIBLE, ELIGIBLE

1	*DECLINE*	-	-	*INEGIBLE*
2	-	false	-	*INEGIBLE*
3	-	-	< 18	*INEGIBLE*
4	-	-	-	*ELIGIBLE*

図80:適格性ルールの意思決定ロジック

Pre-bureau risk category	
Pre-bureau risk category table	
Existing Customer	Applicant data . ExistingCustomer
Application Risk Score	Application risk score

Pre-Bureau risk category

Pre-Bureau risk category table

既顧客　　Applicant data. ExistingCustomer

申込みリスクスコア Applicant risk score

図81:信用調査前のリスクカテゴリーカテゴリーの意思決定ロジック

Pre-bureau risk category table

U	Existing Customer	Application Risk Score	Pre-Bureau Risk Category
1	false	< 100	*HIGH*
2		[100..120)	*MEDIUM*
3		[120..130]	*LOW*
4		> 130	*VERY LOW*
5	true	< 80	*DECLINE*
6		[80..90)	*HIGH*
7		[90..110]	*MEDIUM*
8		> 110	*LOW*

Pre-bureau risk category table

U 既顧客 申込みリスクスコア 信用調査前のリスクカテゴリーカテゴリー

1 false < 100 *HIGH*

2 false [100..120] *MEDIUM*

3 false [120..130] *LOW*

4	false	> 130	*VERY LOW*
5	true	< 80	*DECLINE*
6	true	[80..90]	*HIGH*
7	true	[90..110]	*MEDIUM*
8	true	> 110	*LOW*

図82:信用調査前のリスクカテゴリーカテゴリー表の意思決定ロジック

Application risk score	
Application risk score model	
Age	Applicant data . Age
Marital Status	Applicant data . MaritalStatus
Employment Status	Applicant data . EmploymentStatus

Application risk score

Application risk score model

年齢　　Applicant data. Age

婚姻状況　Applicant data. MaritalStatus

雇用状況　Applicant data. EmploymentStatus

図83:申込みリスクスコアの意思決定ロジック

Application risk score model

C+	Age	Marital Status	Employment Status	Partial score
	[18..120]	*S, M*	*UNEMPLOYED, EMPLOYED, SELF-EMPLOYED, STUDENT*	
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	*S*	-	25
7	-	*M*	-	45
8	-	-	*UNEMPLOYED*	15
9	-	-	*STUDENT*	18
10	-	-	*EMPLOYED*	45
11	-	-	*SELF-EMPLOYED*	36

Application risk score model

C+	年齢	婚姻状況	雇用状況	部分点
	[18..120]	S, M	*UNEMPLOYED, EMPLOYED, SELF-EMPLOYED, STUDENT*	
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>= 50	-	-	48
6	-	S	-	25
7	-	M	-	45
8	-	-	*UNEMPLOYED*	15
9	-	-	*STUDENT*	18
10	-	-	*EMPLOYED*	45
11	-	-	*SELF-EMPLOYED*	36

図84:申込みリスクスコアモデルの意思決定ロジック

Routing	
Routing rules	
Bankrupt	Bureau data . Bankrupt
Credit Score	Bureau data . CreditScore
Post-Bureau Risk Category	Post-bureau risk category
Post-Bureau Affordability	Post-bureau affordability

Routing

Routing rules

破産　　Bureau data. Bankrupt

信用度　　Bureau data. CreditScore

信用調査後のリスクカテゴリーカテゴリー Post-bureau risk category

信用調査後の購入容易性 Post-bureau affordability

図85:ルーティングの意思決定ロジック

Routing rules					
P	Post-Bureau Risk Category	Post-Bureau Affordability	Bankrupt	Credit Score	Routing
				null, [0..999]	*DECLINE, REFER, ACCEPT*
1	-	false	-	-	*DECLINE*
2	-	-	true	-	*DECLINE*
3	*HIGH*	-	-	-	*REFER*
4	-	-	-	< 580	*REFER*
5	-	-	-	-	*ACCEPT*

Routing rules

P 信用調査後の　信用調査後の　破産　信用度　ルーティング

リスクカテゴリーカテゴリー　購入容易性　null, [0..999] *DECLINE, REFER, ACCEPT*

1　-　false　-　-　*DECLINE*

2　-　-　true　-　*DECLINE*

3　*HIGH*　-　-　-　*REFER*

4　-　-　-　< 580　*REFER*

5　-　-　-　-　*ACCEPT*

図86:ルーティング・ルールの意思決定ロジック

Post-bureau risk category	
Post-bureau risk category table	
Existing Customer	Applicant data . ExistingCustomer
Credit Score	Bureau data . CreditScore
Application Risk Score	Application risk score

Post-bureau risk category

Post-bureau risk category table

既顧客　Applicant data . ExistingCustomer

信用度　Bureau data . CreditScore

申込みリスクスコア Application risk score

図87:信用調査後のリスクカテゴリーカテゴリーの意思決定ロジック

Post-bureau risk category table

U	Existing Customer	Application Risk Score	Credit Score	Post-Bureau Risk Category
1	false	< 120	< 590	*HIGH*
2			[590..610]	*MEDIUM*
3			> 610	*LOW*
4		[120..130]	< 600	*HIGH*
5			[600..625]	*MEDIUM*
6			> 625	*LOW*
7		> 130	-	*VERY LOW*
8	true	<= 100	< 580	*HIGH*
9			[580..600]	*MEDIUM*
10			> 600	*LOW*
11		> 100	< 590	*HIGH*
12			[590..615]	*MEDIUM*
13			> 615	*LOW*

Post-bureau risk category table

U	既顧客	申込みリスクスコア	信用度	信用調査後のリスクカテゴリーカテゴリー
1	false	< 120	< 590	*HIGH*
2	false	< 120	[590..610]	*MEDIUM*
3	false	< 120	> 610	*LOW*
4	false	[120..130]	< 600	*HIGH*
5	false	[120..130]	[600..625]	*MEDIUM*
6	false	[120..130]	> 625	*LOW*
7	false	> 130	-	*VERY LOW*
8	true	<= 100	< 580	*HIGH*
9	true	<= 100	[580..600]	*MEDIUM*
10	true	<= 100	> 600	*LOW*
11	true	> 100	< 590	*HIGH*
12	true	> 100	[590..615]	*MEDIUM*
13	true	> 100	> 615	*LOW*

図88：信用調査後のリスクカテゴリーカテゴリー表の意思決定ロジック

Pre-bureau affordability	
Affordability calculation	
Monthly Income	Applicant data . Monthly . Income
Monthly Repayments	Applicant data . Monthly . Repayments
Monthly Expenses	Applicant data . Monthly . Expenses
Risk Category	Pre-bureau risk category
Required Monthly Installment	Required monthly installment

Pre-bureau affordability

Affordability calculation

月収　　Applicant data . Monthly . Income

月次返済　　Applicant data . Monthly . Repayments

月次支出　　Applicant data . Monthly . Expense

リスクカテゴリーカテゴリー　Pre-bureau risk category

要求された月賦　Required monthly installment

図89:信用調査前の購入容易性の意思決定ロジック

Post-bureau affordability	
Affordability calculation	
Monthly Income	Applicant data . Monthly . Income
Monthly Repayments	Applicant data . Monthly . Repayments
Monthly Expenses	Applicant data . Monthly . Expenses
Risk Category	Post-bureau risk category
Required Monthly Installment	Required monthly installment

Post-Bureau Affordability

Affordability calculation

月収　　Applicant data . Monthly . Income

月次返済　　Applicant data . Monthly . Repayments

月次支出　　Applicant data . Monthly . Expense

リスクカテゴリーカテゴリー　Post-bureau risk category

要求された月賦　Required montyly installment

図90:信用調査後の購入容易性の意思決定ロジック

Affordability calculation		
(Monthly Income, Monthly Repayments, Monthly Expenses, Risk Category, Required Monthly Installment)		
Disposable Income	Monthly Income – (Monthly Repayments + Monthly Expenses)	
Credit Contingency Factor	Credit contingency factor table	
	Risk Category	Risk Category
Affordability	if Disposable Income * Credit Contingency Factor > Required Monthly Installment then true else false	
Affordability		

Affordability calculation

(Monthly Income, Monthly Repayments, Monthly Expenses, Risk Category, Required Monthly Installment)

可処分所得　　Monthly Income - (Monthly Repayments + Monthly Expenses)

信用不確実性ファクター　Credit contingency factor table

　　リスクカテゴリーカテゴリー　Risk Category

購入容易性　　if Disposable Income * Credit Contingency Factor > Required Monthly Installment

　　then true

　　else false

購入容易性

図91:購入容易性の計算の意思決定ロジック

Credit contingency factor table		
U	Risk Category	Credit Contingency Factor
1	*HIGH, DECLINE*	0.6
2	*MEDIUM*	0.7
3	*LOW, VERY LOW*	0.8

Credit contingency factor table

U リスクカテゴリーカテゴリー　信用不確実性ファクター

1　*HIGH, DECLINE*　　0.6

2　*MEDIUM*　　0.7

3　*LOW, VERY LOW*　　0.8

図92:信用不確実性ファクターの意思決定ロジック

Required monthly installment	
Installment calculation	
Product Type	Requested product . ProductType
Rate	Requested product . Rate
Term	Requested product . Term
Amount	Requested product . Amount

Required monthly installment

Installment calculation

プロダクト型 Requested product . ProductType

率　　Requested product . Rate

期間　　Requested product . Term

額　　Requested product . Amount

図93:要求される月賦額の意思決定ロジック

Installment calculation	
(Product Type, Rate, Term, Amount)	
Monthly Fee	if Product Type = “STANDARD LOAN” then 20.00 else if Product Type = “SPECIAL LOAN” then 25.00 else null
Monthly Repayment	PMT(Rate, Term, Amount)
Monthly Repayment + Monthly Fee	

Installment calculation

(Product Type, Rate, Term, Amount)

月次料金 if Product Type = “STANDARD LOAN”

then 20.00

else if Product Type = “SPECIAL LOAN”

then 25.00

else null

月次返済 PMT(Rate, Term, Amount)

Monthly Repayment + Monthly Fee

図95:割賦計算の意思決定ロジック

11.5 意思決定モデルの実行

意思決定モデル（この場合は、ふたつの意思決定サービスを呼び出すことによる）を実施するために、起動によって関数パラメータに引数を割り当てるのとまったく同じで、事案のデータがインプットデータに結びつけられなければならない。しかし、意思決定の要求のインプットがその意思決定で要求される知識のパラメータに割り当てられる起動の場合と異なり、事案のデータをインプットデータに割り当てることは、意思決定モデルの一部ではない。

FEELは、コンテキストとその他の数式が、事案のデータ（10.3.3のことXMLデータと10.6.1のコンテキストも参照のこと）を表現するために使用されることを許容する。インプットデータはアイテムの定義（7.3.2のItemDefinitionメタモデル））と関連付けられ、事案のデータはアイテム定義によって特定化される同型のそしてその他の制約をもたなければならない。事案のデータ

はFEEL領域にマッピングされなければならない。例えば、XLMインスタンスのデータは10.3.3のXMLデータで記述されているようにFEEL領域にマッピングされる。

わかりやすさのため、我々はXMLの代わりに囲み式を使って事案のデータを明記する。図95、図96、図97は、申込み者データ、要求された商品、信用調査データのために事案のデータを定義する囲みコンテキストを示す。

Applicant data		
Age	51	
MaritalStatus	*M*	
EmploymentStatus	*EMPLOYED*	
ExistingCustomer	false	
Monthly	Income	10,000.00
	Repayments	2,500.00
	Expenses	3,000.00

Applicant data
年齢　51
婚姻状況 M
雇用状況 EMPLOYED
既顧客 false
月次　収入 10,000.00
　　支払額 2,500.00
　　支出額 3,000.00

図95：申込み者データのインプットデータの例

Requested product	
ProductType	*STANDARD LOAN*
Rate	0.08
Term	36
Amount	100,000.00

Requested product
商品型　*STANDARD LOAN*

率　　0.08

期間　　36

額　　100,000.00

図96:要求された商品のインプットデータの例

Bureau data	
Bankrupt	false
CreditScore	600

Bureau data

破産　false

信用度 600

図97:信用調査データのインプットデータ例

Bureau Strategy Decision Serviceは、Applicant dataとRequested product case dataとともに呼び出されるとき、図98で示されるコンテキストを返す。

Strategy	*THROUGH*
Bureau Call Type	*NONE*

戦略　　*THROUGH*

信用調査会社の呼び出し型 *NONE*

図98:信用調査戦略の意思決定サービスのアウトプット

Routing Decision Serviceは、Applicant dataとRequested product、Bureau data case dataとともに呼び出されるとき、図99に示されるコンテキストを返す。

Routing	*ACCEPT*

ルーティング　ACCEPT

図98:ルーティングの意思決定サービスのアウトプット

12 変換フォーマット

12.1 完成前のモデルの詳細化

DMNモデルは、完成する前に詳細化されるのが通常である。 これは、あるユーザー（知識ソースのエキスパートやビジネスユーザーなど）が最初に概要レベルのモデルを定義し、それを他の人に渡してモデルを完成または改良する反復モデリングを行うときに頻繁に見受けられる。このような「完成前」のモデルは、必須のモデル属性のすべてがまだ満たされているわけではないか、または下限に位置する属性と関連のカーディナリティが満たされていないモデルである。
XMIは、このような完成前のモデルの詳細化を可能にする。 DMNでは、この機能をDMN XMLスキーマに基づくXMLファイルの交換に拡張する。 このようなXMLファイルでは、実装者は次の方法でこのファイルの交換をサポートすることが期待される。
・DMN XMLスキーマで「必須」にマークがついている属性の欠落を無視する。
• "minOccurs"が0より大きい要素の下限を減らす。

12.2 マシン読み取り可能ファイル

XSD、XMI、およびXMLファイルを含むすべてのマシン読み取り可能ファイルは、フラットzipファイルであるOMG Document dtc / 15-11-12に見つけられる。
•DMN XMIモデルの場合、メインファイルはDMN.xmiである。
•DMN XSD 交換（適合レベル1、2、3をサポート）では、メインファイルはDMN.xsdである。
•第11章の例のリストはch11exampleで提供されている。

12.3 XSD

12.3.1 ドキュメント構造

ドメイン固有のモデル要素の一式は、DMNファイルで交換されなければならない。 各ファイルのルート要素は、<DMN：Definition>であること。 ファイル一式は自己完結型でなければならない。すなわち、ファイル内で使用されるすべての定義は、<DMN：Import>要素を使用して直接的または間接的にインポートされなければならない。
各ファイルは一つのモデルの複数のファイル間で異なる「namespace」を宣言しなければならない。
含まれている要素が外部定義を使用する場合、DMNファイルは非DMNファイル（XSDやPMMLなど）をインポートしてもよい。

12.3.2 DMN XSD内の参照

参照する必要のあるDMN要素の多くはIDを含み、BPMN XSD内で要素への参照はこれらのIDを使用して表される。 XSD IDREF型は、IDで参照する従来のメカニズムであり、同じファイル内の要素のみを参照できる。 DMNElementReference型のDMN要素は、値が有効なURI参照でなければならないhref属性を利用して、ファイル間でIDを参照することをサポートする[RFC 3986]。そのパス構成要素は絶対または相対であり、参照はクエリコンポーネントでなく、 参照の要素は、その参照されるDMN要素のidの値で構成される。
たとえば、次のDecisionを考えてみる。

<decision name="Pre-Bureau Risk Category" id="prebureauriskDec01">…</decision>

この意思決定が、例えば別のファイルに定義された意思決定のInformationRequirementによって参照される場合、参照は次の形式をとることができる。

<requiredDecision href = "http://www.example.org/Definitions01.xml#prebureauriskDec01" />

ここで、「http://www.example.org/Definitions01.xml」は、「Pre-Bureau Risk Category」意思決定（例えば、対応するImport要素のlocationURI属性の値）が定義され、「prebureauriskDec01」は意思決定のid属性の値である。
URI参照のパス・コンポーネントが相対的である場合、相対参照が適用される基底URIは、[RFC 3986]の指定に従い決定される。 その仕様によれば、 "基底URIが埋め込まれておらず、表現が他のエンティティ内にカプセル化されていない場合、式を取り出すためにURIが使用された場合、そのURIは基底URIとみなされる" [RFC 3986] セクション5.1.3を参照）。
つまり、参照がxml：base属性[XBASE]のスコープ内にない場合、フラグメントとパスコンポーネントを含まないhref属性の値は、XMLファイルの同じインスタンスで定義されているDMN要素を参照する参照要素として使用する。 以下の例では、requiredDecision要素がxml：base属性のスコープ内にないと仮定すると、IDが "prebureauriskDec01"のDMN要素を同じXML文書で定義しなければならない。

<requiredDecision href=”#prebureauriskDec01” />

ある意思決定を使用するBPMNプロセスとタスクは、href属性を使用して参照されることに留意する。実際には、BPMN 2.0定義の外部ProcessインスタンスとTaskインスタンスを参照するシステムと互換性がある。

データ型（XSD属性@typeRef）への参照は、IDではなく名前による。 インポートされた型をサポートするために、@typeRefはQName型であり、接頭辞はインポートされたXSDまたはDMNファイルの名前空間を参照する。 基底タイプとインポートされていないItemDefinitionsを参照するには、接頭辞を省略す

ることができる。

付属書類

すべての付属書類は有益な知識を提供する。
付属書 Aでは、BPMNと組み合わせたDMNの適用に関する問題について説明する。 このセクションは、実践者にいくつかの方向を提供することを意図している、非規範的である。
付属書Bは、仕様の理解を助けるための非規範用語集を提供する。

附属書A：BPMNとの関係

(有益な知識)

1. BPMNとDMNの目的

OMG「ビジネスプロセス・モデルと表記法」標準は、ビジネスプロセスをタスクのオーケストレーションとして記述するための標準的な表記法を提供する。BPMNの成功はDMNにとって大きな動機となり、DMNを使用して記述されたビジネス上の意思決定は、BPMNを使用して記述されたビジネスプロセスで一般的に展開されることが期待される。

下記のBPMNに関連するすべての記述は、特に明記しない限り、OMGドキュメント参照（formal / 11-01-03）からのものである。
BPMNの目的は仕様に記載されており、DMNと簡単に比較できる。
•目的1：「BPMNの主な目的は、すべてのビジネスユーザーが容易に理解できる共通の表記法を提供することである。その表記法は、まずプロセスの初期ドラフトを作成するビジネスアナリスト、つぎにそれらのプロセスを実行するテクノロジーの実装を担当する技術開発者、そして最終的にはそれらのプロセスを管理および監視するビジネスユーザーが容易に理解できるものである。したがって、BPMNは、ビジネスプロセス設計とプロセス実装の間のギャップを解消するための標準的な架け橋を創出する。」 DMNユーザーは、ビジネスアナリスト（意思決定の設計者）であり、ビジネスユーザー（デシジョンテーブルなどの意思決定モデルの作成者）でもある。 また技術開発者は、ビジネス用語を適切なデータ技術にマッピングする責任を負うことがある。したがって、DMNはビジネスアナリストによる意思決定の設計と、意思決定実行技術を使用した意思決定の実装とをつなぐ架け橋と言えるだろう。

•目的2： "... WSBPEL（Web Services Business Process Execution Language）などのビジネスプロセスの実行用に設計されたXML言語をビジネス指向の表記で視覚化できるようにすること。"これはDMNの明白な目標ではない 他のXML言語（W3C RIFやOMG PRRなど）を視覚化できるようにするには、実際には、DMNがそのような言語のMDA仕様層を提供することが期待される。 ただし、実行可能形式（プロダクションルールなど）を表すDMN（デシジョンテーブルなど）の使用を排除するものではない。

•目的3：「BPMNの意図は、さまざまなモデリングの表記法や視点に照らして、ビジネスプロセス・モデルと表記法を標準化することである。 そうすることで、BPMNはプロセス情報を他のビジネスユーザー、プロセス実装者、顧客、サプライヤに伝える簡単な手段を提供する。」DMNの目的は、意思決定モデルと表記法を、広く意味論的に類似したモデルのさまざまな実装全体で

標準化することである。 そうすることで、DMNはビジネスコミュニティやツール間で意思決定情報の伝達を促進する。

2. BPMNのタスクとDMNの意思決定

ほとんどのBPMNダイアグラムには、DMNでモデル化できる意思決定を含むいくつかのタスクが含まれている。 これらのタスクは、プロセスの前半で取得または生成された入力データを取り込み、プロセスの後半で使用される決定出力を生成する。 意思決定出力は、次の2つの主要な方法で使用することができる。

•意思決定アウトプットは別のプロセス・タスクで消費される可能性がある。

•意思決定アウトプットはゲートウェイからのシーケンス・フローの選択に影響を与える可能性がある。

後者の場合、どのサブプロセスまたはタスクを（プロセスの意味で）実行するかを決定するために意思決定アウトプットが使用される。

そのため、DMNはBPMNを補完する。これは、意思決定モデリングがプロセスモデリングを補完するためである（オーケストレーションや作業タスクを定義するという意味で）。

たとえば、図100は、BPMN定義プロセスの例[1]を示す。

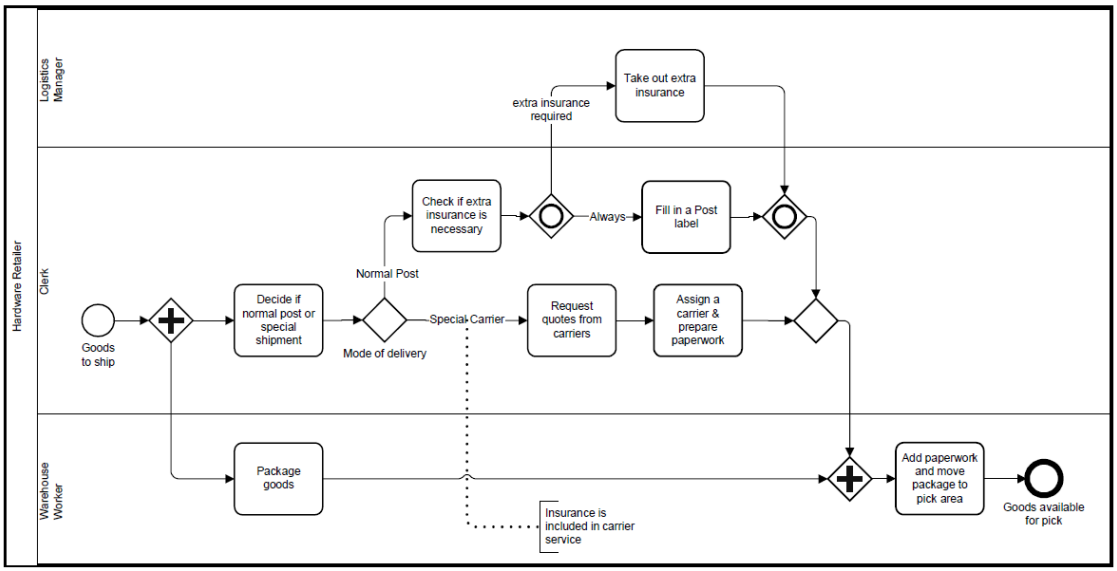


図100：BPMNにおける意思決定

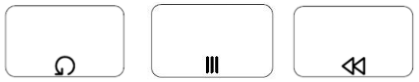
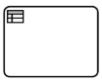
これを分析すると、次のことが分かる。

•タイトルが "Decide ..."で始まるタスクでは、通常出荷または特別出荷を使用するかどうかを意思決定し、その意思決定結果を使用する排他的なゲートウェイに先行するタスクである。
•タイトルが "Check ..."で始まるタスクで、追加の保険が必要かどうかを決定する。これは、決定結果に基づいて追加のプロセスパスが実行される包括的なゲートウェイに先行する。
•タイトルが「Assign ...」で始まるタスク。いくつかの選択基準に基づいてキャリアを選択することを意味する。 前のタスクは、この決定のためのデータを効果的に収集している。 自動化されたシステムでは、これはおそらく意思決定やその他の活動（「書類作成」など）を埋め込んだサブプロセスになるかもしれない。

この例から、**BPMN**の単純なビジネスプロセスでさえ、いくつかの意思決定タスクがあることがわかる。

3. DMNに関連するBPMNタスクのタイプ

BPMNは、意思決定の役割とみなされるさまざまなタイプのタスクを定義している。 関連するタスクは表70のとおり。

	タイプ	意思決定の役割
1	ループ　マルチ・インスタンス　補償 	非明示。 意思決定プロセスは反復またはループ（Reteベースのルールエンジンで Run To Completionサイクルを実行する製造ルールなど）を行うことができるが、これらはビジネス・モデリング・レベルとの関連はみなされない。
2	サービス・タスク 	意思決定タスクは、意思決定サービスによって（自動化された場合）実行される。 ただし、意思決定モデルはビジネスプロセスで自動的に実行されることは保証されない。
3	ユーザー・タスク 	ワークフロー指向のビジネスプロセスの一部として手動で実行される意思決定タスクは、ユーザータスクとして指定できることがある。
4	ビジネスルール・タスク 	ビジネスルール・タスクは**BPMN2**で（ビジネスルール駆動の）意思決定のプレースホルダーとして定義され、意思決定タスクの自然なプレースホルダーである。 注意：ビジネスルール（OMG SBVRで定義される）は、ビジネス上の決定だけでなく、あらゆるタイプのプロセス・アクティビティを制限することができる。
5	スクリプト・タスク 	意思決定タスクは、今日、ビジネスプロセス・スクリプト言語を使用してコード化されることがある。

BPMNの将来のバージョンでは、意思決定モデリング要求と**DMN**をよりよく一致させるためのタスクの定義を明確にし、拡張して、**BPMN**決定タスクを**DMN**でモデル化された意思決定を行うために使用するタスクとして定義できる。 当面、ビジネスルール・タスクは、この機能を表現する最も自然な方法である。 しかし、第5.2.2項 自動化される意思決定モデリング要求と第6.3.6項 アーキファクトメタモデルに記載されているように、**DMN**の意思決定は任意のタスク

に関連付けることができ、実装の柔軟性を可能にしている。

4. プロセス・ゲートウェイと意思決定

プロセス・ゲートウェイは2つのタイプが考えられる。

1. 既存のデータに基づくひとつのプロセスルートまたは複数のプロセスルートを決定するゲートウェイ。
2. プロセス内のいくつかの先行タスクによって決定される意思決定の結果に基づく、ひとつのプロセスルートまたは複数のプロセスルートを決定するゲートウェイ。

後者の場合、意思決定タスク（**DMN**を使用して意思決定を行うために使用されるタスク）は、決定タスクとそれを使用するゲートウェイとの関係を明確にするために拡張表記を必要とすることがある。

5. BPMNとDMNモデルのリンク

DMNは、**BPMN**のビジネスプロセス・モデルと意思決定モデルを結びつける2つのアプローチを提供する。ひとつは規範的であり、もうひとつは非規範的である。

a） タスクとプロセスへの意思決定の関連付け

6.3.6項 「Artifactメタモデル」で説明したように、DMN 1.1では、Decisionのインスタンスのプロセス・コンテキストは、OMG BPMN 2で定義されているProcessのインスタンスである任意の数のusingProcesses、およびOMG BPMN 2で定義されているTaskのインスタンスである任意の数のusingTaskとの関連によって定義される。

したがって、各意思決定は、ビジネスプロセス（そのプロセス中に意思決定が行われたことを示す）、や特定のタスク（タスクには意思決定が含まれる）と関連することがある。 実装は、これらの関連付けが各意思決定のために定義されることを許すべきですである。

実装は、2つのモデル（BPMNとDMN）に対して妥当性を確認して、次のことを実行する。

・意思決定は、意思決定に関連付けられていないプロセスの一部であるタスクには関連付けられていない。

・意思決定は、意思決定に関連するプロセスの一部ではないタスクにも関連付けられていない。

開発時には、意思決定のみをプロセスに関連付けるのが適切かもしれないが、タスクとプロセスの関連付けの間の不一致は許容されない。

このアプローチでは、ビジネスプロセス・モデルと意思決定モデルの関係を定義し、検証することができるが、**DMN**でモデル化された意思決定を**BPMN**でモデル化されたプロセスによって自動的に実行することはできない。

b）意思決定サービス

意思決定の自動化に対するひとつのアプローチは、非標準的に附属書Aに記載されている：BPMNタスクから呼び出される「意思決定サービス」におけるDMN意思決定のカプセル化（例えば、上記付録A.3で議論されるサービス・タスクまたはビジネスルール・タスク）である。usingProcessesプロパティとusingTasksプロパティを使用すると、**BPMN**と**DMN**間の関連付けの定義と検証が可能になる。 デシジョン・サービスの定義では、必要なインタフェースの詳細な仕様が提供される。

付録B:用語集

(参考情報)

A

Aggregation
集合

デシジョンテーブル上で、複数のヒットから一つの結果を生成する**DMN**では、ヒット・ポリシーの集合で4つの集合演算子を指定できる
すなわち：+（合計）、<（最小）、>（最大）、#（カウント）
演算子が指定されていない場合、集約ヒット・ポリシー(Collect hit policy)の結果は集合されない

Any
任意

意思決定が重複する**シングルヒット**の**デシジョンテーブル**の**ヒット・ポリシー**
規則：本ポリシーの下では、すべての一致を使用できる

Authority Requirement
根拠要求線

他の要素上にある意思決定要求図の要素の１つに従属し、その要素を導くまたは知識の源泉としての役割を示す

B

Binding
バインディング

起動では、バインディング式を使用して、呼び出し元の式の入力変数と起動式のパラメータの関連付けする

Boxed Context
囲みコンテキスト

オプションの結果値を含むn（名前、値）のペアを集めた**囲み式**のフォーム

Boxed Expression
囲み式

意思決定論理を小さな部品に分解する表記法で、小さな部品は**DRD**の要素と図形的に関連付けることができる

Boxed Function
囲み関数

関数の種類とパラメータおよび本体を示す**囲み式**のフォーム

Boxed Invocation
囲み起動

囲み式のフォームで、ビジネス知識モデルの本体の評価のためのコンテキストを提供するパラメータバインディングを示す

Boxed List
囲みリスト

n 個の要素からなるリストを表す**囲み式**のフォーム

Boxed Literal Expression
囲み文字式

文字式を表す**囲み式**のフォーム

Business Context Element
ビジネスコンテキルト要素

意思決定のビジネス・コンテキストを表す要素で、**部署**または**業績評価指標**

Business Knowledge Model
ビジネス知識モデル

意思決定または他の**ビジネス知識モデル**によって呼び出される、再使用可能な関数としてカプセル化された、**意思決定ロジック**（例、デシジョンテーブル）

C

Clause
句

デシジョンテーブルでは、句は対象を明確にする。その対象は入力式または出力ドメインによって定義され、デシジョンテーブルによって記述される意思決定ロジックの部分に関連する対象ドメインのサブドメインの有限集合を含む

Collect
集約

重複した**意思決定ルール**をもつ**マルチヒット・デシジョンテーブル**のための**ヒット・ポリシー**：
このポリシーでは、すべてのマッチは任意の順序のリストとして返される演算子を追加して、出力に適用する関数を指定することができる集合を参照のこと

Context
コンテキスト

FEEL において、**コンテキスト・エントリー**と呼ばれるキー値のペアのマップ

Context Entry
コンテキスト・エントリー

コンテキスト毎に 1 つあるキー値のペア

Crosstab Table
クロス・テーブル

表の2軸が2つの**入力式**となり、**出力エントリー**が2次元グリッドを形成する**デシジョンテーブル**の**幾何学的配置**

D

Decision
意思決定

入力からの出力の決定方法を定義した**意思決定ロジック**を使用して、いくつかの**入力値**から**出力値**を決定する行為

Decision Logic
意思決定ロジック

意思決定や**ビジネス知識モデル**の**値式**として**DMN**で定義され、**囲み式**として視覚的に表現された、意思決定に使用されるロジック

Decision Logic Level
意思決定ロジック・レベル

意思決定と**ビジネス知識モデル**に関連した**値式**からなる**DMN**におけるモデリングの詳細なレベル

Decision Model
意思決定モデル

意思決定要求および**意思決定ロジック**として**DMN**で表現された、意思決定の領域で形式化されたモデル

Decision Point
意思決定ポイント

BPMN 2.0でビジネスルール・タスクとしてモデル化された、ビジネスプロセス上の意思決定が行われるポイントで、決定サービスの呼出しとして実装される可能性がある

Decision Requirements Diagram
意思決定要求ダイアグラム

DRG のビューを表す（概ねフィルターリングされた）図

Decision Requirements Graph
意思決定要求グラフ

要求線によって結ばれた**DRG要素**（**意思決定**、**ビジネス知識モデル**、**入力データ**）のグラフ

Decision Requirements Level
意思決定要求レベル

1つ以上の**DRD**で表された**DRG**から構成されるDMNでのモデリングのより抽象的なレベル

Decision Rule
意思決定ルール

デシジョンテーブルでは、意思決定ルールは、ワンセットの意思決定結果またはワンセットの条件（**入力エントリー**）を伴う結果（**出力エントリー**）

Decision Service
意思決定サービス

意思決定モデルをカプセル化し、それをサービスとして公開するソフトウェア・コンポーネント、(例えば)**BPMN**プロセスモデルを内蔵するタスクにより使用される

Decision Table
デシジョンテーブル

出力エントリーが**入力エントリー**の特定のセットに適用されるかを示す**意思決定ルール**に編成される、関連する入力式と出力式のセットを表現した表

Definitions
定義

DMN意思決定モデルのすべての要素を入れたコンテナでは、**DMN**ファイルの交換は、常に1つ以上の定義を通じて行われる

DMN Element
DMN 要素

DMN意思決定モデルの要素には、**DRG要素**、**ビジネス・コンテキスト要素**、**式**、**定義**、**要素コレクション**、**情報項目**または**アイテム定義**がある。

DRD

Decision Requirements Diagram 参照

DRG

Decision Requirements Graph 参照

DRG Element
DRG 要素

DRGのコンポーネントには、**意思決定**、**ビジネス知識モデル**、**入力データ**または**知識ソース**がある。

E

Element Collection
要素コレクション

定義内の **DRG 要素**の名前付きグループを定義するために使用される

Expression
式

DMNの**意思決定モデル**の**決定ロジック**の一部を定義するために使用される**文字式**、**デシジョンテーブル**、または**起動など**。解釈されるときに単一の値を返す

F

FEEL

Friendly Enough Expression Language
DMNのデフォルトの式言語

First
最初

意思決定ルールが重複する**シングルヒット**の**デシジョンテーブル**の場合の**ヒット・ポリシー**。このポリシーでは、最初に合致したものが使用され、**意思決定ルール**の順序に基づいて決定する。

Formal Parameter
形式化されたパラメータ

関数の本体で使用する、関数の呼び出しで**情報項目**を提供するために使用される、名前付きの型付きの値

H

Hit
ヒット

デシジョンテーブルにおいて、**意思決定ルール**のすべての**入力式**の合致成功したこと、結論を結果に含めることができる

Hit Policy
ヒット・ポリシー

重複する意思決定ルールの解釈方法の指示。**シングルヒット・テーブル**は1つのルールの出力のみを返すのに対し、**マルチヒット・テーブル**は複数のルールの出力または出力の**集合**を返すことができる

Horizontal
水平

意思決定ルールが行として句が列として表現されるデシジョンテーブルを意味する幾何学的配置

I

Information Item
情報項目

モデリングに使用される**DMN要素**、**DMN意思決定モデル**の**意思決定ロジック・レベル**にある**変数**または**パラメータ**のいずれか

Information Requirement
情報要求線

入力データ要素に対する**意思決定**、または別の**意思決定**で使用される**変数**を提供するための依存関係

Input Data
入力データ

1つ以上の**意思決定**によって入力として使用される情報を示す。その値は、意思決定モデル外で定義される

Input Entry
入力エントリー

デシジョンテーブル（**意思決定ルール**と入力**句**の交点）内の条件セルを定義する**式**

Input Expression

デシジョンテーブルの入力**句**の**入力エントリー**と比

入力式

較される項目を定義する**式**

Input Value
入力値

デシジョンテーブルの入力**句**の期待値の限定された範囲を定義する**式**

Invocation
起動

複数の**バインディング**を使用して、値を別の値で評価できるようにするメカニズム

Item Definition
項目定義

FEELやXML Schemaなどの型付き言語を使用して、**入力データ**の構造と値の範囲と**意思決定**の結果をモデル化するために使用さる

K

Knowledge Requirement
知識要求線

意思決定ロジックの評価において呼び出されなければならない、**意思決定**または**ビジネス知識モデル**の**ビジネス知識モデル**の依存関係

Knowledge Source
知識ソース

定義された根拠、**意思決定**または**ビジネス知識モデル**、例えばそれらを定義または維持することを担当するドメイン専門家、またはビジネス知識モデルが導出されるソース文書、あるいは意思決定の一貫性をテストできるテストケースのセットなど

L

Literal Expression
文字式

出力値が入力値からどのように導かれるかを記述することによって**意思決定ロジック**を表す文字列。例：平文の英語またはデフォルトの式言語FEELを使用する

M

Multiple Hit
マルチヒット

複数の**意思決定ルール**からの**出力エントリー**を返す**デシジョンテーブル**の型

O

Organisational Unit
部署

意思決定を行う、または所有する組織の単位を表す**ビジネス・コンテキスト要素**

Orientation
幾何学的配置

デシジョンテーブルの表示スタイル：水平（意思決定ルールを行として、句を列として）、垂直（ルールを列として、句を行として）、またはクロス・テーブル

（2 軸から構成されるルール）。

Output Entry
出力エントリー

デシジョンテーブル（**意思決定ルール**と出力**句**の交点）に結論セルを定義した**式**

Output Order
出力順

重複する**意思決定ルール**を持つ**マルチヒット**の**デシジョンテーブル**の**ヒット・ポリシー。**このポリシーでは、すべてのマッチは優先順位の低い順にリストとして戻される。 出力の優先順位は、順序付けられた値のリストで指定される。

Output Value
出力値

デシジョンテーブルの出力**句**のドメイン値の範囲を限定する**式**

P

Performance Indicator
業績評価指標

意思決定の影響を受ける業績評価の指標を表す**ビジネス・コンテキスト要素**

Priority
優先度

重複する意思決定ルールを持つ**シングルヒット**の**デシジョンテーブル**の**ヒット・ポリシー**：このポリシーでは、出力優先度が最も高いマッチが使用される。 出力の優先順位は、順序付けられた値のリストで指定される

R

Relation
関係

リレーショナルテーブルのように、名前リストの一番上に一度しか表示されない水平**コンテキスト**（結果セルなし）のある垂直リストを表示する**囲み式**のフォーム

Requirement
要求線

ある**DRG要素**の別のDRG要素への依存。**情報要求線**、**知識要求線**または**根拠要求線**のいずれか。

Requirement Subgraph
要求線サブグラフ

DRG要素の要求線の推移閉包に起因する有向グラフ。 すなわち、特定の要素によって要求されるすべての意思決定を表す**DRG**のサブグラフ。

Rule Order
ルール順

意思決定ルールが重複している**マルチヒット**の**デシジョンテーブル**の**ヒット・ポリシー**：このポリシーでは、すべてのマッチが意思決定ルールの定義順にリストとして戻される

S

S-FEEL

単純な**式**のみを使用する**意思決定モデル**のための**FEEL**の単純なサブセット。特に、意思決定ロジックがモデル化されている意思決定モデルまたは概ね**デシジョンテーブル**のみを使用する意思決定モデル

Single Hit
シングルヒット

単一の**意思決定ルール**の**出力エントリー**のみを返す**デシジョンテーブル**の型。

U

Unique
ユニーク

重複が起こらず、すべての決定ルールが排他的である**シングルヒット**の**デシジョンテーブル**の**ヒット・ポリシー**。 1つのルールのみを合致することができる

V

Variable
変数

意思決定に入力される値、意思決定ロジックの説明、または関数にパラメータとして渡される値を表す

Vertical
垂直

意思決定ルールが列として句が行として提示されるデシジョンテーブルの幾何学的配置

W

Well-Formed
整形式

参照整合性、非循環性などの制約に従うことを示す**DRG要素**または**要求線**の使用