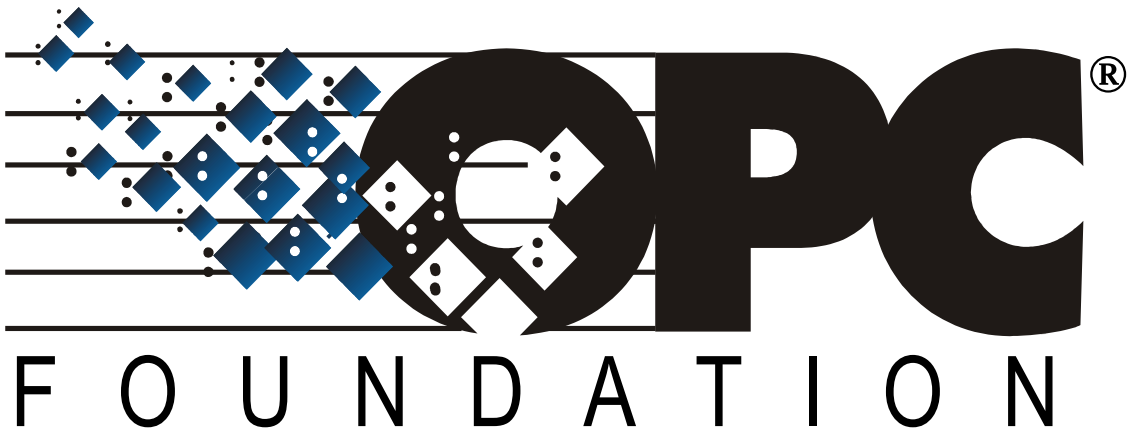


OPC UA 情報モデルの設計ガイドライン

バージョン 1.0

26 August 2019

日本 OPC 協議会　技術部会

CONTENTS

1 はじめに

1.1 対象読者と範囲

本ガイドラインの読者は OPC UA の一般的な仕様、とくに Part3 Address Space Model について既読であることを想定している。つぎについての理解がされていることを前提とする。

- UA ノードとそのノードクラス: Object、ObjectType、Variable、VariableType など
- ノード属性: NodeId、BrowseName、Value、DataType など
- UA ノード間の参照: Reference、ソースノード、ターゲットノードなど

本ガイドラインは、OPC UA 仕様書バージョン 1.04 に基づいている

1.2 背景

OPC Unified Automation (OPC UA)は、情報モデルのモデリングルールを定義している。情報モデルは、産業オートメーションや他の市場における相互運用性の基盤に成り得るが、相互運用性を実現するためには一般に認知されているモデルが必要である。OPC UA は幾つかの機能を実現するため基本的なモデル、DA、A&C、HA などの用途にビルドインで定義した情報モデル、既に他標準化団体とのコラボレーションで作成した情報モデルを提供しているが、それだけで自分達が扱いたい情報を充分に表現できないのであれば、新しい情報モデルを設計しなければならない。更に、その情報モデルを新たなコンパニオンスペックとして登録すれば、それに従ったマルチベンダー製品同士のコミュニケーションが実現でき、各ベンダーはそれらを利用したソリューションの開発に注力できる。

新しい OPC UA 情報モデルの設計のためには、以下のものを定義する。

タイプ情報: UA サーバーが UA クライアントに公開するメタモデル。メタモデルは、ObjectType、VariableType、ReferenceType および/または DataType ノードクラスを定義することによって規定される。

インスタンス情報: メタモデルからインスタンス化されるインスタンス。UA クライアントがそれらのオブジェクトをブラウズできるように AddressSpace の仕様が存在し、標準で定義されている'Objects'というフォルダノードから全てのオブジェクトを辿れることが求められる。'Objects'ノードの他に View ノードを定義して、各 View それぞれの視点について AddressSpace 内の UA ノードをフィルタリングする指定もできる。

新しい OPC UA 情報モデルを設計するプロセスはまず、既存の OPC UA の仕様でカバーされないタイプ情報で交換されるべき情報に何があるのかを収集することから始まる。次に、このような情報を OPC UA 情報モデルとして記述する。しかし、OPC UA 情報モデルはマシン可読形式であるため、その記述には専門知識が必要である。OPC UA のメタモデルはデバイスやマシンの中に存在するソフトウェアが通信できることを目的にしているからである。

1.3 本ガイドラインの目的と内容

本ガイドラインでは上記のハードルを低くするために、OPC UA 情報モデルをどのように設計するかについての基本的な情報をつぎに分類して記述する。

仕様: 本ガイドラインは、読者の理解を助けるために OPC UA オブジェクトモデルの仕様を記述する。いくつかの記述は、オリジナルの OPC UA 仕様書からの転用を和訳している。さらに、本ガイドラインは、対象となる情報が標準の OPC UA 仕様書の中のどこに記述されているかも記載する。読者は、どのベースタイプ情報から新しいタイプを導出するべきかを検討する際に、それらを参照することが期待される。

推奨: 本ガイドラインでは、技術情報のみならず、OPC UA 情報モデルを設計するための推奨事項についても記載する。1 つの効果を実現する場合にも、OPC UA オブジェクトモデルには複数の手段が存在する。設計した OPC UA 情報モデルの意味を容易に理解できるように、どの手段を選択するべきかのガイドラインを示すことが有益な場合がある。

本ガイドラインは OPC UA 情報モデルを設計するためのガイドラインとなる情報をつぎの2つの対象に分けて、以降の章で紹介する。

OPC UA メタモデル: OPC UA 情報モデルを記述する為のルール。

ビルドイン情報モデル: OPC UA メタモデルに従って定義されている、基本的な情報モデル。

2 OPC UA メタモデル

OPC UA オブジェクト・モデルを記述する為のコアとなるルール。

OPC UA 情報モデルを公開するために OPC UA ノードクラスをどのように利用するかという規則。

本章では、OPC UA のコア仕様の仕様書（Part3 から Part7）で記述されている項目の中から、情報モデルを定義するときに留意しなければならない要素や、ある機能を表現するときに参照すべき設計パターンなど、必要と思われるトピックを選択して各節に記載する。

2.1 Namespace

OPC UA において Namespace とは、独自に定義した UA ノードの識別子が他で定義した UA ノードの識別子と重複しないことを保証する為に、参照すべき UA ノードの集合を整理して編成したものを識別するメカニズムを指す。

それぞれの Namespace は、"http://opcfoundation.org/UA/"のような固有の URI を有する。

どのような場合に、新しい名前空間を定義すべきか?

仕様

コンパニオンスペックを定義する場合、既存の Namespace を扱うプロジェクトによる機能拡張を除き、新たな情報モデル群が対象とするドメインを表現するための Namespace を定義すべきである。Namespace を表す URI は、標準の URI の形式に準拠して一意性を確実にすることが求められる。

推奨

ベンダー固有の UA サーバーを開発する場合、その UA サーバーが公開するインスタンスを管理する為の Namespace を用意することを推奨する。URI はそのサーバーを指すものになる。もしもそのサーバー独自のベンダー固有のタイプ情報を定義している場合は、そのタイプ情報も同じ Namespace で管理することを勧める。無闇に Namespace 定義の数を増やすことは得策ではない。

2.2 DataType

DataType ノードは、Variable および VariableType ノード内の Value 属性のデータ型を表現するために使用される。Variable および VariableType ノードは DataType 属性を持ち、その値は Value 属性が持つ値のデータ型を表す DataType ノードの NodeId になっている。DataType ノードは、NodeClass ノード属性値が DataType の UA ノードである。DataType ノードは、HasEncoding 参照を介してエンコーディング情報を参照する。

意味および/または構造を制限する派生を表現するために、DataType ノードにサブタイプを定義することが可能である。例えば、Number という DataType ノードのサブタイプに UInteger が定義され、さらにそのサブタイプの1つに UInt16 という DataType ノードが存在する。独自にサブタイプを定義する場合、この目的のために、DataType のサブタイプに追加の Property を定義する場合や、標準の Property の値を特定する場合がある。

DataType の情報モデルでは、Simple DataType や Structured DataType を定義することができる。Simple DataType は上述の例のように、上位となる標準で定義された1つの値の型を表す DataType ノードから、その意味に更なる制約を加える為に定義する。Structured DataType はプログラミング言語の構造体型などの複数のデータフィールドからなるデータ型を定義する。

参考文献

- Part3 Address Space Model Release 1.04 : 5.8 DataTypes
- Part5 Information Model Release 1.04 : Annex D DataTypeDictionary
- Part5 Information Model Release 1.04 : Annex E OPC Binary Type Description

2.2.1 既存の DataType ノード

どのような標準 DataType ノードが、OPC UA 仕様書内で規定されているか?

仕様

コア仕様に関して、OPC UA 標準の DataType ノードは、以下の仕様書に列挙されている。

- Part3 Address Space Model Release 1.04 : 8 Standard DataTypes
- Part4 Services Release 1.04: 7 Common parameter type definitions
- Part5 Information Model Release 1.04: 12 Standard DataTypes

推奨

特化する特別な理由がない限り、標準タイプを使用すべきである。

String 型と LocalizedText 型を選択する基準は何か?

仕様

LocalizedText は、文字列データおよびそのロケールを有する Structured DataType である。したがって、文字列データがロケールの概念を有するべきである場合、LocalizedText を選択する。

String は、ロケールの概念を持たない文字列データに使用される Simple DataType である。

推奨

UA サーバーは、DataType が LocalizedText である文字列データに対してローカライゼーションは必須ではない。英語またはニュートラルな言語のサポートのみでも構わない。推奨は、ロケールの概念を有することができる文字列データに対しては LocalizedText を使用することである。

NodeId/ExpandNodeId 型の指定と Reference による参照を選択する基準は何か?

推奨

どちらもターゲットノードを参照する方法であり、選択する明確な基準を規定することは困難である。技術的には、Reference を使用する場合、UA クライアントは Browse サービスを使用してターゲットノードを参照する。DataType が NodeId または ExpandNodeId である Variable ノードを使用する場合、UA クライアントは Read サービスを使用してターゲットノードの NodeId を取得して参照する。どちらの手段が UA クライアントに自然に見えるかで判断すべき。一般に、Reference による参照は、UA クライアントが参照対象の UA ノードを柔軟に絞り込むことを可能にする。NodeId/ExpandNodeId データ型変数を使用する動機付けのひとつには間接参照があり、抽象的に用意したテンプレートとなるモデルを動的に具体的な UA ノードに紐付けて、共通の処理を複数の UA ノードに対して都度に流用したい場合に用いる。NodeId/ExpandNodeId を使用する別の動機付けには、配列によって複数の UA ノードを管理することが適切な場合がある。

2.2.2 サブタイプ定義

既存の DataType を特化するにはどうすればよいか?

仕様

既存の DataType のサブタイプとして、新しい DataType を定義する。

例えば、Part4 では、UInt32 DataType のサブタイプとして Counter DataType を紹介している。Counter は、カウント処理という限られた用途に適用されることが期待される。

サブタイプされた DataType への Property の追加や、既に定義されている Property の値を特定することもサブタイプを定義する際の動機付けである。例えば、Enumeration DataType の多くのサブタイプは、列挙型として利用可能な値を表現するために、標準 Property である'EnumStrings'または'EnumValues'を追加してその値を定義する。(「列挙型を示す DataType を定義するにはどうすればよいか?」を参照)

配列データ型を示す DataType ノードを定義できるか?

仕様

定義できない。OPC UA で配列を表現する仕様は別に存在する。Variable および/または VariableType が、その Value 属性に配列値を有する場合、DataType とは別に ValueRank および ArrayDimmensions 属性を用いて値が配列であることを表現する。つまり、Simple DataType Node は、配列を表現するものではない。

例外は、String および ByteString の DataType ノードのみである。この DataType ノードを用いれば、ValueRank および ArrayDimmensions 属性を指定せずに文字およびバイト値の一次元配列を表現する。

列挙型を示す DataType を定義するにはどうすればよいか?

仕様

まず、Enumeration という標準の DataType のサブタイプ(またはそのサブタイプ)である DataType ノードを定義する。定義した DataType ノードに EnumStrings という Property を追加する。EnumStrings には、列挙値のラベルを示す LocalizedText の配列を定義する。オプションで EnumValues という Property を指定しない限り、列挙型のインデックスは EnumStrings 値が示す配列の 0 始まりの位置となる。インデックスが0で開始しない、または各インデックスの値の幅が1以外に定義したい場合は、EnumValues を追加して、EnumStrings が持つ配列内の各要素に対応するインデックス値を Int32 の配列で定義する。

注記 1: Enumeration DataType から直接に派生した DataType でない場合、継承元となる列挙型の DataType ノードに定義した列挙値のサブセットのみが派生側の DataType で定義可能である。1

どのような場合に、Structured DataType を定義すべきか?

推奨

一貫性の観点から複数のデータが一度にアクセスされるべきである場合、それらのデータを 1 つに纏めた Structured DataType を定義すべきである。複数のデータをグループ化する必要があるがアクセスの一貫性が要求されない場合は、そのような変数(DataVariable および/または Property)をグループ化する新しい ObjectType および/または VariableType を指定するべきである。

注記 1: Structured DataType を定義するかどうかを決定することが難しい場合は、Structured DataType を使用しないことを推奨する。OPC UA 1.04 から構造体や共用体のメタ情報を定義する方法が簡易化されたが、それでも UA クライアントの中には標準ではない Structured DataType のエンコード/デコードが未サポートな為に相互運用性に影響を及ぼす場合がある。1

参考文献

- Part3 Address Space Model Release 1.04 : A.4.3 Many Variables and / or structured DataTypes

Structured DataType のメタ情報を定義するにはどうすればよいか?

仕様

Structured DataType のメタ情報は、DataType ノードの DataTypeDefinition 属性の値を指定することで定義できる。以下の仕様書を参照。

- Part3 Address Space Model Release 1.04 : 8.49 StructureDefinition

上記の仕様を用いて構造体や共用体のデータ型を示す DataType ノードを定義する例が以下の仕様書に記載されている。

- Part6 Mappings Release 1.04 : 5.2.6 Structures
- Part6 Mappings Release 1.04 : 5.2.7 Structures with optional fields

DataType と VariableType のどちらを定義するべきか?

推奨

DataType を定義するということは、ソフトウェアの間で交換するデータの形式を定義することを意味する。VariableType を定義するということは、指定した DataType のデータ形式による値をもつ変数のメタ情報を定義することを意味する。一般的に、VariableType を定義することは、ある DataType の値を持つ変数

の用途に応じて補足情報となる Property や追加の変数、そのタイプの変数のみが持ち得る参照関係などを追加することになる。

しかしながら、OPC UA の仕様の制約によりどちらを選択するかが決まってしまう場合がある。

- 一度のアクセスで複数のデータを常に同時に取得することが必須の場合は、それらのデータをデータフィールドに持つ Structured DataType を定義するべきである。複数のデータをグループ化する必要があるが一貫性のアクセスが必須の要件でない場合は、そのようなデータを持つ Variable ノード(DataVariable および/または Property)をグループ化する新しい ObjectType および/または VariableType が指定されるべきである。(注記 1)
- 配列を表すメタ情報を表現する場合は VariableType を定義する。DataType は、String および ByteString 以外の配列データを表すことはできない。

注記 1: Structured DataType のデータフィールドを VariableType の Property として定義することで、用途に応じて一括アクセスと部分アクセスを使い分ける方法も存在する。

参考文献

- Part3 Address Space Model Release 1.04 : A.4.3 Many Variables and / or structured DataTypes

独自の Simple DataType を定義するにはどうすればよいか?

推奨

独自の Simple DataType を定義することは推奨しない。UA クライアントが標準ではない Simple DataType のエンコード/デコードが未サポートな為に相互運用性に影響を及ぼす場合がある

独自の DataType の定義は上述の列挙型や Strucutred DataType の範囲に留めることを勧める。

独自の意味づけした Simple DataType の定義が必要な場合、Structured DataType や VariableType の定義で代用できないかを検討することを勧める。代用が困難な場合は OPC UA WG に提案して標準タイプ化するアプローチを進める。

DataType ノードを記述するための表記法は何か?

推奨

DataType ノードの定義を記述するには、次のテンプレートテーブルを使用する。

使用可能な属性と Reference については、次のドキュメントを参照。

- Part3 Address Space Model Release 1.04 : 5.8.3 DataType NodeClass

表 1-DataType 定義の例

Attribute	**Value**				
BrowseName	MyDataType				
IsAbstract	True				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *<MyParentDataType> de*fined in <Clause number>.					
HasSubtype	DataType	MyAnotherDataType	Defined in <Clause number>		
HasProperty	Variable	EnumStrings	LocalizedText[]	PropertyType	Mandatory
Notes – Notes referencing footnotes of the table content.					

- 前段の行では、DataType ノードに必要な属性とその値を指定する。

- DataType を記述する場合、BrowseName および IsAbstract ノード属性を記述するので充分であるが、UA サーバーを開発する際には他の必須の属性も必要になる。

- 後段の行では、DataType ノードからの Reference を記述する。
  - 親 DataType ノードは、上の表の形式に倣って指定する。
  - DataType の場合、Reference は主に HasSubtype および HasProperty 参照を指定する。HasEncoding も指定できるが、自明な情報なので上記の表に記載する必要はない。
  - HasSubtype を指定した場合、上記と同じ形式の表を新たに用いてサブタイプを記述することが推奨される。
  - 'References'列は、使用可能な参照タイプ名を指定する。
  - 'NodeClass'列は、ターゲットノードの NodeClass ノード属性値を指定する。
  - 'BrowseName'列は、ターゲットノードの BrowseName ノード属性値を指定する。
  - 'DataType'列は、ターゲットノードが Property の場合に、その DataType、ValueRank、ArrayDimensions のノード属性値に合せて、以下の表記法の例に倣って指定する。

表 2–データタイプの例 1

表記	DataType	ValueRank	ArrayDimensions	説明
Int32	Int32	−1	省略または NULL	スカラーInt32
Int32[]	Int32	1	省略または{0}	サイズが任意の Int32 の一次元配列
Int32[][]	Int32	2	省略または{0,0}	両次元のサイズが任意な Int32 の二次元配列
Int32[3][]	Int32	2	{3,0}	Int32 の二次元配列で、1 次元目のサイズは 3、2 次元目のサイズは任意
Int32[5][3]	Int32	2	{5,3}	1 次元目のサイズが 5、2 次元目のサイズが 3 である、Int32 の二次元配列
Int32{Any}	Int32	−2	省略または NULL	任意次元数の配列、またはスカラを取り得る Int32 の値
Int32{ScalarOrOneDimension}	Int32	−3	省略または NULL	一次元配列またはスカラのいずれかである Int32

  - 'TypeDefinition'列は、ターゲットノードが Property の場合に、それが参照する VariableType ノードの BrowseName ノード属性値を指定する。
  - 'ModellingRule'列は、DataType ノードが参照する UA ノードが Property の場合に、その ModellingRule を指定する。(「2.11 ModellingRule」を参照)ModellingRule
  - ターゲットノードが Property でかつそのデフォルト値を定義する必要がある場合、表1の様式の表の下に各 Property について説明する項目を追記し、その中で Value ノード属性値についても記述することを推奨する。

- DataType が Structured DataType である場合、その構造を記述するための表を追記する。以下は例。

- **表 3-構造体定義の例(BuildInfo)**

Name	Type	Description
BuildInfo	structure	Information that describes the build of the software.
productUri	String	URI that identifies the software
manufacturerName	String	Name of the software manufacturer.
productName	String	Name of the software.
softwareVersion	String	Software version
buildNumber	String	Build number
buildDate	UtcTime	Date and time of the build.

- DataType が列挙型(Enumeration のサブタイプ)である場合、値を記述するための表を追記する。表記は、<EnumStrings 設定値>_<EnumValues 設定値>を推奨。EnumValues 値の定義が不要な場合でも下記のようにデフォルトの値を付記するケースが多い。

- **表 4-列挙値定義の例(RedundancySupport 型の列挙値)2**

Value	Description
NONE_0	None means that there is no redundancy support.
COLD_1	Cold means that the server supports cold redundancy as defined in **エラー! 参照元が見つかりません。**.
WARM_2	Warm means that the server supports warm redundancy as defined in **エラー! 参照元が見つかりません。**.
HOT_3	Hot means that the server supports hot redundancy as defined in **エラー! 参照元が見つかりません。**.
TRANSPARENT_4	Transparent means that the server supports transparent redundancy as defined in **エラー! 参照元が見つかりません。**.
HOT_AND_MIRRORED_5	HotAndMirrored means that the server supports HotAndMirrored redundancy as defined in **エラー! 参照元が見つかりません。**.

2.3 Variable

Variable ノードは、値を表すために使用され、Value 属性を有する。値の型を指定するために、DataType、ValueRank、ArrayDimensions の属性を持つ。Variable ノードは、NodeClass ノード属性値が Variable の UA ノードである。

Variable ノードには Property と DataVariable の 2 種類が存在する。その種別は、意味的にはそれらが表すデータの種類により区別される。構造的にはその Variable ノードが他の Variable ノードを要素として持ち得るかにより区別される(Property はその要素に Variable ノードを持ち得ないが、DataVariable は可能)。

参考文献

- Part3 Address Space Model Release 1.04 : 4.4 Variables
- Part3 Address Space Model Release 1.04 : 5.6 Variables

2.3.1 Property

どのような場合に、Property を定義すべきか?

仕様

Property は、Object、DataVariable を始めとする全ての UA ノードに対して、特性を表現するための変数として定義できる。ただし Property を表す Variable ノードには Property を定義することはできない。

推奨

基本的な考え方は上記に記載されている通りだが、Poperty を採用する観点をつぎに示す。

- 判断例として、ある UA ノードが何かを表現するために用いられるデータで、その UA ノードが生成されたときに初期設定された値がそのまま変更されずに使用されるものは Property で表現するべきかもしれない。UA ノードが生成された後に変更されるとしても、1つの値が設定されてからその UA ノードを運用する間は変更されない構成データのようなものも特性と考えられるかもしれない。後者の例としては EU(工業単位)や EURange(工業単位レンジ)が挙げられる。
- 別の判断例は、その値が独立して変更されるのか、他の値に依存して変更されるのかにより特性か否かを検討できる。以下の仕様書には、ファイルオブジェクトの例を記述している。ファイル内部のバイトストリームは、バイト配列を表す DataVariable と考えられ、Property を使用して、ファイルオブジェクトの更新時間および所有者を公開することができる。これらの Property は、独立して変更することはできず、ファイル内部のバイト配列データが新規作成または更新される場合に連動して変更されるので特性と考えることができる。

  Part3 Address Space Model Release 1.04 : 4.4.3 DataVariables

ノード属性と Property の違いは何か?

仕様

ノード属性と Propery は根本的に異なる。ノード属性は UA ノードの構成要素であり、UA ノードそのものの特性を表現する。Property は情報モデルの構成要素であり、UA ノードを用いて表現した情報モデルの特性を表現する。

推奨

ノード属性と Property は仕様として異なるが、どちらを使用するか混乱する場合がある。例えば、すべての UA ノードには Description というノード属性があるが、UA ノードを用いて表現したモデルにも Description という名前の Property を定義すべきかで迷う声がある。

本ガイドラインは、設計検討対象の Property と同一の意味およびデータ型を有するノード属性が存在する場合、ノード属性を使用することを推奨する。意味が異なる場合は、ノード属性とは異なる名前の Property が定義されるべきである。

Property を定義するにはどうすればよいか?

仕様

Property は下記の流れにて定義を行う。

対象の Property を表す Variable ノードを定義する。

Property であることを示すために HasTypeDefinition の Reference を定義し、その参照先を PropertyType(またはそのサブタイプ)を示す VariableType ノードにする。

BrowseName ノード属性の値は、Property を持つ親 UA ノードの要素の中で、一意でなければならない。

DataType、ValueRank および/または ArrayDimensions ノード属性の値は、HasTypeDefinition で参照する VariableType ノードに定義されている同名ノード属性値と同じまたはサブタイプ(注記 1) を設定する。

Property と、それを持つ親 UA ノードの参照関係を定義する。Property を持つ親 UA ノードから対象 Property に対して HasPeoperty(またはそのサブタイプ)の Reference を定義する。

注記 1: 指定可能なサブタイプについては「Part3 Address Space Model Release 1.04:6 Type Model for ObjectTypes and VariableTypes」を参照。

注記 2: Property を ObjectType および/または VariableType に定義する場合、Mandatory または Optional の ModellingRule の定義を考慮しなければならない。ModellingRule の定義については、「2.11 ModellingRule」を参照。1ModellingRule

Property に Property を定義したい場合にはどうすればよいか?

仕様

Property に対して Property を定義することはできない。

推奨

Property に対して Property を定義したいと考える場合は、両者の Property（特性）が示す意味を考えて設計を見直す必要がある。以下に幾つかの観点を紹介する。

- それらの特性の位置が適切であるかを再考する。特性の情報が更に特性を持つということが適切なのか。どこか他にその特性情報を持つべき Object や Variable があると思われる場合には、Property の位置を変更すべきである。
- それらの特性の情報が意味的にグループ化できる場合、それらのデータを含む 1 つの Structured DataType を定義する方法がある。
- 親となる Property が示す値は本当に特性であるのかを再考する。データ変数と考えて問題がないならば、親となる Peoperty を DataVariable として再設計する方法がある。

2.3.2 DataVariable

どのような場合に DataVariable を定義すべきか?

仕様

DataVariable は、Object、ObjectType、Variable、および VariableType に対して、変数として定義することができる。DataVariable は、オブジェクトおよび/またはそのメタ情報が持つ要素の値を表すことが主目的であり、その表現方法の派生として Variable や VariableType にも定義できる（ただし Property を示す Variable ノードには DataVariable を定義することはできない）。

推奨

「どのような場合に、Property を定義すべきか？」を参照。Property として推奨できない Variable は DataVariable とする。どのような場合に、Property を定義すべきか?

DataVariable を定義するにはどうすればよいか?

仕様

DataVariable は下記の流れにて定義を行う。

対象の DataVariable を示す Variable ノードを定義する。

DataVariable であることを示すために HasTypeDefinition の Reference を定義し、その参照先を BaseDataVariableType（またはそのサブタイプ）を示す VariableType ノードにする。

DataType、ValueRank および/または ArrayDimensions ノード属性の値は、HasTypeDefinition で参照する VariableType ノードに定義されている同名ノード属性値と同じまたはサブタイプ(注記 1) を設定する。

DataVariable ノードと、それを持つ親 UA ノードの参照関係を定義する。DataVariable ノードを持つ親 UA ノードから対象 DataVariable ノードに対して HasComponent（またはそのサブタイプ）の Reference を定義する(注記 3)。

注記 1: 指定可能なサブタイプについては「Part3 Address Space Model Release 1.04:6 Type Model for ObjectTypes and VariableTypes」を参照。

注記 2: DataVariable を ObjectType および/または VariableType に定義する場合、ModellingRule を考慮しなければならない。ModellingRule の定義については、「2.11 ModellingRule」を参照。1ModellingRule

注記 3: Browse サービスのレスポンスで返る UA ノードの順序を明確に定義する必要がある場合は、HasComponent の代わりに HasOrderedComponent（またはそのサブタイプ）を指定する。1ModellingRule

2.3.3 ヒストリアクセス対応

Variable ノードがヒストリデータアクセスをサポートしていることを公開するにはどうすればよいか?

仕様

Variable ノードの AccessLevel ノード属性に HisroryRead および/または HistoryWrite ビットを設定する。

注記 1: Historizing ノード属性は、サーバーが変数のヒストリ収集を行っているか否かを公開する為のフラグとして使用する。1

Variable ノードのヒストリ構成を公開するにはどうすればよいか?

仕様

「3.4 Historical Access」を参照。Historical Access

2.4 VariableType

VariableType は、Variable ノードのタイプ定義を表すために使用される。VariableType ノードは、その NodeClass ノード属性値が VariableType の UA ノードである。

VariableType ノードは、Variable ノードから HasTypeDefinition(またはそのサブタイプ)の Reference を介して参照される。

Variable ノードをインスタンス化する場合、参照する VariableType ノードのノード属性のうち Value(オプションでデフォルト値)、DataType、ValueRank および ArrayDimensions(ValueRank が配列を指定する場合のみ)の値は Variable ノードのノード属性にコピーされる。

VariableType ノードの派生関係を表現する為のサブタイプ定義が可能。

参考文献

- Part3 Address Space Model Release 1.04 : 4.5 TypeDefinitionNode
- Part3 Address Space Model Release 1.04 : 5.6.5 VariableType NodeClass
- Part3 Address Space Model Release 1.04 : 5.6.6 Client-side creation of Variables of an VariableType
- Part5 Information Model Release 1.04 : :5.4 VariableTypes

2.4.1 既存の VariableType ノード

どのような標準 VariableType ノードが、OPC UA 仕様書内で規定されているか?

仕様

コア仕様に関して、OPC UA 標準の VariableTypes ノードは、以下の仕様書に列挙されている。

- Part5 Information Model Release 1.04 : 7 Standard VariableTypes

他の標準 VariableType ノードが、Part8 DataAccess、Part9 Alarm and Conditions などのような特定の機能ドメインに対するビルトインの情報モデルとして規定されている。ビルトイン情報モデル

推奨

設計対象と同じ意味の VariableType がある場合は、相互運用性のために標準タイプを使用すべきである。

追加情報が必要な場合は、標準 VariableType をサブタイプ化すべき。

使用する予定のない情報がある場合でも、未使用情報にダミー値を指定するなどして、標準 VariableType の使用を検討すべき。

2.4.2 サブタイプ定義

VariableType に継承関係を定義するにはどうすればよいか?

仕様

派生元の VariableType ノードから派生先 VariableType ノードに対して HasSubtype の Reference を定義する。

抽象タイプの VariableType を定義するにはどうすればよいか?

仕様

IsAbstract ノード属性を True に指定する。

VariableType ノードに動的メンバーである Variable を定義するにはどうすればよいか?

仕様

対象の Variable ノードを定義する。「2.3 Variable」の「Property を定義するにはどうすればよいか？」または「DataVariable を定義するにはどうすればよいか？」を参照。

動的メンバーであることを示すためには、VariableType ノードから対象 Variable ノードへの Reference を定義する際に、ModellingRule(注記 1)を定義する。

注記 1：ModellingRule の定義については、「2.11 ModellingRule」を参照。ModellingRule

VariableType ノードに静的メンバーである Variable を定義するにはどうすればよいか?

仕様

対象の Variable ノードを定義する。「2.3 Variable」の「Property を定義するにはどうすればよいか？」または「DataVariable を定義するにはどうすればよいか？」を参照。

静的メンバーであることを示すためには、VariableType ノードから対象 Variable ノードへの Reference を定義する際に、ModellingRule を定義しないこと。表記上は None とする。

VariableType を派生するとき、派生元 VariableType の Variable 要素をオーバーライドするにはどうすればよいか?

仕様

下記の流れにて行う。

派生先の VariableType ノードに、オーバーライド対象と同じ BrowseName と NodeClass ノード属性値をもつ Variable ノードを定義する。

オーバーライドできるデータは、BrowseName と NodeClass 以外のノード属性値、TypeDefinition および/または ModellingRule である。

オーバーライドの対象でないデータに対しては派生元の Variable ノードと同じ内容を再定義する必要がある。

オーバーライドで変更できる内容にはいくつかの制約がある。基本的な考え方としては、オーバーライドされるデータは、派生元の Variable ノードに定義されている内容のサブタイプに限定される。例として、派生元の Variable ノードの DataType 属性が Number である場合、派生先のオーバーライド対象 Variable ノードの DataType 属性に Integer を指定することはできるが、String は指定できない。詳細な規則については、参考文献を参照。

参考文献

- Part3 Address Space Model Release 1.04 : 6 Type Model for ObjectTypes and VariableTypes1

VariableType ノードに、インスタンス化した Variable ノードがイベントソースになることを公開するにはどうすればよいか?

仕様

VariableType ノードから、インスタンス化した Variable ノードが発生の原因になり得るイベントのタイプを表す EventType ノードに対して GeneratesEvent(またはそのサブタイプ)の Reference を定義する。

注記 1： GeneratesEvent の Reference のソースノードに指定できる UA ノードは、ObjectType および VariableType、Method である。1

注記 2: GeneratesEvent の定義はオプションなので、そこで公開した以外のイベントタイプのイベントが発生することも仕様としては許される。1

注記 3: Variable ノードから直接にイベントを発生することはできない。その Variable ノードを参照する Object ノードが発行する。1

VariableType のみに適用できる(インスタンス化した Variable ノードには適用されない)Reference を定義するにはどうすればよいか?

仕様

VariantType ノードからその参照先の UA ノードに対する Reference の ModellingRule を定義しない。表記上は None とする。

VariableType からインスタンス化した Variable ノードに適用される Reference を定義するにはどうすればよいか?

仕様

VariantType ノードから参照先の UA ノードに対する Reference に何等かの ModellingRule を定義する。

VariableType ノードを記述するための表記法は何か?

推奨

VariableType ノードの定義を記述するには、次のテンプレートテーブルを使用する。

使用可能な属性と Reference については、次のドキュメントを参照。

- Part3 Address Space Model Release 1.04 : 5.6.5 VariableType NodeClass

表 5—VariableType 定義の例 3

Attribute	Value				
BrowseName	MyVariableType				
IsAbstract	True				
Value	<Optional. Specify when devalut value is to be specified for the instance Nodes. Otherwise, remove the line.>				
ValueRank	−2 (−2 = Any)				
ArrayDimensions	<Optional. Specify when ValueRank specifies non-abstract meaning. Otherwise, remove the line.>				
DataType	BaseDataType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *<MyParentVariableType> def*ined in <Clause number>.					
HasSubtype	VariableType	MyAnotherVariableType	Defined in <Clause number>		
HasProperty	Variable	MyProperty	UInt32	PropertyType	Mandatory
HasComponent	Variable	MyDataVariable	Duration	BaseDataVariableType	Optional
HasComponent	Variable	<MyDataVariableSet>	UInt32	BaseDataVariableType	Mandatory Placeholder
HasComponent	Variable	MyStaticVariable	Float	BaseDataVariableType	None
GeneratesEvent	ObjectType	MyEventType	Defined in <Clause number>		
<other Reference>	VariableTypes may contain other References. Modellingrule should be defined if th Reference can be instantiated by Variables defined by this VariableType. Define appropriate attribute columns depending to the NodeClass of the target.				
Notes − Notes referencing footnotes of the table content.					

- 前段の行では、VariableType ノードに必要な属性とその値を指定する。
    - VariableType の場合は、BrowseName、IsAbstract、ValueRank および DataType ノード属性の記述が必要である。デフォルト値が必要な場合

は Value ノード属性の記述が必要になる。ValueRank ノード属性の値が配列を示す場合には ArrayDimensions ノード属性の記述も必要になる。ただし UA サーバーを開発する際には他の必須ノード属性の定義も必要になる。

- 後段の行では、VariableType ノードからの Reference を記述する。
    - 親 VariableType ノードは、上の表の形式に倣って指定する。
    - VariableType の場合、Reference は主に HasSubtype、HasProperty、HasComponent 参照を指定する。GenerateEvent は、この VariableType からインスタンス化した Variable ノードが何らかのイベントを発生する原因になり得ることを示したい場合に指定する。
    - HasSubtype を指定した場合、上記と同じ形式の表を新たに用いてサブタイプの記述することが推奨される。
    - 'References'列は、使用可能な参照タイプ名を指定する。
    - 'NodeClass'列は、ターゲットノードの NodeClass ノード属性値を指定する。
    - 'BrowseName'列は、ターゲットノードの BrowseName ノード属性値を指定する。ターゲットノードの数が複数になり得るために BrowseName が固定できない場合、上記のように'<xxxxx>'という表記が許されている。xxxxx には参照先の UA ノードの意味を示す抽象的な名称を記載する。上記の例は、'ModellingRule'列が MandatoryPlaceholder なので複数の参照先 UA ノードを持つ可能性があるため、このような表記をしている。
    - 'DataType'列は、ターゲットノードが Variable または VariableType である場合、ターゲットノードの DataType ノード属性値を指定する。これは、「表 2-データタイプの例」に書かれた表記に従う。表 2-データタイプの例 1
    - 'TypeDefinition'列は、ターゲットノードの TypeDefinitionNode（ObjectType および VariableType ノード）の BrowseName ノード属性値を指定する。この列は、ターゲットノードが Variable または Object である場合に必要となる。
    - 'ModellingRule'列は、VariableType ノードが参照する UA ノードが InstanceDeclaration である場合に ModellingRule を含む。静的メンバーの場合は'None'を指定する。詳細については「2.11 ModellingRule」を参照。ModellingRule
    - Property および/または DataVariable のデフォルト値を定義する必要がある場合、表 5 の様式の表の下に各 Property および/または DataVariable について説明する項目を追記し、その中で Value ノード属性値についても記述することを推奨する。

2.5 Object

Object ノードは、システム、システムを構成する要素、現実世界の実体、ソフトウェアの実体などを表すために使用される。Object ノードは、その NodeClass ノード属性値が Object の UA ノードである。

Object ノードは、それをインスタンス化するときのテンプレートとなる ObjectType ノードを持つ。Object ノードは、テンプレートとなる ObjectType ノードと HasTypeDefinition の Reference で関連づける。

Object ノードはその要素として、Object、Variable、および/または Method の UA ノードを持つことができる。

参考文献

- Part3 Address Space Model Release 1.04 : 4.2 Object Model
- Part3 Address Space Model Release 1.04 : 5.5 Objects

2.5.1 Object の定義

どのよう場合に Object ノードを定義するのか?

仕様

1つの例は、AddressSpace に対して Object ノードをどのように配置するかを記述する場合である。'Objects' や' Types' などの OPC UA の標準ルートフォルダの下に、設計対象の UA ノードを階層構造で整理して配置する為のサブフォルダとなるフォルダノードを定義する場合がある。

もう1つの例は、InstanceDeclaration となる Object ノードを記述する場合である。InstanceDeclaration の Object ノードは、ObjectType ノードを設計するときに使用する。設計対象の ObjectType ノードをインスタンス化して Object ノードを生成するとき、その Object ノードが要素として保持すべきメンバーとなる Object ノードの情報を既定するものである。

注記 1: InstanceDeclaration の説明については「2.11 ModellingRule」を参照。1ModellingRule

Object を定義するにはどうすればよいか?

仕様

Object は下記の流れにて定義を行う。

対象の Object ノードを定義する。

Object ノードのメタ情報を指定するために HasTypeDefinition の Reference を定義し、その参照先を BaseObjectType(またはそのサブタイプ)を示す ObjectType ノードにする。

定義した Object ノードが持つ要素(Object、Variable、Method)や Reference は HasTypeDefinition で参照する ObjectType ノードに定義されているもののサブセットが基本である。しかし、ObjectType ノードに定義されていない要素や Reference を追加することも許されている。

Object ノードに要素や Reference を定義する方法は、ObjectType ノードに対して動的メンバーを定義する方法と、ModellingRule の指定が不要である点を除いて、同じである。「2.6 ObjectType」の動的メンバーの追加方法を参照。

定義した Object ノードを公開するにはどうすればよいか?

仕様

クライアントが利用可能なすべての Object ノードは、UA サーバーの AddressSpace 内に標準のルートフォルダとして決められている'Object' という名称(BrowseName ノード属性値)のフォルダ(FolderType という標準の ObjectType からインスタンス化した Object ノード)から直接または間接にアクセス可能であるように配置する。(図 1 参照)

フォルダの下に Object を配置する為には、フォルダと対象 Object に Reference を定義する。フォルダから対象 Object 対して Organizes(またはそのサブタイプ)の Reference を定義する。

フォルダも Object ノードであるため、フォルダにサブフォルダを配置する場合には、フォルダとサブフォルダの間に Organizes の Reference を定義すれば良い。

注記 1: 例外(‘Object’フォルダから直接または間接にアクセスできない場合)として、InstanceDeclaration として定義した Object ノードがある。InstanceDeclaration については「2.11 ModellingRule」を参照。

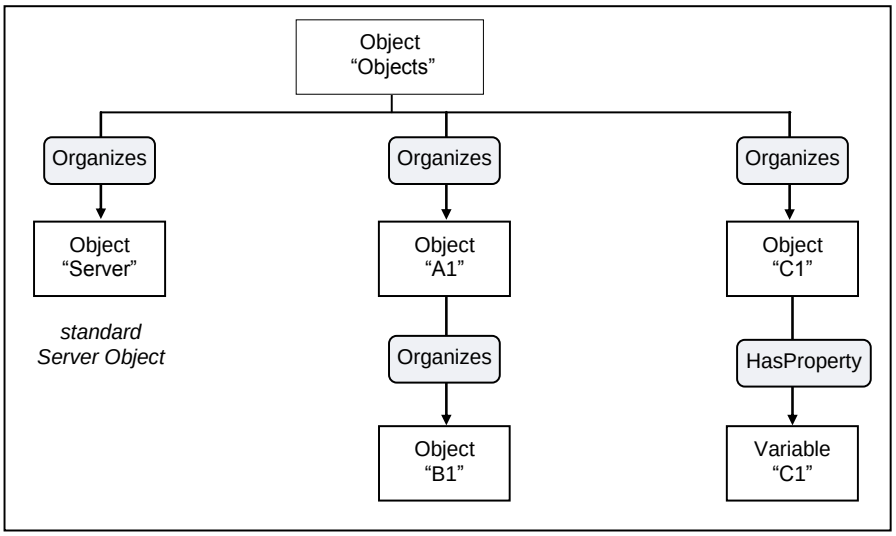


図 1 – Object ノードの配置 1

参考文献

- Part5 Information Model Release 1.04 : 8.2 Objects used to organise the AddressSpace structure

2.5.2 イベント

Object ノードからイベントが発行することを公開するにはどうすればよいか?

仕様

Object ノードの EventNotifier ノード属性に'SubscribeToEvents'を指定する。

対象の Object ノードがイベントの実際の発生元ではない場合、イベントを発行する UA ノードに対して HasEventSource または HasNotifier の Reference を定義する。一般的に、ターゲットノードがイベントを発行できない仕様の UA ノード(Variable や Method)の場合には HasEventSource で参照し、ターゲットノード(Object や View)が発行するイベントを対象の Object ノードでも発行できるようにしたい場合には HasNotifier で参照する。

このように EventNotifier 属性と HasEventSource または HasNotifier の Reference を利用して、クライアントがイベント通知の受信を指定できる Object ノードと、実際のイベントソースを示す UA ノードからイベント通知を行う Object ノードまでのリンクを表現できる。

Object ノードがイベントヒストリアクセスをサポートしていることを公開するにはどうすればよいか?

仕様

Object ノードの EventNotifier ノード属性に HistoryRead および/または HistoryWrite ビットを設定する。

注記 1: Object ノードの EventNotifier 属性が SubscribeToEvents は未指定だが、HistoryRead および/または HistoryWrite を指定することも可能である。この場合、その Object ノードはイベントヒストリアクセスをサポートするが、イベントの発行はしないことを意味する。

Object ノードのイベントヒストリ構成を公開するにはどうすればよいか?

仕様

「3.4 Historical Access」を参照。Historical Access

2.5.3 Object ノードの表記

Object ノードを記述するための表記法は何か?

推奨

Object ノードの定義を記述するには、次のテンプレートテーブルを使用する。

使用可能な属性と Reference については、次のドキュメントを参照。

- Part3 Address Space Model Release 1.04 : 5.5.1 Object NodeClass

表 6-Object 定義の例 ： フォルダノード 4

Attribute	Value				
BrowseName	MyFolder				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Organized by the Objects Folder defined in **エラー! 参照元が見つかりません。**					
HasTypeDefinition	ObjectType	FolderType			
Organizes	Object	<MyObjectsSet>		BaseObjectType	
<other Reference>	Objects may contain other References. Modellingrule shall not be defined in case of defining Folder in AddressSpace.				
Notes – Notes referencing footnotes of the table content.					

表 7– Object 定義の例 ： InstanceDeclaration5

Attribute	Value				
BrowseName	MyInstanceDeclaration				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasTypeDefinition	ObjectType	BaseObjectType			
HasComponent	Variable	<MyDataVariableSet>		BaseDataVariableType	OptionalPlaceholder
<other Reference>	Objects may contain other References. Modellingrule should be defined if th Reference can be instantiated by Objects defined by this ObjectType. Define appropriate attribute columns depending to the NodeClass of the target.				
Notes – Notes referencing footnotes of the table content.					

- 前段の行では、Object ノードに必要な属性とその値を指定する。
    - Object の場合、BrowseName ノード属性の記述が必要である。ただし UA サーバーを開発する際には他の必須の属性も必要になる。
- 後段の行では、Object ノードからの Reference を記述する。
    - フォルダノードを定義する場合は、表 6 に記載してあるような形式で、始めに親となる Object ノード の BrowseName ノード属性値を記載する。続けて HasTypeDefinition （FolderType またはそのサブタイプとなる ObjectType の参照となる）と Organizes 参照を指定する。Organizes 参照のターゲットノードはサブフォルダを示す Object ノードか、そのフォルダが保持すべき Object ノードの種類を記載する。4
    - InstanceDeclaration を定義する場合は、表 7 に記載してあるような形式で、HasTypeDefinition、HasProperty、HasComponent を指定する。フォルダノードの場合とは異なり、InstanceDeclaration の場合は対象 Object ノードを保持する親の ObjectType ノードや Object ノードを記載する必要はない。なぜならばこの InstanceDeclaration を保持する側の定義で表記するべきものだからである。
    - 'References'列は、使用可能な参照タイプ名を指定する。

- 'NodeClass'列は、ターゲットノードの NodeClass ノード属性値を指定する。
- 'BrowseName'列は、ターゲットノードの BrowseName ノード属性値を指定する。ターゲットノードの数が複数になり得るために BrowseName が固定できない場合、上記のように'<xxxxx>'という表記が許されている。xxxxx には参照先の UA ノードの意味を示す抽象的な名称を記載する。上記の例は、'ModellingRule'列が OptionalPlaceholder なので複数の参照先 UA ノードを持つ可能性があるため、このような表記をしている。
- 'DataType'列は、ターゲットノードが Variable または VariableType である場合、ターゲットノードの DataType ノード属性値を指定する。これは、「表 2-データタイプの例」に書かれた表記に従う。表 2-データタイプの例 1
- 'TypeDefinition'列は、ターゲットの TypeDefinitionNode（ObjectType および VariableType ノード）の BrowseName ノード属性値を指定する。この列は、ターゲットノードが Variable または Object である場合に必要となる。
- 'ModellingRule'列は、Object ノードが参照する UA ノードが InstanceDeclaration である場合に ModellingRule を含む。静的メンバーの場合は'None'を指定する。詳細については「2.11 ModellingRule」を参照。ModellingRule
- Property および/または DataVariable のデフォルト値を定義する必要がある場合、表 6 または 7 の様式の表の下に各 Property および/または DataVariable について説明する項目を追記し、その中で Value ノード属性値についても記述することを推奨する。

2.6 ObjectType

ObjectType は、Object ノードのタイプ定義を表すために使用される。ObjectType ノードは、その NodeClass ノード属性値が ObjectType の UA ノードである。

ObjectType ノードは、Object ノードから HasTypeDefinition(またはそのサブタイプ)の Reference を介して参照される。

ObjectType ノードは、Object、Variable、および/または Method を要素として持つことができる。

ObjectType ノードの派生関係を表現する為のサブタイプ定義が可能。

参考文献

- Part3 Address Space Model Release 1.04： 4.5 TypeDefinitionNode
- Part3 Address Space Model Release 1.04： 5.5.2 ObjectType NodeClass
- Part3 Address Space Model Release 1.04： 5.5.3 Standard ObjectType FolderType
- Part3 Address Space Model Release 1.04 : 5.5.4 Client-side creation of Objects of an ObjectType

2.6.1 既定の ObjectType ノード

どのような標準 ObjectType ノードが、OPC UA 仕様書内で規定されているか?

仕様

コア仕様に関して、OPC UA 標準の ObjectType ノードは、以下の仕様書に列挙されている。

- Part5 Information Model Release 1.04 : 6 Standard ObjectTypes

他の標準 ObjectType ノードが、Part8 DataAccess、Part9 Alarm and Conditions など特定の機能ドメインに対するビルトインの情報モデルとして規定されている。ビルトイン情報モデル

推奨

設計対象と同じ意味の ObjectType がある場合は、相互運用性のために標準タイプを使用すべきである。

追加情報が必要な場合は、標準 ObjectType をサブタイプ化すべきである。

使用する予定のない情報がある場合でも、未使用情報にダミー値を指定するなどして、標準 ObjectType の使用を検討すべきである。

2.6.2 サブタイプ定義

ObjectType に継承関係を定義するにはどうすればよいか?

仕様

派生元の ObjectType ノードから派生先 ObjectType ノードに対して HasSubtype の Reference を定義する。

抽象タイプの ObjectType を定義するにはどうすればよいか?

仕様

IsAbstract ノード属性を True に指定する。

ObjectType を派生するとき、派生元 ObjectType の持つ要素(Object、Variable、Method)をオーバーライドするにはどうすればよいか?

仕様

下記の流れにて行う。

派生先の ObjectType ノードに、オーバーライド対象と同じ BrowseName と NodeClass ノード属性値をもつ要素(Object、Variable、Method)を定義する。

オーバーライドできるデータは、BrowseName と NodeClass 以外のノード属性値、TypeDefinition および/または ModellingRule である。

オーバーライドの対象でないデータに対しては派生元の要素(Object、Variable、Method)と同じ内容を再定義する必要がある。

オーバーライドで変更できる内容にはいくつかの制約がある。基本的な考え方は、オーバーライドされるデータは、派生元の要素(Object、Variable、Method)に定義されている内容のサブタイプに限定される。例として、Variable ノードをオーバーライドする場合、派生元の対象 Variable ノードの DataType 属性が Number である場合、派生先のオーバーライド対象 Variable ノードの DataType 属性に Integer を指定することはできるが、String は指定できない。詳細な規則については、参考文献を参照。

参考文献

- Part3 Address Space Model Release 1.04 : 6 Type Model for ObjectTypes and VariableTypes

2.6.3 サブタイプ定義 : Property

ObjectType ノードに動的なメンバーである Property を定義するにはどうすればよいか?

仕様

対象の Property を表す Variable ノードを定義する。「2.3 Variable」の「Property を定義するにはどうすればよいか？」を参照。

動的メンバーであることを示すためには、ObjectType ノードから対象 Variable ノードへの Reference を定義する際に、ModellingRule を定義する。Property の場合、ModellingRule は Mandatory または Optional と考えられる。

注記 1：ModellingRule の定義については、「2.11 ModellingRule」を参照。ModellingRule

ObjectType ノードに静的メンバーである Property を定義するにはどうすればよいか?

仕様

対象の Property を表す Variable ノードを定義する。「2.3 Variable」の「Property を定義するにはどうすればよいか？」を参照。

静的メンバーであることを示すためには、ObjectType ノードから対象 Variable ノードへの Reference を定義する際に、ModellingRule を定義しないこと。表記上は None とする。

2.6.4 サブタイプ定義：DataVariable

ObjectType ノードに動的なメンバーである DataVariable を定義するにはどうすればよいか?

仕様

対象の DataVariable を表す Variable ノードを定義する。「2.3 Variable」の「DataVariable を定義するにはどうすればよいか？」を参照。

動的メンバーであることを示すためには、ObjectType ノードから対象 Variable ノードへの Reference を定義する際に、ModellingRule を定義する。

注記 1：ModellingRule の定義については、「2.11 ModellingRule」を参照。ModellingRule

ObjectType ノードに静的メンバーである DataVariable を定義するにはどうすればよいか?

仕様

対象の DataVariable を表す Variable ノードを定義する。「2.3 Variable」の「DataVariable を定義するにはどうすればよいか？」を参照。

静的メンバーであることを示すためには、ObjectType ノードから対象 Variable ノードへの Reference を定義する際に、ModellingRule を定義しないこと。表記上は None とする。

2.6.5 サブタイプ定義：Method

ObjectType ノードに動的なメンバーである Method を定義するにはどうすればよいか?

仕様

対象の Method ノードを定義する。「2.9 Method」の「Method を定義するにはどうすればよいか？」を参照。

動的メンバーであることを示すためには、ObjectType ノードから対象 Method ノードへの Reference を定義する際に、ModellingRule を定義する。ModellingRule は Mandaatory または Optional であることが要求される。

注記 1：ModellingRule の定義については、「2.11 ModellingRule」を参照。ModellingRule

ObjectType ノードに静的なメンバーである Method を定義するにはどうすればよいか?

仕様

対象の Method ノードを定義する。「2.9 Method」の「Method を定義するにはどうすればよいか？」を参照。

静的メンバーであることを示すためには、ObjectType ノードから対象 Method ノードへの Reference を定義する際に、ModellingRule を定義しないこと。表記上は None とする。

2.6.6 サブタイプ定義：Object

ObjectType ノードに動的メンバーである Object を定義するにはどうすればよいか?

仕様

下記の流れにて行う。

対象の Object ノードを定義する。「2.5 Object」の「Object を定義するにはどうすればよいか？」を参照。

ObjectType ノードの要素に対象 Object ノードを追加するためには、ObjectType ノードから対象 Object ノードに対して HasComponent（またはそのサブタイプ）の Reference を定義する（注記 2）。

動的メンバーであることを示すためには、ObjectType ノードから対象 Object ノードへの Reference を定義する際に、ModellingRule を定義する。

注記 1： ModellingRule の定義については、「2.11 ModellingRule」を参照。ModellingRule

注記 2： Browse サービスで返される要素の順序を保証することを表現したい場合、HasOrderedComponent(またはそのサブタイプ)を定義する。

ObjectType ノードに静的メンバーである Object を定義するにはどうすればよいか?

仕様

下記の流れにて行う。

対象の Object ノードを定義する。「2.5 Object」の「Object を定義するにはどうすればよいか？」を参照。

ObjectType ノードの要素に対象 Object ノードを指定するためには、ObjectType ノードから対象 Object ノードに対して HasComponent（またはそのサブタイプ）の Reference を定義する（注記 2）。

静的メンバーであることを示すためには、ObjectType ノードから対象 Object ノードへの Reference を定義する際に、ModellingRule を定義しないこと。表記上は None とする。

注記 1： ModellingRule の定義については、「2.11 ModellingRule」を参照。ModellingRule

注記 2： Browse サービスで返される要素の順序を保証することを表現したい場合、HasOrderedComponent(またはそのサブタイプ)を定義する。

2.6.7 サブタイプ定義：Reference

ObjectType のみに適用される(インスタンス化した Object には適用されない)参照関係を定義するには、どうすればよいか?

仕様

ObjectType ノードに定義した Reference の ModellingRule を定義しないこと。表記上は None とする。

ObjectType からインスタンス化した Object にも適用される参照関係を定義するにはどうすればよいか?

仕様

ObjectType ノードに定義した Reference に ModellingRule を定義する。

注記 1：ModellingRule の定義については、「2.11 ModellingRule」を参照。ModellingRule

他 UA ノードとの関連を表現するのに、Reference を使用すべきか否かの判断基準は何か?

推奨

「2.2 DataType」の「2.2.1 既存の DataType ノード」の「NodeId/ExpandNodeId 型の指定と Reference による参照を選択する基準は何か？」 を参照。

2.6.8 サブタイプ定義：イベント

インスタンス化した Object ノードからイベントが発行することを定義するにはどうすればよいか?

仕様

ObjectType ノードから EventType ノードに対して GeneratesEvent(またはそのサブタイプ)の Reference を定義する。

注記 1： GeneratesEvent の Reference のソースノードに指定できる UA ノードは、ObjectType および VariableType、Method である。1

注記 2: GeneratesEvent の定義はオプションなので、そこで公開したイベントタイプ以外のイベントが発生することも仕様としては許される。1

注記 3: 発行されたイベントが一過性のものではなく、ある条件が満たされるまで保持されるものであることを定義したい場合には EventType の定義で対応する。その定義方法については'3.2 Alarm & Conditions'を参照。**エラー! 参照元が見つかりません。**

2.6.9 ObjectType ノードの表記

ObjectType ノードを記述するための表記法は何か?

推奨

ObjectType ノードの定義を記述するには、以下のテンプレートテーブルを使用する。

使用可能な属性と Reference については、次のドキュメントを参照。

- Part3 Address Space Model Release 1.04 : 5.5.2ObjectType NodeClass

表 8—ObjectType 定義の例

Attribute	**Value**				
BrowseName	MyObjectType				
IsAbstract	True				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *<MyParentObjectType> de*fined in <Clause number>.					
HasSubtype	ObjectType	MyAnotherObjectType	Defined in <Clause number>		
HasProperty	Variable	MyProperty	UInt32	PropertyType	Mandatory
HasComponent	Variable	MyDataVariable	Duration	BaseDataVariableType	Optional
HasComponent	Variable	<MyDataVariableSet>	UInt32	BaseDataVariableType	Mandatory Placeholder
HasComponent	Method	MyDynamicMethod	Defined in <Clause number>		Mandatory
HasComponent	Method	MyStaticMethod	Defined in <Clause number>		None
GeneratesEvent	ObjectType	MyEventType	Defined in <Clause number>		
<other Reference>	ObjectTypes may contain other References. Modellingrule should be defined if th Reference can be instantiated by Objects defined by this ObjectType. Define appropriate attribute columns depending to the NodeClass of the target.				
Notes – Notes referencing footnotes of the table content.					

- 前段の行では、ObjectType ノードに必要な属性とその値を指定する。
    - ObjectType の場合、BrowseName および IsAbstract ノード属性の記述が必要である。ただし UA サーバーを開発する際には他の必須の属性も必要になる。
- 後段の行では、ObjectType ノードからの Reference を記述する。
    - 親 ObjectType ノードは、上の表の形式に倣って指定する。
    - ObjectType の場合、Reference は主に HasSubtype、HasProperty、HasComponent 参照を指定する。GeneratesEvent は、この ObjectType からインスタンス化した Object ノードから何らかのタイプのイベントを発行する原因になり得ることを示したい場合に指定する。
    - HasSubtype を指定した場合、上記と同じ形式の表を新たに用いてサブタイプを記述することが推奨される。
    - 'References'列は、使用可能な参照タイプ名を指定する。

- 'NodeClass'列は、ターゲットノードの NodeClass ノード属性値を指定する。
- 'BrowseName'列は、ターゲットノードの BrowseName ノード属性値を指定する。ターゲットノードの数が複数になり得るために BrowseName が固定できない場合、上記のように'<xxxxx>'という表記が許されている。xxxxx には参照先の UA ノードの意味を示す抽象的な名称を記載する。上記の例は、'ModellingRule'列が MandatoryPlaceholder なので複数の参照先 UA ノードを持つ可能性があるため、このような表記をしている。**エラー! 参照元が見つかりません。**
- 'DataType'列は、ターゲットノードが Variable または VariableType である場合、ターゲットノードの DataType ノード属性値を指定する。これは、「表 2–データタイプの例」に書かれた表記に従う。表 2–データタイプの例 1
- 'TypeDefinition'列は、ターゲットの TypeDefinitionNode（ObjectType および VariableType ノード）の BrowseName ノード属性値を指定する。この列は、ターゲットノードが Variable または Object である場合に必要となる。
- 'ModellingRule'列は、ObjectType ノードが参照する UA ノードが InstanceDeclaration である場合に ModellingRule を含む。静的メンバーの場合は'None'を指定する。詳細については「2.11 ModellingRule」を参照。ModellingRule
- Property および/または DataVariable のデフォルト値を定義する必要がある場合、表 8 の様式の表の下に各 Property および/または DataVariable について説明する項目を追記し、その中で Value ノード属性値についても記述することを推奨する。

2.7 EventType

EventType は、イベントのタイプ定義を表すために使用される。EventType は ObjectType のサブタイプである。EventType を表す UA ノードはその NodeClass ノード属性値が ObjectType である。

イベントを発行できる UA ノードは Object ノードおよび View ノードのみであり、どちらも EventNotifier ノード属性が存在する。クライアントはこのノード属性に SubscribeToEvents というビット値が設定されている UA ノードからのみ、イベント発行を待ち受けることができる。

発行されるイベントを表す Object ノードは、HasTypeDefinition(またはそのサブタイプ)の Reference で EventType を参照する。EventType は BaseEventType という標準 ObjectType のサブタイプである。

EventType ノードの派生関係を表現する為のサブタイプ定義が可能。

注記 1：EventType 自体は ObjectType から派生しているため、「2.6 ObjectType」も参照。ModellingRule

参考文献

- Part3 Address Space Model Release 1.04 : 4.6 Event Model

2.7.1 既定の EventType ノード

どのような標準 EventType ノードが、OPC UA 仕様書内で規定されているか?

仕様

コア仕様に関して、OPC UA 標準の EventType ノードは、以下の仕様書に列挙されている。

- Part3 Address Space Model Release 1.04 : 9 Standard EventTypes
- Part5 Information Model Release 1.04 : 6.4 ObjectTypes used as EventTypes

- Part5 Information Model Release 1.04 : :B.4.16 TransitionEventType
- Part5 Information Model Release 1.04 : :B.4.17 AuditUpdateStateEventType
- Part5 Information Model Release 1.04 : :F.4 RoleMappingRuleChangedAuditEventType

他の標準 EventType ノードが、Part8 DataAccess、Part9 Alarm and Conditions など特定の機能ドメインに対するビルトインの情報モデルとして規定されている。ビルトイン情報モデル

推奨

設計対象と同じ意味の EventType がある場合は、相互運用性のために標準タイプを使用すべきである。

追加情報が必要な場合は、標準 EventType のサブタイプを定義する。

使用する予定のない情報がある場合でも、未使用情報にダミー値を指定するなどして、標準 EventType の使用を検討すべきである。

2.7.2 サブタイプ定義

※ 「2.6 ObjectType」の「2.6.2 サブタイプ定義」の内容も参照。

EventType をサブタイプ定義する場合に名前規則はあるか?

推奨

「EventType」という文字列を最後に付記することが暗黙の慣例になっている(例： xxxxxxxEventType)。

定義する EventType が監査証跡用である場合は上記に加えて「AuditTrail」という文字列から始める例が多い(例： AuditTrailxxxxxEventType)。

EventType のサブタイプ定義で追加すべき要素はなにか?

推奨

一般にイベントを表す Object ノード(HasTypeDefinition の Reference が BaseEventType またはそのサブタイプを参照する Object ノード)は、一過性の事象の発生を表すために用いられる。その Object ノードは AddressSpace 上に公開されておらず、Borwse サービスで検索されるものではなく、その Object ノードに対して Method を呼び出すようなユースケースも存在しない。この特性のため、新しく EventType のサブタイプを定義する場合、通常は EventType への Property 追加のみを行う。その用途は、イベントの通知を認識した UA クライアントが追加 Prpperty 情報を UA クライアントアプリケーション内部で確認することとなる。

ただし例外があり、'Part9 Alarms & Conditions Release 1.04'で紹介されている Condition がそれに該当する。その ObjectType が ConditionType(およびそのサブタイプ)であるイベントが発生すると、その Object ノードは ConditionType で定義されている EnabledState という Property の値が False になるまでは AddressSpace に存在する。したがって、ConditionType から派生した EventType の定義においては、Object、Variable、および/または Method などの UA ノードもその要素として定義することができる。

EventType を記述するための表記法は何か?

推奨

EventType は、ObjectType のサブタイプである。したがって、その表記法は「2.6.9 ObjectType ノードの表記」に記載されているものと同じ様式になる。ObjectType ノードを記述するための表記法は何か?

2.8 ReferenceType

Reference は UA ノードを互いに関連付けるために使用される。参照する側の UA ノードはソースノードと呼ばれ、参照される側の UA ノードはターゲットノードと呼ばれる。複数の UA ノードが参照関係によって関連付けられて AddressSpace を構築し、UA クライアントは Browse サービスや QueryFirst サービスなどを用いてそれらの UA ノードにアクセスする(注記 1)。

Reference は、ReferenceType ノードのインスタンスとして定義されるがそれ自体は UA ノードではなく、ソースノードとターゲットノードの間に存在して、UA ノード間の参照関係を構築する為の独立した機構である。対して、ReferenceType ノードは、その NodeClass ノード属性値が ReferenceType である UA ノードであり、UA ノード間の参照関係の意味を定義する。

ReferenceType ノードの派生関係を表現する為のサブタイプ定義が可能で、親の ReferenceType が表現する意味を更に特化した意味を定義できる。

注記 1：Browse サービスと QueryFirst サービスの違いを記す。Browse サービスは指定した UA ノードが直接に Reference のソースノードおよび/またはターゲットノードとなっている対象の UA ノードしか抽出することができない。対して、QueryFirst サービスは指定した UA ノードから複数の Reference を介してでも辿ることのできる参照先および/または参照元も検索対象として抽出される。ModellingRule

参考文献

- Part3 Address Space Model Release 1.04 : 4.3.4 References
- Part3 Address Space Model Release 1.04 : 5.3 ReferenceType NodeClass
- Part3 Address Space Model Release 1.04 : A.7 Definint ReferenceTypes

2.8.1 既定の ReferenceType

どのような標準 ReferenceType ノードが、OPC UA 仕様書内で規定されているか?

仕様

コア仕様に関して、OPC UA 標準の ReferenceType ノードは、以下の仕様書に列挙されている。

- Part3 Address Space Model Release 1.04 : 7 Standard ReferenceTypes
- Part5 Information Model Release 1.04: 11 Standard ReferenceTypes

参考として’Part3 Address Space Model Release 1.04 : 7 Standard ReferenceTypes’に記載されている図を下記に示す。

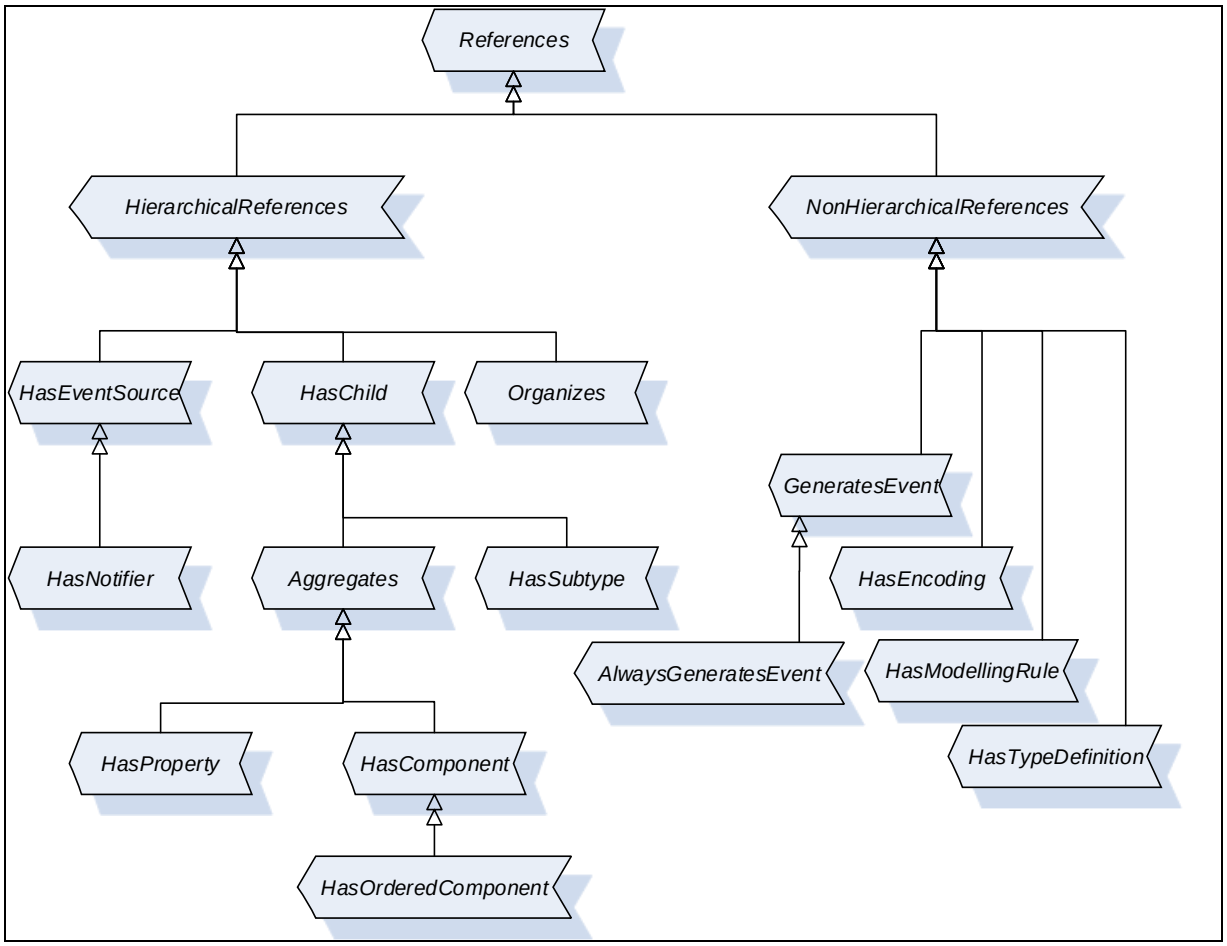


図 2 標準 ReferenceType の階層 2

他の標準 ReferenceType ノードが、Part8 DataAccess、Part9 Alarm and Conditions など特定の機能ドメインに対するビルトインの情報モデルとして規定されている。ビルトイン情報モデル

推奨

設計対象と同じ意味の ReferenceType がある場合は、相互運用性のために標準タイプを使用すべきである。

2.8.2 サブタイプ定義

新たに ReferenceType をサブタイプ定義する動機は何か?

推奨

Reference に新たに特化した意味を定義することによって AddressSpace 内の UA ノードの検索が効果的であると判断される場合に、その特化した意味を表現する ReferenceType のサブタイプ定義を検討する。特化した ReferenceType ノードが無い場合は汎用的な意味を持つ上位の ReferenceType で検索した後に UA クライアント側で独自にフィルタリングするロジックを実装することが考えられるが、そのような手間を行う頻度と ReferenceType ノードが増えることによる煩雑さのどちらが許容できるのかで判断するのだろう。

ReferenceType に継承関係を定義するにはどうすればよいか?

仕様

派生元の ReferenceType ノードから派生先 ReferenceType ノードに対して HasSubtype の Reference を定義する。

抽象タイプの ReferenceType を定義するにはどうすればよいか?

仕様

IsAbstract ノード属性を True に指定する。

新たに抽象タイプの ReferenceType をサブタイプ定義する動機は何か?

推奨

AddressSpace 内の検索が、新たに定義した ReferenceType を利用することによって効果的であると判断される場合に、その特化した意味を表現する ReferenceType のサブタイプ定義を検討する。

ユースケースの例:

UA ノードを検索するための Browse サービスや QueryFirst サービスには'includeSubtypes'というパラメータが存在する。その値を True に指定すると検索対象に指定した ReferenceType およびそこから派生した全ての ReferenceType の Reference を対象に検索を行うことができる。False を指定すると指定した ReferenceType のみが検索対象になる。特化した ReferenceType を利用することが有効と思えるユースケースが多いと判断されるならば ReferenceType のサブタイプ定義を検討すると良い。

ReferenceType をサブタイプ定義する場合に名前規則はあるか?

推奨

ReferenceType に特定の名前規則はない。

慣例的には、ソースノードを主語、ターゲットノードを対象と考えた場合の動詞(必要ならばその補助となる目的語)になる言葉を、その ReferenceType の BrowseName ノード属性値に設定しているものが多い。その場合に助詞(英語では前置詞)を含める例はない。

動詞のみの例 : Organaizes

動詞＋目的語の例 : HasComponent

ReferenceType ノードを記述するための表記法は何か?

推奨

ReferenceType ノードの定義を記述するには、以下のテンプレートテーブルを使用する。

使用可能な属性と Reference については、次のドキュメントを参照。

- Part3 Address Space Model Release 1.04 : 5.3 ReferenceType NodeClass

表 9—ReferenceType 定義の例 6

Attribute	Value				
BrowseName	MyReferenceType				
IsAbstract	True				
Symmetric	False				
InverseName	<Specify if Symmetric Attribute Value is True>				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *<MyParentReferenceType> de*fined in <Clause number>.					
HasSubtype	ReferenceType	MyAnotherReferenceType	Defined in <Clause number>		
HasProperty	Variable	MyProperty	UInt32	PropertyType	None
Notes – Notes referencing footnotes of the table content.					

- 前段の行では、ReferenceType ノードに必要な属性とその値を指定する。

- ReferenceType の場合、BrowseName、IsAbstract、Symmetric、および（Symmetric ノード属性値が True の場合に限り）InverseName ノード属性の記述が必要である。ただし UA サーバーを開発する際には他の必須の属性も必要になる。

- 後段の行では、ReferenceType ノードからの Reference を記述する。
  - 親 ReferenceType ノードは、上の表の形式に倣って指定する。
  - ReferenceType の場合、Reference は主に HasSubtype 参照を指定する。HasProperty 参照を定義する場合は、ReferenceType に特性情報を追加することに留意すること（Reference は UA ノードではないので Property は持てない）。
  - HasSubtype を指定した場合、上記と同じ形式の表を新たに用いてサブタイプを記述することが推奨される。
  - 'References'列は、使用可能な参照タイプ名を指定する。
  - 'NodeClass'列は、ターゲットノードの NodeClass ノード属性値を指定する。
  - 'BrowseName'列は、ターゲットノードの BrowseName ノード属性値を指定する。
  - 'DataType'列は、ターゲットノードが Variable または VariableType である場合、ターゲットノードの DataType ノード属性値を指定する。これは、「表 2-データタイプの例」に書かれた表記に従う。表 2-データタイプの例 1
  - 'TypeDefinition'列は、ターゲットの TypeDefinitionNode（VariableType ノード）の BrowseName ノード属性値を指定する。この列は、ターゲットノードが Property である場合に必要となる。ReferenceType は Property のみを要素に持つことができるため、'TypeDefinitoin'は PropertyType またはそのサブタイプとなる。
  - ReferenceType の場合、Property を定義する場合の'ModellingRule'列は常に'None'と指定する。Property は常に ReferenceType ノードの静的メンバーとなるためである。
  - Property のデフォルト値を定義する必要がある場合、表 9 の様式の表の下に各 Property について説明する項目を追記し、その中で Value ノード属性値についても記述することを推奨する。

2.9 Method

Method は、Method ノードを要素に持つ Object ノードや ObjectType ノードに限定された範囲で、軽量な関数処理を行うために使用される。Method ノードは、NodeClass ノード属性値が Method の UA ノードである。

Method ノードを要素に持てる UA ノードは Object および ObjectType のみである。

UA クライアントが Call サービスを実行することで Method が起動され、UA サーバー上で関数処理が行われる。関数処理を完了した後に、UA サーバーは関数処理の結果を UA クライアントに返す。

Method は 0 個以上の入力引数および出力引数などのメタデータを持つことができる。

参考文献

- Part3 Address Space Model Release 1.04 : 4.7 Methods
- Part3 Address Space Model Release 1.04 : 5.7 Method 1NodeClass

Method を定義するにはどうすればよいか?

仕様

下記の流れにて行う。

対象の Method ノードを定義する。Method の BrowseName ノード属性値は、その Method を持つ親 UA ノードの要素の中で一意でなければならない。引数が必要な場合は InputArguments および/または OutputArguments という Propaerty を定義する 。

Method ノードを持つ親 UA ノード(Object ノードまたは ObjectType ノード)と対象 Method ノードの参照関係を設定する。Method ノードを持つ親 UA ノードから対象 Method ノードに対して HasComponent(またはそのサブタイプ)の Reference を定義する(注記 2)。

注記 1: Method を ObjectType に定義する場合、Mandatory または Optional の ModellingRule の定義を考慮しなければならない。ModellingRule の定義については、「2.11 ModellingRule」を参照。

注記 2: Browse サービスで返される要素の順序を保証することを表現したい場合、HasOrderedComponent(またはそのサブタイプ)を定義する。

ObjectType に Method を定義する場合にどのような ModellingRule を定義すべきか?

仕様

Method を静的にする(ObjectType に対してのみ呼出し可能な Method であることを表す)ためには、ModellingRule を定義しないこと。表記上は None とする。

Method を動的にする(インスタンス化した Object ノードが Method を持つことを表す)ためには、ModellingRule を Mandatory または Optional として定義すること。Optional とした場合、インスタンス化した Object ノードが Method を持つか否かは UA サーバーの仕様による。

Method ノードに、その関数処理の実行がイベントソースになることを公開するにはどうすればよいか?

仕様

Method ノードから EventType ノード(関数処理がイベント発生の原因になり得るイベントのタイプを表す)に対して GeneratesEvent(またはそのサブタイプ)の Reference を定義する。特に、Method が表す関数処理の実行中に常に発生するイベントの場合は、AlwaysGeneratesEvents(またはそのサブタイプ)リファレンスを定義する。

注記 1: GeneratesEvent の Reference のソースノードに指定できる UA ノードは、ObjectType および VariableType、Method である。AlwaysGeneratesEvents の Reference のソースノードに指定できる UA ノードは Method のみである。1

注記 2: GeneratesEvent の定義はオプションなので、そこで公開したイベントタイプ以外のイベントが発生することも仕様としては許される。1

注記 3: Method ノードから直接にイベントを発生することはできない。その Method ノードを要素に持つ Object ノードがイベントを発行する。とくに Method ノードが静的メンバーである場合、その親となる ObjectType ノードはイベントを発行できない仕様になっているので留意が必要。1

Method ノードを記述するための表記法は何か?

推奨

Method ノードが表すものは関数処理であるため、慣例としてそのファンクションシグニチャと引数の説明を始めに記載している。以下はその例。

```
FetchTransferResultData(
  [in] Int32              TransferID,
  [in] Int32              SequenceNumber,
  [in] Int32              MaxParameterResultsToReturn,
  [in] Boolean             OmitGoodResults,
  [out] FetchResultType       FetchResultData);
```

Table 10 –FetchTransferResultData Method Arguments

Argument	Description
TransferID	Transfer Identifier returned from TransferTo- or TransferFromDevice.
SequenceNumber	The sequence number being acknowledged. The *Server* may delete the result set with this sequence number. "0" is used in the first call after initialising a transfer and also if the previous call of *FetchTransferResultData* failed.
MaxParameterResultsToReturn	The number of *Parameters* in *TransferResult.ParameterDefs* that the *Client* wants the *Server* to return in the response. The *Server* is allowed to further limit the response, but shall not exceed this limit. A value of 0 indicates that the *Client* is imposing no limitation.
OmitGoodResults	If TRUE, the Server will omit data for Parameters which have been correctly transferred. Note that this causes all good results to be released.
FetchResultData	Two sub-types are possible: • TransferResultError Type – is returned if the Transfer failed completely • TransferResultData Type – is returned if the Transfer was performed. Status information is returned for each transferred *Parameter*.

ファンクションシグニチャの後に、Method ノードの持つ属性や Reference を定義する表を付記する。上記の処理関数を表す Method ノードの場合の定義例をつぎに示す。

Table 11 – FetchTransferResultData Method AddressSpace Definition

Attribute	Value				
BrowseName	FetchTransferResultData				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

以降に、上記の Method ノード定義用のテンプレートテーブルについて記す。

使用可能な属性と Reference については、次のドキュメントを参照。

- Part3 Address Space Model Release 1.04 : 5.7 Method NodeClass

表 12—Method 定義の例 7

Attribute	Value				
BrowseName	MyMethod				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory
GeneratesEvent	ObjectType	MyEventType	Defined in <Clause number>		
AlwaysGeneratesEvent	ObjectType	MyAnotherEventType	Defined in <Clause number>		
<other Reference>	Methods may contain other References. Modellingrule should be defined if th Reference can be instantiated by Mehods defined by this InstanceDeclaration Method. Define appropriate attribute columns depending to the NodeClass of the target.				
Notes – Notes referencing footnotes of the table content.					

- 前段の行では、Method ノードに必要な属性とその値を指定する。
    - Method の場合、BrowseName ノード属性の記述が必要である。ただし UA サーバーを開発する際には他の必須の属性も必要になる。
- 後段の行では、Method ノードからの Reference を記述する。

- 要に応じて定義可能な ReferenceType を指定し、Method ノードの Reference 情報を定義することができる。

  入力引数および出力引数を表す必要がある場合には InputArguments と OutputArguments という BrowseName ノード属性値の Property を指定する。

  Method の表す関数処理を実行すると、何らかのイベントを発行すると予想される場合、GenerateEvent および/または AlwaysGenerateEvent を任意に指定することができる。

  ModellingRule を指定することで、この Method を要素に持つ ObjectType ノードから Object ノードをインスタンス化する場合に、その Object ノードも同じ ReferenceType の Reference を持ち得ることを表現することができる。（ModellingRule の定義については、「2.11 ModellingRule」を参照）。ModellingRule

- References'列は、使用可能な参照タイプ名を指定する。

- 'NodeClass'列は、ターゲットノードの NodeClass ノード属性値を指定する。

- 'BrowseName'列は、ターゲットノードの BrowseName ノード属性値を指定する。Method の場合、'InputArguments'および'OutputArguments'という BrowseName ノード属性値を持つ標準 Property を定義することが多い。ターゲットノードの数が複数になり得るために BrowseName が固定できない場合、'<xxxxx>'という表記が許されている。xxxxx には参照先の UA ノードの意味を示す抽象的な名称を記載する。**エラー! 参照元が見つかりません。**

- 'DataType'列は、ターゲットノードが Variable または VariableType である場合、ターゲットノードの DataType ノード属性値を指定する。これは、「表 2–データタイプの例」に書かれた表記に従う。表 2–データタイプの例 1

- 'TypeDefinition'列は、ターゲットの TypeDefinitionNode（ObjectType および VariableType ノード）の BrowseName ノード属性値を指定する。この列は、ターゲットノードが Variable または Object である場合に必要となる。Method を定義する場合、'InputArguments'および'OutputArguments'という標準の Property が定義されることが多く、それらの DataType ノード属性値は'PropertyType'である。

- 'ModellingRule'列は、Method ノードが参照する UA ノードが InstanceDeclaration である場合に ModellingRule を含む。静的メンバーの場合は'None'を指定する。詳細については「2.11 ModellingRule」を参照。Method を定義する場合、'InputArguments'および'OutputArguments'という標準の Property を定義することが多く、それらの ModellingRule は'Mandatory'である。引数が必要でない場合は、これらの Property は定義しない。ModellingRule

- Property および/または DataVariable のデフォルト値を定義する必要がある場合、表 12 の様式の表の下に各 Property および/または DataVariable について説明する項目を追記し、その中で Value ノード属性値についても記述することを推奨する。

2.10 View

View ノードは、AddressSpace 内の UA ノードを閲覧することによる負担を回避するために、AddressSpace 内の UA ノードのサブセットを定義するために使用される。View ノードは、NodeClass ノード属性値が View の UA ノードである。

View ノードは、Browse サービスおよび QueryFirst サービスの呼び出し時のパラメータとして利用される。この場合、各サービスが検索対象にする UA ノードは、View ノードに指定されたものに限定される。View ノードが指定されていない場合、AddressSpace 全体がデフォルトビューとして検索対象になる。

View 内に定義する UA ノードは、AddressSpace 内に定義されている全ての Reference のサブセットに限定される。AddressSpace 内に存在しない Reference を View で定義した空間内に追加することはできない。

参考文献

- Part3 Address Space Model Release 1.04 : 5.4 View NodeClass

2.10.1 既定の View

どのような標準 View ノードが、OPC UA 仕様書内で規定されているか?

仕様

OPC UA コア仕様で定義されている標準の View ノードは存在しない。

2.10.2 View の定義

新たに View を定義する動機は何か?

推奨

デフォルトの AddressSpace 内に存在する UA ノードに対して、何らかの観点によって可視できる UA ノードの範囲を限定したい場合、その観点ごとに新しい View ノードを定義する。

View を定義するにはどうすればよいか?

仕様

下記の流れにて行う。

View ノードとして、NodeClass ノード属性値が View である UA ノードを定義する。この View ノードが特定の観点に可視範囲を限定した AddressSpace のルート UA ノードとして機能する。

作成した View ノードから、限定して公開したい AdderssSpace のトップレベルの UA ノードに対して、ReferenceType が HierarchicalReference(またはそのサブタイプ)である Reference を作成する。ここでトップレベルの UA ノードを何にするかは View の設計に委ねるが、'Server'オブジェクトをトップレベルの UA ノードに指定することは許されない。(注記 1)

トップレベルの UA ノードから Reference を介して参照されている UA ノードのうちどれがその View が示す AddressSpace の対象であるかを情報モデルとして公開するルールは存在しない。どの UA ノードが Browse サービスや QueryFirst サービスの結果として返されるかは、そのサービスを実装する UA サーバーの設計に依存する。

上記で決定したトップレベルの UA ノードから、Reference が示す参照を前方方向に辿っていった場合に再びそのトップレベルの UA ノードに行き着く場合、View ノードの ContainsNoLoops ノード属性に False を指定しなければならない。そのような循環ループが存在しないならばこのノード属性値に True を指定する。

注記 1: View ノードは任意の HierarchicalReference(またはそのサブタイプ)である Reference のソースノードになり得るが、NonHierarchicalReference のソースノードにしてはならない。1

定義した View を公開するにはどうすればよいか?

仕様

クライアントが利用可能なすべての View ノードは、UA サーバーの AddressSpace 内に標準のルートフォルダとして決められている'Views'という BrowseName ノード属性値のフォルダノード(FolderType という標準の ObjectType からインスタンス化した Object ノード)から直接または間接にアクセス可能であるように配置する。(図 3 参照)

推奨

図 3 では、'Views'フォルダノードから直接に View ノードを参照する場合と、Object ノードを介した間接参照で View ノードを参照している場合を示している。ここでの'Engineering'という Object ノードも Organizes で参照されていることからフォルダであると推測できる。基本的に View ノードは'Views'フォルダまたはそのサブフォルダから参照される構成で公開することが望ましい。

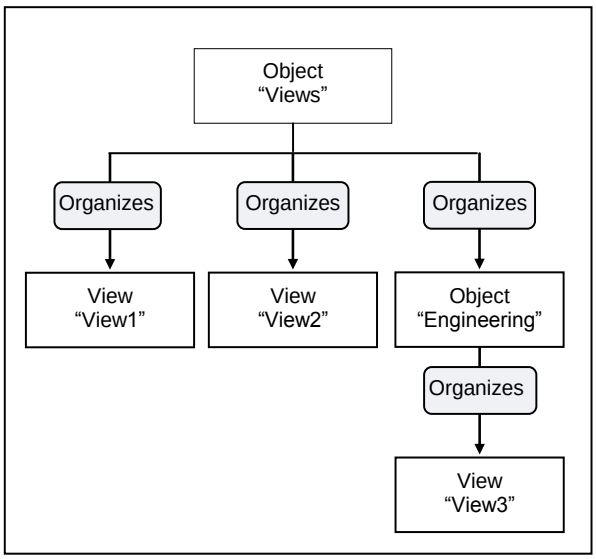


図 3-View ノードの配置 3

アクセス制限手段として、View ノードを用いる場合と、AccessLevel および/または Role を用いる場合の選択基準は何か?

仕様

これらの機能は根本的に考え方が異なる。

View ノードは AddressSpace 内のサブセットに特定の観点を定義する。したがって、Browse サービスや QueryFirst サービスのパラメータに View が指定されている場合、View に定義されていない UA ノードが返ることは有り得ない。

他の機能は、UA クライアントの種別および/またはサーバーの状態に基づいて、UA ノードに対するサービス呼び出しの可否を制御する。それらの目的は、UA ノードを隠蔽することではない。

2.10.3 イベント

View ノードからイベントが発行することを公開するにはどうすればよいか?

仕様

View ノードの EventNotifier ノード属性に'SubscribeToEvents'を指定する。

View ノードがイベントの実際の発生元ではない場合、イベントを発行する UA ノードに対して HasEventSource または HasNotifier の Reference を定義する。一般的に、ターゲットノードがイベントを発行できない仕様の UA ノード(Variable や Method)の場合には HasEventSource で参照し、ターゲットノード(Object や View)が発行するイベントを対象の View ノードでも発行できるようにしたい場合には HasNotifier で参照する。

View ノードがイベントヒストリアクセスをサポートしていることを公開するにはどうすればよいか?

仕様

View ノードの EventNotifier ノード属性に HistoryRead および/または HistoryWrite ビットを設定する。

注記 1: View ノードの EventNotifier ノード属性に対して、SubscribeToEvents は未指定だが、HistoryRead および/または HistoryWrite を指定することが可能である。この場合、その View ノードはイベントヒストリアクセスをサポートするが、イベントの発行はしないことを意味する。

View ノードのイベントヒストリ構成を公開するにはどうすればよいか?

仕様

「3.4 Historical Access」を参照。Historical Access

2.10.4 View ノードの表記

View ノードを記述するための表記法は何か?

推奨

View ノードの定義を記述するには、次のテンプレートテーブルを使用する。

使用可能な属性と Reference については、次のドキュメントを参照。

- Part3 Address Space Model Release 1.04 : 5.4 View NodeClass

表 13–View 定義の例 8

Attribute				
BrowseName				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**
<Subtype of HierarchicalReferences>	ObjectType	<MyObjectsSet>		BaseObjectType
HasProperty	Variable	ViewVersion	UInt32	PropertyType

- 前段の行では、ObjectType ノードに必要な属性とその値を指定する。
    - View の場合、BrowseName ノード属性の記述が必要である。ただし UA サーバーを開発する際には他の必須の属性も必要になる。
- 後段の行では、View ノードからの Reference を記述する。
    - View の場合、References は主に HierarchicalReferences のサブタイプであり、それには HasProperty も含まれる。HasProperty 以外で許される HierarchicalReferences のサブタイプは View ノードが示す AdderssSpace のトップレベルの UA ノードに関連する。
    - 'References'列は、使用可能な参照タイプ名を指定する。
    - 'NodeClass'列は、ターゲットノードの NodeClass ノード属性値を指定する。
    - 'BrowseName'列は、ターゲットノードの BrowseName ノード属性値を指定する。ターゲットノードの数が複数になり得るために BrowseName が固定できない場合、'<xxxxx>'という表記が許され、xxxxx に参照先の UA ノードの意味を示す抽象的な名称を記載する。
    - 'DataType'列は、ターゲットノードが Variable または VariableType である場合、ターゲットノードの DataType ノード属性値を指定する。これは、「表 2–データタイプの例」に書かれた表記に従う。表 2–データタイプの例 1

- 'TypeDefinition'列は、ターゲットの TypeDefinitionNode（ObjectType および VariableType ノード）の BrowseName ノード属性値を指定する。この列は、ターゲットノードが Variable または Object である場合に必要となる。

- Property のデフォルト値を定義する必要がある場合、表 13 の様式の表の下に各 Property について説明する項目を追記し、その中で Value ノード属性値についても記述することを推奨する。

2.11 ModellingRule

ModellingRule は、InstanceDeclaration（後述）のメタデータであり、InstanceDeclaration として指定された UA ノードがインスタンス化の際にどのように生成され得るかを定義する。

InstanceDeclaration とは構造的には、HasModellingRule タイプの Reference で ModellingRule という Object ノード（その ObjectType は ModellingRuleType）を参照する Object ノード、Variable ノード、または Method ノードである。

InstanceDeclaration となる UA ノードは、TypeDefinitionNode(後述)または別の InstanceDeclaration から HierarchicalReference のサブタイプの Reference で参照されるターゲットノードである。

TypeDefinitionNode は ObjectType および VariableType ノードを指し、Object や Variable ノードのメタ情報を定義する。

TypeDefinitionNode を定義する際、ModellingRule を参照しない構成要素は、静的メンバーとして扱われる。ModellingRule を参照している構成要素は、動的メンバーとして扱られ、そのような要素を InstanceDeclaration と呼ぶ。TypeDefinitionNode から Object ノードや Variable ノードをインスタンス化する際、ModellingRule を参照する動的メンバー（InstanceDeclaration）のみがインスタンス化される。

InstanceDeclaration となる UA ノードが持つ要素も InstanceDeclaration である(特にそれを意識する場合は InstanceDeclarationHierarchy と呼ぶ)。この場合の下位階層側の InstanceDeclaration は、それがインスタンス化される際に上位階層側で自身を定義している ModellingRule を参照する。例えば、上位階層側の InstanceDeclaration を示す UA ノードが ModellingRule を参照しない UA ノードを要素として定義している場合、その下位階層側の UA ノードが ModellingRule を定義した要素を定義していても、その要素はインスタンス化されない。

標準の ModellingRule にはつぎが存在する。それぞれの仕様については以降の項目を参照。

- Mandatory
- Optional
- MandatoryPlaceholder
- OptionalPlaceholder
- ExposesItsArray

ModllingRule はガイドラインにすぎないことに注意されたい。UA サーバーが表現する実システムや運用の要件によっては新しい ModellingRule をサブタイプ定義で新たに定義しなければならない。

参考文献

- Part3 Address Space Model Release 1.04 : 6 Type Model for ObjectTypes and VariableTypes
- Part3 Address Space Model Release 1.04 : A.8 Defining ModellingRules

ModellingRule を指定しないとどうなるのか?

仕様

InstanceDeclaration は TypeDefinitionNode の静的メンバーの要素であるとみなされる。

TypeDefinitionNode がインスタンス化される場合、ModellingRule のない UA ノード（Object、Variable および/または Mothod などの要素）はインスタンス化されない。同様に、InstanceDeclaration 内に定義されている ModellingRule のない UA ノードの要素は、親の InstanceDeclaration がインスタンス化される場合にもインスタンス化されない。

ModellingRule に Mandatory を指定するとどうなるのか?

仕様

親 UA ノード(TypeDefinitionNode または、InstanceDeclarationHierarchy 内の親 InstanceDeclaration)が対象の InstanceDeclaration を必ず 1 つ動的メンバーとして持つことを表現する場合には Mandatory を指定する。

InstanceDeclarationHierarchy で最上位にあたる TypeDefinitionNode がインスタンス化されるとき、InstanceDeclaration からコピーされたインスタンスの UA ノードは、InstanceDeclaration と同じ NodeClass および BrowseName のノード属性値を持つことが期待される。InstanceDeclaration が Varible ノードである場合、DataType ノード属性値は同じであるか、またはそのサブタイプ（DataType ノードの継承関係で特化されているもの）が求められる。インスタンス化された UA ノードに定義される Reference の ReferenceType は、コピー元の InstanceDeclaration に許可されている ReferenceType と同じか、またはそのサブタイプ（ReferenceType ノードの継承関係で特化されているもの）が求められる。

ModellingRule に Optional を指定するとどうなるのか?

仕様

親 UA ノード(TypeDefinitionNode または、InstanceDeclarationHierarchy 内の親 InstanceDeclaration)が対象の InstanceDeclaration を 0 または 1 つの動的メンバーとして持つことを表現する場合には Optional を指定する。実際に InstanceDeclaration からコピーされてインスタンス化される UA ノードの数は UA サーバーの仕様による。

InstanceDeclarationHierarchy で最上位にあたる TypeDefinitionNode がインスタンス化されるとき、InstanceDeclaration からコピーされたインスタンスの UA ノードは、InstanceDeclaration と同じ NodeClass および BrowseName のノード属性値を持つことが期待される。InstanceDeclaration が Varible ノードである場合、DataType ノード属性値は同じであるか、またはそのサブタイプ（DataType ノードの継承関係で特化されているもの）が求められる。インスタンス化された UA ノードに定義される Reference の ReferenceType は、コピー元の InstanceDeclaration に許可されている ReferenceType と同じか、またはそのサブタイプ（ReferenceType ノードの継承関係で特化されているもの）が求められる。

ModellingRule に MandatoryPlaceholder を指定するとどうなるのか?

仕様

親 UA ノード(TypeDefinitionNode または、InstanceDeclarationHierarchy 内の親 InstanceDeclaration)が対象の InstanceDeclaration を 1 つ以上の動的メンバーとして持つことを表現する場合には MandatoryPlaceholder を指定する。実際に InstanceDeclaration からコピーされてインスタンス化される UA ノードの数は UA サーバーの仕様による。

InstanceDeclarationHierarchy で最上位にあたる TypeDefinitionNode がインスタンス化されるとき、InstanceDeclaration からコピーされたインスタンスの UA ノードは、InstanceDeclaration と同じ NodeClass のノード属性値を持つことが期待される。InstanceDeclaration が Varible ノードである場合、DataType DataType ノード属性値は同じであるか、またはそのサブタイプ（DataType ノードの継承関係で特化されているもの）が求められる。インスタンス化された UA ノードに定義される Reference の ReferenceType は、コピー元の InstanceDeclaration に許可されている ReferenceType と同じか、またはそのサブタイプ（ReferenceType ノードの継承関係で特化されているもの）が求められる。インスタンス化された UA ノードの BrowseName ノード属性値は、UA サーバーの仕様によって決定される。したがって、InstanceDeclaration の BrowseName ノード属性値は、TypeDefinitionNode から参照されるためだけに使用される（注記 1）。

注記 1: InstanceDeclaration の BrowseName ノード属性値は、InstanceDeclaration からインスタンス化される複数の UA ノードをグループ化するための抽象名であることが推奨される。また、TypeDefinitionNode の定義テーブルでは、

<SampleBrowseNameOfInstanceDeclaration>のように < と > で InstanceDeclaration の BrowseName ノード属性値を記述するのがルールである。1

ModellingRule に OptionalPlaceholder を指定するとどうなるのか?

仕様

親 UA ノード(TypedeDefinitionNode または、InstanceDeclarationHierarchy 内の親 InstanceDeclaration)が対象の InstanceDeclaration を 0 または 1 つ以上の動的メンバーとして持つことを表現する場合には OptionalPlaceholder を指定する。実際に InstanceDeclaration からコピーされてインスタンス化される UA ノードの数は UA サーバーの仕様による。

InstanceDeclarationHierarchy で最上位にあたる TypeDefinitionNode がインスタンス化されるとき、InstanceDeclaration からコピーされたインスタンスの UA ノードは、InstanceDeclaration と同じ NodeClass のノード属性値を持つことが期待される。InstanceDeclaration が Varible ノードである場合、DataType DataType ノード属性値は同じであるか、またはそのサブタイプ(DataType ノードの継承関係で特化されているもの)が求められる。インスタンス化された UA ノードに定義される Reference の ReferenceType は、コピー元の InstanceDeclaration に許可されている ReferenceType と同じか、またはそのサブタイプ(ReferenceType ノードの継承関係で特化されているもの)が求められる。インスタンス化さらた UA ノードの BrowseName ノード属性値は、UA サーバーの仕様によって決定される。したがって、InstanceDeclaration の BrowseName ノード属性値は、TypeDefinitionNode から参照されるためだけに使用される(注記 1)。

注記 1: InstanceDeclaration の BrowseName ノード属性値は、InstanceDeclaration からインスタンス化される複数の UA ノードをグループ化するための抽象名であることが推奨される。また、TypeDefinitionNode の定義テーブルでは、<SampleBrowseNameOfInstanceDeclaration>のように < と > で InstanceDeclaration の BrowseName ノード属性値を記述するのがルールである。1

ModellingRule に ExposesItsArray を指定するとどうなるのか?

仕様

ExposesItsArray は、その DataType ノード属性値が一次元または多次元の配列である VariableType に適用可能なもので、配列の各値もそこからインスタンス化した Variable ノードの DataVariable として公開されることを示す。

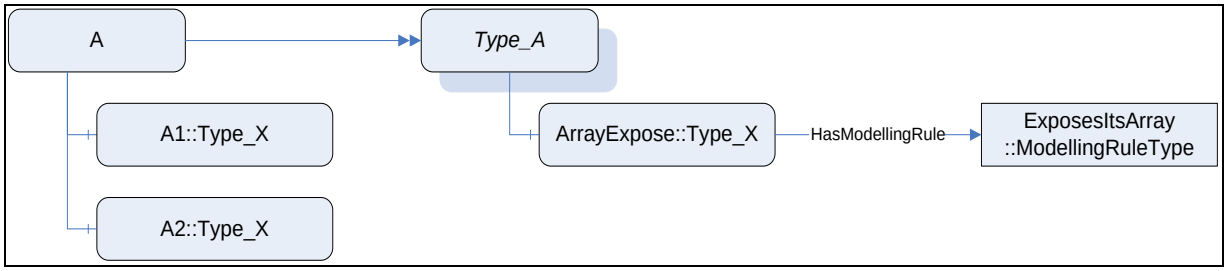


図 4–ExposesItsArray の使用例 4

上の図では、Type_A は VariableType ノードであり、その DataType 属性は Type_X であり、ValueRank 属性は一次元または多次元配列を示す。A は、Type_A からインスタンス化された Variable ノードであり、その Value 属性は、その配列要素が Type_X データ型の配列である。したがって、Type_X の DataVariable が A に要素として追加されるとき、A の Value 属性は、その配列値要素として追加された DataVariable を含まなければならない。上図において、A は、要素が A1 および A2 である配列値を有すると期待される。

ExposesItsArray は、Browse サービスと Read サービスの両方によって Value 属性の要素にアクセスする必要がある場合に指定する(注記 1)。

InstanceDeclarationHierarchy で最上位にあたる TypeDefinitionNode がインスタンス化されるとき、InstanceDeclaration からコピーされたインスタンスの UA ノードは、InstanceDeclaration と同じ NodeClass のノード属性値を持つことが期待される。InstanceDeclaration が Varible ノードである場合、DataType DataType ノード属性値は同じであるか、またはそのサブタイプ(DataType ノードの継承関係

で特化されているもの）が求められる。インスタンス化された UA ノードに定義される Reference の ReferenceType は、コピー元の InstanceDeclaration に許可されている ReferenceType と同じか、またはそのサブタイプ（ReferenceType ノードの継承関係で特化されているもの）が求められる。インスタンス化さらた UA ノードの BrowseName ノード属性値は、UA サーバーの仕様によって決定される。したがって、InstanceDeclaration の BrowseName ノード属性値は、TypeDefinitionNode から参照されるためだけに使用される（注記 2）。

注記 1：つぎの項目を参照。1 注記 1: InstanceDeclaration の BrowseName ノード属性値は、InstanceDeclaration からインスタンス化される複数の UA ノードをグループ化するための抽象名であることが推奨される。また、TypeDefinitionNode の定義テーブルでは、<SampleBrowseNameOfInstanceDeclaration>のように < と > で InstanceDeclaration の BrowseName ノード属性値を記述するのがルールである。1

ModellingRule に ExposesItsArray を指定するとどうなるのか?

注記 2: InstanceDeclaration の BrowseName ノード属性値は、InstanceDeclaration からインスタンス化される複数の UA ノードをグループ化するための抽象名であることが推奨される。また、TypeDefinitionNode の定義テーブルでは、<SampleBrowseNameOfInstanceDeclaration>のように < と > でで InstanceDeclaration の BrowseName ノード属性値を記述するのがルールである。1

複数の UA ノードを管理する場合、NodeId/ExpandNodeId 配列型の Variable と、MandatoryPlaceholder/OptionalPlaceholder の ModellingRule で InstanceDeclaration を選択する基準は何か?

推奨

両方とも、複数の UA ノードを参照する方法であり、選択する明確な基準を提供することは困難である。一般的に、InstanceDeclaration を使用する場合、Browse サービスを使用して UA ノードを参照する。DataType ノード属性値が NodeId または ExpandNodeId である Variable ノードを使用する場合、UA クライアントは Variable ノードを認識し、Read サービスを呼び出してその要素である UA ノードを参照する。

複数の UA ノードにアクセスする方法は、ユースケースに基づいて決定されたい。

注記 1: 両方のユースケースが存在する場合、ExposesItsArray ModellingRule の使用を考慮することが考えられる。「ModellingRule に ExposesItsArray を指定するとどうなるか?」を参照。1 注記 1: InstanceDeclaration の BrowseName ノード属性値は、InstanceDeclaration からインスタンス化される複数の UA ノードをグループ化するための抽象名であることが推奨される。また、TypeDefinitionNode の定義テーブルでは、<SampleBrowseNameOfInstanceDeclaration>のように < と > で InstanceDeclaration の BrowseName ノード属性値を記述するのがルールである。1

ModellingRule に ExposesItsArray を指定するとどうなるのか?

MorellingRule をサブタイプ化するべきか?

推奨

構造的には、ModellingRule とはその ObjectType が ModellingRuleType の Object ノードである。ModellingRuleType のサブタイプ定義や、ModellingRule となる新たな Object ノードを追加定義して利用することは可能である。

しかし、本ガイドラインは、標準 ModellingRule のサブタイプを定義することを推奨しない。クライアントにとって、派生した ModellingRule の追加の意味を理解することは困難に思われる。

標準 ModellingRule のサブタイプを定義する代わりに、対象の InstanceDeclaration になる UA ノードに対して追加したい意味を表す定義をすることを推奨する。

2.12 Role

Role は、UA クライアントが UA サーバーにアクセスするときに、その接続状態によってどのようなサービスを利用できるか、認可の手段を扱うための仕様である。Role は、認証(クライアントの識別)と認可(クライアントが何をすることを許可されているかの決定)を分離するための仕組みを提供する。

通常 UA サーバーの設計者は、Role に関連する ObjectType ノードや VariableType ノードのサブタイプ定義を行う必要はない。その代わり、規定の ObjectType ノードや VariableType ノードを用いて対象の UA サーバーが公開する権限情報を表す Object ノードや Variable ノードを定義する。

Role に関連する情報モデルを下図に示す。

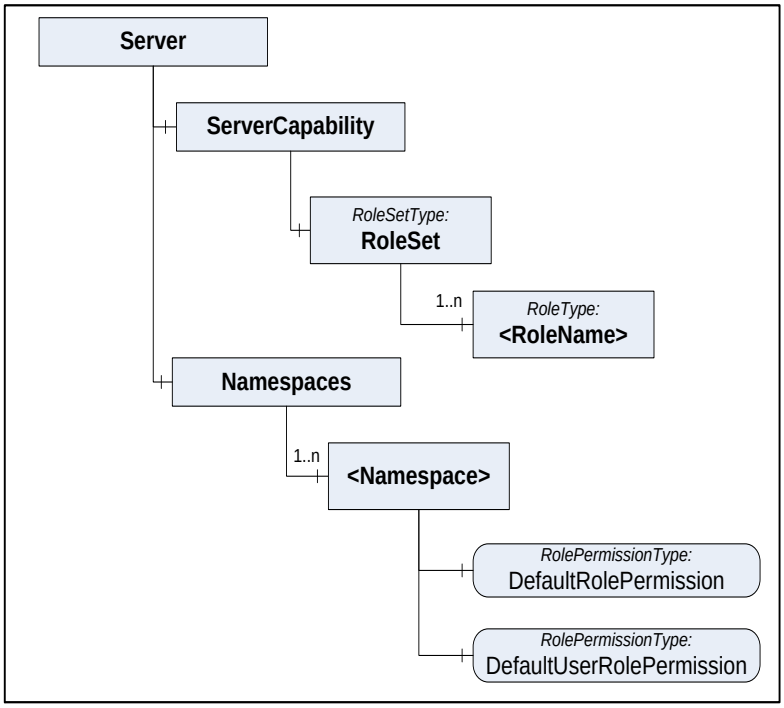


図 5 ユーザ認証モデル 5

- 標準の UA ノード’Server’およびその下の UA ノード’ServerCapability’オブジェクトの下に、’RoleSet’という BrowseName ノード属性値のオブジェクト(その TypeDefinitionNode は RoleSetType)を用意し、当該 UA サーバーが公開する Role 群を管理する。

- ‘RoleSet’オブジェクトにはその要素として、RoleType という ObjectType からインスタンス化されたオブジェクトを複数定義できる。この要素の Object ノードが個々の Role を表す。RoleType には Role を決定するための条件を指定する Property が定義されている。UA サーバーは、UA クライアントが下記に示すどの条件で接続しているかと、個々の Role の Property の設定内容によって、その Role の有効／無効を決定する。

  - セッション作成時に指定したクライアント ID
  - UA クライアントのアプリケーション ID
  - 接続時に指定した UA サーバーのエンドポイント

- 全ての UA ノードは、オプションとして’RolePermissions’および’UserRolePermissions’ノード属性を持ち、Role 毎に呼出し可能なサービスを指定できる。呼出し可否の指定ができるサービスは Read サービス、Write サービス、Call サービスに限定される。

- ’RolePermissions’および’UserRolePermissions’ノード属性の定義がない UA ノードについては、UA サーバー内の Namespace 毎に、Role 毎のデフォルトのサービス呼び出し可否を指定することができる。これらの定義は、上図の Namespaces フォルダオブジェクトの下に配置されている個々の Namespace を指す Object ノードの’DefaultRolePermision’および’DefaultUserRolePermission’の Property で表現する。

- Part3 Address Space Model Release 1.04 : 4.8 Roles
- Part5 Information Model Release 1.04 : Annex F (normative) User Authorization

どのような標準 Role が、OPC UA 仕様書内で規定されているか?

仕様

標準の Role として UA サーバーにサポートされるべきものは、以下の仕様書に列挙されている。

- Part3 Address Space Model Release 1.04 : 4.8.2 Well Known Roles

2.13 StateMachine

StateMachie は、あるものの状態 または/および その状態が取り得る状態遷移を表現する。SateMachine の情報をモデル構築して公開したい場合に利用されるべき ObjectType ノード、VariableType ノード、および ReferenceType ノードが OPC UA の標準情報モデルとして提供されている。

つぎの OPC UA 仕様書にそれらの UA ノードや使用法について記載されている。

- Part5 Information Model Release 1.04 : Annex B (normative) StateMachines

状態遷移の情報をモデル構築するために、上記の仕様に従うことが必ず求められるわけではないが、UA アプリケーション間の相互運用性という観点からは、ここで規定している仕様に従うことが強く推奨される。

実際に、OPC UA の以下のビルトイン情報モデルには、この仕様を元に SateMachine の情報モデルが設計されている。

- Part9 Alarm and Conditions Release 1.04 : 5.8.10 ShelvedStateMachineType
- Part9 Alarm and Conditions Release 1.04 : F.2 SystemStateStateMachineType
- Part10 Programs Release 1.04: 5.2 ProgramStateMachineType

StateMachie を定義するためにどのような情報モデルが規定されているか?

仕様

主な情報モデルは以下の通り。

StateMachineType: すべての StateMachie を定義するベースとなる ObjectType。必須の Variable ノードとして StateMachie の現在のステートを表す'CurrentState'がある。その VariableType ノードは StateVariableType(後述)またはそのサブタイプである。StateMachineType のインスタンスは、重要な状態遷移が起こるときはいつでもイベントを発行する。特に重要なイベントを表す EventType に対しては、StateMachineType から対象となる EventType 派生 ObjectType ノード(複数可)に対して GeneratesEvent の Reference を定義する。

StateVariableType: StateMachine の現在のステートを表す Variable ノードのベースとなる VariableType ノードで、その DataType 属性は LocalizedText であり現在のステートを示す文字情報を格納する。'Id'という必須の Property をもち、現在のステートを示す数値データを格納する。

FiniteStateMachineType: StateMachieType のサブタイプで、StateMachineType が現在のステートを表す為のモデルであるのに対して、FiniteStateMachineType は StateMachine の中で定義されるステートが取り得る全ての値、およびそれらの遷移条件などを明示的に定義する。この目的のために、OPC UA StateMachie では FinitieStateMachineType の定義に必要な ObjectType、VariableType、ReferenceType を定義している。

TransitionEventType: BaseEventType のサブタイプで、状態遷移が発生したこをイベント通知するためのベースとなる ObjectType ノード。

AuditUpdateStateEventType: コア仕様で定義されている AuditUpdateMethodEventType のサブタイプで、状態遷移が発生したこを監査証跡の形式でイベント通知するためのベースとなる ObjectType ノード。

StateMachie を定義するにはどうすればよいか?

仕様

下記の流れにて行う。

FinitestateMachineType のサブタイプとなる ObjectType ノードを定義する。

StateMachie の中で起こり得るすべてのステートを定義する。各ステートは、定義された FiniteStateMachineType サブタイプの要素として定義する。その要素は 'StateType' という ObjectType からインスタンス化される(State ノード)。ステートマシンで表現するステートの中で初期状態になるものは、' StateType' のサブタイプである' IntialStateType' からインスタンス化したものを指定する。

StateMachie の中で起こり得る遷移を定義する。各遷移の事象を表す Object ノードを、定義された FiniteStateMachineType サブタイプの要素として定義する。その要素は、'TransitionType' という ObjectType からインスタンス化される(Transition ノード)。

上記で定義した Transition ノードが示す遷移の詳細を定義する。Transition ノードからその遷移発生前のステートを示す State ノードに対して、'FromState'の Reference を定義する。同様に、その Transition ノードから遷移後のステートを示す State ノードに対して、'ToState'の Reference を定義する。(注記 1)。

上記で定義した遷移が 1 つの Method を呼び出すことによって引き起こされる場合、その Method ノードを定義し、当該の Transition ノードからその Method ノードに対して'HasCause'の Reference を定義する。一般に、Method ノードは、FinitestateMachineType 派生 Object ノードの要素、または FinitestateMachineType 派生 Object ノードを InstanceDeclaration として有する ObjectType の要素として定義する(「2.11 ModellingRule」を参照) (注記 1)。ModellingRule

遷移が発生したときにイベントを発行することを定義する場合は、当該の Transition ノードから EventType 派生 ObjectType ノードに対して'HasEffect'の Reference を定義する。これらの遷移を定義している FiniteStateMachineType サブタイプは、'GeneratesEvent'の Reference でそれらの EventType を参照すべきである。

注記 1: 2018 年末に、遷移先のステートを条件によって複数の中から選択する表現を行う追加仕様が審議されている。1

注記2: StateMachie における状態遷移は、StateMachieType に定義した Method の起動により引き起こされる設計にすることが前提となっている。1

以下は定義例。

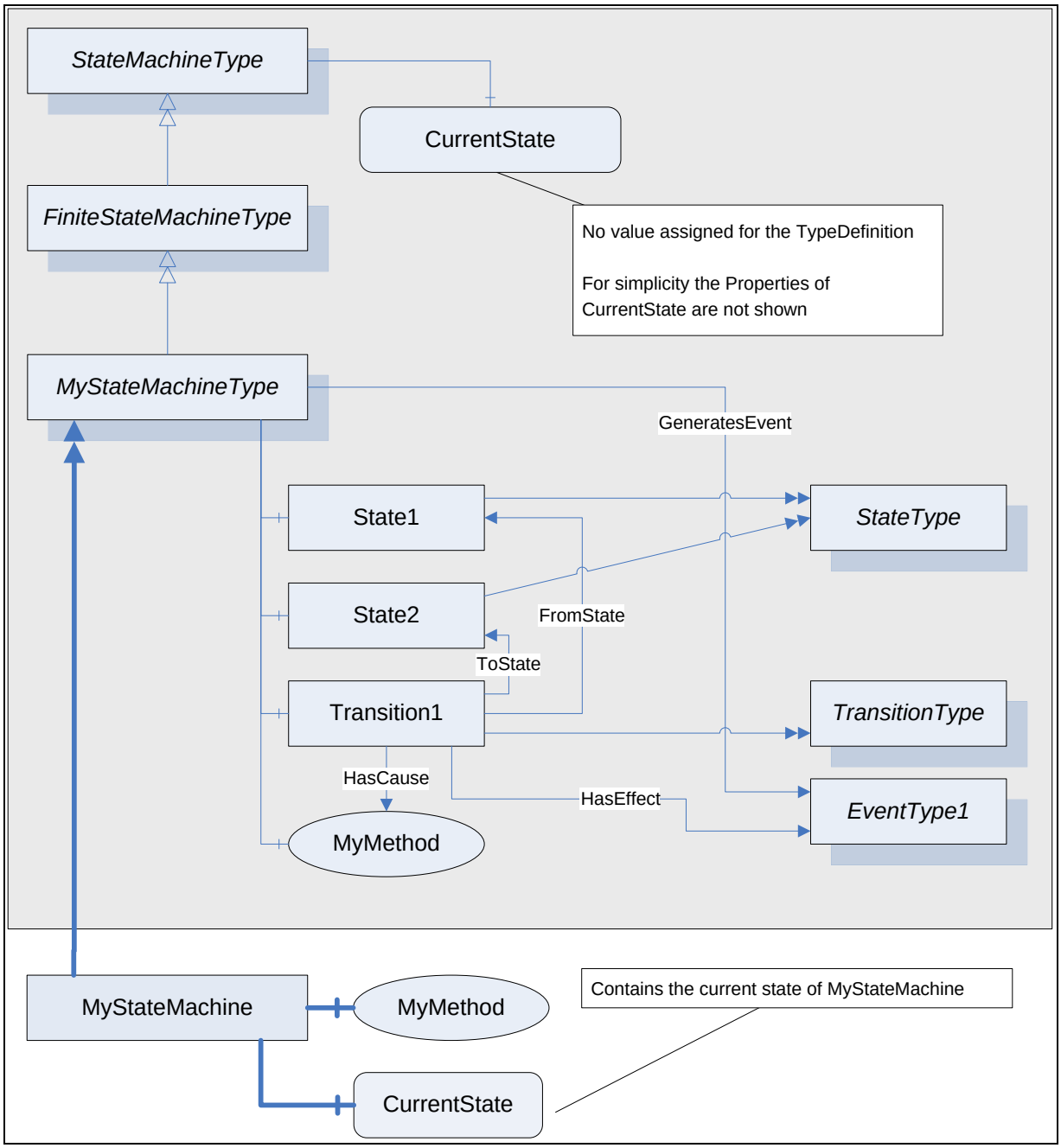


図 6 – StateMachie の例 6

StateMachine にサブステートの StateMachine を定義するにはどうすればよいか?

仕様

FinitestateMachineType のサブタイプである ObjectType ノードを 2 つ用意する。各 FiniteStateMachineType サブタイプを定義するには、「StateMachie を定義するにはどうすればよいか?」を参照。これはメインの StateMachine とサブステートの StateMachine を表す。StateMachie を定義するにはどうすればよいか?

サブステートとなる FiniteStateMachineType サブタイプの InstanceDeclaration(「2.11 ModellingRule」を参照)を、メインの FiniteStateMachineType サブタイプの要素として定義する。ModellingRule

メインとなる FinitestateMachineType サブタイプ内に定義している State ノードの中で、サブステートに展開するものから、FinitestateMachineType サブタイプの InstanceDeclaration に対して

'HasSubStateMachine'の Reference を定義する。これでメインステートがサブステートを持つことを表現する。

以下は例であり、つぎが定義されている。

- MyStateMachineType のサブステートマシンとして、MySubMachine が要素として定義されている。
- MyStateMachineType の State1 ステートに上記のサブステートマシンが定義されていることを表すために、State1 から MySubMachine に HasSubStateMachine の Reference が定義されている。

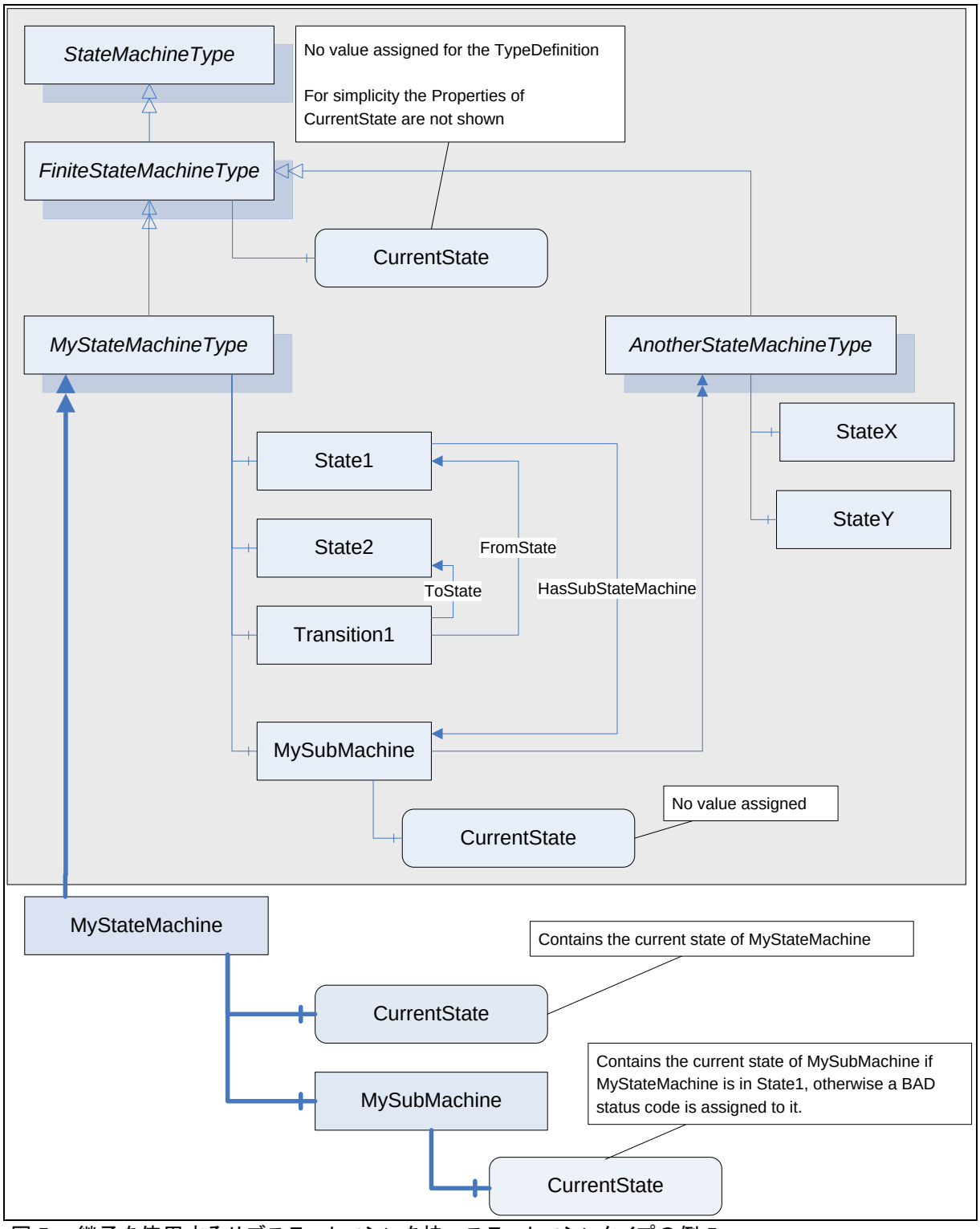


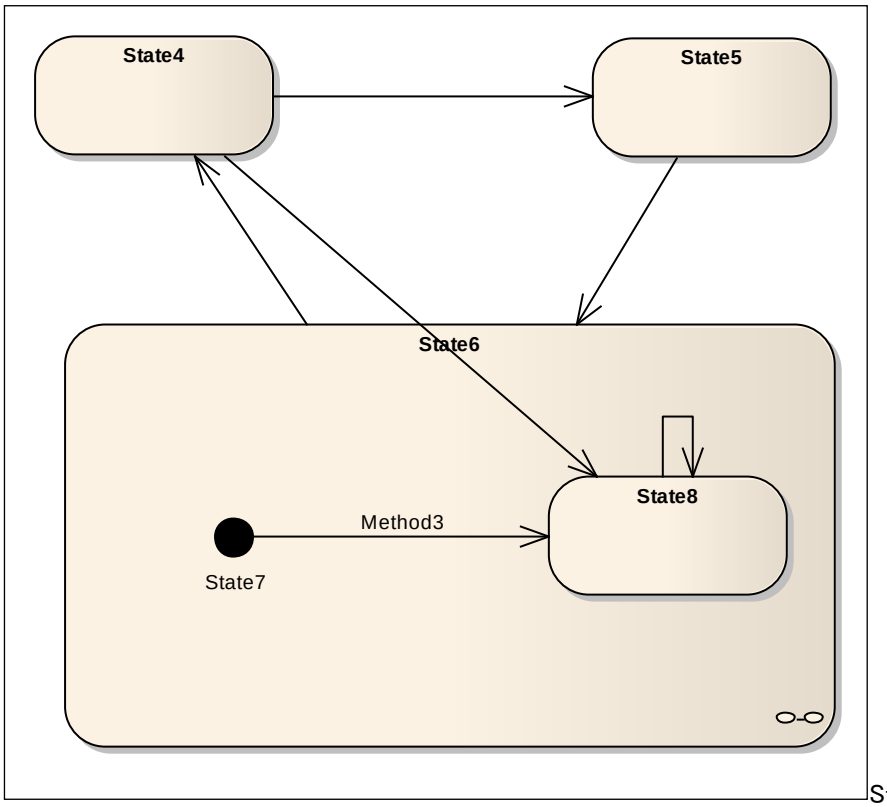
図 7 – 継承を使用するサブステートマシンを持つステートマシンタイプの例 7

上記の例では、メインステートマシンとサブステートマシンとの間の遷移は定義されていない。この場合の仕様は、メインステートマシンの中でサブステートマシンを持つ State ノード（上記の State1）に遷移すると、デフォルトでサブステート内の State ノードの中で InitialSateType からインスタンス化されているものが初期ステートになる。

もしもデフォルトと異なる遷移を表現する場合にはつぎの定義を行う。

メインとなる FiniteStateMachineType サブタイプが持つ State ノードと、サブステートを表す FiniteStateMachineType サブタイプ InstanceDeclaration 内の State ノードの間にも'FromState'、'ToState'の Reference を定義する。これにより、メインステートとサブステート間の遷移を表現する。'HasEffect'や'HasCause'の Reference、およびそれに必要な Method ノードおよび EventType ノードがあれば定義する。

メインとサブのステートマシンの State ノード間の遷移を表現する例を下記にしめす。ここにおいて、State4 からは明示的にサブステートマシンの State8 に遷移することが定義されている。しかし State5 から State6 への遷移が発生する場合には、サブステートマシンの初期ステートである State7 に遷移することになる。また、下記の例ではサブステートマシンのステートに関わらず、ある遷移条件が発生すれば State6 から State4 に遷移することが定義されている。



StateMachie を定義するにはどうすればよいか?

図 8 – サブステートマシン内の初期ステートの例 8

どのような種類の StateMachine が、OPC UA 仕様書内で規定されているか?

仕様

コア仕様に関して、OPC UA 標準の StateMachine は、以下の仕様書に列挙されている。

- Part5 Information Model Release 1.04: Annex B (normative) StateMachines

ビルトインの情報モデルでは、以下に StateMachine が記載されている。

- Part9 Alarm and Condition 1.04: 5.8.10 ShelvedStateMachineType

- Part9 Alarm and Condition 1.04: F.2 SystemStateStateMachineType
- Part10 Programms 1.04: 5.2 ProgramStateMachineType

推奨

StateMachine が設計対象の StateMachine と同じ意味を持つ場合は、相互運用性のため、標準タイプを使用することを推奨する。

追加情報が必要な場合は、標準 StateMachine をサブタイプ化すべきである。

使用する予定のない情報がある場合でも、未使用情報にダミー値を指定するなどして、標準の StateMachine の使用を検討すべきである。

2.14 File Transfer

FileTransfer は、ファイル転送に関する情報モデルである。ファイルは、DataType ノード属性が ByteStrings またはそのサブデータタイプである Variable ノードとしてモデル化することができる。しかし、この設計でファイル処理を運用することは下記のような幾つかの問題がある。

- OPC UA にデータ交換するメッセージのサイズが UA サーバー上のリソースやセキュリティの条件などにより制約を受けて、実際のファイルサイズを処理できない
- 複数のクライアントが同時に共通のファイルにアクセスする場合のアクセスロック

このような課題に対応するために、OPC UA が規定する FileTransfer の情報モデルは、個々のファイル、ファイルシステム、およびファイル転送機能を表すモデルを定義する。

つぎの OPC UA 仕様書にそれらの UA ノードや使用法について記載されている。

- Part5 Information Model Release 1.04 : Annex C (normative) File Transfer

ファイル転送を扱う場合に File Transfer の仕様に従うことが必ず求められるわけではないが、UA アプリケーション間の相互運用性という観点からは、ここで規定している仕様に従うことが強く推奨される。

3 ビルトイン情報モデル

2章では、OPC UA のコア機能を紹介した。この章では、OPC UA コア機能を応用した、Data Access、Alarm and Conditions、Historical Access などの OPC Classic で扱っていた仕様を扱う情報モデルを説明する。

同様の機能を実現する場合には、ここで規定されている情報モデルをベースにすることを推奨する。

OPC UA メタモデル

3.1 Data Access

Data Access(以下 DA)は、産業オートメーションのデータを取り扱うための情報モデルを規定している。その中で任意のオートメーションデータとの結びつきを表現するものを DataItem と呼んでいる。

DA は、いくつかの種類の DataItem に対応する VariableTypes ノードを定義し、それらの VariableType に必要な DataType ノードを定義している。

参考文献

- Part8 Data Access 1.04: 4 Concepts

3.1.1 既定モデル

DA にはどのような情報モデルが規定されているか?

仕様

主な情報モデルは以下の通り。

DataItemType: DataItem(任意のオートメーションデータとの結びつきを表現するもの)の一般的な特性を定義する VariableType ノード。他のすべての DataItem を表す為の VariableType ノードは DataItemType から派生している。DataItemType の標準のサブタイプには以下がある。

- AnalogItemType
- DiscreteItemType (および幾つかのそのサブタイプ)
- ArrayItemType (および幾つかのそのサブタイプ)

DA はこのほかに幾つかの DataType ノードを定義している。これらの DataType ノードは、主に DataItemType のサブタイプ定義に追加される Property の DataType ノード属性値として参照されるために用いられる。

参考文献

- Part8 Data Access 1.04: 5.3 Variable Types
- Part8 Data Access 1.04: 5.6 Data Types

推奨

設計対象の情報モデルの意味が DA で標準定義している情報モデルの意味と同一である場合は、相互運用性のために標準タイプを使用すべきである。

追加情報が必要な場合は、標準タイプをサブタイプ化すべきである。

使用する予定のない情報がある場合でも、未使用情報にダミー値を指定するなどして、標準タイプの使用を検討すべきである。

3.1.2 サブタイプ定義

新たに VaribleType をサブタイプ定義する動機は何か?

推奨

DA の場合、新しい VariableType を指定することは、DataItemType の新しいサブタイプの記述を意味する。

したがって、1 つの動機付けは、その DataType ノード属性値が既存の DataItemType サブタイプによってサポートされていないものを指定したい場合に、新しい VaribleType の定義を検討すべきである。

別の動機付けは、追加の Property を指定することによって既存の DA VariableType に情報を追加することである。

新たに DataType をサブタイプ定義する動機は何か?

推奨

既存の DataType では VariableType またはその Property の Value ノード属性で表現したい情報を表現できない場合、その情報を表現する為の新しい DataType の定義を検討すべきである。

以下は、新しい DataType を検討する例である。

- 一貫性の観点から一度のアクセスで複数のデータを同時に処理すべき場合、それらのデータを纏めた 1 つの Structured DataType を検討すべきである。
- 既存の DataType に追加の情報や意味を指定したい場合、そこから派生した新しい DataType に Property を追加することを検討すべきである。

注記 1: 詳細については「2.2 DataType」を参照。1 **エラー! 参照元が見つかりません。**

3.2 Alarms & Conditions

Alarms & Conditions(以下 AC)は、OPC UA メタモデルで規定されている EventType の拡張により、Condition という概念を導入する。Condition はシステムまたはその構成要素の状態を表現する。OPC UA における通常のイベントは、そのデータが一時的でイベント発生後に UA サーバーの中に残らない(イベントヒストリは除く)。これに対して、Condition は状態を示す情報を持ち、それが表すシステムまたはその構成要素の状態がある条件を満たすまでは、イベントデータが UA サーバーに残り続けるモデルを定義している。

AC においては、Condition のモデルを応用した情報モデル(確認機能を含むアラーム発生状態や、応答機能を含むダイアログ表示状態など)が定義されている。

参考文献

- Part9 Alarms & Conditions 1.04:4 Concepts

3.2.1 既定モデル

AC にはどのような情報モデルが規定されているか?

仕様

主な情報モデルは以下の通り。

TwoStateVariableType: 2 値(真または偽)の状態を定義する VariableType。TwoStateVariableType の DataType ノード属性は LocalizedText であり、Value ノード属性は True または False 状態を表す人間が読み取り可能な文字列となる。

必須の Property として'Id'があり真または偽の状態を表す論理値を持つ。TwoStateVariableType は HasTrueSubState および/または HasFalseSubState の Reference を使って、True および False の両方の状態についてサブステータスを定義できる。この Reference の参照先はサブステータスに取り得る状態の数によって TwoStateVariableType(2 つの状態)または(後述の)StateMachineType(2 を超える状態)のいずれかになる。TwoStateVariableType や StateMachineType は「2.13 StateMachine」で紹介した StateMachine を表す情報モデルを利用している。StateMachie を定義するためにどのような情報モデルが規定されているか?

StateMachineType: この ObjectType は「2.13 StateMachine」で紹介しているコア仕様の ObjectType ノードである。OPC UA で StateMachine を表現する場合のベースとなる。AC では状態が 2 を超える StateMachine を表す場合に StateMachineType またはそこから派生した ObjectType を用いる。StateMachie を定義するためにどのような情報モデルが規定されているか?

ConditionType: EventType から派生した ObjectType である。イベントとしての情報に加えてシステムまたはその要素の状態も定義する。Condition は、ConditionType のインスタンスとなる Object ノードであり、通常のイベントのようにそのライフタイムが一時的ではない。

Condition がサーバーの AddressSpace にも公開されるかどうかは UA サーバーの仕様に依存する。

Condition が AddressSpace に公開されない場合にも、各 Condition を特定するために Condition は一意の識別子（ConditionType の派生元である BaseEventType に定義されている EventId Property）を有する。

Condition は、現在と過去に発生した状態付きイベントを同時に管理できるように、オプションで ConditionBranch 機能を利用できる。Condition は必須で EnabledState という Variable ノードを持つ。その VariableType は（前述の）TwoStateVariableType であり当該の Condition が表すイベント状態（有効状態か無効状態）を示す。無効状態の場合、UA サーバーはその Condition を削除することが許される。有効状態の場合には通常、追加でサブステータスを定義することで StateMachine を拡張する。

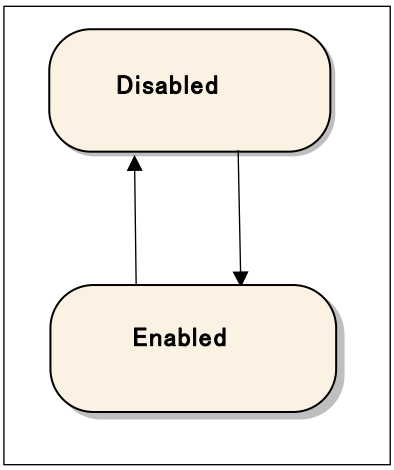


図 9–基本となる Condition の状態モデル 9

AC は ConditionType のサブタイプとして、アラームを表現する AcknowledgeableConditionType およびダイアログ操作状態を表現する DialogConditonType を規定している。これらのモデルを管理／操作する為の ObjectType、VariableType、ReferenceType、AuditEventType、Method の各 UA ノードが AC の仕様書で規定されている。

注記 1: TwoStateVariableType は、AC の名前空間内で定義されている。したがって、AC を必要としないドメインで 2 つの状態を有する StateMachine を表現するためには、TwoStateVariableType の代わりに StateMachineType を用いることが考えられる。[1]

参考文献

- Part5 Information Model Release 1.04 : Annex B (normative) StateMachines
- Part9 Alarms & Conditions 1.04: 5 Model

推奨

設計対象の情報モデルの意味が AC で標準定義している情報モデルの意味と同一である場合は、相互運用性のために標準タイプを使用すべきである。

追加情報が必要な場合は、標準タイプをサブタイプ化すべきである。

使用する予定のない情報がある場合でも、未使用情報にダミー値を指定するなどして、標準タイプの使用を検討すべきである。

3.2.2 サブタイプ定義:

ConditionType にサブステートを定義するにはどうすればよいか?

仕様

下記の流れにて行う。

サブステートを表現する情報モデルを定義する。サブステートは、TwoStateVariableType（2 つの状態の場合)または StateMachineType（2 つより多い状態の場合)のいずれかを使用して定義できる。

そのサブステートを持つべき ConditionType の派生のサブタイプを定義し、サブステータスを表現する情報モデルを要素として定義する（「2.11 ModellingRule」を参照して、Mandatory や Optional などを確認する)。

親となる ConditionType の派生サブタイプが要素として持つ'EnableState' Variable ノードからサブステートを表現する情報モデルに対して'HasTrueSubState'（Enabled 状態のサブステートの場合）および/または'HasFalseSubState'（Disabled 状態のサブステートの場合）の Reference を定義する。
ModellingRule

2 つの独立した TwoStateVariableType で表現するサブステートを Enabled の拡張として定義する例として、AcknowledeableConditiontype を以下に示す。

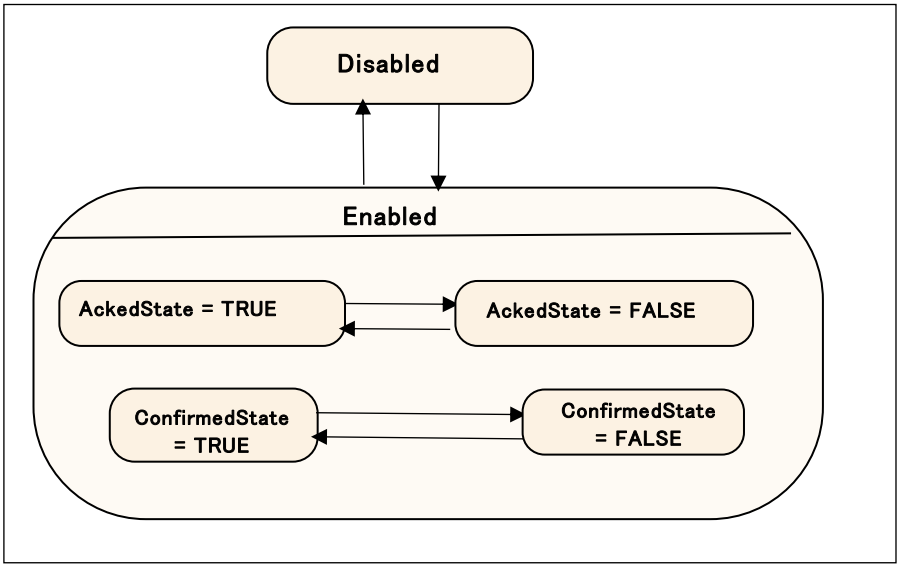


図 10 – AcknowledgeableConditions 状態モデル

上記の情報モデルは以下のように定義される。

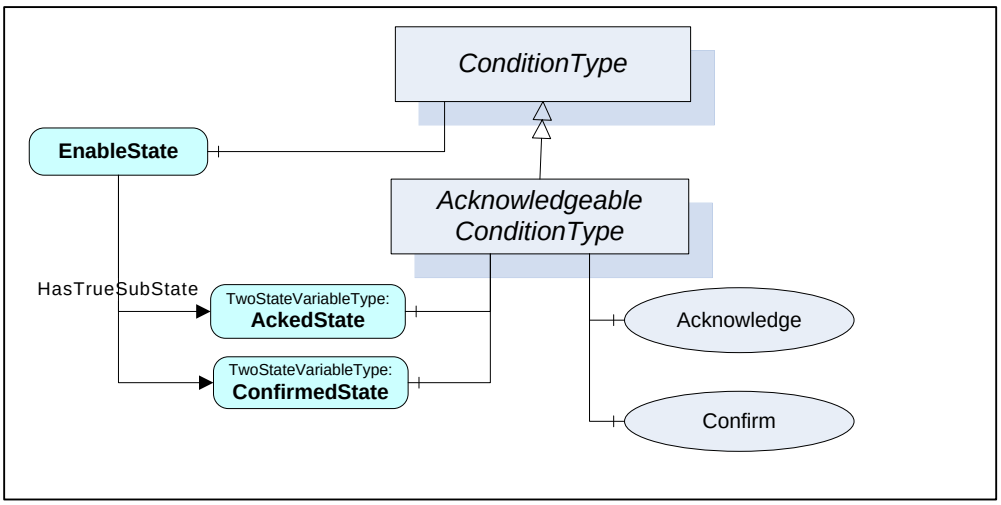


図 11–AcknowledgeableConditionType の構成 10

ConditionType 内のサブステートに更にサブステートを定義するにはどうすればよいか?

仕様

親となるサブステートのタイプ定義が TwoStateVariableType または StateMachineType により方法が異なる。

TwoStateVariableType の場合、前述の「ConditionType にサブステートを定義するにはどうすればよいか？」で説明した方法とほとんど同じである。違いは、親となるサブステートが「EnableState」という Variable ノードを要素として持っていないことである。親となる TwoStateVariableType の要素として子となるサブステートの UA ノードを定義するだけである。ConditionType にサブステートを定義するにはどうすればよいか?

StateMachineType の場合、「2.13 StateMachine」の「StateMachine にサブステートの StateMachine を定義するにはどうすればよいか？」に記載されているサブステートの定義方法に従う。ここで注意をすることがある。AC でしか定義されていない TwoStateVariableType の Variable ノードは、StateMachineType で表現するステートに定義することがサポートされていない。子となるサブステートが2つの状態しか持たない場合にも、StateMachine にサブステートの StateMachine を定義するにはどうすればよいか?StateMachineType を利用する必要がある。

新たに ConditionClass をサブタイプ定義する動機は何か?

仕様

ConditionClass とは、ある Condition がどのドメインまたはどの目的で使用されるかを示す為に、複数の Condition をグループ化するための仕組みである。ConditionType は必須で'ConditionClassId'という Property を持つ。その Value ノード属性は BaseConditionClassType（ObjectType ノード）から派生して定義したサブタイプの ObjectType ノードの NodeId を持つ。

いくつかの ConditionClasse を示す為に、ConditionClassType のサブタイプが標準 ObjectType ノードとして定義されている。それよりも更に特化した意味の ConditionClass を定義する必要がある場合には、既存の CondiitonClassType のサブタイプにすることを検討するべきである。既存の ConditionClassType ノードとは全く異なる意味のユースケースなどを表現したい場合には、新たな ConditionClass が検討される。

参考文献

- Part9 Alarm & Conditions 1.04: 5.9 ConditionClasses

3.2.3 AddressSpace

Condition を発行する Object や ObjectType を定義するにはどうすればよいか?

仕様

いくつかの方法があるが、一例を説明する。

Condition を発行する Object ノードおよび／または ObjectType ノードから、対象となる Conditoin を表す ConditionType 派生ノードに対して、HasCondition(またはそのサブタイプ)の Reference を定義する。

以下の例は Object ノードの例であるが、TankA から MySystemAlarmType へ、また DeviceB から MyAlarmTypeA:Conditoin1 と MyAlarmTypeA:Conditoin2 に HasCondition の Reference を定義している。ObjectType ノードの場合も同様に HasCondition の Reference を ObjectType ノードから参照すればよい。

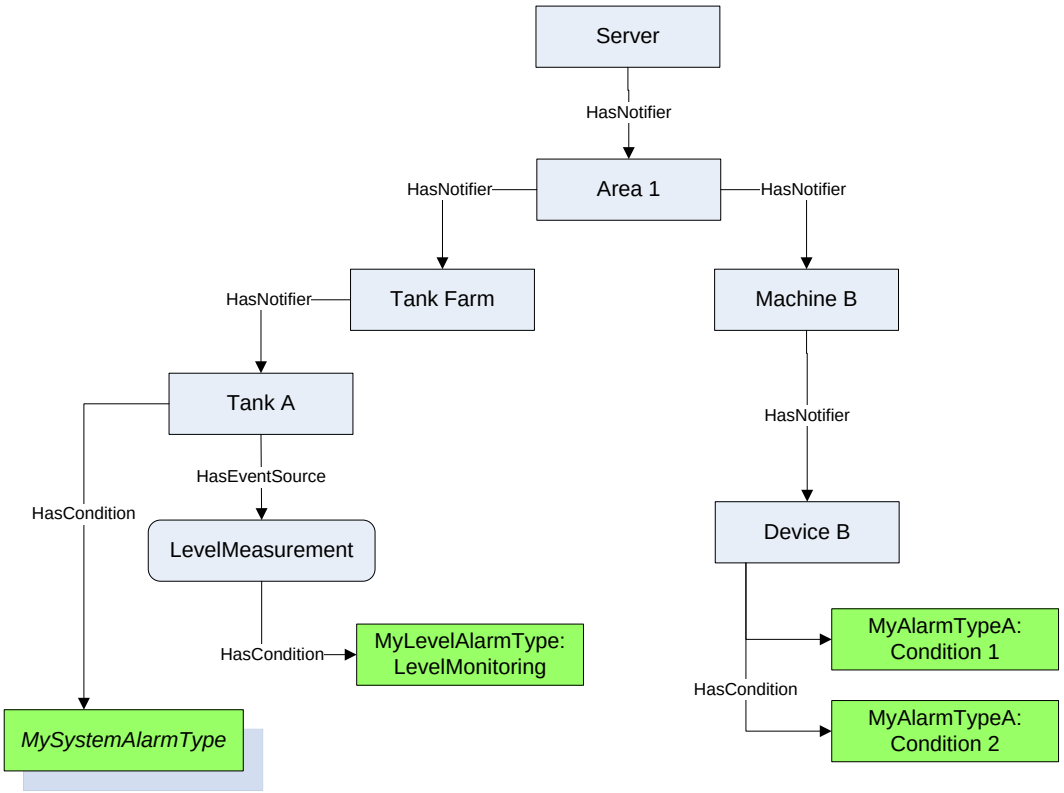


図 12–イベント階層における HasCondition の使用例 11

Condition 発行の原因となる Variable ノードを持つ ObjectType を定義するにはどうすればよいか?

仕様

いくつかの方法があるが、一例を説明する。

Condition 発行の原因となる Variable ノードから対象の ConditionType(またはそのサブタイプ)に対して' HasCondition'(またはそのサブタイプ)の参照を定義する。この場合、「2.5 Object」の「2.5.2 イベント」の「Object ノードからイベントが発行することを公開するにはどうすればよいか？」に記載されているように、Variable ノードがイベントソースである場合には、実際に Condition を発行する Object を示す ObjectType ノードから対象の Variable ノードに対して'HasEventSource'(またはそのサブタイプ)の参照を定義する必要がある(Variable ノード自体はイベントを発行できない仕様)。

図 13 に示すように、TankType からインスタンス化した TankA の Object ノードにも同様の Reference を構成する。

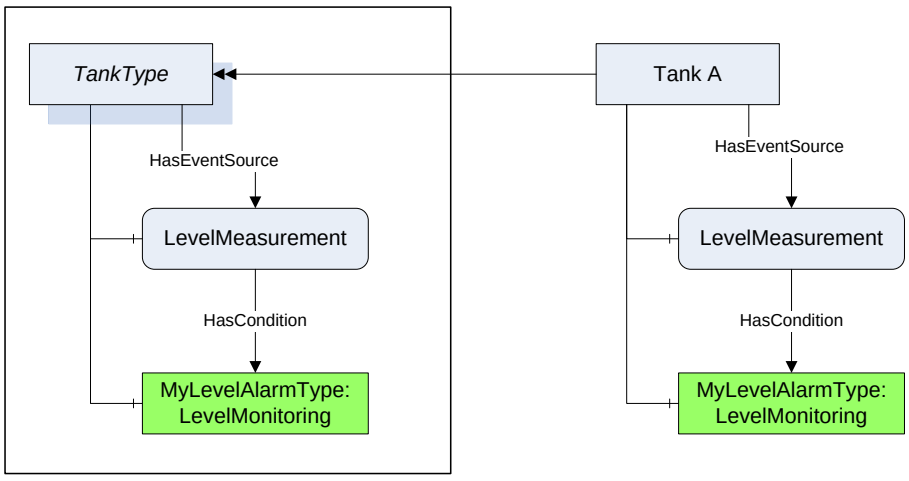


図 13–InstanceDeclaration における HasCondition の定義例 12

3.3 Programs

OPC UA は、関数を処理する手段を公開するための 2 つのメカニズムを提供する。1つは Method ノードによる公開であり、もう 1 つは Program ビルドイン情報モデルによる公開である。どちらを選択するかの基準は、UA サーバー(および基礎となるシステム)において実行される当該関数処理の処理範囲、処理のライフタイム、および複雑さによる。Method ノードは、クライアントによって呼び出すことができる UA サーバー内の軽い基本的な機能を表す。対照的に、Program は、システム内のより複雑でライフタイムの長い処理を表す。

Program は UA サーバー(およびその基盤となるシステム)における複雑な処理機能をモデル化するために使用される。その処理は UA クライアントによって呼び出され、UA クライアントは処理実行ステータスを管理することができる。Program が表現する処理機能はそのシステムにおいて、UA クライアントによる制御または介入が必要とされる。その進捗状態の監視が要求される処理には任意のレベルを表すことができる。

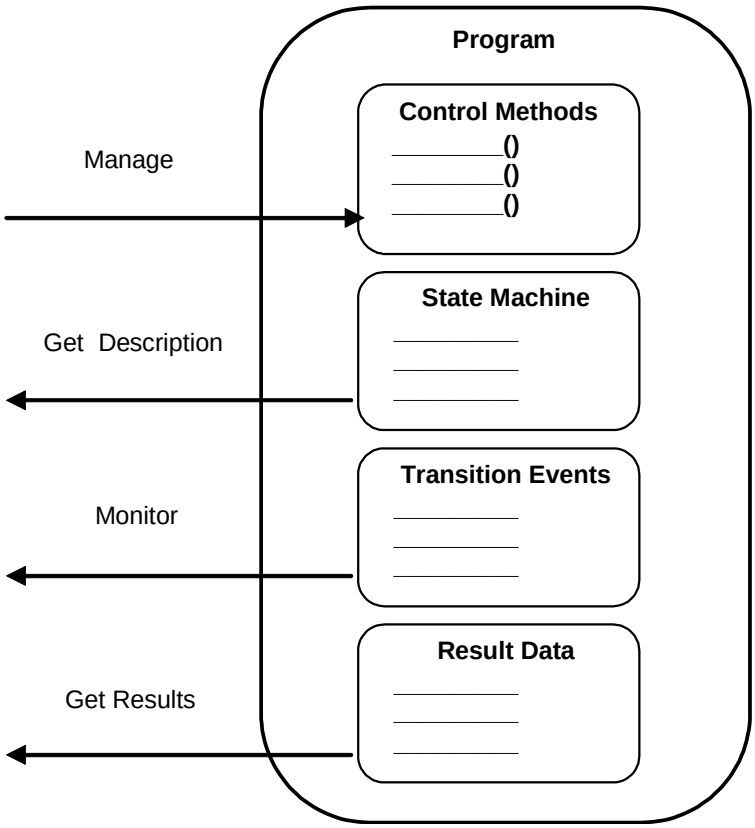


図 14 – Program の構造 13

参考文献

- Part10 Programs 1.04: 4 Concepts

3.3.1 既定モデル

Program にはどのような情報モデルが規定されているか?

仕様

主な情報モデルは以下の通り。

ProgramStateMachineType: Program が表現する処理のスコープ、挙動、ライフタイム、および複雑さを表すモデルのベースになる ObjectType。ProgramStateMachineType は、「2.13 StateMachine」で紹介されている FiniteStateMachineType から派生したものである。ProgramStateMachineType は以下のような Program の StateMachine を構成する基本状態、遷移、原因、および効果を定義する。オプションとして、BrowseName ノード属性値が'FinalResultData'である Object ノードを要素に含

めることができ、Program がその機能を完了したときの最終結果データを含むことができる。**エラー! 参照元が見つかりません。**

表 14—Program StateMachine 9

No.	Transition Name	Cause	From State	To State	Effect
1	HaltedToReady	Reset Method	Halted	Ready	Report Transition 1 Event/Result
2	ReadyToRunning	Start Method	Ready	Running	Report Transition 2 Event/Result
3	RunningToHalted	Halt Method or Internal (Error)	Running	Halted	Report Transition 3 Event/Result
4	RunningToReady	Internal	Running	Ready	Report Transition 4 Event/Result
5	RunningToSuspended	Suspend Method	Running	Suspended	Report Transition 5 Event/Result
6	SuspendedToRunning	Resume Method	Suspended	Running	Report Transition 6 Event/Result
7	SuspendedToHalted	Halt Method	Suspended	Halted	Report Transition 7 Event/Result
8	SuspendedToReady	Internal	Suspended	Ready	Report Transition 8 Event/Result
9	ReadyToHalted	Halt Method	Ready	Halted	Report Transition 9 Event/Result

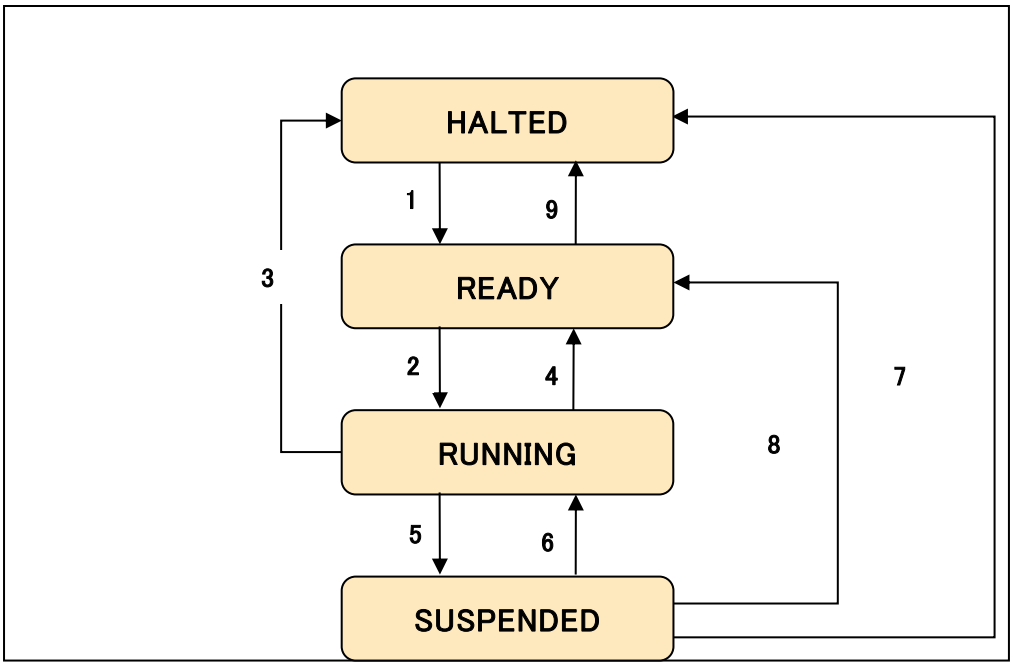


図 15–Program の状態と遷移 14

ProgramTransitionEventType: ProgramTransitionEventType は、「2.13 StateMachine」に記載されている TransitionEventType のサブタイプである。ProgramTransitionEventType は Program の処理において発生する、任意の状態遷移を通知するイベントを表現するためのベースになる EventType ノードである。これは、状態遷移に関連する中間または最終結果または他のデータを取得するために Program での処理において使用される。

Program は、任意の状態遷移に対して一意の ProgramTransitionEventType 定義を有することができる。各 ProgramTransitionEventType は、その ProgramStateMachineType 上の指定された状態遷移イベント発行時に、そのイベントに関する中間データまたは結果データをイベント情報を表す Object ノードの Property として指定する。**エラー! 参照元が見つかりません。**

AuditProgramTransitionEventType: AuditProgramTransitionEventType は、「2.13 StateMachine」に記載されている AuditUpdateStateEventType のサブタイプである。これは、任意のクライアントが呼び出した Program 上の Method ノードを実行したときの監査証跡イベントを表現する手段を提供するために使用される。**エラー! 参照元が見つかりません。**

参考文献

- Part5 Information Model Release 1.04 : Annex B (normative) StateMachines
- Part10 Programs 1.04: 5 Model

推奨

設計対象の情報モデルの意味が Program で標準定義している情報モデルの意味と同一である場合は、相互運用性のために標準タイプを使用すべきである。

追加情報が必要な場合は、標準タイプをサブタイプ化すべきである。

使用する予定のない情報がある場合でも、未使用情報にダミー値を指定するなどして、標準タイプの使用を検討すべきである。

3.3.2 サブタイプ定義:

ProgramStateMachineType にサブステートを定義するにはどうすればよいか?

仕様

OPC UA メタモデルで規定している StateMachine のメカニズムを流用するので、「2.13 StateMachine」の「StateMachine にサブステートの StateMachine を定義するにはどうすればよいか？」を参照。この拡張に従って、新しい Method および/または ProgramTransitionEventType サブタイプを定義することになる。StateMachine にサブステートの StateMachine を定義するにはどうすればよいか?

Program の処理の中間または最終結果データを公開するにはどうすればよいか?

仕様

ProgramStateMachineType のサブタイプを定義し、Variable ノードの要素を追加して中間や結果データを保持する。

また、別の手段として ProgramTransitionEventType のサブタイプを定義して中間や結果データを表す Property を追加する方法もある。この方法であれば、データをできるだけ早くクライアントに通知することができる。

'Start'メソッドが Program の処理を開始する唯一の方法か?

仕様

UA サーバーは、Program の処理を起動するために'Start'メソッドを提供する。

推奨

UA サーバーの仕様によっては、例えば何かの Object ノードのもつ他の Method や Variable ノードへの書込みによって Program を起動する設計も可能ではある。

しかし OPC UA アプリケーションとの相互運用性の観点から、上記のような場合でもその処理に置いて最終的には Program の’Start’ Method ノードを呼び出す設計にすることを強く推奨する。

‘Suspend’、’Reset’、’Halt’ および/または ‘Resume’ メソッドはサポートしなければならないか?

推奨

本ガイドラインは、それらは必須ではないと考えている。しかし、ProgramStateMachineType にこれらの Method は定義済で削除できない。サポートしない場合は、その Method ノードの'Executable'属性と'UserExecutable'属性に False を設定しておかなければならない。

3.3.3 AddressSpace

Program を管理する ObjectType を定義するにはどうすればよいか?

仕様

いくつかの方法があるが、一例を説明する。

Program を管理する Object ノードを表す ObjectType ノードから ProgramStateMachineType(またはそのサブタイプ)のタイプの要素に対して'HasComponent'(またはそのサブタイプ)の参照を定義する。

3.4 Historical Access

Historical Access (以下 HA)は、OPC UA におけるヒストリデータおよびイベントヒストリデータへのアクセス処理を定義する。AddressSpace におけるヒストリデータおよびイベントヒストリデータの公開方法についての仕様も含まれる。HA は、ヒストリアクセスが利用可能な UA ノードに対してどのような仕様のヒストリアクセスがサポートされているかを公開する情報モデルも定義する。

本ガイドラインでは独自の HA の情報モデルのメタ情報を定義する必要性は低く、既存の情報モデルをそのまま使用することが相互運用性の観点からも有用であると考えるので、これ以上の記載について割愛する。

必要に応じて以下の仕様書を参考されたい。

参考文献

- Part11 Historical Access 1.03
- Part13 Aggregates 1.04

注記 1: 2019 年 4 月現在、'Part 11 Historical Access 1.04'は、OPC Foudation のサイト内で公開されていない。

———————