

SEMI E97-0200A (Reapproved 0307)

CIM フレームワークグローバル宣言および抽象インタフェースに関する暫定仕様

本スタンダードは、global Information & Control Committee で技術的に承認されている。現版は 2006 年 11 月 21 日、global Audits and Reviews Subcommittee にて発行が承認された。2007 年 2 月に www.semi.org で、そして 2007 年 3 月に CD-ROM で入手可能となる。初版は 2000 年 2 月に発行された。

注意: 本文書は 2000 年 2 月 (0200) の発行サイクル期間中に 2 度発行された。

1 目的

1.1 本文書は CIM フレームワークの他のコンポーネントすべてが使用する宣言を定義し、CIM フレームワークを機能的に統合するための共通アーキテクチャ・パターンを規定する。マテリアル・アーキテクチャは製品管理コンポーネント、耐久消費財管理コンポーネントおよび消耗品管理コンポーネントに共通の機能を定義する。ファクトリ・リソース・アーキテクチャはさまざまなファクトリ・リソースの関係および共通機能を定義する。ジョブ・アーキテクチャはさまざまな製造タスクを実行するファクトリ・ジョブをコントロールするために使用するファクトリ全体のモデルを定義する。上記の仕様は別々のグループに分けられているので、いったん規定した後、必要に応じてそれらを論理的に組み込む、あるいは継承させることができる。

2 範囲

2.1 本仕様は製造実行システムが必要とする共通インタフェースを提供する。その目的は下記のとおりである:

- 表記法を一貫させるための共通データ構造のタイプ定義を提供する。
- CIM フレームワークで一貫して使われる共通例外の定義を提供する。
- ファクトリ内のマテリアルの特定、グルーピング、移動、位置特定、およびトラッキングに共通のマテリアル・アーキテクチャ・インタフェースを提供する。
- 装置、センサ、耐久消費財、および人間を含むファクトリ・リソースの用途の定義、整理、トラッキング、およびそれらリソースの維持管理に共通のファクトリ・リソース・アーキテクチャ・インタフェースを提供する。
- ファクトリ内の作業の生成、実行および管理に共通のジョブ・アーキテクチャ・インタフェースを提供する。ジョブ・アーキテクチャはマテリアル処理ジョブ、マテリアル搬送ジョブ、リソース・メンテナンス・ジョブ、および製品マテリアルをプロセス・フロー内で流すファクトリ・ジョブに特化されている。

注意: このスタンダードはその使用に関連したすべての安全問題を取り扱うことを意図していない。このスタンダードの使用者は、その責任において、適切な安全および健康上実施すべき事柄を確立し、また使用前に法規制やその他の制限への適用性を判断するものである。

3 制限

3.1 暫定ステータス

3.1.1 本仕様は未完成の部分があるため、暫定とされている。今後出されるバロットをフル・スタンダードにするために処理しなければならない箇所を下記に要約しておく。

3.1.2 本仕様ではとところどころで IDL の typedef “any” を使用している。これによって柔軟性が得られる一方、スタンダード・インタフェースを介して対話する別々のインプリメンテーションで使用する値の解釈に違いがあるため、相互運用性が損なわれるという影響もある。暫定からフル・スタンダードにする前に “any” typedefs を明確なデータ・タイプに替えなければならない。

3.1.3 CIMフレームワーク・オブジェクトに関連した履歴の検索に使うインタフェースの定義を、本文書のCIMフレームワーク・ファクトリ・サービス・コンポーネントの履歴機能に関する完全な仕様のあとに記されている抽象インタフェースに追加する必要があるかもしれない。

3.1.4 公開状態変更イベントで報告されるCIMフレームワーク状態の規定は現在のところ、状態値を列挙してそれをタイプ定義したものに基づいている。ネストされた状態および並列状態の意味をよりよく理解できる別の状態表記法があるかもしれない。CIMフレームワークに使われている状態表記法は検討され、フル・スタンダードにグレードアップされる前にたぶん変更されるだろう。

3.1.5 SEMI E81 に定義されているリソース・モデルには本仕様にはまだ掲載されていない拡張がいくつか含まれている。さまざまなリソースから持ってきたリソースの混成、リソースに関連した性能、およびリソースのトラッキング機能およびメンテナンス機能の提携などである。フル・スタンダードにグレードアップする前にこういった拡張について処理することになるだろう。

3.1.6 MESFactory, Area, およびComponentManagerについて規定されているインタフェースは、SEMI E81 の中で、ファクトリ・オペレーションに含まれるファクトリ・コンポーネントに関する部分が担当する。こういったインタフェースは将来のパロッドでは上記のコンポーネントへ移す必要があるだろう。

4 参照スタンダードおよび文書

4.1 SEMIスタンダード

SEMI E5 — SEMI Equipment Communications Standard 2 Message Content (SECS-II)

SEMI E10 — Standard for Definition and Measurement of Equipment Reliability, Availability, and Maintainability (RAM)

SEMI E81 — Provisional Specification for CIM Framework Domain Architecture

4.2 その他のスタンダード

UML Notation Guide, Version 1.1, document number ad/97-08-05, Object Management Group¹

ISO/IEC International Standard 14750 (also ITU-T Recommendation X.920) — Information Technology — Open Distributed Processing — Interface Definition Language²

注意： 別途指示のない場合、引用されたすべての文書は最新の出版バージョンである。

5 用語

5.1 定義

5.1.1 抽象インタフェース(*abstract interface*) — インプリメンテーション用ではなく継承のためだけに規定されたインタフェースで、特化されたすべてのインタフェースが共有する共通機能を標準化するのが目的である。

6 要求条件

6.1 グローバル・タイプの定義

6.1.1 本項には本質的にグローバルなデータ・タイプ定義と例外が記されている。「グローバル」というのはCIMフレームワークのあらゆるコンポーネントでも使えるようにデザインされているということである。それをCIMフレームワーク準拠のアプリケーションに実装する方法については本仕様に記されていない。そのレベルの詳細は開発作業の領域である。

6.1.2 グローバル・タイプ定義はあらゆるCIMフレームワーク・インタフェースが参照できるIDL宣言として規定されている。CIMフレームワークではtypedefというキーワードを使って基本的なオブジェクト・タイプの別名を定義して

1 UML Notation Guide v1.1 is available to the general public at <http://www.omg.org/cgi-bin/doclist.pl>, +1-508-820 4300, Object Management Group, Inc., Framingham Corporate Center, 492 Old Connecticut Path, Framingham, MA 01701.

2 ISO Central Secretariat, 1, rue de Varembe, Case postale 56, CH-1211 Genève 20, Switzerland
(国内連絡先) 日本規格協会 〒107-0052 東京都港区赤坂4-1-24 03-3583-8003

いるが、それには別の意味もある(例えば、ユニット(*units*)は文字列(*string*)を定義されていて、その内容はSEMI E5 に書かれている定義に準拠している)。構造(*struct*)というキーワードの後には、名前が付けられていてタイプが決められている値の集合であるレコード構造のタイプ宣言が続く。タイプ定義の 3 番目の形式は予約語*enum*で始まる列挙タイプである。列挙タイプはそのタイプの値として使用できるトークンのリストを宣言するのに用いられる。列挙タイプは主にCIMフレームワークイベントで通信に関してオブジェクトの状態を表すのに使われる。最後に、値の集合はシーケンス*sequence*というキーワードで宣言される。この種のタイプ定義ではシーケンスの構成要素の順番に意味がある場合とない場合がある。順番に関する制約が記されていない場合はシーケンスの構成要素の順番には意味がないものとされる。

注 1: 下記の定義で“//”あるいは“/* */”はコメントの区切りである。

読者への注: (上記注 1 に記されている)コメントは、コメントの内容に書かれている項目の直前に記されている。

6.1.3 CIMフレームワーク仕様はすべて、CIMフレームワークの全コンポーネントに及ぶCIMFWモジュールのコンテキストで宣言されている。CIMFWモジュールでは、グローバル・タイプ定義はGlobalと呼ばれる第 2 レベルモジュールの中で宣言される。

モジュール: Global

```
typedef string Identifier;
typedef unsigned long Flags;
struct NamedValue {
    Identifier name;
    any argument;
    long len;
    Flags arg_modes;
};
typedef NamedValue NameValue;
typedef sequence <NameValue> NameValueSequence;
typedef string PropertyName;
struct Property {
    PropertyName property_name;
    any property_value;
};
typedef sequence <Property> Properties;
```

/*下記のタイプ定義はファクトリ・パラメータ、計測などの単位を表し、単位の表記法については SEMI E5 スタandardに準拠する。SEMI E5 スタandardでは文字列には単位の値を表すコードが含まれる。例えば“ns”はナノ秒を表し、“A”はアンペア、“wfr”はウェーハを表す。*/

```
typedef string Unit;
typedef string Units;
```

/*下記のタイプ定義は文字列のシーケンスを表す。*/

```
typedef sequence <string> StringSequence;
```

/*下記のタイプ定義はあらゆる型の値のシーケンスを表す。*/

```
typedef sequence <any> AnySequence;
```

/*下記のタイプ定義は long 値のシーケンスを表す。*/

```
typedef sequence <long> LongSequence;
```

/*下記の列挙タイプはイベントのプライオリティを示し、イベント定義一つ一つに使われる。*/

```
enum PriorityOfEvent {  
    Low,  
    Medium,  
    High,  
    Alarm };
```

/*下記の列挙タイプはオブジェクトのライフ・サイクル状態を示す。状態変更のイベント通知で使われる。*/

```
enum LifecycleState {  
    Undefined,  
    Created,  
    Deleted,  
    Moved,  
    Copied };
```

/*下記の列挙タイプは確保できるオブジェクト状態 (Lot, Durable および Machine)を示す。状態変更のイベント通知で使われる。*/

```
enum ReservationState {  
    UndefinedReservationState,  
    Reserved,  
    UnReserved };
```

/*下記の列挙タイプはマシン・リソースおよびサポート・リソースの SEMI E10 状態を表す。状態変更のイベント通知で使われる。*/

```
enum E10State {  
    E10Productive,  
    E10Standby,  
    E10Engineering,  
    E10ScheduledDowntime,  
    E10UnscheduledDowntime,  
    E10NonscheduledTime };
```

/* *TimeT* は *ulonglong* 値 (64 ビット) で、基準時間 (グリニッジ標準時での時刻を表す万国標準時表記での 1582 年 10 月 15 日 00:00) から経過した時間を 100 ナノ秒刻みで表す。*TimeT* の仕様は下記の通り: */

```
struct ulonglong {  
    unsigned long low;  
    unsigned long high;  
};
```

```
typedef ulonglong TimeT;
```

/* *TimeStamp* はデータタイプ *TimeT* に対応する。*/

```
typedef TimeT TimeStamp;
```

```
typedef sequence <TimeStamp> TimeStampSequence;
```

/*開始時間および終了時間を表す間隔という概念は *IntervalT* と呼ばれる構造で表される。*/

```
struct IntervalT {  
    TimeT lower_bound ;  
    TimeT upper_bound ;  
};
```

/* *TimeWindow* はデータタイプ *IntervalT* に対応する。*/

```
typedef IntervalT TimeWindow;
```

/*持続時間はデータタイプ *TimeT* に対応する。*/

```
typedef TimeT Duration;
```

/*下記の構造はひとつのスケジュール・インスタンスを表すためのものである。“EndTime”は絶対に“StartTime”より前にきてはいけないことに注意する。*/

```
struct ResourceSchedule {  
    TimeStamp plannedStartTime;  
    TimeStamp plannedEndTime;  
    TimeStamp actualStartTime;  
    TimeStamp actualEndTime;  
};
```

/* ResourceSchedules のシーケンスの定義。このシーケンスは時間が早い順に並べられ、シーケンスをどのように操作した場合でもその順番は保たれなければならない。*/

```
typedef sequence <ResourceSchedule> ResourceScheduleSequence;
```

6.2 グローバル例外宣言

6.2.1 本項にはコンポーネントのオペレーションによって発生する可能性のある標準的なCIMフレームワーク例外が記述されている。

モジュール: Global

/*下記のシグナルは探索が失敗した場合に発行される。*/

```
exception NotFoundSignal { string errorMessage; };
```

/*下記の信号は追加しようとした識別子を持つオブジェクトがすでに存在していたために追加が失敗した場合に発行される。インタフェースが<ObjectType>DuplicateSignalを定義して発行する場合もある*/

```
exception DuplicateIdentifierSignal {  
    string errorMessage;  
    string duplicateIdentifier; };
```

/*下記の信号はあるオブジェクトについて無効な状態遷移要求が出された場合に発行される。*/

```
exception InvalidStateTransitionSignal {  
    string errorMessage; };
```

/*下記の信号は“set”属性の値が範囲外だった場合に発行される。*/

```
exception SetValueOutOfRangeSignal {  
    string errorMessage; };
```

/*下記の信号は正しくない TimePeriod が使われた場合に発行される。*/

```
exception TimePeriodInvalidSignal {  
    string errorMessage; };
```

/*下記の信号はプロパティ名が有効でない場合に発行される。*/

```
exception InvalidPropertyNameSignal {};
```

/*下記の信号は指定された名前をもつプロパティが定義されていない場合に発行される。*/

```
exception PropertyNotFoundSignal {};
```

/*下記の信号はプロパティがサポートされていない場合に発行される。*/

```
exception UnsupportedPropertySignal {};
```

/*下記の信号はプロパティが読み込み専用で設定不可の場合に発行される。*/

```
exception ReadOnlyPropertySignal {};
```

/*下記の信号はエラー状態が定義されていなかった場合に発行される。*/

```
exception FrameworkErrorSignal {
    string errorMessage;
    unsigned long errorCode;
    any errorInformation; };
```

/* FrameworkErrorSignal のフィールドの定義:

errorMessage 例外の状況を説明するテキストフィールド。開発者が例外のデバ깅に使用する。

errorCode 指定された例外のコードを表す数値フィールド。

errorInformation 例外の状況に関連した詳細なデバ깅情報。

ErrorCode には CIM フレームワークで定義され標準化されている予約値がある。

- 0000～0999 CIM フレームワークのために予約。
- 1000～1999 CIM フレームワークの拡張のために予約。
- 2000～2999 特化された CIM フレームワークインタフェースのために予約。
- 3000～maximum 実装者のために予約。/*

/* サプライヤはインプリメンテーションを提供しないが、オペレーションを実行しても何も起きていないとユーザに伝える必要がある場合には下記の errorCode を使わなければならない。*/

```
const unsigned long NOT_IMPLEMENTED = 0;
```

/* そのオペレーションを特化インタフェースが実装するとサプライヤが考えている場合には下記の errorCode を使わなければならない。この例外を受信すると、インタフェースが適切に特化していないことがユーザにわかる。*/

```
const unsigned long IMPLEMENTED_BY_SUBCLASS = 1;
```

/* インプリメンテーションは未定義の例外をキャッチしたが、クラッシュしないでその「未定義」例外をこの定義済み例外にマッピングできるという場合には、下記の errorCode を使わなければならない。プログラムのデバ깅の役には立たないだろうが、プログラムのクラッシュは避けられるだろう。*/

```
const unsigned long UNKNOWN_EXCEPTION = 2;
```

/* 未定義エラーが発生してサーバー・オブジェクトがあいまいな状態になっている場合はこの errorCode を使わなければならない。*/

```
const unsigned long COMPLETION_UNKNOWN = 3;
```

6.3 抽象インタフェースのタイプ定義

6.3.1 参照宣言

6.3.1.1 下記の宣言は本仕様の一部ではないが、抽象インタフェース・エレメントが参照しなければならないものである。そういった参照宣言は別の文書で定義されているが、IDL コンパイルに出てくる従属物として特記しておく。

モジュール: EquipmentTracking

インタフェース: Machine

```
module EquipmentTracking {
    interface Machine {}; // Stub
    typedef sequence <Machine> MachineSequence;
    exception MachineDuplicateSignal { };
    exception MachineNotAssignedSignal { };
    exception MachineRemovalFailedSignal { };
}; // module EquipmentTracking
```

モジュール: Labor

インタフェース: Person

```
module Labor {  
  interface Person {}; // Stub  
  typedef sequence <Person> PersonSequence;  
  exception PersonDuplicateSignal { };  
  exception PersonNotAssignedSignal { };  
  exception PersonRemovalFailedSignal { };  
}; // module Labor
```

6.3.2 抽象インタフェース宣言

モジュール: AbstractIF

/*下記の IDL インタフェースの完全な定義は後述されている。これらのインタフェースはシーケンス typedef をサポートするために前方参照としてここで宣言されている。*/

```
interface Resource;  
interface Material;  
interface MaterialGroup;  
interface JobSupervisor;  
interface Job;  
interface JobRequestor;  
/*インタフェース・インスタンスのシーケンスのタイプ定義。*/  
typedef sequence <Resource> ResourceSequence;  
typedef sequence <Material> MaterialSequence;  
typedef sequence <MaterialGroup> MaterialGroupSequence;  
typedef sequence <Job> JobSequence;  
typedef sequence <JobSupervisor> JobSupervisorSequence;
```

6.4 リソース抽象インタフェース・グループ

6.4.1 リソース抽象インタフェース・グループはグローバルに利用できる抽象セットを提供する。図 1 はリソース抽象インタフェース・グループの情報モデルである。

Resource Abstract Interface Group

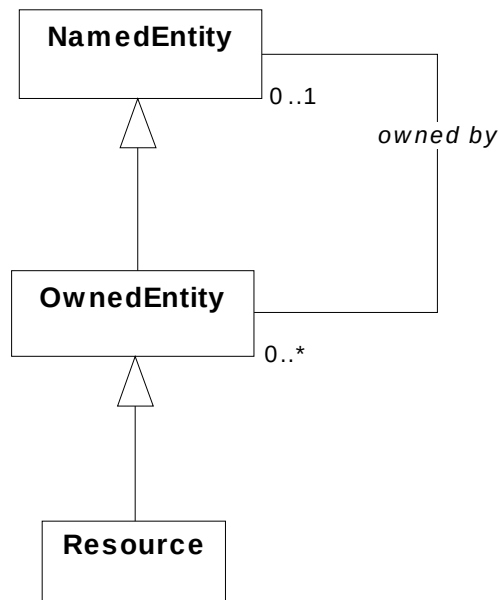


図 1

リソース抽象インタフェース・グループの情報モデル

6.4.1.1 CIMフレームワーク・インタフェースはすべて図 1 に描かれているインタフェースのひとつから継承する。NamedEntityは最も基本的なネーミング機能を提供する。OwnedEntityは所有というコンセプトをサポートする機能を持ったNamedEntityである。Resourceは製品の製造に対し能動的な役割を持つOwnedEntityである。

6.4.2 名前つきエンティティのインタフェース

モジュール: AbstractIF

インタフェース: NamedEntity

継承インタフェース: インプリメンテーションによって異なる。

説明: 抽象インタフェース NamedEntity は名前が付けられているアイテムというコンセプトを提供する。このインタフェースによってスタンダード・オブジェクトを使つての名前の比較および変換ができる。

例外: なし。

公開イベント: なし。

提供されたサービス:

interface NamedEntity {

/*名前を設定し、取得する。NamedEntity インタフェースは名前の範囲を指定せず、名前のユニーク性も強制しない。そのため、ある NamedEntity の別個のインスタンスがおなじ文字列を名前として使用することができる。*/

void setName (in string name)
 raises (Global::FrameworkErrorSignal);

```
string getName ( )
    raises (Global::FrameworkErrorSignal);

/*引数として与えられた名前の同一性をテストする。*/
boolean isNamed (in string testName)
    raises (Global::FrameworkErrorSignal);
}; // NamedEntity
```

短縮されたサービス: なし。

動的モデル: なし。

6.4.3 所有されたエンティティのインタフェース

6.4.3.1 CIMフレームワークでは所有権というコンセプトはひとつのオブジェクトが他のオブジェクトを「所有する」という場合に定義される階層型構造に関連している。資産としての物の価値に関する所有権というビジネスコンセプトと混同してはならない。

モジュール: AbstractIF

インタフェース: OwnedEntity

継承インタフェース: NamedEntity

説明: 抽象インタフェース OwnedEntity は「所有された」エンティティというコンセプトを提供する。OwnedEntity のインスタンスにはそれぞれ「所有者」がひとつだけある。OwnedEntity は所有者と通信してサービスを要求したり関心のある情報を転送したりできる。階層は所有の繋がりにより作られる。

例外: なし。

公開イベント: なし。

提供されたサービス:

```
interface OwnedEntity : NamedEntity {
/*所有者の設定および取得。*/
void setOwner (in NamedEntity owner)
    raises (Global::FrameworkErrorSignal);
NamedEntity getOwner ( )
    raises (Global::FrameworkErrorSignal);
}; // OwnedEntity
```

短縮されたサービス: なし。

動的モデル: なし。

6.4.4 リソース・インタフェース

モジュール: AbstractIF

インタフェース: Resource

継承インタフェース: OwnedEntity

説明: Resource は、工場内で製品が(価値が付加される)製造ライフサイクルの中で能動的に関わるあらゆるエンティティに関する継承された抽象インタフェースである。これには工場自体、職員、製作、企画およびスケジューリングのリソース、およびマテリアルの処理、搬送および保存に使用される装置全部が含まれる。Resource はモニタリングおよびコントロールに関する

る共通サービスを提供する。Resource はリソース階層の構築のために NamedEntity と OwnedEntity の特性を一緒に利用する。

Resource の状態と、Resource が表す物理的エンティティの状態ははっきり区別しなければならない。例えばマシンはリソースのひとつだが、マシンが「稼働中止」であるという事実が、現場では物理的装置がシャットダウンしているということを意味するのではない。実際には装置はマニュアルモードで作動しているかもしれない。Resource 状態は Resource オブジェクトが使用できてファクトリ・システムの作業を受け付けられるかどうかを表している。

例外: なし。

公開イベント: なし。

提供されたサービス:

```
interface Resource : OwnedEntity {
```

```
/*下記の Resource に関してスタートアップ活動を実行する。Resource の特化インタフェースは実装しなければならない。*/
```

```
void startup ( )
    raises (Global::FrameworkErrorSignal);
```

```
/*下記の Resource に関して正常なシャットダウン活動を実行する。正常とは Resource がその時点で実行中の活動をすべて完了させて「きれいに」シャットダウンできる状態と定義される。*/
```

```
void shutdownNormal ( )
    raises (Global::FrameworkErrorSignal);
```

```
/*下記の Resource に関して即時シャットダウン活動を実行する。即時とはその時点で実行中の活動をすべてアボートあるいは終了させてできるだけ早く活動をストップさせることと定義される。Resource の特化インタフェースは実装しなければならない。*/
```

```
void shutdownImmediate ( )
    raises (Global::FrameworkErrorSignal);
```

```
/* Resource 階層のレシーバのレベルを答える。Resource のタイプごとに“resourceLevel”識別子を提供するよう Resource で規定されている。
```

```
string resourceLevel ( )
    raises (Global::FrameworkErrorSignal);
```

```
/*下記のサービスは Resource の名前の範囲を提供する。Resource の名前の範囲は「リソース・レベル」という考え方と所有権階層を利用している。例えば Machine 内で MachineResources にユニークな識別子をつけることは可能だが、それを Machine 外で特定するにはその所有権に関する追加情報が必要になる。
```

つまり: “Chamber” という名前の ProcessResource に送信された nameQualifiedTo (“Machine”) の答は下記となる。

“TestMachine>Chamber”。

その Machine が “TestFactory” という名前の Factory によって所有されているなら;

ProcessResource に送信された nameQualifiedTo (“Factory”) の答は下記となる。

“TestFactory>TestMachine>Chamber”

ここでようやく ProcessResource はこの Factory に関してユニークに特定されることになる。

このようにして指定するレベルの数に限度はない。nameQualifiedTo (string); の実装により、ある Resource レベルが識別できない場合、必ずしも名前をつなげなくてもよい。*/

```
string nameQualifiedTo (in string resourceLevel)
    raises (Global::FrameworkErrorSignal);
```

/*指定された Resource の従属 Resource を返す。*/

```
ResourceSequence subResources ( )
    raises (Global::FrameworkErrorSignal);
```

/*そのリソースが正常稼動中かどうかを答える。正常稼動中とは、そのリソースが機能していて通常のタスクを受け付けて実行する用意が整っているという意味である。Resource の派生インタフェースで、こういった余分な問題をキャパシティ、「通常」作業対メンテナンス、などとしてはっきりと処理するものはこの状態を拡張させる(例えばサブインタフェースを追加する)ことになっている。*/

```
boolean isInService ( );
```

/*そのリソースが稼動中止中であるかどうかを答える。稼動中止とはリソースが新しいタスクを受け付けるあるいは開始することができないことを意味する。すでに開始されているタスクは続行できる場合もある。*/

```
boolean isOutOfService ( );
```

```
}; // Resource
```

短縮されたサービス: なし。

動的モデル:

Resource のインプリメンテーションは状態モデルを拡張して、ここに定義されている状態にすっぽり含まれる下位状態を追加してもよい。状態遷移を追加して状態モデルを拡張することは Resource のサブタイプのオプションでもある。

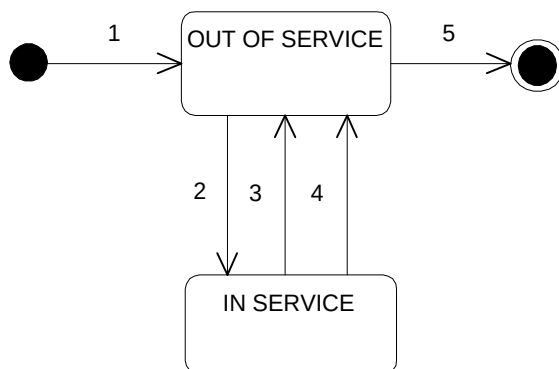


図 2
リソースの動的モデル

表 1 リソースの状態定義および問い合わせ表

状態	定義	状態問い合わせに使うメッセージ
IN SERVICE	リソースはフルサービス・インタフェースを使って他のリソースと対話できる	Resource のインスタンスに送信されたブール数 <code>isInService ()</code> ; は TRUE を返す。
OUT OF SERVICE	リソースはサービスを提供することができない。	Resource のインスタンスに送信されたブール数 <code>isOutOfService ()</code> ; は TRUE を返す。

表 2 リソースの状態遷移

#	現在の状態	トリガー	新しい状態	アクション	コメント
1	Non Existent	オブジェクトの生成	OUT OF SERVICE	なし	デフォルトで入る状態。
2	OUT OF SERVICE	startUp ()あるいはオブジェクトの初期化	IN SERVICE	Resource およびサブリソースを起動する	startUp()サービスがトリガーを起動させる。
3	IN SERVICE	shutdownNormal ().	OUT OF SERVICE	リソースがその時点で実行している活動を正常に完了させる	shutdownNormal()サービスがトリガーを起動させる。
4	IN SERVICE	shutdownImmediate ().リソースを稼動中止にする。	OUT OF SERVICE	リソースが実行している活動を即座にストップさせる。	shutdownImmediate()サービスがトリガーを起動させる。
5	OUT OF SERVICE	リソースを除去する。	Non Existent	なし	

6.5 マテリアル抽象インタフェース・グループ

6.5.1 CIMフレームワークではマテリアルの特定、グルーピング、および位置特定のために共通アーキテクチャを使用する。どのマテリアルにもID、履歴、位置、およびそのマテリアルが入っているマテリアル容器に関連した情報がある。マテリアルは物理的(同じ位置あるいはキャリア)あるいは論理的(同じロット、同じプロセスバッチ、同じプロダクトファミリーなど)に関連している複数のマテリアルグループに属することができる。マテリアルを特化したものにはプロダクト、耐久消費財、および消耗品がある。マテリアル・グループを特化したものにはロット、およびプロセスグループがある。

Material Abstract Interface Group

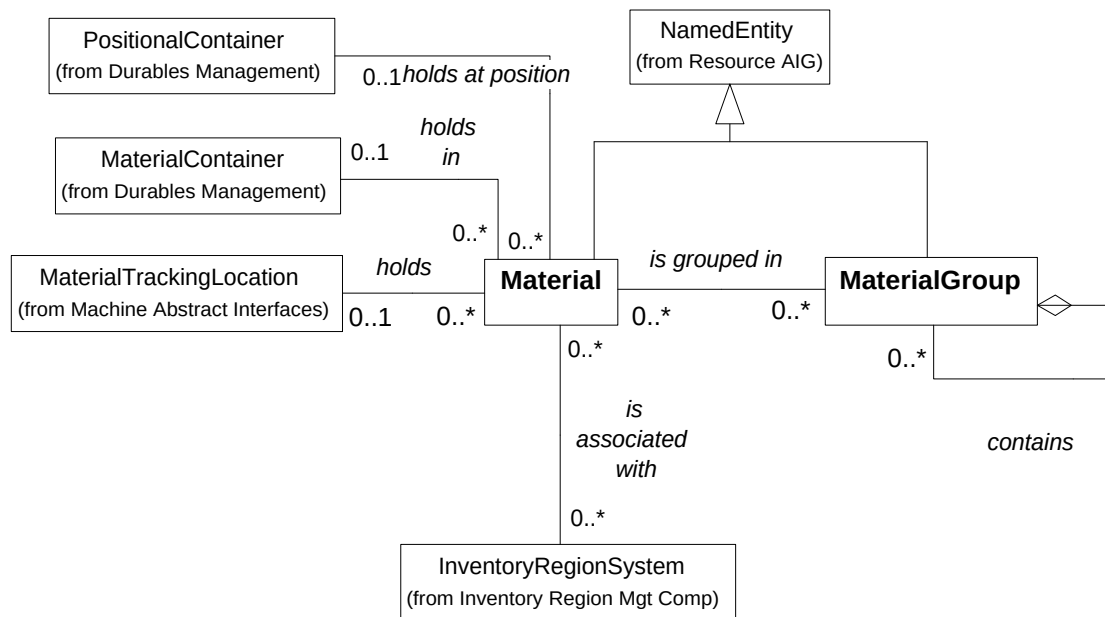


図 3

マテリアル抽象インタフェース・グループの情報モデル

6.5.2 マテリアル・インタフェース

モジュール: AbstractIF

インタフェース: Material

継承インタフェース: NamedEntity

説明: **Material** は製造プロセスにインプットするものとして必要な物理アイテムあるいは物質の抽象インタフェースである。これには製品それ自体、製造プロセスで使用される消耗品および耐久消費財が含まれる。マテリアルを製品に変えるのに使われるリソースは含まれない。

例外: なし。

公開イベント: なし。

提供されたサービス:

```
interface Material : NamedEntity {
```

```
/*マテリアルの識別子を設定し、取得する。識別子はひとつの MES ファクトリのマテリアル全部の中でユニークなものとする。*/
```

```
string getIdentifier ( )  
    raises (Global::FrameworkErrorSignal);
```

```
void setIdentifier (in string identifier)  
    raises (Global::FrameworkErrorSignal,  
           Global::DuplicateIdentifierSignal);
```

```
/*レシーバが属しているマテリアル・グループを返す。*/
```

```
MaterialGroupSequence materialGroups ( )  
    raises (Global::FrameworkErrorSignal);
```

```
/*引き数で示されているマテリアル・グループにレシーバが属しているかどうかを返す。*/
```

```
boolean isMemberOf (in MaterialGroup aMaterialGroup)  
    raises (Global::FrameworkErrorSignal);
```

```
}; // Material
```

動的モデル: なし。

6.5.3 マテリアル・グループ・インタフェース

モジュール: AbstractIF

インタフェース: MaterialGroup

継承インタフェース: NamedEntity

説明: **MaterialGroup** は **Material** の集まりの抽象インタフェースである。

```
interface MaterialGroup : NamedEntity {
```

例外:

```
/*すでにグループに属している Material を MaterialGroup に追加しようとする信号を送る。*/
```

```
exception DuplicateMaterialSignal {Material aMaterial;;}
```

```
/*すでに属している MaterialGroup を MaterialGroup に追加しようとする信号を送る。*/
```

```
exception DuplicateMaterialGroupSignal {Material aMaterialGroup;;}
```

```
/*この MaterialGroup には存在していない Material を削除しようとする信号を送る。*/
```

```
exception MaterialRemovalFailedSignal {Material aMaterial;;}
```

```
/*この MaterialGroup には存在していない MaterialGroup を削除しようとする信号を送る。*/
```

```
exception MaterialGroupRemovalFailedSignal {MaterialGroup aMaterialGroup;;}
```

公開イベント: なし。

提供されたサービス:

```
/*グループのユニークな識別子を設定し、取得する。*/
string getIdentifier ( )
    raises (Global::FrameworkErrorSignal);

void setIdentifier (in string identifier)
    raises (Global::FrameworkErrorSignal,
            Global::DuplicateIdentifierSignal);

/*レシーバが持っている Material の集まりに引き数である MaterialSequence を追加する。*/
void addMaterials (in MaterialSequence aMaterialSequence)
    raises (Global::FrameworkErrorSignal,
            DuplicateMaterialSignal);

/*レシーバが持っている Material の集まりに引き数である Material を追加する。*/
void addMaterial (in Material aMaterial)
    raises (Global::FrameworkErrorSignal,
            DuplicateMaterialSignal);

/*表示された Material を MaterialGroup から削除する。見つからない場合は例外を発生させる。*/
void removeMaterial (in Material aMaterial)
    raises (Global::FrameworkErrorSignal,
            MaterialRemovalFailedSignal,
            Global::NotFoundSignal);

/* MaterialGroup から Material を全部削除し、返す。*/
MaterialSequence removeAllMaterials ( )
    raises (Global::FrameworkErrorSignal);

/*この MaterialGroup に引き数である MaterialGroup を追加する。*/
void addMaterialGroup (in MaterialGroup aMaterialGroup)
    raises (Global::FrameworkErrorSignal,
            DuplicateMaterialGroupSignal);

/*この MaterialGroup から引き数である MaterialGroup を削除する。*/
void removeMaterialGroup (in MaterialGroup aMaterialGroup)
    raises (Global::FrameworkErrorSignal,
            MaterialGroupRemovalFailedSignal,
            Global::NotFoundSignal);

/*この MaterialGroup からすべての MaterialGroup を削除し、返す。*/
MaterialGroupSequence removeAllMaterialGroups ( )
    raises (Global::FrameworkErrorSignal);

/*この MaterialGroup にある material の集まりを返す。*/
MaterialSequence allMaterials ( )
    raises (Global::FrameworkErrorSignal);

/*この MaterialGroup にある MaterialGroup 全部を返す。*/
MaterialGroupSequence allMaterialGroups ( )
    raises (Global::FrameworkErrorSignal);

/* MaterialGroup にあるアイテム数(カウント)を返す。*/
long count ( )
    raises (Global::FrameworkErrorSignal);
```

```
}; // MaterialGroup
```

動的モデル: なし。

6.6 ジョブ監視抽象インタフェース・グループ

6.6.1 ジョブ監視抽象インタフェース・グループは「ジョブ」の生成、実行および管理に共通した抽象を提供する。ここでジョブとはJobSupervisorが要求できる何らかのシステム・レベル・オペレーションと定義できる。ジョブはかなりの時間にわたって、システム内の複数のリソースを使うことがよくある。システムの問題を解決するため、特化インタフェースは独自のジョブ・スーパーバイザとジョブを提供することになっている。

Job Supervision Abstract Interface Group

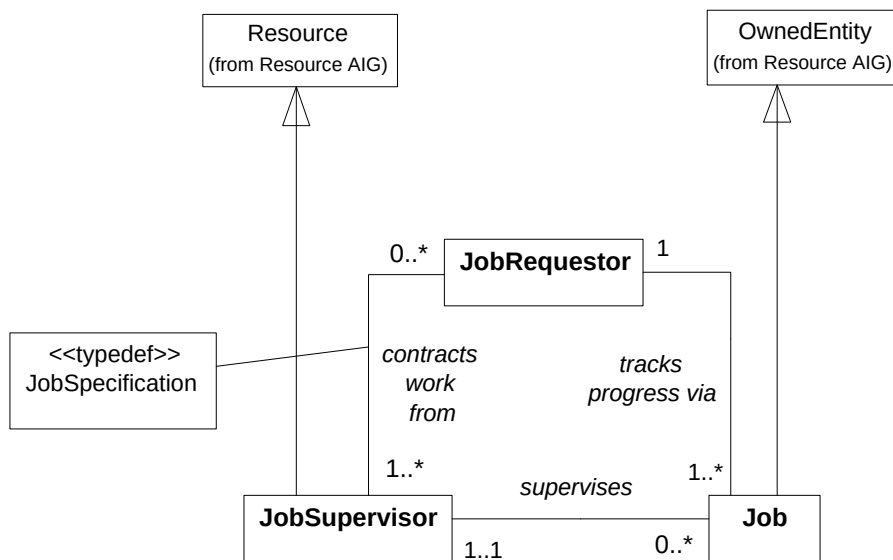


図 4

ジョブ監視抽象インタフェース・グループの情報モデル

6.6.1.1 基本的なジョブ監視抽象インタフェース・グループは他のコンポーネントと対話しないが、グループのインタフェースを他のコンポーネントがインスタンス化するという程度の係わり合いはある。図 4 にはジョブ監視のインタフェースが互いここのような関係にあるかが描かれている。

6.6.1.2 ジョブ仕様は実行すべき仕事を十分に定義するのに必要なパラメータを含むプロパティのシーケンスである。このシーケンスはJobRequestorがジョブ要求メッセージでJobSupervisorに渡す。詳細についてはJobSupervisorインタフェースを参照のこと。

6.6.1.3 JobSupervisionレベルは階層状になっている。一つのレベルがジョブを受け付け、自分より下のレベルにそのジョブの一部を委託する。しかしジョブは純粋な階層状にはなっていない。あるJobSupervisorが受け付けたジョブは分割され、おなじコンポーネントの他のジョブと一緒に、ファクトリの活動の効率を最高にするため、必要に応じて再構成されることがある。

6.6.1.4 例えばProductRequestが特定の製品のウェーハを 15 枚要求したとする。ProductRequestManagerは、そのウェーハ 15 枚と別のProductRequestからのウェーハ 10 枚が入っているロットを使うファクトリ・オペレーションにLotJobを委託する。ファクトリでは、このロットは分割されてより小さなグループになってさまざまな段階で処理されるかもしれない。あるいはスクラップのためにウェーハ数が減ったので、別の小さなロットと組み合わせられてもっと効率的なプロセス・グループになるかもしれない。ジョブ監視のインプリメンテーションはパフォーマンスを最適にするために相当自由が許されている。このインプリメンテーションの唯一の要求事項はジョブの仕様を満たすことである。

6.6.2 JobSupervisor インタフェース

モジュール: AbstractIF

インタフェース: JobSupervisor

継承インタフェース: Resource

説明: JobSupervisor は JobSupervisor を実装しているコンポーネントが実行しているジョブすべてを管理する。JobSupervisor は仕事に関する要求を受信し、そのタスクのジョブの生成を助け、そのジョブ (の呼び名) を返す。

JobSupervisor は作業の実行を依頼できる、十分に定義された領域を持つ。領域とは作業を委託するなら CIM フレームワーク・リソースかもしれないし、自分で作業を実行するなら内部リソースかもしれない。ある JobSupervisor が自分に対して発行し、自分で受け付けるのは、自分の領域内で遂行できる活動要求だけでなければならない。

ジョブは NamedEntity のサブタイプであり、ジョブに関する問い合わせに名前の属性を使用できるように JobSupervisor がジョブの名前を付ける。別の JobSupervisors からきたジョブが同じ名前をもっている場合もある。

実行すべき作業の定義は JobSpecification である。これは名前/値を一組としてそれがいくつも連なったシーケンスである (「プロパティ」の定義参照)。ジョブ監視の特化インタフェースは JobSpecification に特別なプロパティを必要とする場合がある。一般的に利用できるプロパティがいくつか下記の表に定義されている。できるなら、特化インタフェースはこれらの定義を再利用すべきである。特化インタフェースは自分がサポートしている許容プロパティおよび必須プロパティを文書に記しておくべきでもある。詳細についてはジョブの定義を参照のこと。

```
interface JobSupervisor : Resource {
```

タイプ定義:

/*ジョブ実行の結果を返すためのタイプ*/

```
typedef Global::NameValueSequence Results;
```

JobSpecification のプロパティ:

名前	値のタイプ	説明
"JobType"	文字列	ランするジョブの種類。JobSupervisor がひとつ以上のタイプのジョブを起動できる時には役に立つ。このプロパティに確保されている文字列の値は下記の通り: ProductRequestJobType, LotJobType, AreaJobType, ProcessMachineJobType, TransportJobType, PMJobType, ControlStrategyJobType, AlgorithmJobType.
"Priority"	ロング	1 から 99 までの整数値。1 が最高プライオリティ, 99 が最低。
"Deadline"	タイムスタンプ	ジョブはデッドラインの指定値より早く完了することになっている。

例外:

/*要求されたジョブは拒否された。*/

```
exception JobRejectedSignal {  
    string errorMessage; };
```

/*要求されたジョブが見つからなかった。*/

```
exception JobNotFoundSignal {
    string errorMessage;
    string missingJobName; };
```

公開イベント:

/*トラッキングのためにライフサイクルイベントを提供する。*/

```
const string JobLifecycleSubject =
    "/JobSupervision/JobSupervisor/JobLifecycle";
```

/*ここで(他のすべてのイベントでも)使っている「名前」は、イベントで言及しているジョブの名前あるいは識別子を表す文字列値を指している。フィルタリングはオブジェクトの呼び名の比較を行わないので、オブジェクトの「名前」でフィルタリングを行わなければならない。*/

```
struct JobLifecycleFilters {
    Global::Property name;
    Global::Property lifecycle;
};
```

JobLifecycleFilters のプロパティ:

名前	値のタイプ	説明
"Name"	文字列	ジョブの名前
"Lifecycle"	Global::LifecycleState	新しいライフサイクルの状態

```
struct JobLifecycleEvent {
    string eventSubject;
    Global::TimeStamp eventTimeStamp;
    JobLifecycleFilters eventFilterData;
    Global::Properties eventNews;
    Job aJob; // on Delete, aJob is nil
};
```

提供されたサービス:

/*参照している仕様に従って作業を実行するよう要求する。後で参照できるよう、その作業を表すジョブが返される。このオペレーションの事後条件は指定されたジョブの生成の成功である。*/

```
Job requestJob (
    in Global::Properties aJobSpecification,
    in JobRequestor aJobRequestor)
    raises (Global::FrameworkErrorSignal,
    JobRejectedSignal);
```

/*ジョブ仕様に従って作業を実行するよう要求する。このオペレーションはジョブが完了するまでブロックする。ジョブによって発生した結果が返される。このオペレーションの事後条件は指定されたジョブの実行の成功である。このインタフェースは、要求者が JobRequestor インタフェース(例, informJobStateChange オペレーション)をサポートしていなくてもよいので、**requestJob** の代わりになる。runJob によってジョブが生成される場合とされない場合があるが、生成された場合は、JobSupervisor のジョブ問い合わせオペレーションでそのジョブにアクセスできる。*/

```
Results runJob (
    in Global::Properties aJobSpecification)
    raises (Global::FrameworkErrorSignal,
    JobRejectedSignal);
```

/* RequestJob メッセージあるいは runJob メッセージがいま発行されたら、JobSpecification で指定されたジョブが現在進行中の処理あるいは(キューイングされて)この後の処理で受け付けられるかどうかをたずねる。*/

```
boolean canPerform (in Properties aJobSpecification)
    raises (Global::FrameworkErrorSignal);
```