

SEMI E42-0704

レシピ管理スタンダード:コンセプト, 挙動, およびメッセージサービス

免責事項:この SEMI スタンダードは, 投票により作成された英語版が正式なものであり, 日本語版は日本の利用者各位の便宜のために作成したものです。万が一英語と日本語とに差異がある場合には英語版記載内容が優先されます。

本スタンダードは, Global Information and Control Committee で技術的に承認されたもので Japanese Information and Control Committee が直接責任を負うものである。現版は 2004 年 4 月 30 日に Japanese Regional Standards Committee にて承認された。2004 年 6 月にまず www.semi.org で入手可能になり, 2004 年 7 月発行に至る。初版は 1995 年 9 月発行。前版は 2000 年 2 月に出版された。

目次

1. 序文

- 1.1 目的
- 1.2 範囲
- 1.3 参照文書
- 1.4 定義
 - 1.4.1 サービス
 - 1.4.2 形式
 - 1.4.3 レシピ管理
- 1.5 表記法
 - 1.5.1 テキスト文字列の制約
 - 1.5.2 ハレル状態モデル
 - 1.5.3 オブジェクト
 - 1.5.3.1 OMT オブジェクト情報モデル
 - 1.5.3.2 オブジェクト属性の表示
 - 1.5.4 サービスメッセージの表現
- 1.6 要求事項
- 1.7 文書の構成
- 1.8 適用文書

2. RMS の概要

- 2.1 レシピ管理モデル
- 2.2 RMS の主なオブジェクト
 - 2.2.1 レシピ
 - 2.2.2 レシピネームスペース
 - 2.2.3 レシピネームスペースマネージャ
 - 2.2.4 レシピエグゼキュータ

3. レシピ

- 3.1 動機
- 3.2 基本的なコンセプト
 - 3.2.1 レシピのタイプ
 - 3.2.2 レシピの構造
 - 3.2.2.1 レシピ本体
 - 3.2.3 レシピの識別
 - 3.2.3.1 レシピ名
 - 3.2.3.2 レシピクラス
 - 3.2.3.3 バージョン
 - 3.2.3.4 レシピ識別子

- 3.2.4 レシピの先進機能
 - 3.2.4.1 マルチパートレシピ
 - 3.2.4.2 変数パラメータ
- 3.2.5 属性
 - 3.2.5.1 記述子
- 3.3 フルレシピモデルおよび最少レシピモデル
- 3.4 管理レシピ
 - 3.4.1 包括的属性
 - 3.4.1.1 タイムスタンプ属性
 - 3.4.1.2 長さ属性
 - 3.4.1.3 記述子
 - 3.4.2 管理レシピオブジェクト属性の定義
 - 3.4.2.1 包括的属性の定義
 - 3.4.2.2 エージェント固有の属性の定義
 - 3.4.2.3 最少の管理レシピ
- 4. レシピネームスペース
 - 4.1 動機
 - 4.2 ネームスペースモデル
 - 4.3 ネームスペースの仕様
 - 4.4 メンバーエージェント
 - 4.5 イラストレーション
 - 4.6 属性定義表
- 5. 分散レシピネームスペース
 - 5.1 動機
 - 5.2 概要
 - 5.3 分散レシピネームスペースの問題
 - 5.3.1 オブジェクトサービス
 - 5.3.2 論理レシピ
 - 5.3.3 変更要求
 - 5.4 分散レシピネームスペースセグメント
 - 5.4.1 マスターおよび専用セグメント
 - 5.4.2 変更の制約
 - 5.5 分散レシピネームスペースレコーダ
 - 5.6 分散レシピネームスペース管理情報
 - 5.7 分散レシピネームスペース
 - 5.8 分散レシピネームスペースマネージャ
 - 5.8.1 変更管理
 - 5.9 分散レシピネームスペースの構築
 - 5.10 壊れた分散レシピネームスペースの再構築
 - 5.11 オブジェクト属性定義表
 - 5.11.1 分散レシピネームスペースセグメント属性の定義
 - 5.11.2 分散レシピネームスペースレコーダ属性の定義
 - 5.11.3 分散レシピネームスペース属性の定義
 - 5.11.4 分散レシピネームスペースマネージャ属性の定義
- 6. レシピエグゼキュータ
 - 6.1 動機
 - 6.2 説明
 - 6.3 実行レシピ
 - 6.3.1 管理レシピと実行レシピの比較
 - 6.3.2 ダウンロードされたレシピ

- 6.3.3 実行レシピの識別子
 - 6.3.4 実行レシピの記述子
 - 6.3.5 実行レシピ属性の定義
- 6.4 デフォルトネームスペース
- 6.5 レシピ保存
- 6.6 変更コントロール
 - 6.6.1 レシピ生成
 - 6.6.2 レシピの圧縮
 - 6.6.3 保存されたレシピへの変更
 - 6.6.4 最終値
- 6.7 制作
- 6.8 レシピエグゼキュータの属性
- 7. エージェント
 - 7.1 定義
 - 7.2 RMS リソース
 - 7.3 エージェントの属性
- 8. レシピ管理オペレーション
 - 8.1 レシピライフサイクル
 - 8.2 オペレーションの説明
 - 8.2.1 一般的な要求
 - 8.2.2 レシピ創作
 - 8.2.2.1 レシピ作成
 - 8.2.2.2 レシピアップデート
 - 8.2.3 レシピ構築
 - 8.2.3.3 レシピのリンク解除
 - 8.2.3.4 変数パラメータ修正
 - 8.2.4 レシピ認証
 - 8.2.4.1 レシピ承認
 - 8.2.5 レシピ保護
 - 8.2.6 保護解除
 - 8.2.7 情動的オペレーション
 - 8.3 レシピ状態モデル
 - 8.4 オペレーション表
- 9. ネームスペース管理オペレーション
 - 9.1 オブジェクトサービスのアプリケーション
 - 9.1.1 オブジェクト識別子
 - 9.1.2 必要なオブジェクトサービス
 - 9.2 ネームスペースオペレーション
 - 9.2.1 ネームスペース作成
 - 9.2.2 ネームスペース削除
 - 9.2.3 ネームスペースリネーム
 - 9.3 ネームスペース情報オペレーション
 - 9.3.1 使用可能ストレージ取得
 - 9.3.2 レシピステータス検査
 - 9.3.3 ベストバージョン取得
 - 9.4 ネームスペース情報オペレーション
 - 9.4.1 レシピ作成
 - 9.4.2 レシピ削除
 - 9.4.3 レシピ保存
 - 9.4.4 レシピ検索

- 9.4.5 レシピコピー
- 9.4.6 レシピリネーム
- 9.4.7 レシピ検証
- 9.4.8 レシピダウンロード
- 9.4.9 レシピアップロード

9.6 オペレーション表

9.7 ネームスペースイベント

10. 分散レシピネームスペース管理オペレーション

10.1 分散レシピネームスペースセグメントのオペレーション

10.1.1 オブジェクトサービス

10.1.1.1 属性 Read/Write

10.1.1.2 生成および削除オペレーション

10.1.1.3 オブジェクト連結オペレーション

10.1.2 セグメントレシピ管理オペレーション

10.1.2.1 承認のための要求条件

10.1.2.2 セグメント変更承認

10.1.2.3 セグメント変更要求のシナリオ

10.2 分散レシピネームスペースレコーダ

10.2.1 オブジェクトサービス

10.2.1.1 属性 Read/Write

10.2.1.2 オブジェクト生成および削除オペレーション

10.2.1.3 オブジェクト連結オペレーション

10.2.2 セグメントレコードの追加

10.2.3 セグメントレコードの削除

10.2.4 変更要求レコードの追加

10.2.5 変更要求レコードの削除

10.2.6 変更要求レコードの取得

10.3 分散レシピネームスペース管理オペレーション

10.3.1 オブジェクトサービス

10.3.2 分散レシピネームスペースの削除

10.3.3 監督されているオブジェクトの連結および連結解除

10.3.3.1 監督されているオブジェクトの連結

10.3.3.2 監督されているオブジェクトの連結解除

10.3.4 変更要求管理

10.3.4.1 外部からの変更要求

10.3.4.2 内部からの変更要求

10.3.4.3 許可できる変更要求

10.3.4.4 変更要求レコードの定義

10.3.4.5 変更要求のライフサイクル

10.3.4.6 変更要求の完了

10.3.4.7 変更管理の例

10.3.5 セグメント変更要求

10.3.6 セグメントアクション完了

10.3.7 セグメント通知

10.3.8 変更要求の取得

10.3.9 分散レシピネームスペースの再構築

10.4 オペレーション表

10.4.1 セグメントのオペレーション表

10.4.2 レコーダのオペレーション表

10.4.3 マネージャのオペレーション表

11. レシピエグゼキュータオペレーション

11.1 オブジェクトサービスオペレーション

11.1.1 実行レシピの指定子

11.2 オペレーションの説明

11.2.1 レシピのダウンロードおよび検証

11.2.2 レシピ検証

11.2.2.1 派生オブジェクト形式レシピ

11.2.2.2 検証 ID

11.2.3 レシピアップロード

11.2.4 レシピリネーム

11.2.5 使用可能メモリの獲得

11.2.6 レシピ削除

11.2.7 レシピ選択

11.2.7.1 複数選択

11.2.7.2 妥当性チェック

11.2.7.3 委任

11.2.7.4 変数パラメータ

11.2.7.5 メッセージシナリオ

11.2.7.6 選択済みレシピ

11.2.8 レシピ選択解除

11.2.9 実行レシピ記述子獲得

11.2.10 変更コントロール

11.2.10.1 既存のレシピ変更

11.2.10.2 新規レシピ生成

11.2.10.3 派生形式レシピの構築

11.2.10.4 最終値の保存

11.2.10.5 変更通知

11.3 オペレーション表

11.4 レシピエグゼキュータのイベント

12. レシピネームスペースのサービス

12.1 レシピ管理メッセージパラメータ辞書

12.2 メッセージフロー

12.3 RMNCreateNS

12.4 RMNDeleteNS

12.5 RMNRenameNS

12.6 RMNSpaceInquire

12.7 RMNRcpStatuInquire

12.8 RMNVersionInquire

12.9 RMNCreate

12.10 RMNUpdate

12.11 RMNStore

12.12 RMNRetrieve

12.13 RMNCopy

12.14 RMNRename

12.15 RMNAction

12.16 RMNVarPar

12.17 RMNGetDescriptor

12.18 RMNComplete

13. 分散レシピネームスペースのサービス

13.1 分散レシピネームスペースのメッセージパラメータディクショナリ

13.2 分散レシピネームスペースセグメントのサービス

13.2.1 RMDSApproveAction

- 13.3 分散レシピネームスペースレコーダのサービス
 - 13.3.1 RMDRAddSegRecord
 - 13.3.2 RMDRDelSegRecord
 - 13.3.3 RMDRAddChgRecord
 - 13.3.4 RMDRDelChgRecord
 - 13.3.5 RMDRGetChgRecord
- 13.4 分散レシピネームスペースマネージャのサービス
 - 13.4.1 RMDComplete
 - 13.4.2 RMDNotify
 - 13.4.3 RMDSegChange
 - 13.4.4 RMDGetChangeRequests
 - 13.4.5 RMDRebuild

14. レシピエグゼキュータサービス

- 14.1 レシピエグゼキュータメッセージパラメータディクショナリ
- 14.2 メッセージフロー
- 14.3 RMEDnldVer
- 14.4 RMEVerify
- 14.5 RMEUpload
- 14.6 RMERename
- 14.7 RMESpaceInquire
- 14.8 RMEDelete
- 14.9 RMESelect
- 14.10 RMEDeselect
- 14.11 RMEGetDescriptor
- 14.12 RMEChange
- 14.13 RMEComplete

15. レシピ管理準拠

- 15.1 準拠範囲
- 15.2 管理レシピ
- 15.3 管理レシピ準拠表
- 15.4 レシピネームスペース管理
 - 15.4.1 レシピネームスペース
 - 15.4.2 レシピネームスペースマネージャ
 - 15.4.3 レシピネームスペース管理準拠表
- 15.5 分散レシピネームスペース管理に関する準拠
 - 15.5.1 分散レシピネームスペース
 - 15.5.2 分散レシピネームスペースマネージャ
 - 15.5.3 分散レシピネームスペース管理準拠表
- 15.6 分散レシピネームスペースセグメント
 - 15.6.1 分散レシピネームスペースセグメント準拠表
- 15.7 分散レシピネームスペースレコーダ
 - 15.7.1 分散レシピネームスペースレコーダ準拠表
- 15.8 実行レシピ準拠
 - 15.8.1 実行レシピ準拠表
- 15.9 レシピエグゼキュータ
 - 15.9.1 レシピエグゼキュータ準拠表

16. 用語

関連情報 1

- R1-1 RMS スタンダード化されたオブジェクト
- R1-2 RMS 要求／コンセプトマップ
- R1-3 背景
 - R1-3.1 追跡可能性

- R1-3.2 レシピのライフサイクル
- R1-3.3 レシピ編集
- R1-3.4 レシピ共有
- R1-3.5 保護およびプロセスコントロール
- R1-3.6 レシピ選択
- R1-4 承認レベルの工場での実装の例**
- R1-5 変数パラメータの例**
- R1-6 オブジェクトサービスのアプリケーション**
 - R1-6.1 範囲
 - R1-6.2 フィルタ
 - R1-6.3 複合属性
- R1-7 RMS アプリケーションの例**

INACTIVE

SEMI E42-0704

レシピ管理スタンダード:コンセプト, 挙動, およびメッセージサービス

^E 本スタンダード出版に関して編集上 1999 年 9 月付けで修正が行われ, 暫定版ではなくなった。修正は 1.2 項に対して行われた。

1 序文

このスタンダードはレシピ管理に必要とされるコンセプト, レシピ管理スタンダード(RMS)が規定するオペレーションあるいは挙動, および, サービスプロバイダとユーザ間のインタフェースを介してそのサービスを行うのに使われるメッセージを定義する。

1.1 目的 — このスタンダードの目的は二つある:

- 一体となってレシピ管理を形成するスタンダードのコンセプト, 挙動, およびメッセージサービスの存在を前提に, それらを利用したアプリケーションソフトウェアの開発を可能にすること。
- レシピ管理能力を提供するソフトウェアの開発を可能にすること。

1.2 範囲 — これは, 半導体工場内での自動化されたレシピ管理の統合化をサポートするためのコンセプト, 挙動, およびサービスを定義するスタンダードである。これらのサービスは, 従来の**ホスト/装置**の関係およびクラスタツールコントローラ/付属モジュール通信およびコントロールの関係も含む, さまざまな関係に適用できる。

本スタンダードは, システムが半導体製造装置およびシステム内で材料が正しく処理されるようにレシピを転送し, 管理することができるような通信サービスを提供する。RMS では完璧を期するために, SEMI E39 (オブジェクトサービススタンダード(OSS):コンセプト, 挙動およびサービス)にも準拠することが要求されている。

この文書にはいくつかの異なる階層関係が書かれている:監視するものと監視されるもの, レシピとそのサブレシピ, レシピクラスとそのサブクラスなど。こういった階層関係は, 電話切り替えシステムやディレクトリツリーといった数多くのさまざまな種類のシステムに反映されている, 自然で組織的・分類的な構造を提供する。このスタンダードの意図は, システム構造に厳密な階層関係を実装するよう要求することではなく, むしろ, 論理的な階層関係をサポートすることである。

RMS では定義されたサービスをどこに実装するかについては制約を設けない。

1.3 参考文献

1.3.1 *Semiconductor Equipment and Materials International (SEMI)*¹

1.3.1.1 *SEMI Equipment Automation/Hardware Volume*

SEMI E10 — Specification for Definitions and Measurements of Equipment Reliability, Availability, and Maintainability (RAM)

1.3.1.2 *SEMI Equipment Automation/Software 2 Volume*

SEMI E30 — Generic Model for Communications and Control of SEMI Equipment (GEM)

SEMI E39 — Object Services Standard: Concepts, Behavior, and Services

SEMI E53 — Event Reporting

1.3.2 その他の参考文献

¹ Semiconductor Equipment and Materials International (SEMI), 805 East Middlefield Road, Mountain View, CA 94043, USA 650.964.5111, Fax 650.967.5375

(国内連絡先) SEMI ジャパン 〒102-0074 東京都千代田区九段南 4-7-15 03-3222-5755

James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, Object-Oriented Modeling and Design, Englewood Cliffs, New Jersey: Prentice-Hall, 1991

D. Harel, "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming 8, 1987

1.4 定義 – オブジェクト、サービスおよび形式についての基本的な定義は SEMI E39, 4.1 項から 4.4 項に書かれている。この項には追加された定義も書かれている。

1.4.3 項の定義は RMS 独自のものである。

1.4.1 サービス

サービスプロバイダ(*service provider*) – サービスユーザにサービスを提供する責任を負ったアプリケーション(エージェントのコンポーネント)。

サービスユーザ(*service user*) – サービスユーザとはサービスを使うアプリケーションである。

1.4.2 形式

バイナリ(*binary*) – ビット値の文字列(複数のゼロと 1)で、フォーマットは、指定されていないか、または最も重要なビットを最初においたビットポジションで指定されたもののどちらか。文字列の長さの合計は 8 の倍数である。メッセージングプロトコルが長さに制約を設けることがある。

1.4.3 レシピ管理 – この項では RMS で使用されている基本的な用語を紹介する。見出しはアルファベット順である。追加された定義および仕様は別の項で後述されている。

エージェント(*agent*) – 1 つ以上のサービスリソースを提供し、他のエージェントのサービスを使用する、工場のインテリジェントシステム。ホスト、装置、セル、クラスタ、クラスタモジュール、ステーションコントローラ、ワークステーション等の普遍的概念。エージェントは、コンピュータープラットフォームを含む、物理的システムまたは、物理的システムの集合と関連づけられる。

認可されたユーザ(*authorized user*) – ある装置用のレシピを認証するといった特定の、活動に要求されるレベルの権限を持っているユーザであることを、エージェントにユーザ認知されているもの。

収集イベント(*collection event*) – サービスユーザが関心を持っている、検出可能な出来事。

コンポーネントエージェント(*component agent*) – スーパーバイザーエージェントにサービスを提供する下位エージェント。

ダウンロード(*download*) – レシピを実行ストレージエリアに移す作業。

編集する(*edit*) – 新しいレシピ本体を作る、あるいは既存のレシピの本体を変更する作業。

エディタ(*editor*) – ユーザがレシピを編集できるようにするサービス。RMS ではエディタの仕様は決められていない。

装置(*equipment*)² – 最低限レシピ実行サービスを提供する関連ハードウェアを持つエージェント。

イベント(*event*) – イベントとは装置にとって重要な、検出可能な出来事である。

実行(レシピ実行)(*execution (recipe execution)*) – レシピの内容を読み、その指示、プロセス、パラメータ、あるいは自分の処理に必要なその他の情報を実装するプロセス。

実行エージェント(*executing agent*) – レシピ実行能力を提供するエージェント。

実行エリア(*execution area*) – その時点で実行を選択されている(実行の準備が整っている)レシピの保存場所。

ホスト(*host*) – 下位にあるものにとっては工場に相当するスーパーバイザーエージェント。

論理レシピ(*logical recipe*) – 特定の属性のセットおよび本体を持つレシピで、物理的な所在に関係なく考えられたもの。論理レシピが複数の実装実体あるいはコピーを持つこともある。

2 「装置」という用語は RMS では「知的装置」に限定されている。

名前(name) — あるオブジェクトについての、テキストベースの属性で、オブジェクトの識別子の一部あるいは全部として使用される。

ネームスペース(namespace) — 一般的には同じオブジェクト識別子がこの中にはない領域。RMS ではネームスペースという用語は別に定義されていない限りレシピネームスペースの同義語として扱われる。

オペレータ(operator) — エージェントのインタフェースを介してエージェントをローカルにやりとりするユーザ。

レシピ(recipe) — プロセッシングエージェントの制御下で、あらかじめ計画され再利用可能な命令、設定、パラメータ集合の 1 部。それは材料が遭遇するプロセッシング環境を決定する。レシピは実行またはプロセッシングサイクル間に変更されることがある。

レシピクラス(recipe class) — 共通の言語シンタックスと機能を持つレシピを正式にグループに分けたもの。

レシピエグゼキュータ(recipe executor) — レシピを格納し、実行するモジュール部分。

レシピネームスペース(recipe namespace) — レシピの格納、管理の責任を有する論理的な管理領域。それはレシピ識別子の一意性を保証し、当該領域内に格納されたレシピに関係するサービスを提供する。

レシピパラメータ(recipe parameter) — エージェントの処理に影響を与えるコントロール値。

選択(select) — レシピを実行する準備を行う行為。

設定(setting) — ユーザがひとつあるいは複数の方法を使ってアクセスすることができる静的数値で、装置はこの値を使って自分の処理をコントロールする。設定にはセットポイント値が含まれるがそれだけではない。通常はレシピ内で設定を指定することができる。

ストレージエリア(storage area) — オブジェクトおよびデータが保存されるエリア。

下位エージェント(subordinate agent) — 別のエージェントのコンポーネントである、あるいは別のエージェントに管理されているエージェント。

スーパーバイザーエージェント(supervisory agent) — ひとつあるいは複数の下位エージェントを監督する責任を持つエージェント。

タイムスタンプ(timestamp) — あるイベントが発生した日付および時刻を記すもの。

アップロード(upload) — 実行ストレージエリアからレシピを転送する操作。

ユーザ(user) — エージェントの人的インタフェースを介して直接、あるいはエージェントのスーパーバイザを介して間接的に、エージェントとやりとりする人。

確認(validate) — 実行に先だってレシピの内容を調べ、パラメータタイプと範囲がその装置の構成に対して有効であることを保証する行為。[validationとverificationの使い方が異なることに注意する。]

変数パラメータ(variable parameter) — レシピ本体内で定義されていて、正式に定義された変数(設定)。外部から実際の値を与えることができる。

検証する(verify) — レシピの内容を読み込んで、それがシンタックス的に正しく、公にされなければならない要素をはっきり示していることを保証する作業。

バージョン(version) — レシピが受け継いで来たものを示すために使われるレシピの識別子の一部。

1.5 表記法 — 本文書では下記の規則が使用されている:

- RMS 独自の用語(1.4.1 項から 1.4.3 項までで定義されている、複数のスタンダードに共通の用語を除く)を強調するため、定義される用語に関しては、初めて出てきた箇所およびその用語が定義されている箇所では太字で書く。独自の意味を持つこういった用語に対して読者の注意を向けさせるためである。
- オブジェクトおよびオブジェクトサービスに関連した用語は SEMI E39(オブジェクトサービススタンダード:コンセプト、挙動、およびサービス)に準拠する。オブジェクトについての短い文章が本文書の付属書に書かれている。

- 属性名には下線を引く。
- RMS で「ネーム」と呼ばれる属性は通常は、標準化されたオブジェクトの ObjID 属性に使われることになっている。
- 「XxxList」という名前の膨大な数のメッセージパラメータを定義しなくてすむように、本文書では、リストを「Xxx (のリスト)」と呼ぶことにする。この場合、リストではなく、パラメータ Xxx の定義が与えられる。「リスト」という用語は同一のデータタイプのゼロ個あるいはそれ以上の項目の集まり(あるいはセット)を指す。

リストである属性についてはこの規則は守られていない。というのは、属性全体にリストとして独自の名前を割り当てなければならないからである。

1.5.1 テキスト文字列の制約 — 属性名、属性値、あるいはメッセージパラメータに使用するテキスト文字列は OSS が定義した制約に従う。すなわち ASCII でのテキストは、クエスチョンマーク“?” (3F₁₆)、アスタリスク“*” (2A₁₆)、およびティルデ“~” (7E₁₆)を除いて、20₁₆から 7D₁₆、の間のキャラクタに限るということである。

オブジェクト識別子 ObjIDとして、あるいはその中で使われているテキスト文字列は OSS の要求に準拠するため、「より大きい」符号“>” (2E₁₆)およびコロンのキャラクタ“:” (3A₁₆)も除外するよう追記する。

1.5.2 ハレル状態モデル — 本文書では定義されているオブジェクトの動的な挙動を説明するのにハレルの状態図を使用している。この表記法についての概要は SEMI E30 の付属書に書かれている。この表記法の正式な定義は D. Harel 著 1987 年発行の Science of Computer Programming 8, "Statecharts: A Visual Formalism for Complex Systems"に書かれている。

本文書ではオブジェクトの削除を示すのに Rumbaugh およびその他の人たちが使っているようにハレルの表記法の拡張図を採用している(1.5.2.1 項参照)。

1.5.3 オブジェクト — RMS で定義されている標準化されたオブジェクトは SEMI E39(オブジェクトサービススタンダード:コンセプト, 挙動, およびサービス)の要求に準拠する。RMS では、標準化されたオブジェクトはオブジェクトモデルの中では太線で描く、という決まりを採用している。標準化されていないオブジェクトは、コンセプトおよび関係を示すために使われるが、正式に定義されてはおらず、オブジェクトサービスを介してアクセスすることはできない。RMS が定義した標準化されたオブジェクトのリストは関連情報 1 に含まれている。

1.5.3.1 OMT オブジェクト情報モデル — オブジェクトモデルは、Rumbaugh その他が Object-Oriented Modeling and Design, PrenticeHall, Englewood Cliffs, NJ, 1991 で発表したオブジェクトモデル作成技法を使って描かれている。この表記法の概要は SEMI E39(オブジェクトサービススタンダード:コンセプト, 挙動, およびサービス)の付属書に書かれている。

1.5.3.2 オブジェクト属性の表示 — 標準化されたオブジェクトのためのオブジェクト情報モデルは下記の見出しのついた属性定義表でサポートされている:

属性名	定義	アクセス	要求	形式
属性の正式なテキスト名	含まれている情報の説明	ROあるいはRW	YあるいはN	下記参照

アクセスの欄では、RO(Read Only)あるいはRW(Read and Write)を使って、サービスのユーザが属性に対してどのようなアクセスを行えるかを示している。

要求の欄の「Y」あるいは「N」は、サービスに基本的に準拠するためには、その属性をサポートしていなければならないかどうかを示している。

形式の欄は、属性のフォーマットを示すのに使われる(定義については 1.4 項参照)。

1.5.4 サービスメッセージの表現

サービスリソースの定義

サービスの定義表では下記の表のように、あるサービスリソースに関して決められているメッセージを定義している:

メッセージサービス名	タイプ	説明
メッセージ名	N または R	サービスの目的

タイプは N=通知, あるいは R=要求, のどちらかとなる。

通知タイプのメッセージは, サービスプロバイダが起動し, ユーザから応答が返ることを想定していない。

要求メッセージはサービスユーザが起動する。要求メッセージは, プロバイダにデータあるいは活動を依頼する。要求メッセージでは決まった応答メッセージを待つ(メッセージの内容については推定しない)。

サービスパラメータ辞書

サービスパラメータ辞書の表では下記の表のように, ひとつあるいは複数のサービスに関するパラメータを定義している:

パラメータ	形式	説明
パラメータ X	データタイプ	X という名前のパラメータは A において B である。

表ではサービスのパラメータひとつずつに 1 行があてられている。最初の欄にはパラメータの名前が書かれている。続いて, 形式および対応するプリミティブの内容が説明されている。

形式の欄はパラメータに含まれているデータのタイプを示すのに使われる(定義については 1.4 項参照)。

サービスパラメータ辞書の表の説明の欄にはパラメータの意味, そのパラメータがとることのできる値, それ以外のパラメータとのあらゆる相互関係が書かれている。

“XxxList”という名前の膨大な数のパラメータを定義しなくてすむよう, ここではそういったリストを“Xxx(のリスト)”と呼ぶ方法を採用している。この場合, リストの定義ではなく変数 Xxx の定義が与えられる。「リスト」という用語は同じデータタイプのゼロ個以上の項目の集合(あるいはセット)を指す。要求と応答の両方にリストが使われる場合には, 要求でのリストの順序が応答でもそのまま使われる。要素 0 個ということが特に許可されていない限り, リストには少なくとも要素が 1 個は含まれていなければならない。

サービスメッセージの定義

サービスメッセージの定義の表では下記の表のように, サービスに使われるパラメータが定義されている:

パラメータ	要求/指示	応答/確認	説明
パラメータ X	(下記参照)	(下記参照)	サービスの説明

要求/指示, および応答/確認と書いてある欄はパラメータとメッセージの方向を結び付ける。起動した側が送信したメッセージは「要求」と呼ばれ, レシーバがそれを受けるとそのメッセージは「指示」あるいは要求と呼ばれる。それからレシーバは「応答」を送信し, それをオリジナルセNDERは「確認」と呼ぶ。

下記のコードは要求/指示の欄, および応答/確認の欄に出てくるものでパラメータの定義(例えば, それぞれの方向で各々のパラメータはどのように使われるか)に使用される:

“M” – 必須パラメータ(Mandatory parameter) – 有効な値が与えられなければならない。

“C” – 条件付きパラメータ(Conditional parameter) – ある状況では定義され, それ以外の状況では定義されない。値が与えられるかどうかは完全に任意であるか, またはそれ以外のパラメータの値次第である。

“U” – ユーザ定義パラメータ(User-defined parameter)

“_” – このパラメータは使用されていない。

“=” – (応答のみに使用) 応答でのこのパラメータの値は, (定義されていれば)プライマリ値と一致しなければならない。

1.6 要求事項 レシピ管理の要求は多様である。例えば基本的に、レシピごとにその情報をとっておく、要求があればその情報を提供する、ずっと使用できるようにレシピといっしょにその情報を移す、ということが必要である。レシピを分類する、それを再使用する、それを複数の装置と共有する、ということも必要である。レシピの共有という必要性は、レシピの識別子とその内容との関係を守って、レシピが勝手に変更されないように、そして、複数の装置で使用されているレシピの識別子が同じレシピを指すことを保証する、という要求を新たに生み出す。

工場でのレシピ管理の要求は表 1.1 に書かれている。表 1.1 ではレシピの機能上の分野と特に関連する問題に沿って要求が書かれている。関連情報の表 R1.1 にはこの要求ひとつひとつを説明するために RMS で定義されている独自のコンセプトが書かれている。

表 1.1 要求事項

機能分野	問題点	要求
レシピの管理	識別	システム内でひとつしかないものとしてレシピを記憶、識別、選択する。
		レシピの識別が容易で明確である。
	複数のアクセス	装置同士でレシピを共有する。
		装置間で共有しているレシピの変更が同時にできるようにする。
	履歴および追跡可能なこと	レシピの使用法の履歴を捉える。 レシピ変更の履歴を捉える。
	ライフサイクル管理	プロセス開発および製造のためのレシピの承認を管理する。
操作	保護	予定外の変更からレシピを守る
		オペレーション中の誤りからレシピを守る
	簡便性	コマンドを実行する装置外でレシピを作成、編集、変更する。
	再利用性	特定の装置のためのレシピを変更する。
		ある装置で開発されたレシピを別の装置で使用する。 個々の相違点を調整して同じ種類の装置間でレシピを共有する。
実行	安全性	シンタックス的に正しくないレシピを実行してはいけない。
		特定の(選ばれた)レシピをエラーを起こさずに実行する。
		別の活動によって偶然レシピに生じた変更が実行されないように保護する。
	柔軟性	ランの最中に、あるいはランとランの間に系統的方法でレシピのパラメータを変更する。
システム操作	システムノード操作	装置と通信ネットワークとの接続を動的に入れたり切ったりする。
	独自の操作	通信リンクなしにレシピを実行する。
		通信リンクなしにレシピを作成、変更する。
		装置間で移されたレシピ、あるいは可動式のメディアを使ってオフラインストレージから移されたレシピを管理する。
		独自の装置をスムーズにオンラインのファクトリシステムに統合できるようにする。

1.7 文書の構成 – 図 1.1 は、目的、コンセプト、挙動、およびメッセージサービスで構成されているレシピスタンダードの領域を描いている。

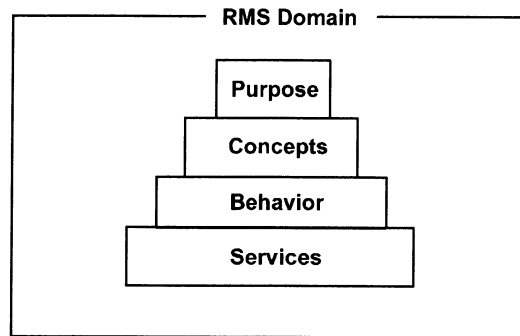


図 1.1
レシピ管理領域

目的は、レシピ管理能力に対する動機を提供しており、1.6 項で扱われている。コンセプトは RMS の標準化されたオブジェクト、その属性、および別のオブジェクトとの関係を詳細に述べている。挙動ではそういったオブジェクトによって、あるいはオブジェクトで、行われる操作を説明している。最後に、メッセージサービスはメッセージおよびそのパラメータを、それが実装されているプロトコルとは別個に定義している。コンセプトと挙動の両方は RMS アプリケーションの「中の視点」を表わしているのに対して、サービスは外部からのインタフェースを提供している。

- 第 1 章: RMS の形式の紹介。
- 第 2 章: RMS の主なオブジェクトおよびその関係の概要。この後に書かれている技術的な詳細のための一般的なコンテキストを提供する。
- 第 3-6 章: レシピ、レシピネームスペース、分散レシピネームスペース、およびレシピエグゼキュータについてのコンセプト。
- 第 7 章: エージェントおよびサービスリソースのコンセプト。RMS 実装のための、いっそう完璧なコンテキストを提供する。
- 第 8-11 章: レシピ管理、ネームスペース管理、分散レシピネームスペース管理、およびレシピエグゼキュータの挙動（オペレーション）。
- 第 12-14 章: レシピネームスペース、分散レシピネームスペース、およびレシピエグゼキュータサービスリソースのためのメッセージサービスの定義。
- 第 15 章: RMS 準拠。
- 第 16 章: 用語集。参照に便利なように。
- 関連情報: RMS およびアプリケーションの例の要求に関する背景的な情報を提供している。

1.8 適用文書 – ISO/TR 8509:1987, 情報処理システム、オープンシステムの相互接続 – サービスの規定。

2 RMS の概要

本章には RMS の主なオブジェクトの紹介およびその概要が書かれている。

2.1 レシピ管理モデル – RMS で仕様が決められている機能の主なエンティティあるいは分野を、より明確に理解してもらうため、RMS モデルのオブジェクトは OMT 記述法を使って説明されている。

オブジェクトベースのモデルに描かれている関係を表わす線はオブジェクト間に直接通信リンクが結ばれているという意味ではなく、あるオブジェクトが別のオブジェクトに関して持っている知識および関連を示している。

モデルは二つのオブジェクトの相対的な位置関係を示してはいない。

2.2 RMSの主なオブジェクト – レシピ管理は、図 2.1 に描かれている機能の主な分野: レシピ³、持続的なレシピ保存を行うレシピネームスペース、ネームスペースの管理とそのレシピへのアクセスを行うレシピネームスペースマネージャ、および、レシピを実行するレシピエグゼキュータ、に関係している。

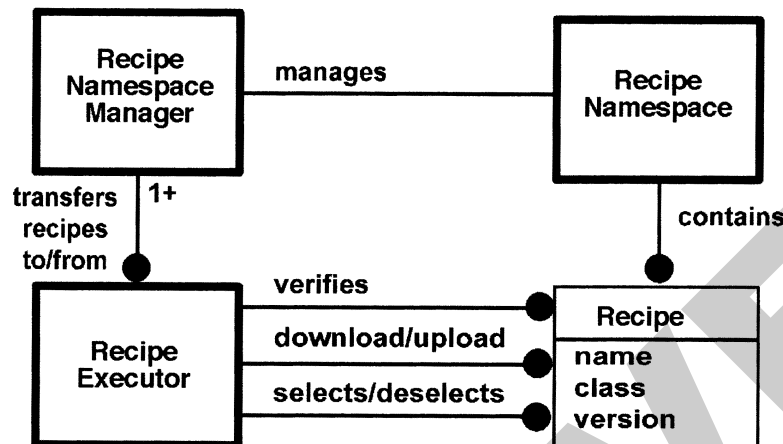


図 2.1
RMS の主なオブジェクト

2.2.1 レシピ – レシピは、ユーザがある結果をもたらす手順および設定を選択することができるように、操作可能、柔軟性、再利用性がある形式を提供する。装置はさまざまな方法でレシピを使って処理環境をコントロールし、時には数種類のレシピを一緒に使うこともある。レシピを使ってキャリブレーションやクリーニングといったメンテナンス活動を行うこともできる。

レシピは修正可能であり、ある環境から別の環境へコピーすることができる。その形態の柔軟性を管理できない限り、柔軟性が尽きない問題の種ともなる。誤処理、つまりある製品にうっかりして間違っただけのレシピを処理させることは工場にとっては法外に高くつく。

RMS はレシピを、内容だけでなく属性 – レシピについての情報 – をもったオブジェクトとして定義する。レシピのライフサイクルの間中ずっとレシピを本当に管理できるのは、レシピの持つ属性のおかげである。

2.2.2 レシピネームスペース – RMS におけるレシピネームスペースは賢いファイルディレクトリに似ている。ネームスペースは長期にわたるレシピ保存能力を提供する。

基本的な操作はネームスペース仕様の一部である。例えば、レシピはあるネームスペースから別のネームスペースへのコピー、ネームスペース内でのコピー、リネーム、作成、アップデート、および削除が可能である。レシピはレシピエグゼキュータという名前のアプリケーションへのダウンロード、そこからのアップロードが可能である。いくつかのレシピをリンクして 1 組にすることができる。

ネームスペースは同じタイプの装置グループが共有可能なレシピプールとしての役目も果たす。分散レシピネームスペースとは、そういった装置グループが提供するレシピ記憶領域を利用、管理することができるネームスペースである。

2.2.3 レシピネームスペースマネージャ – レシピネームスペース自体は受動的で、レシピを入れておく容器にすぎない。レシピネームスペースマネージャ(マネージャ)はネームスペースを管理する動的な要素を提供する。マネージャは外部の世界に対してはネームスペースのインタフェースとなり、ネームスペース内では内部での決定権を持つものとなる。

2.2.4 レシピエグゼキュータ – レシピエグゼキュータ(エグゼキュータ)は、レシピの内容を理解し、それがシタックス的に正しいかどうか検証でき、現在の設定環境でそれを実行してよいかどうかを確認することができるエー

3 図 2.1 のレシピオブジェクトそれ自体は標準化されたオブジェクトではない。しかし第 3 項で定義されているレシピサブタイプ 2 つは標準化されたオブジェクトである。

ジェントのコンポーネントである。これはレシピの内容を読み取り、期待した結果を得るために適切に適用して、そのレシピを**実行**する。

レシピエグゼキュータは追加的な限られた保存能力も提供することがあるが、思いがけない変更、あるいは望ましくない変更を避けるために、特別な条件のもと以外は、既存のレシピの修正をすることはできない。

2.3 エージェント – RMS で使われているように、エージェントというのは工場内にあり、物理的な面を持っているシステムあるいはサブシステムである。これは図 2.2 に描かれているように、レシピネームスペースマネージャやレシピエグゼキュータ等のサービスリソースを提供し、また／あるいは使用するひとつもしくは複数のアプリケーションから成っている。

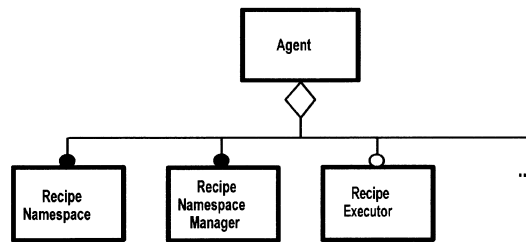


図 2.2
エージェントと複数のコンポーネントアプリケーションの例

エージェントというのは、従来の装置、ステーション、セル、クラスタといった監視システムあるいは統合システム、クラスタの中にあるプロセスモジュールのような、装置内にある知的なサブシステムをひとまとめにして呼ぶ名前である。レシピネームスペース管理専用のもの、といった別のタイプのエージェントも実装可能である。

レシピエグゼキュータを提供するエージェントは**実行エージェント**と呼ばれる。**装置**はレシピが適用され、仕事を行うのに使用する関連ハードウェアを持つ実行エージェントのひとつである。

2.4 実装 – RMS は工場内の異なったプラットフォームに実装することができる。ホストコントローラはレシピネームスペース管理を行うだけなのに対して、ディスク不要のクラスタモジュールはレシピ実行だけを行う。単体モードでの操作が可能である従来の装置はレシピエグゼキュータだけでなく、立ち上げの時に使用できるレシピネームスペースも提供する必要がある。RMS 実装の例は関連情報 1 に載っている。

RMS は外部インタフェースを介して提供されるサービスを定義している。エージェントの内部となるアプリケーション間、あるいはアプリケーション内の通信は、属性および操作が RMS の要求に準拠している限りは、RMS ではカバーしない。

3 レシピ

本章ではレシピに関するコンセプトを述べている。

3.1 動機 – レシピの構造、属性、および操作アドレスの定義では下記の問題を扱っている：

- 管理および追跡ができるように行うレシピの属性のメンテナンス
- レシピ本体と一緒に転送するのと同様、属性だけを別個に転送するための標準的な方法に関する規定
- 専用のレシピエディターの必要性を減らす、あるいはなくす。
- フォーマットあるいは言語シンタックスといった複数のレシピタイプを必要とするアプリケーションのサポート
- R2R コントロール
- 工場の承認および認可レベルでのソーティングのサポート
- 包括的なレシピをずっと同じエージェントに使用するだけでなく、特別な使い方をしたエージェント用に調整する能力
- 同じレシピシンタックスあるいは言語をサポートしている実行エージェント間でレシピを共有する能力

3.2 基本的なコンセプト – RMS では二つのタイプのレシピを扱っている。レシピネームスペースに保存されている**管理レシピ**と、**レシピ実行ストレージ**に保存されている**実行レシピ**である。この項では両方のタイプのレシピに関する基本的なコンセプトを述べる。

3.2.1 レシピのタイプ – 図 3.1 のように、レシピオブジェクトには 3 つのタイプがある。

レシピ管理の大部分はレシピネームスペースに保存されているレシピの管理に関係している。こういうレシピは**管理レシピ**と呼ばれている。レシピは**レシピ実行ストレージ**内のレシピエグゼキュータにも保存されていて、そういったレシピは**実行レシピ**と呼ばれている。

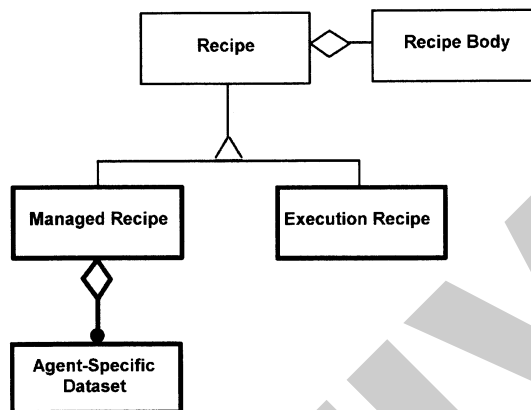


図 3.1
レシピタイプ

レシプスーパータイプは、共通の要素および、管理レシピと実行レシピの属性、およびその相違点を示している抽象タイプである。レシプスーパータイプ自体は実装するものではない。つまり、RMS で形式が定義されているレシピオブジェクトタイプは管理レシピか、実行レシピのどちらかである、ということである。細い線で描かれているのでわかるように、スーパータイプは標準化されたオブジェクトではない。

3.2.2 レシピの構造 – 管理レシピおよび実行レシピの両方とも**本体**⁴および属性を一組持っている。レシピの本体(内容)にはレシピエグゼキュータが実行プロセスで使用するデータが入っている。本体も標準化されたオブジェクトではなく、オブジェクトサービスを介してアクセスすることはできない。

図 3.2 には、管理レシピと実行レシピに共通の RMS で定義された属性が、すべて描かれている。RMS を最小限に実装する場合には、この属性がすべて必要なわけではない。しかし、ある属性の値を設定する際の規則は、管理レシピと実行レシピとは異なっている。

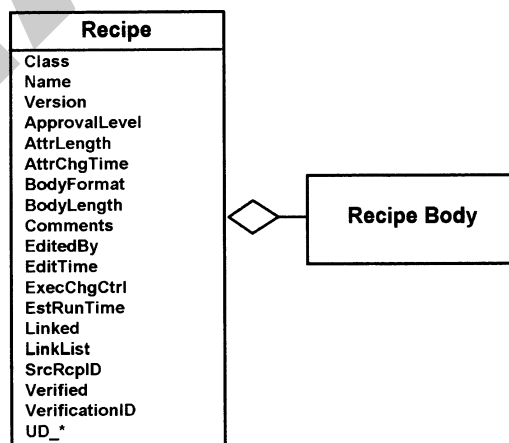


図 3.2
レシプスーパータイプ

4 レシピの本体は SEMI E5 および SEMI E30 の「プロセスプログラム」に対応している。

識別属性というのは識別の目的のために使用される属性で、これについては 3.3 項で述べる。それ以外の、個々のレシピタイプ独自の属性は後述する。

管理レシピも、単一の実行エージェント固有の属性を保持することがある。**エージェント固有**の属性は技術的にはレシピと実行エージェントとの間の関連についての属性である。しかし RMS ではこの属性は、レシピの別の属性と一緒にネームスペース内で管理できるよう、**エージェント固有のデータセット**と呼ばれるレシピのコンポーネントの属性として扱われる。エージェント固有の属性のおかげでレシピが特定の実行エージェント(つまり特定のレシピエグゼキュータ)のものとなる。

管理レシピは複数のエージェントに適用できるので、その管理レシピは複数のエージェント固有のデータセットを持つことができる。レシピがレシピネームスペースからレシピエグゼキュータへダウンロードされると、エージェント固有のデータセットからそのエージェント(そのレシピエグゼキュータ)のエージェント固有の属性を取り出し、それらを実行レシピの属性とマージすることにより、管理レシピを実行レシピへと変換する。

レシピあるいはレシピのコンポーネントを保存する方法は RMS には書かれていない。しかし、レシピの本体とその属性との関連は気を付けて保持すること。

3.2.2.1 レシピ本体 — レシピ**本体**には再使用可能な指示、設定、パラメータ、およびその他レシピエグゼキュータがレシピを実行しそのオペレーションをコントロールするために読み込むデータが含まれている。

レシピ本体は二つの基本形:**ソース**(テキスト)あるいは**オブジェクト**のどちらかの形式を取っている。ソース形式の目的は人間が読めて編集できるテキストを使用するよう奨めることである。オブジェクト形式は、レシピエグゼキュータが作成して使用する専有フォーマットだけでなく、視覚システムあるいは CAD プログラムで作られたシステムなどの、マシンおよびその他の自動化されたシステムだけが読めて使えるタイプをサポートしている。

3.2.2.1.1 ソース形式 — エディタ形式で作成され修正されたレシピは、ソース形式で使えなければならない。レシピ本体の**ソース形式**はASCIIあるいはJIS-8 のようなテキストファイルと同じもので、どのような「標準的な」テキストエディタ⁵でも作成、読み込み、印刷、修正ができる。実行エージェントはそういったテキストエディタを提供するよう要求されていないが、そのレシピエグゼキュータは、ソース形式のレシピの読み書きが両方ともできなければならない。編集は、RMSではカバーしていない。

レシピのソース形式はネームスペースから別のネームスペースへ、ネームスペースからレシピエグゼキュータのレシピ実行ストレージへとコピーすることが可能であり、あるネームスペースへアクセスできるレシピエグゼキュータは、すべて同じレシピ言語と同じ機能を使うものと見なされている。

エージェント間で移されたテキストの解釈に関して制約が二つ設けられている:

- ASCII では、有効なキャラクタはスペース(20₁₆)から“~”(7D₁₆) プラス、タブ(09₁₆)、改行(0A₁₆)、キャリッジリターン(0D₁₆)、およびフォームフィード(0C₁₆)キャラクタである。スペースおよびタブは**横方向のホワイトスペース**キャラクタとみなされ、キャリッジリターンおよびフォームフィードキャラクタは**縦方向のホワイトスペース**とみなされる。改行キャラクタはテキストのラインの終点(eol)を決めるのに使用される。
- レシピの意味は、内蔵されたコントロールキャラクタ、独自のホワイトスペースキャラクタ、あるいは行の最後にあるホワイトスペースによって異なってはならない。解釈のため、横方向のホワイトスペースキャラクタは縦方向のホワイトスペースキャラクタと同じ様に扱わなければならない。それ以外のキャラクタはすべて、ここで指定されている以外は、使わない方がよく、見つけた場合には転送の前にテキストから取り除かなければならない。

一般的に言って、レシピ本体で使用されているレシピ言語およびシンタックスはレシピ管理の範囲外である。しかし、ユーザごとに勝手な違いがでないように、下記の「規則」が推奨される:

- 比較あるいは意味という目的から言うと大文字か小文字かということは重要ではないが、読みやすくするために使われていることが多いので、出てきたらそのままにしておく。例えば、トークンの `alarmlevel`、`ALARMLEVEL`、`AlarmLevel` はどれも同じ意味でなければならない。

⁵ この意図は、テキストエディタの仕様を決めることではなく、ユーザが、適用できる明確なガイドラインと制約を基にしてテキストを修正できるようにすることである。ソース形式のレシピは、言語ごとの専用のエディタが増えないようにし、その必要性を減らすために、非常に望ましいものである。

- コメントをテキストに入れてレシピの解釈の際には無視することができる。コメントは二つのタイプが定義されている: “/” というキャラクタで始まっているコメントはその行の最後まで続き, “/*” というキャラクタで始まっているコメントは “*/” でしか終わらない。2 番目のタイプのコメントは, 最初に出てきた “/*” で終了する。コメントをネストにはいけない場合がある。ネストになったコメントは大事にされず, シンタックスエラーを起こすことがある。
- 1 行の最短のものは改行キャラクタ 1 個で構成されている。

さらに, 別のレシピへの**外部参照**をサポートしているレシピ言語は, レシピ識別子 (3.2.3.4 項) および**レシピ指定子** (3.2.4.1.2 項) のために定義されたテキスト形式を使わなければならない。

レシピエグゼキュータのサプライヤは, そのエージェントがサポートしているレシピ言語ひとつひとつを正式に定義している文書を提供する。文書は二つの形式のものが必要とされる。ひとつはユーザのために, ユーザは正しいレシピを書くことができなければならない。もうひとつは, **Backus-Naur Form (BNF)** などの形式記述プロトコルを使い, 言語のシンタックスおよび語彙の構造の両方をカバーしているものである。2 番目の形式の文書があれば, レシピをダウンロードする前にそのレシピのシンタックスおよび語彙が正しいかどうかを前もってチェックする能力を持つ解析ソフトを組み立てることができる。事前チェックは, レシピエグゼキュータがレシピのチェックをする必要性を減らすことになるので効率と作業性能を向上させる。

レシピの正しさを決定する最終的な責任はレシピエグゼキュータにある。

3.2.2.1.2 オブジェクト形式 — 意味のあるソース形式に翻訳できないレシピであり, 特別なハードウェアごとの「ティーチ」操作を使って視覚システムあるいは機械的なシステムから引き出されたデータセットを含む。こういったレシピは専用のオブジェクト形式のままでもよく, ハードウェアに依存しているため, 別のレシピエグゼキュータに適用できる場合とできない場合とがある。

より効率を上げるために, レシピエグゼキュータはソースレシピを書き直して(例えば**トークン化**して), レシピの専用のオブジェクト形式を作ってもよい。これは**派生オブジェクト形式**と呼ばれ, ソース形式では決して存在しないオブジェクト形式のレシピと区別している。エージェントのメーカーはレシピの移動のためにもこの形式を使えるようにすることを選択してもよい。スーパーバイザの観点から言うと, オブジェクト形式をとっているレシピの本体は構造化されていないバイナリのベクトルあるいは文字列である。一般的に, これはスーパーバイザでは編集できず, レシピのソース形式がやはり必要である。複数のレシピエグゼキュータで使えない派生オブジェクト形式のレシピは共有されているレシピネームスペースに保存してはいけない。

ソース形式と派生オブジェクト形式の両方が存在している場合には, レシピのオブジェクト形式のための識別子は, 何かははっきり認識できる形でソース形式の識別子と区別する。レシピエグゼキュータのサプライヤは, 識別子で両者を見分ける手段を書いた文書を提供する。この手段は, オリジナルのソースレシピからの派生オブジェクト形式のレシピの関係がユーザにはっきり判るようなものでなければならない(例については 3.2.3.4 項参照)。ソース形式のレシピと派生オブジェクト形式のレシピとの関係を保持するために, レシピの属性である **SrcRcpID** が与えられる。

3.2.3 レシピの識別 — レシピを適切にそして明白に識別する能力はレシピ管理およびレシピ処理の両方にとって非常に重要である。識別が論理的であるということは, ユーザにとっても重要である。**RMS** では, あらゆるレシピネームスペースおよびレシピエグゼキュータのストレージ内にあるレシピをユニークに識別するのに使う要素を 3 つ定義している。それはレシピの**クラス**, **名前**, および**バージョン**である。各要素はテキスト文字列として表わされ, 連結されて**レシピ識別子** (3.2.3.4 項) となる。

3.2.3.1 レシピ名 — **レシピ名**はユーザが定義するテキスト文字列で, テクノロジ, 階層, 製造エリア, その他の特徴をコード化するのに使うことができる。レシピ名だけはクラス内あるいはネームスペース内で必ずしもユニークである必要はない。

レシピ名は 1.5.1 項で定義されているテキストのための規定に従う。

3.2.3.2 レシピクラス — **レシピクラス**とは共通のシンタックスを持つレシピを, 正式にグループに分けたものである。クラスには**サブクラス**もあり, サブクラス内のレシピは, 親クラスあるいはサブクラス同士, またはその両方とは異なる環境で動作し, あるいは異なったシンタックスを持つ。サブクラス自体がクラスであり, さらにそのサブクラスを持つこともある。

レシピの全体モデルはクラス階層、つまり、レシピエグゼキュータのサプライヤが定義した「クラスツリー」のひとまとまりである。レシピの主なクラスには **PROCESS** クラス、**SERVICE** クラス、そして、たぶんこれ以外にもエージェンツ固有のクラスがある。これらは**プライマリクラス**と呼ばれる。**プライマリクラス**とは別のクラスのサブクラスではないクラスである。

PROCESSクラスは必須プライマリクラスであり、製造材料⁶の製造価値を高めるのが主な目的であるレシピで構成されている。

SERVICE クラスは、装置のオペレーションを維持、準備、計測、あるいはテストするのが目的であるレシピに使われる、オプションのプライマリクラスである。**SERVICE** クラスのよくあるサブクラスには **CALIBRATION** および **CLEANING** がある。通常のプロセスレシピと同じレシピ言語を使う **SERVICE** レシピは、別個のプライマリクラスに入れる必要はない。

SERVICE クラスがあれば、処理以外の特別な目的のための、異なったシンタックスを分けて使用することができる。

PROCESS クラスにも **SERVICE** クラスにもあてはまらないその他のプライマリクラスはレシピエグゼキュータのサプライヤが提供することもある。サプライヤはこういった追加クラスとその機能についての文書を作成しなければならない。

PROCESS クラスにもそのサブクラスにも入らないレシピは製造に使用してはいけない。しかし、**PROCESS** クラスに入っているレシピを製造以外の目的に使用するのはかまわない。

図 3.3 および図 3.4 に、ワイヤボンダーと自動ボートローダーサブシステムを持つ拡散炉のためのクラスとサブクラスのオブジェクトモデルの例を示す。

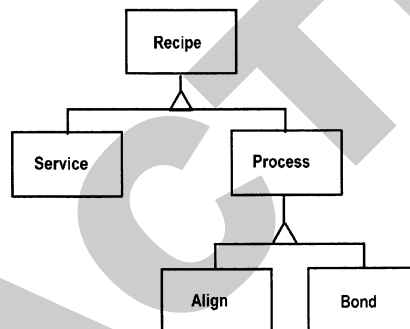


図 3.3
ワイヤボンダーのレシピクラス

レシピクラスのテキスト形式はテキストのための規定(1.5.1 項)に準拠する。レシピ識別子のテキスト形式の中ではクラスは前後を前傾のスラッシュキャラクタ“/”で区切られている。レシピ識別子内の完全なクラス指定子は、プライマリクラスで始まり問題のレシピのクラスで終わっている連結されたクラスとサブクラスで構成されている：

"/PRIMARY CLASS/SUBCLASS₁/SUBCLASS₂/..."

例えば、図 3.4 中の **NORMAL CYCLE** クラスの中にある **DryOx** という名前のレシピのバージョン 4 は下記の識別子を持つ；

"/PROCESS/FURNACE/DIFFUSION/NORMALCYCLE/DryOx; 4".

⁶ 材料ハンドリングシステム、ウェーハ度量衡、ウェーハ計測、ウェーハ検査システム、さらに材料の特性を直接変える処理装置は何らかの方法で材料の価値を高める。

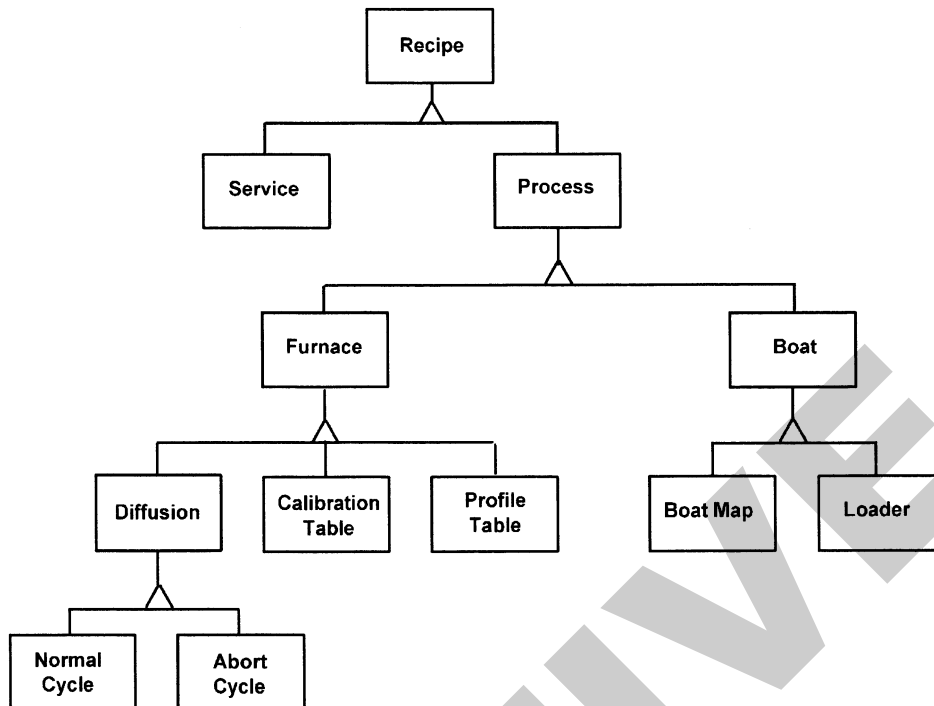


図 3.4
拡散炉のレシピクラス

クラス指定の完全な形は常にプライマリクラスで始まり、常に受け付けられる。しかし、ネームスペース内のクラス名がすべてユニークなものである場合には、サービスプロバイダは単一のクラスで構成されているクラス指定子も受け付けることがある。その場合には、上記の例の識別子はこうなる；

"/NORMAL CYCLE/DryOx; 4".

3.2.3.3 バージョン — レシピは進化し続け、いくつかのバージョンを持つことがある。このためユーザは複数のバージョンでレシピ名を保持してレシピの継承を見せることができる。さまざまなバージョンは、レシピ識別子のバージョンの個所で識別される。注:あるレシピの複数のバージョンが工場内だけではなく 1 台の装置の中でも同時に製造に使用される場合がある。

バージョンは最低 1 個のキャラクタで構成されているテキスト文字列である。新バージョンはレシピネームスペースで自動的に作られるか、またはユーザによって決められる。ユーザは下記以外のテキストキャラクタおよび句読点のあらゆる組み合わせをバージョンにあてることができる；

- テキストの使用法およびオブジェクト識別子に関する規定 (1.5.1 項) で禁じられているキャラクタ
- ホワイトスペースキャラクタ

絶対にではないが、大文字だけを使用する方がよい。

数値バージョン(バージョンナンバー)は“0”から“9”までの数字および小数点 1 個“.”だけで構成されており、純数字に翻訳できる。同じ数値を持つバージョンがいくつもあるという事態と混乱を避けるため、数字のバージョンにはさらに下記のような制約がある：

- 整数(小数点なし)は、ゼロで始まってそのあとに別の数字が続いてはいけない
- 小数(小数点あり)で、小数点で始まる、あるいは小数点で終わる、あるいは小数点のあとがゼロで終わるものはいけない

例えば、“09”、“1.”、“.5”、“1.670”というバージョンナンバーは禁じられている。同じ数値の正しい形は“9”、“1”、“0.5”、“1.67”である。ユーザは“0”というバージョンをあてることができるが“00”はできない。

自動的に割り当てられたバージョンはさらに次のような制約を受ける:

- 小数点を除いた数字のバージョンで最小値は“1”である。
- バージョンは 1 ずつ増やして割り当てられる。あるクラスと名前のレシピにバージョンを割り当てる時、そのクラスと名前を持つレシピがそのネームスペースの中になければ、バージョン“1”が割り当てられる。

でなければ、そのクラスと名前ですでに使われている既存のバージョンの中で最大のものが決められ、新バージョンはその値プラス 1 に等しい値が割り当てられる。例えば、使われている最大のバージョンは 5 という数値を持っているとすると、次のバージョンには“6”が割り当てられることになる。二つのバージョンを比較するために、バージョンは大文字に変換されキャラクタ 1 字ずつ比較される。

3.2.3.4 レシピ識別子 — レシピ識別子はレシピのクラス、名前、バージョンがこの順序でつながったもので作られている:

`"/CLASS1/CLASS2/.../CLASSn/NAME;VERSION"`

ここで CLASS₁ はプライマリクラスであり、CLASS_{i+1} は CLASS_i のサブクラスである。レシピ名がクラスのあとに来る。名前とバージョンは常にセミコロン“;” (3B₁₆) で分けられている。

CLASS_n がユニークなクラス名である場合、形式は次のようになる;

`/CLASS/NAME; VERSION"`

レシピ識別子は OSS で要求されている属性 ObjID として使用され、ObjID に課せられているのと同じ制約に従う (SEMI E39 参照)。さらに ObjID の長さの合計は、プロトコルが設けている制約も受ける。

注: レシピの識別子あるいは名前と、本体および属性が保存されているファイル名との間には必要な関係はない。レシピの識別子とは、レシピを論理的に指すもので、特定のプラットフォームおよび実装とは無関係なものである。特に、オペレーティングシステムが提供するファイルサービスには長さなど、ネーミング上の制約がある場合もあり、その制約は RMS の要求とは相容れない。レシピはシングルフラットファイル、組になった関連ファイル、あるいはデータベースなど、さまざまな方法で内部保存が可能である。実際の保存手段は RMS には見えない。

ソース形式のレシピをサポートしているサブライヤが 3.2.2.1.2 項に書かれているようにそのレシピを書き直す場合には、二つの形式を見分け、同時にオリジナルのソース形式のレシピと派生したオブジェクト形式のレシピとの関係を認識するために、レシピ識別子 (ObjID) に基づいた方法が必要である。その方法は、この目的のために識別子の 3 つの要素のどれを使ってもよい。例えば、オブジェクト形式のレシピは“/OBJ/”という別個のサブクラスに置いてもよいし、ユーザが定義したオリジナルレシピ名に“.obj”という接尾辞を付けてもよい。

3.2.3.4.1 デフォルトのレシピ識別子 — 完全な識別子(クラス、名前、およびバージョンナンバー)を指定してレシピを指すのは常に可能である。3.2.4.1.1 項に書かれているような場合には、レシピのネームスペースを指定する必要もある。しかし、識別子のコンポーネントを常に全部指定する必要があるわけではない。具体的に言うと、レシピが実行のために外部で選択された場合にはクラスやバージョンを省略してもよい。ネームスペースは通常は指定しない。規則ではネームスペースおよびクラスのデフォルト値が定義されており、適切なバージョンを決めるのにも決まりがある。

レシピ名は常に指定しなければならない。

レシピ内の外部参照に関しては、デフォルトのクラスとは参照を行うレシピのクラスである。レシピがリンクされた時に適切なクラスが決められる(3.2.4.1.3 項)。

レシピバージョンが指定されていない場合には、そのクラスと名前で最高の承認レベルを持っているレシピが使われる。同じ承認レベルのレシピがいくつもある場合には、一番高いバージョンのものが使用される。外部参照でバージョンが指定されていない場合には、リンクオペレーションによってリンク済みレシピセットが作られた時に正しいバージョンが決定される。

管理レシピの参照については、ネームスペースが指定されていない場合は、デフォルトは参照を行うレシピのネームスペースとなる。

3.2.4 レシピの先進機能

3.2.4.1 マルチパートレシピ – マルチパートレシピの使用は、レシピエグゼキュータが指定することがよくあり、その処理には異なったクラスのレシピを 1 組必要とする場合がある。さらに、マルチパートレシピのサポートは、ユーザにとっては有用であり、特にこれを使わなければレシピが長くなるという場合に有用である。通常のプログラミング言語でのサブルーチンの使用法に似たサブレシピの使用法を、レシピ言語で定義することができる。このため、異なったレシピがかなり似ている場合には、再利用がよくできるようになる。

ひとつ以上のクラスのレシピが必要とされる場合には、ひとつのレシピが別のレシピを参照できるようにすると便利である。例えば、PROCESS クラスのレシピが別のクラスのレシピを参照してそのレシピの組全体と一緒に実行されるようにすることもできる。

実質的な指示や設定がさまざまなレシピに共通である場合には、ひとつのレシピが同じクラスの別のレシピを参照できるようにするのも便利である。

あるレシピの本体の中で別のレシピを参照することは**外部参照**と呼ばれる。これを行う正確なシンタックスは、使用されているレシピ言語によって決まる。外部参照では参照されるレシピあるいはそのサブレシピのひとつで定義されている変数パラメータの値を指定することもできる。

3.2.4.1.1 サブレシピ – あるレシピが別のレシピを参照する時には、両方のレシピを実行する必要がある。起動されるレシピを**メインレシピ**という。

サブレシピとはメインレシピ、あるいはメインレシピの別のサブレシピが参照するレシピのことである。別のレシピを参照するレシピは、参照されるサブレシピの親レシピである。**親レシピ**が必ずメインレシピである必要はない。メインレシピをそのサブレシピが参照してはならない。

同じネームスペース内にあるレシピへの外部参照は明白なものでもよいし、そうでなくてもよい。つまり、現在のクラス(すなわち検証されるレシピのクラス)への参照ははっきり言葉に出さなくてもよく、バージョンはリンク時にデフォルトのバージョンを決定するための規則に任せることができる。

レシピは別のネームスペースにあるレシピを参照することもできる。これはあるレシピが**代表**になる場合、または複数のエージェントがアクセスするネームスペースにあるレシピが、**デフォルトのネームスペース**にある、ハードウェアごとのレシピあるいはそれ以外のエージェント固有のレシピを参照しなければならない場合のどちらかで発生する。レシピエグゼキュータのデフォルトのネームスペースはレシピ内では常に“Default”という名前のネームスペースとして参照される。この名前は実行時にレシピエグゼキュータが [DefaultNamespace](#) (6.3 項参照) という属性に基づいて判断する。

サブレシピの識別子を持つレシピ内で識別子がテキスト文字列で指定される場合には、識別子の前にはそのネームスペースの識別子の値がきて、後には不等号(より大)符号“>”がくる。これは**レシピの指定子**と呼ばれる。“NS-MOM”という名前のネームスペースの中にある ETCH というレシピのバージョン 5 だと、次のような指定となる；

"NS-MOM>/PROCESS/ETCH; 5"

OSS の規則に従って、それ以外の方法で決められる時にはオブジェクトタイプを省略する場合もある。RMS では決定は順序に従って行われる。

3.2.4.1.2 代表レシピ – レシピは、コンポーネントエージェントが実行することになっていて**代表レシピ**という名前がついている、別のネームスペースにあるレシピを参照することができる。別のレシピに参照されるが、代表レシピは親(参照を行う)レシピのサブレシピとは見なされない。なぜなら代表レシピは別のエージェントが実行することになっており、それ自身がメインレシピでなければならないからである。代表レシピへの参照はスーパーバイザから自動的に送られる“select”および“start”コマンドシーケンスと同じものである。だから通常は選択オペレーションの制約に従う。

代表レシピは、実装ごとの規則次第で、実行エージェント(レシピエグゼキュータ)の名前を必要としてもよいし、そうでなくてもよい。例えば、スーパーバイザは指定されたネームスペースに基づいてランタイムの際に適切な実行エージェントを決めることができる場合もある。しかし、ある装置にとっては処理の結果が要注意であるかもしれないので、実行エージェントを指定するという能力はユーザにとって重要なものである。そのため、代表をサポートして

いるレシピ言語ではある実行エージェントを使用するよう指定できるようにする。この場合の完全な**レシピ指定子**のテキストフォーマットは次のようになる:

"Agent-Name > Namespace-Name > CLASS/Name; Version"

指定子内の識別子がとらなければならない順序は: エージェント, ネームスペース, レシピである。

3.2.4.1.3 リンクされたレシピセット — あるレシピが別のレシピを参照する場合, 一緒に実行されることになっているレシピの組を作るには, メインレシピから始めて, その組のメンバーを決めるために外部参照を集める。この組を**リンクされたレシピセット**と呼び, 参照を集める操作が**リンク操作**である。

3.2.4.2 変数パラメータ — **変数パラメータ**とは, レシピ自身の外から値を割り当てることができる変数である。変数パラメータを使うと, レシピは, 温度設定値, タイムディレイの間隔, およびデータセット名の設定といった操作に定数ではなく変数を使うことができる。この能力はレシピの再利用性を非常に高める。

パラメータはまず, レシピ本体の中で**定義**され, ユニークな**パラメータ名**, **パラメータ初期値**(デフォルト値)が与えられ, その値は無視されない限り, レシピが実行を選択された時に使用される。適用可能な場合には, 定義の中にパラメータに割り当てられた値が有効となるために満たさなければならないひとつあるいはそれ以上の条件を表わす**パラメータ制約**も含まれる。

レシピ本体内のパラメータ定義のシンタックスは, レシピ言語のシンタックスに従っていて必要な要素を持っている限り, 指定しない。変数パラメータの仕様に関するそれ以外の制約を, レシピ言語の仕様を介してレシピエグゼキュータのサブライヤが課す場合もある。

パラメータ領域というのは, **パラメータ制約**(があればその)条件を満たす形式(1.4.3 項参照)がとり得る値全部をまとめたものである。

形式は, レシピ本体で宣言されたパラメータ定義に含まれる必要はない。むしろ, レシピエグゼキュータが検証プロセスの間に, パラメータがレシピ内で適用される方法から導き出される場合がある。例えば, 温度を符号なし整数として内部で持っているなら, 温度設定値と範囲を割り当てるためにレシピ内で使用されるパラメータも, 普通は実行エージェントのサブライヤから符号なし整数で表すことが要求されるであろう。サブライヤは変数パラメータとして使用される可能性のあるパラメータひとつひとつの形式と有効領域を文書にする。

注: パラメータの値はレシピのテキストの中にある初期(デフォルト)値から変更される場合があり, また, サブレシピに渡される「引き数」として設定されることもある。こういった能力はレシピ言語の一機能であり, RMS の範囲外である。

二つのカテゴリのパラメータが定義されている: **数値**と**非数値**である。

3.2.4.2.1 数値パラメータ — 数値パラメータには**パラメータ下限**と**パラメータ上限**の間でフォーマットタイプに合ったあらゆる数値をとることができるパラメータすべてが含まれる。

パラメータ定義を保管するあらゆる属性での数値パラメータに関するパラメータ制約は下記の中の 1 項目に準拠しているテキスト文字列となる:

- “(a,b)⌋UNITS”x の領域は $a < x < b$
- “(a,)⌋UNITS”x の領域は $a < x < +\infty$
- “(,b)⌋UNITS”x の領域は $-\infty < x < b$
- “UNITS”x の領域は $-\infty < x < +\infty$

ここで

1. a および b は“0”から“9”までの数値, プラスおよびマイナス符号“+”, “-”, ピリオド“.”, および浮動小数点用の文字“E”あるいは“e”でできている数値である。
2. ⌋はオプションのホワイトスペースを表わす。
3. UNIT は Units of Measure Identifiers (SEMI E5, 9 章参照)に準拠している**大文字と小文字の区別をする有効な文字列**である。

例えば、下記の文字列は両方とも有効である：“(500, 2000)└┐degC”と“(0, 100)”等号を入れるには左端あるいは右端の括弧をそれぞれ角括弧 “[and/or]” で置き換える。例えば，“[a,b)”は $a \leq x < b$ を表わす。

範囲の制約なしに単位だけを指定するには下限を $-\infty$ に、上限を $+\infty$ に設定する。

しかし、下限および上限は、レシピエグゼキュータのサプライヤがそのパラメータに対して設定した絶対最低および最高の限界を越えない。

パラメータの制約のない数値パラメータは $(-\infty, \infty)$ という領域の (単位なしの) 純数字である。

数値パラメータは下記のフォーマットタイプのうちのどれかである：

- 長さが 80 キャラクタ以内のテキスト文字列
- 同じ長さの符号付きあるいは符号なしの整数
- 同じ長さの浮動小数点の数字

テキスト文字列は領域の下限および上限を表わすための、上記に定義されたキャラクタの組に限られ、領域内で数値に変換される。

3.2.4.2.2 非数値パラメータ — **非数値パラメータ**は、単一の数学的な間隔では表現できない離散数の組が領域であるパラメータを含む、数値パラメータ以外のあらゆるパラメータ、および、名前を表現する文字列である。非数値は置き換え値に制約を設けてもよいし、そうでなくてもよい。例えば、ウェーハマップの名前を含むパラメータは一般的な規則⁷では簡単には制約できないが、熱電対タイプを識別している文字列を含むパラメータは有効な文字列を順不同に集めた定義された組のひとつに制約できる。非数値パラメータについての制約は、値が使用される前に真であると評価される論理表現でも構成されている、あるいはそういう論理表現を含んでいることもある。

レシピ言語が、レシピの対応クラスのために定義されている場合には、制約に関するシンタックスは、ソース形式のレシピ本体の中にあるそのパラメータの形式的な定義から直接取り出される (そのため、そのレシピ言語の仕様に準拠する)。そうでなければ、制約はレシピエグゼキュータのサプライヤがパラメータ属性のために特に定義したシンタックスで表わされる。

例えば、拡散炉のレシピのためのパラメータは、装置に A から F の組がある場合、どの組の PID 値を使うべきかということを指定する。そういったパラメータに関して値に自動的に設けられる制約は、レシピエグゼキュータのサプライヤが仕様で決め、“{A, B, C, D, E, F}” となり、その値に対するデフォルトはキャラクタ “C”，フォーマットタイプは “長さ 1 の ASCII 文字列” であると指定されることもある。

別の例では、制約は列挙した組から値を取る、という場合もある；

“{cassette, lot, batch}”

3.2.4.2.3 エージェント固有のパラメータ — 同じタイプのレシピエグゼキュータすべてで同じ結果が出るようにレシピを「調整する」ためには、個々の実行エージェントのパラメータあるいは異なったパラメータ制約に別々の初期値を与える必要があるかもしれない。特別な編集装置がレシピマネージャから与えられて、パラメータのオリジナル値や制約を特定実行エージェント用に修正できる場合もある。

注：レシピエグゼキュータにダウンロードされるレシピのなかにある対応するオリジナル定義は、エージェント固有のパラメータの定義に置き換えられる。これはランタイム時にはレシピエグゼキュータの選択オペレーションのために指定されたパラメータ値に代わる。

3.2.5 属性 — 属性の名前はそのオブジェクトでユニークでなければならないテキスト文字列である。本文書で定義されている属性の名前は標準的な使用法をするためにとつてある。

識別属性は OSS のオブジェクトを識別するために使用される属性である。この属性は OSS を介して使用できるが、RMS サービスではこれ以外の属性とは切り離して扱われており、「レシピ属性」のためのパラメータとしては転送されない。

⁷ 普通は制約なしの名前であるパラメータの妥当性はランタイム時に確立される。

識別属性および下記に書かれている**必須属性**以外の属性は常に定義された**デフォルト値**を持っている。デフォルト値を持つ属性で、特にその値が指定されていない場合は、デフォルト値として、それぞれの形式に対応する**ナル値**を持つと見なされる。テキスト文字列のナル値は長さゼロの文字列である。数字形式のナル値はゼロである。ブール形式のナル値は偽である。空リストあるいは空構造のナル値は空(長さゼロ)リストである。ナル値を表わす方法はプロトコルに任されている。

属性の値をデフォルト値に設定するとその属性は**クリア**される、あるいは**リセット**される。

必須属性とは、常に非デフォルト値を持つことを要求されているものである。

すべての**必須属性**を含む特定の属性は**要求**とされるものである。**要求属性**とは非デフォルト値でサポートされているものである。例えば、**BodyFormat** 属性は要求である。これは、オブジェクト形式のレシピ (**BodyFormat** はこれに関する非デフォルト値を持っている) がネームスペースでサポートされているということである。

このスタンダードで定義されている属性はどんなものでもその属性に関して指定されている規則に従ってのみ修正される。

管理レシピおよび実行レシピの属性はオブジェクトサービスを介してアクセスできる場合がある。しかし、オブジェクトサービスで設定できるのはその中のいくつかである。

OSS の目的のためには **RMS** で定義されている属性はすべて認識されなければならない。つまり、**RMS** で定義されているオブジェクトのあらゆる属性を参照する **GetAttr** サービス要求は「無効属性」エラーを返してはならない。属性がそのアプリケーションでサポートされていない場合には、形式に適したナル値を持っているものとしてその属性の値を示し、その値を **SetAttr** サービスで設定しようとするのを拒否する。

3.2.5.1 記述子 — 必須属性の中で特に重要な二つのタイプは**タイムスタンプ属性**と**長さ属性**である。この二つの属性は本体、エージェント固有のデータセット、および、本体のタイムスタンプおよび長さ属性を含むレシピオブジェクト自体の属性の組に提供される。

記述子にはレシピのひとつあるいは複数の要素:属性、本体、あるいはエージェント固有のデータセットについての長さ属性およびタイムスタンプ属性が(この順序で)入っている。記述子は別々のネームスペースに、あるいはネームスペースとレシピエグゼキュータのレシピ実行エリアに保存されている二つのレシピを比較するのに使われる。

注:リアルタイムクロックあるいはその他 100 分の 1 秒単位で時間を計れる装置があると、この比較のためのタイムスタンプの価値が非常に向上する。

3.3 フルレシピモデルおよび最少レシピモデル — 図 3.5 には **RMS** で定義されている標準的な属性すべてをサポートしている管理レシピおよび実行レシピのオブジェクトモデルが書かれている。非識別子の属性はアルファベット順に書かれている。こういった属性の多くは最小のアプリケーションには必要ではない。管理レシピおよび実行レシピの最少モデルはそれぞれ 3.4.2.3 項および 3.5.2 項に載っている。

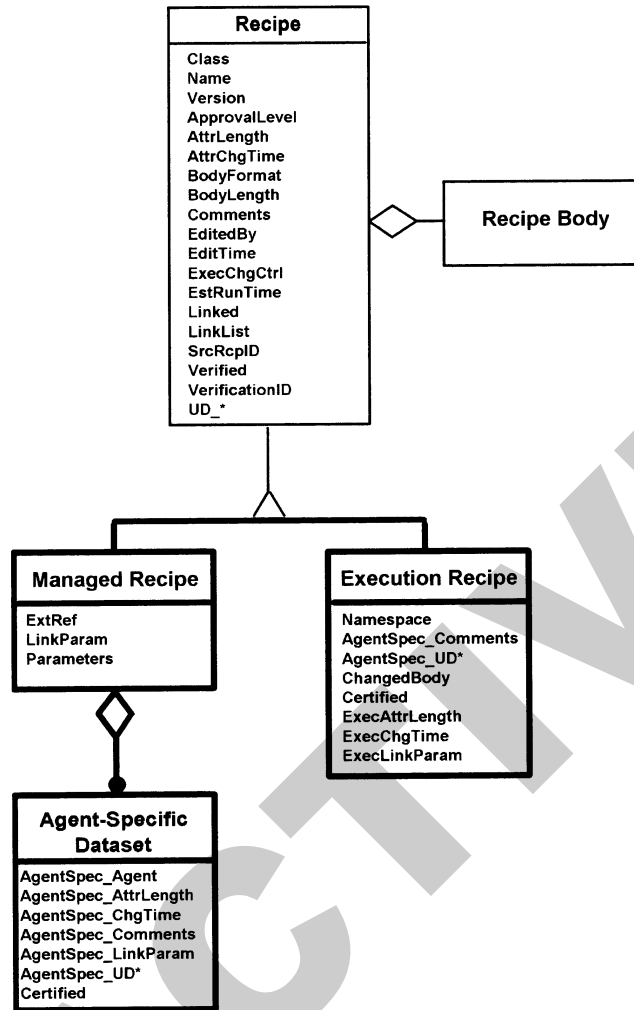


図 3.5
フルレシピオブジェクトモデル

3.4 管理レシピ 本項では管理レシピおよびその属性を定義している。

3.4.1 包括的属性 明確にするために、管理レシピの属性を時に**包括的属性**と呼び、エージェント固有の属性あるいは実行レシピの属性と区別する。

このスタンダードで特に定義されていない属性はアプリケーションあるいはユーザが定義することがある。ネームスペースあるいはレシピエグゼキュータが自分で使うために定義した属性は、その属性の作成者が定義し、文書化した規則でしか変更されない。

非スタンダード属性名と、将来スタンダードになるかもしれない属性との間の重複を避けるため、非スタンダード属性名は“UD_” (“U”および“D”という二つのキャラクタとそのあとにアンダーラインが来る) という接頭辞で始める。作成者は責任をもってユニークな属性名を与える。これ以外の RMS サービスを使う際には非スタンダード属性はすべてのスタンダード属性の後に来る。

ユーザが新しい属性を定義するための方法が提供されているのが望ましい。

3.4.1.1 タイムスタンプ属性 — **タイムスタンプ**情報(最後に行った変更の日付および時刻)はレシピ管理にとっては重要である。タイムスタンプ属性はレシピネームスペースが保持し、それ以外には変更されない。

タイムスタンプ属性には常に、レシピのある面が最後に変更された日付と時刻が入っている。この属性は“yyyymmddhhmmsscc”という形式のテキスト文字列で、yyyy は年、mm は月、dd は日、hh は時、mm は分、ss は秒、そして cc は 100 分の 1 秒である。

本体のタイムスタンプは EditTime と呼ばれ、レシピが最初に作成された時に設定され、どのような方法であれ、本体が修正された時はいつでもアップデートされる。

レシピの属性は本体の変更がなくても変更されることがあるので、レシピの包括的属性とエージェント固有の属性は、それ自体がそれぞれタイムスタンプ属性を持っている。包括的属性のタイムスタンプ属性は AttrChgTime と呼ばれる。(エージェント固有のデータセットの) エージェント固有の属性は、自分のタイムスタンプ属性を持っていて、AgentSpe ChgTime と呼ばれる。

3.4.1.2 長さ属性 — 長さ属性には 3 つある。本体の長さを持つもの、包括的属性の長さを持つもの、そしてエージェント固有の属性の長さを持つものである。長さ属性は、実装ごとに変わる可能性のある、内部保管用の専有フォーマットのようなプロトコルオーバーヘッドにもストレージオーバーヘッドにも関係なく計算される。これで、異なったレシピストレージ方式および異なった通信プロトコルに対して、長さを共通にできる。

本体の長さ属性である BodyLength には本体の長さがバイトではいつている。本体の長さは、プロトコルフォーマット情報に必要なかもしれないようなオーバーヘッドはすべて除いて、本体を転送する際に必要となるバイト数として計算される。

個々の属性の長さは、属性名と属性値の長さの合計として計算される。

属性の組の長さは、長さ属性自体を含めて、計算される時点での非デフォルト値に設定されている個々の属性の長さの合計として計算される。長さが計算される時点でデフォルト値に設定されている属性は、すべて計算から除外される。これは、レシピがネームスペースに出入りする時には非デフォルト値の属性しか移されないからである。また、属性が非デフォルト値に設定されている時には、長さ属性の値にはもっと大きな変化がおきる結果となる。

3.4.1.3 記述子 — 管理レシピの記述子は本体記述子 (BodyLength および EditTime)、包括的属性記述子 (AttrLength および AttrChgTime)、そしてエージェント固有の記述子 (AgentSpec AttrLength および AgentSpec ChgTime) である。管理レシピのレシピ記述子は属性記述子、本体記述子(この順序)で構成されており、その後に既存のエージェント固有のデータセットがあればそのすべての記述子が来ている。

3.4.2 管理レシピオブジェクト属性の定義 — この項ではレシピの包括的属性の定義を行っている。オブジェクトサービスについては、レシピはそのコンポーネントであるすべてのエージェント固有のデータセットのオーナーであると見なされている。本体にはオブジェクトサービスからアクセスできない場合がある。

表 3.1 および表 3.2 の属性は下記の順序で書かれている:

- オブジェクトタイプおよび識別子を含む、識別属性で表 3.1 および表 3.2 の太線の上に書かれているもの。
- 必須属性およびそれ以外の要求属性で、レシピが移された時に出てくる順に書かれている。
- オプションの属性がアルファベット順に書かれている。
- 最後に転送される非スタンダード属性。

RMS サービスでレシピを移す時には識別属性は属性リストには含まれず、要求属性はオプション属性の前に、順番に送信される。デフォルト値の非必須属性は移されない。その属性がないということがどういう値をとっているかということを示すからだ、オブジェクトサービスを介してそれを使用することは常にできる。オプションの属性の順序は命令されない。

3.4.2.1 包括的属性の定義 — 表 3.1 には管理レシピの包括的属性の形式的な定義が書かれている。

表 3.1 管理レシピ属性の定義

属性名	定義	アクセス	要求	型
識別属性				
<u>ObjType</u>	オブジェクトタイプ	RO	Y	テキスト: "MRcp"
<u>ObjID</u>	<u>Class</u> , <u>Name</u> , <u>Version</u> から出された識別子リネーム以外のものを使って識別子の一部を変更してはいけない。	RO	Y	フォーマットされたテキスト。

属性名	定義	アクセス	要求	型
<u>Name</u>	レシピが作成された時あるいはリネームされた時にユーザが割り当てる論理名。	RO	Y	テキスト。
<u>Class</u>	レシピのクラス (例: “/ PROCESS/”あるいは“/PROCESS/LOADER/”)	RO	Y	フォーマットされたテキスト: “CLASS/CLASS/./CLASS/”
<u>Version</u>	レシピのバージョン	RO	Y	テキスト。
必須属性				
<u>AttrLength</u>	包括的属性の長さの合計をバイトで表したもの。必須。	RO	Y	符号なし整数。
<u>AttrChgTime</u>	包括的属性に最後に行われた変更のタイムスタンプ。必須。	RO	Y	フォーマットされたテキスト。
<u>RodyLength</u>	レシピ本体の長さをバイトで表したもの。必須。	RO	Y	符号なし整数。
<u>EditTime</u>	本体がいつ作成されたか、あるいは最後にアップデートされたのはいつか、というタイムスタンプ。必須。	RO	Y	フォーマットされたテキスト。 タイムスタンプフォーマット。
要求属性				
<u>BodyFormat</u>	レシピ本体の形式およびフォーマットを示す。デフォルトはゼロ。	RO	Y	列挙された符号なし整数: 0=ソース, 1=オブジェクト, >1 保留。
<u>Varified</u>	レシピ本体のシンタックスが正しいかどうかを示す。レシピが作成された時、あるいはアップデートされるとリセットされる。デフォルトは偽。	RO	Y	ブール数。
<u>Linked</u>	レシピがリンクされているかどうかを示す。レシピが開始、検証、あるいはリンクを解除されたらリセットする。デフォルトは偽。	RO	Y	ブール数。
オプションの属性				
<u>ApprovalLevel</u>	権限を持つユーザが割り当てた承認レベルを示す。デフォルトはゼロ。レシピが開始あるいはリンクされたらリセットする。リンクされたレシピの承認レベルはそのサブレシピの承認レベルより低くする。	RW	N	符号なし整数。
<u>Comments</u>	ユーザコメント。	RW	N	テキスト。最大長は 80 キャラクタ。
<u>EditedBy</u>	レシピを最後に編集した人の名前。	RO	N	テキスト。最大長は 40 キャラクタ。
<u>EstRunTime</u>	レシピの名目上のあるいは予想される実行(ラン)時間を秒単位で表したもの。レシピが作成、あるいはアップデートされたらリセットする。レシピが検証されたら設定する。 リンクされた時にはメインレシピ用に合計時間を計算し直すことができる。スケジューリングに使用される。計算式は文書に書かれていなければならない。デフォルトは 0。	RW	N	符号なし整数。
<u>ExecChgCtrl</u>	レシピの変更コントロール要求を指定する (6.5 項参照)	RW	N	バイナリ。ビット形式 (MSB=8**) 1- レシピ本体は変更可能 2- 変更通知は必要 3- レシピは変更後に選択してもよい 4- 一番最近のパラメータ設定が保存される
<u>ExtRef</u>	レシピ内で使われるレシピの指定子全部のリスト。はっきりしたバージョンは必要ではない。レシピが作成、アップロード、および検証されたらリセットする。	RO	N	フォーマットされたテキストのリスト。

属性名	定義	アクセス	要求	型
<u>LinkList</u>	あるメインレシピおよびそのサブリストすべての <u>ExtRef</u> 属性の中にあるレシピ指定子の完全なリスト。除かれたコピーおよびはっきり決定されたバージョン全部も含む。リンクされたらメインレシピ用に設定する。レシピが開始あるいは検証されたらリセットする。マルチパートレシピサポートに必要とされる。	RO	N	フォーマットされたテキストのリスト。
<u>LinkParam</u>	あるメインレシピおよびのそのサブレシピすべての <u>Parameters</u> 属性に含まれている変数パラメータ定義全部のリスト。除かれたコピーを含む。レシピが作成、アップデートあるいは検証されたらリセットする。レシピがリンクされたら設定する。変数パラメータサポートに必要とされる。	RO	N	パラメータ名, 初期値, および制約で構成されている。
<u>Parameters</u>	レシピに含まれている変数パラメータ定義のリスト。レシピが作成、アップデートおよび検証されたらリセットする。レシピが検証されたら設定する。変数パラメータサポートにのみ必要とされる。	RO	N	パラメータ名, 初期値, および制約で構成されている。
<u>SrcRepID</u>	オブジェクト形式のレシピが派生するものとのソース形式のレシピの識別子。値はレシピの検証元が決定する。派生オブジェクト形式のレシピのサポートにのみ必要。	RO	N	フォーマットされたテキスト。
<u>VerificationID</u>	レシピの検証者が設定した識別コード。検証し直す必要がある時代遅れのフォーマットを決めるのに使用できる。	RO	N	テキスト, 最大長は 40 キャラクタ。
<u>UD*</u>	サプライヤあるいはユーザが定義した非スタンダード属性。星印は, この定義に使われている属性名の一部を示す。それを定義したエンティティ以外が正確にとってかおなければならない。	RO	N	定義によって変わる。テキスト形式は 80 キャラクタに制限されている。

注: SEMI E4 および E5 は, ビット 1-8 までの番号をつけている。ここで, ビット 8=MSB (最も重要なビット)。

3.4.2.2 エージェント固有の属性の定義 — ユーザが定義する属性の名前は“AgentSpec_UD_”という接頭辞で始まっていなければならない, 転送されたレシピから, 修正なしに保存されなければならない。こういった属性は転送されるとスタンダード属性すべての後につく。

エージェント固有のデータセットは, データセット自体のタイムスタンプ属性および属性長属性以外のエージェント固有の属性が設定された時にのみ存在する。

表 3.2 ではエージェント固有のデータセットオブジェクトの属性が定義されている。

表 3.2 エージェント固有のデータセットオブジェクト属性の定義

属性名	定義	アクセス	要求	型
<u>ObjType</u>	オブジェクトタイプ	RO	Y	テキスト=“MRpASDS”
<u>ObjID</u>	オブジェクトの識別子。AgentSpec Agent の値を含む。	RO	Y	テキスト。
<u>AgentSpec Agent</u>	データセット内にあるこれ以外の属性が適用される実行エージェントの名前。必須。	RO	Y	テキスト。
<u>AgentSpec AttrLength</u>	エージェント固有の属性の長さをバイトで表したもの。必須。	RO	Y	符号なし整数。
<u>AgentSpec ChgTime</u>	エージェント固有の属性が最後に変更された時のタイムスタンプ。必須。	RO	Y	フォーマットされたテキスト。
<u>Agentspec Comments</u>	著者が入力したエージェント固有のコメント。	RW	N	テキスト。 最大長は 80 キャラクタ。
<u>AgentSpec LinkParam</u>	LinkParam のリストから修正された変数パラメータ定義のリスト。リンクされたメインレシピにのみ有効。パラメータ名および形式は変更できない。	RO	N	パラメータ名, 値, および制約で構成されている構造のリスト。

属性名	定義	アクセス	要求	型
<u>Certified</u>	権限のあるユーザが割り当てた、特定のエージェントの認可レベル。 AgentSpec LinkParam が修正されたりリセットする。認可サポートに必要とされる。	RW	N	符号なし整数。
<u>AgentSpecUD*</u>	サプライヤあるいはユーザが定義した非標準化属性。星印は、この定義に入っている属性の一部を示している。定義を行うエンティティ以外は正確にとっておかなければならない。	RO	N	定義が変わる。テキスト形式は 80 キャラクタに制限されている。

3.4.2.3 最少の管理レシピ – 管理レシピの最少のモデルは図 3.6 に示されている。このモデルでは必要とされる属性しかサポートされていない。このモデルは、単体装置が必要とするデフォルトネームスペースのような、専用ネームスペースにあるシングルパートレシピにしか使用できない。エージェント固有のデータセットはない。

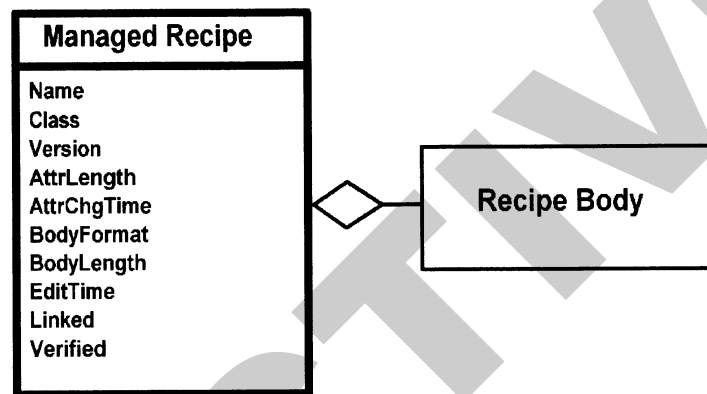


図 3.6
最少の管理レシピのためのオブジェクトモデル

3.5 実行レシピ – レシピエグゼキュータが処理するレシピのタイプは実行レシピと呼ばれる。このタイプのレシピは、属性と本体を持っているがエージェント固有のデータセットは持っていない。管理レシピの包括的属性の大部分と、エージェント固有の属性のいくつかも実行レシピの属性であるが、それ以外の場合には包括的属性とエージェント固有の属性はマージされて新しい実行レシピ属性になる。

管理レシピは、レシピエグゼキュータにダウンロードされると実行レシピに変換される。それと同様に、実行レシピはレシピエグゼキュータからアップロードされると管理レシピに変換される。

実行レシピは 6.3 項で詳細に定義されている。

4 レシピネームスペース

本章ではレシピネームスペースのための基本的な概念的モデルを定義している。レシピネームスペースとはレシピ保存および管理のための論理領域であり、レシピの識別子は、この中ではユニークであることを保証されている。本章に書かれているモデルは集中保存を行う実装に適用できる。第 5 章では分散レシピネームスペースのために基本モデルを拡張したものが定義されている。

一般的に「ネームスペース」という用語はユニークな識別子の領域を指す。ネームスペースの境界という問題は、あらゆるオブジェクトタイプに関して、特に分散環境で存在していて、レシピ独特のものではない。しかし、RMS コンテキスト内では、ネームスペースはレシピネームスペースの同義語として使用されている。

4.1 動機 – レシピの識別子は、異なったネームスペースの間で、あるいは工場内ではユニークでない場合もある（つまり、ある識別子は、ひとつのネームスペース内を除いて、さまざまなレシピに使われる可能性がある、ということである）。ネームスペースの一義的な役割は、あるレシピ識別子のユニーク性が保証されるようなエリアを定義することである。

本章に書かれているレシピ管理のための要求には以下のものが含まれている:

- あるレシピが、他に同じものはないと識別され、保存、検索されることが可能な独自のエリアの境界を定義すること
- レシピ保護およびレシピオペレーションを含む、レシピの管理領域のための属性およびオペレーションを定義すること
- 取り外し可能なメディアをコントロールできるようにすること
- 単体装置がレシピを実行できるようにすること
- 単体装置をオンラインの工場システムにスムーズに統合するのを助けること
- 通信が途切れた時に統合された装置がレシピを実行できるようにすること
- 第 5 章に書かれている分散モデルの基礎を提供すること。それによってスーパーバイザは自分の従属エージェントの保存容量を使用、管理することができる。

4.2 ネームスペースモデル — ネームスペースモデルには二つの大きな目的がある。まず、共通した管理規則を提供する。次に、共通のプロセスタイプ、共通の機能、および共通のレシピ言語を持っている実行エージェントのグループが、レシピを共有できるようにする。

図 4.1 には、レシピ、ネームスペース自身、レシピネームスペースセグメントというネームスペースのコンポーネント、そしてレシピネームスペースマネージャ、という 4 つのオブジェクトを持つ基本的なネームスペースモデルが書かれている。

レシピネームスペースマネージャ(マネージャ)は、外界に対するネームスペースのインタフェースであり、ネームスペース内では内部決定機関である。ネームスペースおよびそのレシピに対するサービスはすべてマネージャが提供する。ネームスペース自体は受動的である。重要な属性を持っているにも関わらず、ネームスペースはオペレーションを行わず、サービスも提供しない。

レシピネームスペースセグメント(セグメント)はマネージャの監督のもと、内部保存要素とネームスペース内での実際のレシピ操作の両方を行う。基本モデルでは公開サービスは行っていない。このモデルでのセグメントとマネージャの関係はアプリケーション内部の関係ということになっている。

図 4.1 は集中保存を行うネームスペースを表わしている。しかし、これを分散レシピネームスペースへ拡大するのは簡単である。分散レシピネームスペースには複数のセグメントがあり、異なった外部エージェントが個々のセグメントを提供している。その場合、マネージャおよびセグメントは互いに公開サービスを提供しなければならない。

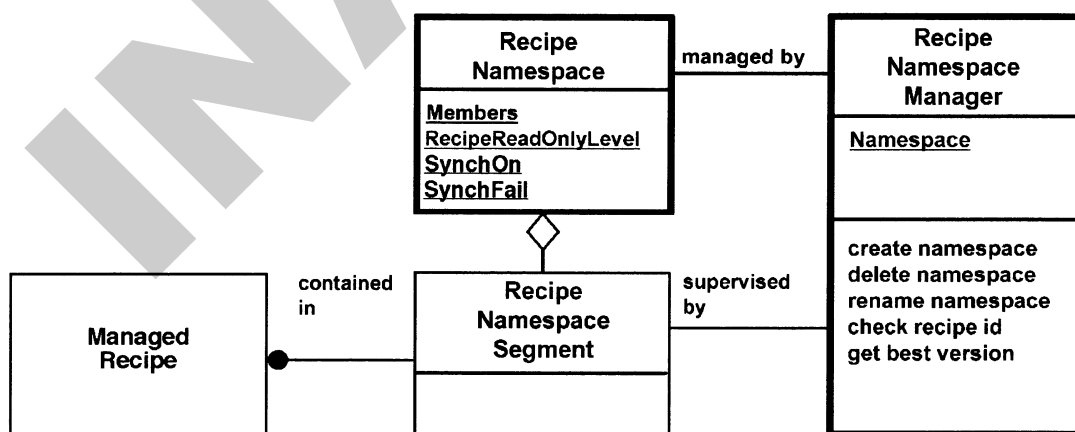


図 4.1
レシピネームスペースモデル

レシピ管理サービスを介してレシピに行われるすべてのオペレーションは、二つの例外を除いて、**ネームスペースマネージャ**によって、あるいはネームスペースマネージャの監督のもとでネームスペース内で行われる。ネームスペースマネージャは、レシピ言語のシンタックスを理解する必要はない。この理由から、レシピの**編集**(レシピ本体の修正)は、ネームスペースの定義の一部ではなく、また検証は、適切な実行エージェント(6章参照)の**レシピエグゼキュータ**が行う。この二つの活動には使用されている特定のレシピ言語について徹底的な理解が要求される。これらの活動はそれ以外から切り離されているため、包括的ネームスペース能力を提供することができるのである。

4.3 **ネームスペースの仕様** — レシピネームスペース、そのセグメントコンポーネント、およびそのマネージャの組み合わせが RMS に準拠したレシピの保存、検索、および管理を行う。

ネームスペースマネージャはネームスペースの完全さ、ネームスペース内のレシピの完全さ、およびレシピ識別子の完全さを責任をもって保持する。ネームスペースマネージャはレシピ、その属性およびそのコンポーネントに関する規則を理解し、責任を持ってその規則を実際に守らせる。例えば、読み込み専用のレシピの変更や削除を許さない。そのため、ネームスペースマネージャは、読み込み専用のレシピにすでに使用されている識別子を持つレシピを受け付けない。

ネームスペースはレシピ全体あるいはレシピの属性に関する**読み取りアクセス**には制約は設けず、また、ネームスペース外に出すような使い方にも関心を示さない。複数のエージェントが同じレシピを共有できるかどうかは、もっぱらエージェントが同じネームスペースにアクセスできるかどうかによって決まる。安全性および信頼性の問題は RMS の範囲外である。

レシピネームスペース、あるいはネームスペースという用語は、その属性、オペレーション、および本項に書かれているネームスペースとその分散サブタイプの両方に共通の要求をも含めて指すような使い方をされている。**集中ネームスペース**という用語は、基本モデルの実装実体(インプリメンテーション)を指すために、さらに分散サブタイプの実装実体に適用されない記述を明確にするために使われている。

集中ネームスペースは、ファイルの単一ディレクトリによく似ている。そこでは、全く同じファイル名は存在を許されない。ネームスペースの要求に付随するいくつかの異なった構成では存在する場合もある。

物理的に保存するために取り外し可能なメディアを使うネームスペースを提供することも可能である。その場合には、このネームスペースと別のどんなネームスペースとでも、レシピをやりとりすることができる。

ネームスペース管理の要求に従っている場合以外は、ネームスペース内にレシピを移すことは絶対にできない。例えば、読み込み専用のレシピを取り替えることはできない。

集中ネームスペースは、いくつかの方法で使うことができる。単体モードで操作される可能性のある、実行能力を持っているエージェントは、単体モードで操作された時に使用されることになる集中ネームスペースを自分自身に提供する。クラスタの中にあるディスク不要のプロセスモジュールといった、これ以外のエージェントはクラスタスーパーバイザが提供するネームスペース能力に頼られると思われる。

4.4 **メンバーエージェント** — 複数のエージェントが使用するレシピを持っているネームスペースは**共有**されている、と言われる。そうでない場合には**専用**、あるいは**共有されていない**、と呼ばれる。図 4.2 にはエッチャーが提供しているものとホストが提供しているものの二つの専用ネームスペースが書かれている。この用語は、明確さと説明のためだけに導入されている。

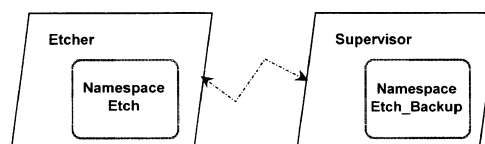


図 4.2
装置ネームスペースのホストバックアップ

ネームスペースの **Members** 属性には、ネームスペースでレシピを使うエージェントの名前(オブジェクト識別子)が入っていて、そのエージェントはそのネームスペースの**メンバーエージェント(メンバー)**と呼ばれる。ネームスペースマネージャは、レシピを検証するのにメンバーエージェントの補助が必要なので、この情報を使用する。

ネームスペースの **Members** 属性に複数のエージェント名が含まれている場合には、そのネームスペースは共有されている。そうでない場合には共有されていない、あるいはまだ完全にはセットアップされていないのかもしれない。ネームスペースが空(ナル)の場合には、そのネームスペースは完全にはセットアップされておらず、レシピ検証を含めたあるオペレーションではネームスペースに内容がなければならないので、完全に機能するとは言えない。

4.5 イラストレーション – 本項には、集中ネームスペースで組み合わせ可能ないくつかの構成が書かれている。

図 4.2 は自分自身のローカルなネームスペースを持っていて RS-232 で自分のスーパーバイザと通信するエッチャーである。この例のスーパーバイザは別個のネームスペースを持っていて特に重要なレシピのバックアップを行っている。

図 4.3 は 4 つのディスクレスの下位装置を持っているスーパーバイザである。この構成ではスーパーバイザは集中ネームスペースから 4 つの下位装置すべてにレシピを提供している。

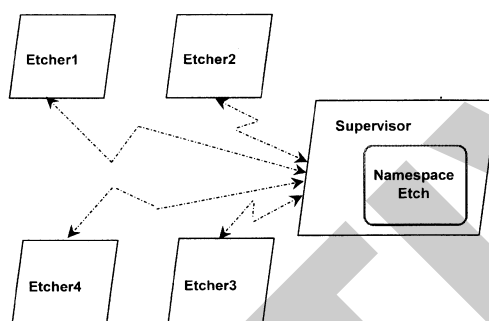


図 4.3
スーパーバイザが提供するネームスペース

図 4.4 はスーパーバイザーエージェントと集中ネームスペースを提供する 6 番目のエージェントを持つ共通ネットワーク上の 4 つの実行エージェントである。各実行エージェントはスーパーバイザとは別個にネームスペースにアクセスできる。

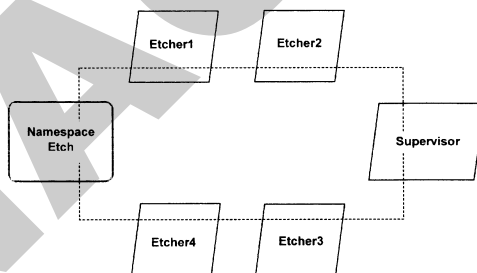


図 4.4
共有されたネットワークアクセス

図 4.3 と図 4.4 は両方とも共有された集中ネームスペースの例である。

4.6 属性定義表 – 表 4.1 はレシピネームスペースの属性を定義したものである。

表 4.1 レシピネームスペース属性の定義

属性名	定義	アクセス	要求	形式
<u>ObjType</u>	オブジェクトタイプ。	RO	Y	テキスト=“RNS”
<u>ObjID</u>	ネームスペースの名前。	RO	Y	テキスト。“Default”という名前は禁止。
<u>RecipeRead OnlyLevel</u>	レシピが読み込み専用である際の承認レベル。	RW	Y	符号なし整数

属性名	定義	アクセス	要求	形式
<u>Members</u>	Members ネームスペース内でレシピを検証し実行することができるエージェントの名前	RW	Y	エージェント名のリスト(識別子)
<u>SynchOn</u>	同期レベル(9.5 項参照)。同期がサポートされている場合には必須	RW	N	符号なし整数: 0ー ディスエーブル, あるいは下記の組み合わせ(合計), のどちらか: 1ー 本体の変更 2ー 新しい実行レシピ 8ー 最終値の変更 16ー 新しい派生オブジェクト形式実行レシピ
<u>SynchFail</u>	同期を取ろうとして失敗した実行レシピの指定子。同期がサポートされている場合には必須	RW	N	フォーマット付きテキストのリスト

表 4.2 はレシピネームスペースマネージャオブジェクトの属性を定義したものである。ネームスペース指定子がもっと広く使われているため、マネージャの名前(識別子)は、一般的にそれほど重要なものではない。しかし、オブジェクトサービスを介してマネージャにアクセスできるというのは重要なことである。

表 4.2 レシピネームスペースマネージャの属性の定義

属性名	定義	アクセス	要求	型
<u>ObjType</u>	オブジェクトタイプ。	RO	Y	テキスト=“RNS_Mgr”
<u>ObjID</u>	マネージャの名前。	RO	Y	テキスト
<u>NamespaceName</u>	管理されているネームスペースの名前。	RO	Y	テキスト

5 分散レシピネームスペース

分散レシピネームスペースとは、レシピを保存するために、複数のエージェントの保管場所を利用するレシピネームスペースである。レシピはさまざまなエージェントが提供する特別なレシピネームスペースセグメントに保存される。こういったセグメントが分散レシピネームスペースセグメントであり、分散レシピネームスペースマネージャに監督される。

本章では、分散レシピネームスペース能力を共同で提供するさまざまなオブジェクトを定義している: すなわち、分散レシピネームスペース、分散レシピネームスペースマネージャ、分散レシピネームスペースセグメント、および分散レシピネームスペースレコーダである。オペレーションの詳細な説明は 10 章に書かれている。

本書では DRNS という略語は「分散レシピネームスペース」のことであり、主に DRNS サブタイプオブジェクトとそのスーパータイプとを区別するのに使用されている。

5.1 動機 — 分散レシピネームスペース能力は、図 5.1 に描かれているように、複数のエージェントの保管場所を使う方法を提供する。この能力によって集中型の工場システムに要求される記憶容量を減らすことができ、エージェントがレシピを使えるようにすることでパフォーマンスを向上させ、ネームスペースを集中管理して同じレシピ識別子のユニークさを保つことを保証する。

図 5.1 には、分散レシピネームスペースと 4 つのセグメントが描かれている。個々のセグメントは、それぞれ別個のエージェントが提供していて、エージェントもそれぞれローカルなネームスペースを持っている。

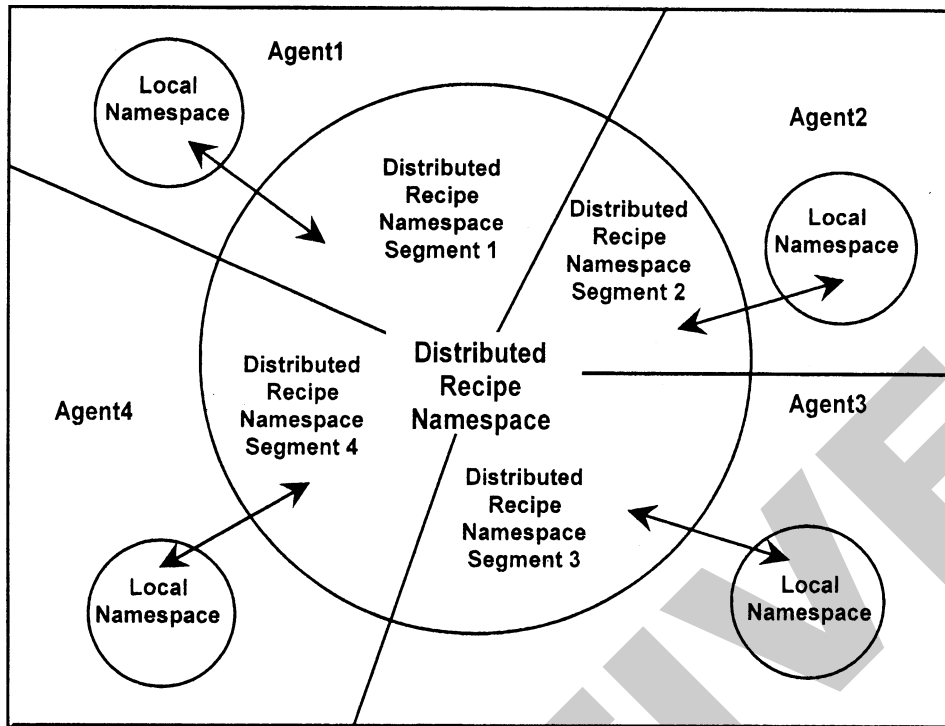


図 5.1

エージェント、セグメント、およびローカルネームスペース図

5.2 概要 — DRNS オブジェクトモデル (図 5.2) は 4 章で紹介されているモデルを特殊化したものである。分散レシピネームスペース、DRNS セグメント、および DRNS マネージャは、それぞれ、レシピネームスペース、レシピネームスペースセグメント、およびレシピネームスペースマネージャのサブタイプである。どれも対応するスーパータイプの属性およびオペレーションを継承している。オブジェクトの説明の箇所には DRNS タイプ独自の属性およびオペレーションだけが書かれている。

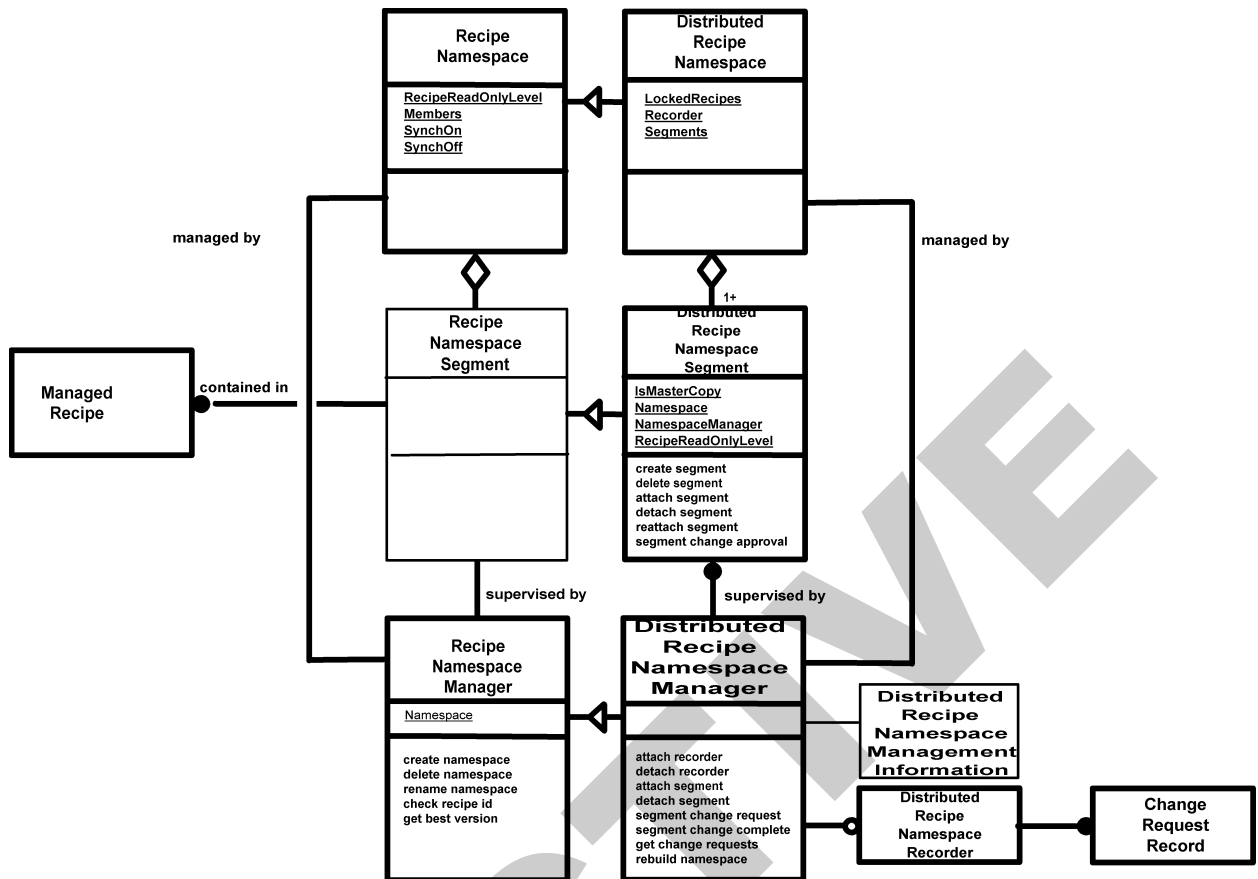


図 5.2
分散レシピネームスペースのモデル

ネームスペースのサブタイプには、保存セグメントがひとつある。保存が集中化されている時には、セグメントは内部にあってネームスペースに属している。しかし、分散レシピネームスペースの場合には、それぞれ別個のエージェントが提供している複数の DRNS セグメントがある場合もある。この理由から、DRNS セグメントは標準オブジェクトである。

分散レシピネームスペースは、特定のレシピ保存領域にあるレシピの管理を、ネームスペース全体の管理から切り離す。特定の保存領域にある個々のレシピの管理は、分散レシピネームスペースセグメントに任される。分散レシピネームスペースマネージャは、さまざまなセグメントおよびネームスペース自体を管理する。マネージャは、ネームスペース全体に保存されているレシピ全部の識別子を知っておかなければならない。マネージャは、同じ名前でもバージョンが違うレシピを区別するのに、レシピの識別子の構造およびバージョンの重要度を知っておかなければならない。

DRNS レコーダは、分散レシピネームスペースの再構築を自動化するのに必要な情報のバックアップ装置となる。これは DRNS ネームスペースおよび DRNS マネージャの外に存在する。

5.3 分散レシピネームスペースの問題 — 本項では分散レシピネームスペース能力の中にあるひとつ以上のオブジェクトに適用される問題について論じる。

5.3.1 オブジェクトサービス — 連結というのは、管理されているオブジェクトとそのマネージャとの間にある、OSS (SEMI E39) で定義されている関係である。マネージャひとつに対して、ひとつのオブジェクトが連結、連結解除、再連結できる。連結されている時には、管理されているオブジェクトは、非常に重要な特定のサービスが自分のマネージャから要求されているということが分かる。セグメントおよびレコーダのオブジェクトは、DRNS マネージャと連結、連結解除、および再連結するには、SEMI E39 (OSS) で定義されているオペレーションおよびサービスに関する要求条件に準拠しなければならない。マネージャは、こういったオブジェクトの本来リードオンリーである指定属性の修正もできる。

セグメントあるいはレコーダを生成あるいは削除する能力はオプションである。

許可されたユーザは、特定のセグメントあるいはレコーダを連結あるいは連結解除するよう、DRNS マネージャに要求できる。

5.3.2 論理レシピ — **論理レシピ**とは、特定の本体および特定の包括的属性値を持つレシピであると定義される。管理レシピで保存されているものはすべて論理レシピの実体である。

分散レシピネームスペースでは、普通、ある論理レシピの実体(コピー)はどんな時でもひとつ以上の DRNS セグメントに存在している。言い換えれば、あるレシピに関して、同じ識別子を持つ複数の複製コピーが分散レシピネームスペース内で共存できる、ということである。ネームスペースを完全な形にしておくため、これが許されるのは、ある識別子を持つレシピのコピーひとつひとつが同じ論理レシピの実体である場合だけとなっている。この論理的同一性を確実に維持するのは DRNS マネージャの責任である。

エージェント固有のデータセットは論理レシピの定義には含まれない。なぜなら、複数のエージェント固有のデータセットが互いに独立して存在する可能性があり、普通、DRNS セグメントを提供しているエージェント固有のデータセットしかそのセグメントには保管されないからである。

5.3.3 変更要求 — **変更要求**は、ユーザ、外部アプリケーション、あるいは連結された DRNS セグメントがレシピに変更を加えるよう、あるいは変更を許可してくれるよう DRNS マネージャに要求する時に常に発生する。変更要求に関する情報は論理の変更要求レコードの形式で保管される。変更要求が出されると、**変更要求レコード**が生成され、変更が完了するまで、あるいは変更が廃棄されるまで保管される。変更要求で表される情報は公開されるので、ある程度の診断能力を提供することになる。

変更要求レコードは正式なオブジェクトでもなく、標準化されてもいないが、サービスを通じて知ることのできる情報である。

変更管理は、DRNS 能力の最も重要な問題であり、詳細は後述されている。

5.4 分散レシピネームスペースセグメント — **分散レシピネームスペースセグメント**(図 5.3)は、保存、検索、削除、およびレシピの属性あるいは本体を変更するオペレーションを含む、直接レシピ操作に関わる活動すべてに責任を負っている。これにはレシピの識別子、その構造、さまざまなレシピ属性の相互関係、レシピの保存方法、および伝達方法についてマイクロレベルの知識が必要とされる。これには、そのセグメントに保存されているレシピの識別子全部についてのマクロレベルの知識も入っている。DRNS セグメントは、ある識別子を持つレシピはその DRNS セグメントには只ひとつしか存在しないようにしなければならない。連結されていない DRNS セグメントの内容はリードオンリーである。

DRNS セグメントおよびその保管場所は、エージェントが提供する。エージェントは装置、独立した「レシピサーバー」、あるいはそれ以外の工場システムかもしれない。

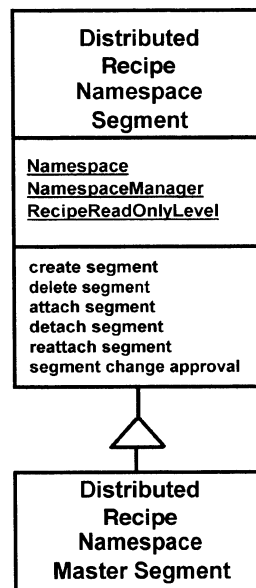


図 5.3
分散レシピネームスペースのセグメント

5.4.1 マスターおよび専用セグメント – マスターセグメントとは、レシピごとにエージェント固有のデータセットを複数保存できる DRNS セグメントを特殊化したものである。マスターセグメントは一台の装置の専用ではなく、そのネームスペースの中にある論理レシピ全部のフルコピー（エージェント固有の既存データセット全部を含む）を保存するのに用いられる。分散レシピネームスペースマネージャはどれも、連結されているマスターセグメントを少なくともひとつはサポートできなければならない。バックアップ保護のためにマスターセグメントがもっとあった方がよいかもしれない。

専用セグメントという言葉は、マスターセグメントでないセグメントを指すのに使われる。専用セグメントは、レシピごとにエージェント固有のデータセットを少なくともひとつ保存する。装置が提供する DRNS セグメントは、別の装置のエージェント固有のデータセットは保管しないことになっているので、普通、装置はマスターセグメント能力を提供していない。

マスターおよび専用セグメントは、同じ属性を持っていて同じメッセージサービスをサポートしているが、エージェント固有のデータセットへの対応は異なっており、異なった目的のために使用される。マスターセグメントは、エージェント固有のデータセットを変更する前に承認が必要である。

5.4.2 変更の制約 – DRNS セグメントは、保存を行っているエージェントの別のコンポーネントが、自分の記憶領域内にあるレシピへアクセスできるようにしている場合がある。しかし、そのセグメントで保存されているレシピの包括的属性も本体も、分散レシピネームスペースマネージャがはっきりと承認しない限り、変更されない。

DRNS マネージャに連結されている専用 DRNS セグメントは、最初に許可を得なくてもエージェント固有のデータセットの内容を変更できる。しかし、変更直後にマネージャにそのことを通知しなければならない。

属性 RecipeReadOnlyLevel は、セグメントが最初に連結された時とネームスペース属性が変更された時は必ず、DRNS マネージャが対応するネームスペースの RecipeReadOnlyLevel 属性の値にセットする。この属性は、ネームスペース属性と同じ機能を持っており、これを使うと、セグメントは 8.2.5 項で定義されているように、レシピの承認レベルに基づいて変更を禁止できる。例えば、セグメントは書き込み保護されているレシピを修正するようにという要求を拒否する。

DRNS セグメントは、連結されていない時に内容を変更することを禁じられている。しかし、オーナーエージェントの別のコンポーネントを含めて別のエンティティは、いつでもレシピおよびレシピ属性を読める。

DRNS セグメントが保存している論理レシピに加えられる変更は、変更が行われる前に、まず DRNS マネージャが承認する。これには本体および包括的属性に加えられる変更も含まれる。これが必要な理由は二つある。第一に、

二つの DRNS セグメントが同時に同じレシピを変更しようとするかもしれず、その活動を整合させなければならない。第二に、ひとつのレシピの実体が複数存在する場合にレシピ識別子を損なわないようにするため、分散レシピネームスペースは確実に、その変更に関して実体全部が適切にアップデートされてから同じレシピに対するそれ以外の変更が許可されるようにしなければならない。

注: 無許可の変更は禁止されていても、許可を待っている間 DRNS セグメント外でそういった変更を保存することはできる。しかし、上記の変更の後および許可を受ける前に別の DRNS セグメントが加えた変更がこういった変更を無効にする場合がある。

RMS で定義されている正式なサービスを必要としないあるいは使用しないエージェントのコンポーネント間の通信は、そのエージェント専用のものであると見なされ、上記の変更に関する制約に従う以外は、RMS ではカバーされておらず除外されてもよい。

Distributed Recipe Namespace Recorder
<u>LockedRecipes</u> <u>Namespace</u> <u>NamespaceManager</u> <u>Segments</u>
create recorder delete recorder attach recorder detach recorder reattach recorder add segment remove segment get segments add change request delete change request get change request

図 5.4
分散レシピネームスペースレコーダ

5.5 分散レシピネームスペースレコーダ 分散レシピネームスペースレコーダは、壊れた分散レシピネームスペースの自動再構築に非常に重要な情報を外部で保存、検索する手段を提供する。これには二つのタイプの情報が含まれている: DRNS マネージャに連結されている DRNS セグメントのリスト、およびレシピごとに進行中の現時点での変更要求レコードである。

DRNS レコーダは、DRNS マネージャと連結および連結解除でき、そのマネージャに再連結できる。DRNS レコーダが持っている情報には誰でもアクセスできるが、変更できるのはそのマネージャだけである。

DRNS レコーダは、自分の Segments 属性の中に、連結されている DRNS セグメントの DRNS セグメント指定子 (オブジェクト指定子) を保存できる。この情報を使うと、ネームスペースあるいは DRNS マネージャが壊れてその情報が失われたあるいはアクセスできなくなった場合に、分散レシピネームスペースを自動的に再構築できる。

DRNS レコーダはレシピごとの現時点での変更要求の保存、変更要求の削除、およびひとつ以上のレシピに関する複数の変更要求を返すこともできる。属性 LockedRecipes は変更要求が出されたレシピの識別子のリストを提供する。

DRNS マネージャは、サービスユーザが DRNS レコーダをネームスペースに連結している場合には、DRNS レコーダのサービスを使う。しかし、レコーダの使用 (連結) はユーザのオプションである。DRNS レコーダが提供されているのは非常に重要な情報を離して保存するためであって、ユーザに全般的な情報源を提供するためではない。インアクティブ変更要求情報は提供できない。

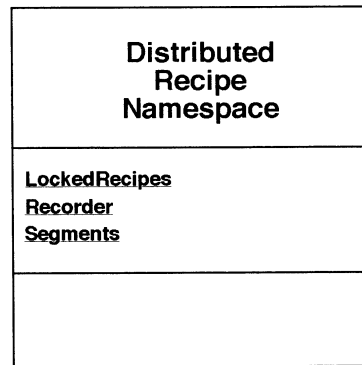


図 5.5
分散レシピネームスペース

5.6 分散レシピネームスペース管理情報 – DRNS 管理情報オブジェクトは DRNS マネージャの専有物で非公開のものである。これは、DRNS マネージャには管理が必要であるという情報の重要性を強調するために図 5.2 にも描かれている。この情報には連結されているセグメントごとに保存されているレシピ識別子および既存の変更要求レコードが全部、さらにそれ以外の情報も入っている。

DRNS マネージャは自分に連結されているセグメントすべての内容を常におかenneなければならない。マネージャは分散レシピネームスペースの中にあるレシピの実体ひとつひとつを識別できなければならない。マネージャは、あるセグメントのレシピに加えられた変更が別のセグメントにアップデートされた時、分散レシピネームスペース内の論理レシピの実体ひとつひとつの現在のステータスを追跡しておく責任がある。DRNS 管理情報はこういった目的のために重要である。

5.7 分散レシピネームスペース – 分散レシピネームスペースには図 5.5 に書かれているように、追加属性が 3 つあり、その属性はリードオンリーで DRNS マネージャがセットする。属性 Segments にはその時点で連結されている DRNS セグメントのオブジェクト指定子のリストが含まれている。属性 Recorder には割り当てられた DRNS レコーダのオブジェクト指定子が含まれている。属性 LockedRecipes には既存の変更要求レコードがあるレシピのレシピ識別子のリストが含まれている。

5.8 分散レシピネームスペースマネージャ – 分散レシピネームスペースマネージャ(図 5.6)は、確実に分散レシピネームスペースと DRNS セグメントが共同して作動するようにする責任、およびネームスペースが全体として損なわれないようにしておく責任がある。マネージャは自分に連結されている DRNS レコーダおよび DRNS セグメントが何であるか、および連結されている DRNS セグメントひとつひとつの内容(レシピ識別子)を常におかenneなければならない。

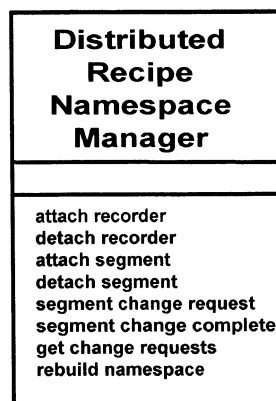


図 5.6
分散レシピネームスペースマネージャ

DRNS マネージャの現時点での連結は DRNS ネームスペース属性 Recorder および Segments の中に、5.5 項に記されている DRNS レコーダのサービスを使って維持されている。DRNS セグメントが連結される度に、DRNS マネージャはネームスペース属性 Segments および DRNS レコーダが持っている現時点のリストにセグメント指定子を追加する。DRNS セグメントの連結が解除される度に、DRNS マネージャはその指定子を Segments 属性および DRNS レコーダのリストから取り除く。

DRNS マネージャは、個々の DRNS セグメントに提供されたあるいは個々の DRNS セグメントが提供したオブジェクトサービスを使って、各 DRNS セグメントにどのレシピが保存されているのかを知ることができる。マネージャは、ネームスペースに保存されているレシピの識別子が表すのは論理レシピひとつだけである、ということを保証することによってネームスペースが損なわれないようにする。これができるのは、DRNS セグメントが保存している論理レシピ(包括的属性あるいは本体)に加える変更にはすべてマネージャがはっきりと承認を与えなければならないからである。

Change Request Record
ChangeType DestRecipeID OperationID RecipeID Segment SegmentRequest Timestamp

図 5.7
変更要求レコード

5.8.1 変更管理 — 変更要求はユーザあるいは連結されているセグメントが特定のタイプの変更をするよう、あるいは変更を許可するよう、分散レシピネームスペースマネージャに要求する度に発生する。変更要求レコード(図 5.7)には誰がどんなタイプの変更を要求したかについての情報が含まれている。この情報を使うと診断が必要になった時に診断を行える。

変更要求レコードは連結された DRNS セグメントが変更を要求する度にレシピひとつに対して生成され、要求された変更が完了したときあるいは廃棄されたときに削除される。変更要求は**インアクティブ**か**アクティブ**のどちらかである。変更要求は要求が出された後、DRNS マネージャが承認するまで**インアクティブ**であると見なされる。承認されると**アクティブ**と見なされる。

変更要求レコードは標準化されたオブジェクトではなく、公開サービスを使っても直接アクセスできない。しかし、変更要求レコードで表される情報は、インアクティブな変更要求もアクティブなものも全部、要求すれば DRNS マネージャから入手できる。そのため、変更要求レコードをオブジェクトとしてモデル化すると便利である。

変更要求が存在するレシピは**ロックされている**という。そうでないものは**ロックされていない**という。

DRNS マネージャは、どんな時でもレシピひとつにつきアクティブな変更要求は絶対にひとつしかないようにしておく責任がある。承認された変更要求の結果は、その後同じレシピに対して別の変更が承認されないうちに、同じ論理レシピのコピーを付けて、変更された DRNS セグメント以外の DRNS セグメントのひとつひとつで適切にアップデートされなければならない。

5.9 分散レシピネームスペースの構築 — 分散レシピネームスペースは数段階に分けて構築される。分散レシピネームスペースとそれに関連した DRNS マネージャは第一段階では別々に生成される。DRNS セグメントがまず個別に生成され、その後 DRNS マネージャに連結される。DRNS マネージャは、少なくとも DRNS セグメントがひとつは連結されていないとレシピを受け入れられない。なぜなら、分散レシピネームスペースはこれ以外に保存手段を持っていないからである。図 5.8 には連結の関係が描かれている。