BridgePoint® Development Suite

チュートリアル

Document Version: 1.1

PROJECT TECHNOLOGY INC.

BridgePoint® Development Suite Tutorial

本社

Project Technology, Inc.
7400 North Oracle Road, Suite 365
Tucson, AZ 85704-6342
USA

フリーダイヤル : 800/845-1489
tel: +1 520/544-2881
fax: +1 520/544-2912
email: info@projtech.com
web: www.projtech.com

国際担当系列会社

Project Technology International, Ltd.
Kendon Business Centre
27 Miller Road
Ayr KA7 2AX
UK

tel: +44 (0) 8705-673056
fax: +44 (0) 8705-673057
email: ukinfo@projtech.com

サポート

email: support@projtech.com
フリーダイヤル : 800/482-3853
tel: +1 520/544-0808
fax: +1 520/544-2912
web: www.projtech.com/support.html

英国内サポート

email:uksupport@projtech.com
voice: +44 (0) 8705-673058
fax: +44 (0) 8705-673057

目次

1

概要

1.1 BridgePoint Development Suite について

BridgePoint 製品群は、きわめて複雑な現代のソフトウェアアプリケーションに適した、統合型でモデルベースのソフトウェア開発環境を提供します。ツールスイートは xTUML をサポートしており、多くの適切なターゲット言語の 1 つにモデルを変換する前に、モデルを完全に検証できます。

特に重要なのは、ツールスイートが、分析と設計を分離するという方法論に基づいていることです。このため、得られた設計ソリューションは、アプリケーションモデルと直接関係しないソフトウェアアーキテクチャ仕様にカプセル化されます。開発者は、オープンモデル変換方式を通して、ソフトウェアアーキテクチャにアクセスしたり、変更したりできます。この方式によって、開発者は、将来のプロジェクトでソフトウェアアーキテクチャを再利用できると同時に、生成されるコードを完全に制御できます。

これらの基本機能やその他の基本機能によって、ツールスイートは、ソフトウェア開発プロセスにおける効率的で信頼性に優れた基盤になり得ます。

1.2 マニュアルセットの構成

1.2.1 The BridgePoint® Development Suite Tutorial

『BridgePoint Modeling Suite Tutorial』は、次の章で構成されています。:

第 2 章「概念の概要」では、Executable Translatable UML ($^{X}_{T}$UML) の背後にある基本概念に焦点を当てます。この開発手法に関連する利点に加えて、プロセス、表記法、分割方式、モデルの実行、および変換について説明します。

この章をお読みになると、$^{X}_{T}$UML について理解し、これが、詳細に記述する従来の方式と異なる理由を理解できます。

第 3 章「演習 1: BridgePoint Model Builder の開始および Model Builder インデックスウィンドウについて」では、$^{X}_{T}$UML を使用してのシステム開発をサポートするために作成されたツールである、BridgePoint Model Builder の基本的な構造とユーザインターフェイスについて説明します。

この章をお読みになると、BridgePoint Model Builder を開始する方法、Model Builder が管理するデータの構成方法、およびそのデータへのアクセス方法を理解できます。

第 4 章「演習 2: 既存のモデルの表示」では、BridgePoint Model Builder が備えている各種の編集機能を使って、既存のモデルのセットを参照する方法について説明します。この章では、エディタを読み取り専用モードで使用します。

この章をお読みになると、BridgePoint が提供する各種のモデリングビュー、使用できる各種のモデル編集機能、およびその編集機能に共通する外観と使い勝手を理解できます。

第 5 章「演習 3: クラス図の作成」では、スクラッチからクラス図を作成する方法について詳しく説明します。クラス図は、分析対象の問題の静的構造を表現するために使用されます。この章では、クラス、クラス属性、および関係（関連）を含む microwave oven アプリケーションのクラス図を作成します。

この章をお読みになると、UML クラス図を作成するにあたって BridgePoint Model Builder が備えているサポート機能を深く理解できます。

第 6 章「演習 4: ステートチャートの作成」は、前の章に基づきます。この章では、microwave クラス図で定義された各種クラスの動的な振る舞いを表現する、microwave oven アプリケーションのステートチャートとアクション仕様を作成する方法について説明します。

この章をお読みになると、分析対象のシステムの動的振る舞いを記述するにあたって BridgePoint Model Builder が備えているサポート機能を深く理解できます。

第 7 章「演習 5: アプリケーションモデルの検証」では、前の章で定義した microwave アプリケーションの実行を、BridgePoint Model Verifier のモデル実行機能を使って調べます。実行モデルのテストとデバッグに関する概念的なステップについて説明します。

この章をお読みになると、Executable Translatable UML ($^{X}_{T}$UML) の、"executable （実行可能）" が持つ意味を理解できます。

第 8 章「演習 6: モデルからコードへの変換」では、microwave アプリケーションモデルを、変換機能を介して実行システムに変換する方法について説明します。Project Technology の多くの標準モデルコンパイラの 1 つ (MC-2020) を使って、この変換を実行します。

この章をお読みになると、Executable Translatable UML ($^{X}_{T}$UML) における "translatable（変換可能）" 部分の実際の動作方法を理解できます。

付録 A「Brigedpoint の監査機能」では、BridgePoint Model Builder が備えている一貫性と完全性の検査機能について説明します。$^{X}_{T}$UML モデルを正しく実行または変換するためには、これらの機能によって実行される必須の検査をパスする必要があります。

この章をお読みになると、BridgePoint Model Builder の完全性と一貫性の検査について理解を深めることができます。

1.2.2 関連マニュアル

『*Tutorial*』は、マルチボリュームマニュアルセットの中の一冊です。

『*Getting Started Guide*』では、各種プラットフォームへの BridgePoint 製品群のインストール方法について説明します。また、サンプルモデルを使って、ツールの機能的および概念的な概要を説明します。付属の『*Release Notes*』には、サポートされるプラットフォームについての情報、オペレーティングシステムの要件、および前バージョンの BridgePoint ツールからアップグレードする場合の重要な情報が記載されています。

『*Tutorial*』は、BridgePoint Modeling Suite の使い方を手順を追って説明するガイドであり、Model Builder を使って $^{x}_{T}$UML の分析を実行する方法、Model Verifier シミュレーション環境でそのモデルを検証する方法、およびモデルコンパイラを使って $^{x}_{T}$UML モデルを C++ 実装に変換する方法を紹介します。

『*Reference Manual*』では、静的、状態機械、コラボレーション、モデル管理、検証ビューなど、$^{x}_{T}$UML アプリケーションの各種 " ビュー " を 1 つずつ詳しく説明します。150 を超える $^{x}_{T}$UML および BridgePoint 用語を定義する参照の節も含まれています。

『*Object Action Language Manual*』では、$^{x}_{T}$UML モデルアクションの処理を指定するための言語である、Object Action Language™ (OAL) について説明します。

『*Design Guide*』では、BridgePoint Generator ツールを使って $^{x}_{T}$UML モデルをコードに変換するプロセスについて説明します。分析モデルが配置される、元になるメタモデルは、アーキタイプ言語（配置されたメタモデルをたどってコードを作成する目的で使用される変換テンプレート言語）のまま詳細に記述されます。

『*Administrator Guide*』では、分析と設計以外の BridgePoint ツールの機能について説明します。これには、環境のカスタマイズ、ライセンス管理、インストール、ツールの基本などが含まれます。

1.3 表記規則

このマニュアルでは、次の表記規則が使用されています。字体を変えることで、文中の異なる要素に目が留まります。

`command`	示されているとおりに入力する必要があるコマンド、ファイル名、またはクラス名を表します。
`<value>`	コマンド行のパラメータやディレクトリパスなど、ユーザが指定する値を表します。
[*スタート*][*プログラム*]	Windows 環境におけるメニュー選択の順序や選択する一連のボタンを指定します。
『*マニュアル名*』	本書以外のマニュアルを参照します。
「節」を参照	本書の節を参照します。

1.4 Project Technology の連絡先

Project Technology にご連絡いただく場合は、次のいずれかの連絡先をご利用ください。

	電話	*Fax*	*E-mail*
営業	1-520-544-2881 1-800-845-1489	1-520-544-2912	sales@projtech.com
サポート	1-520-544-0808 1-800-482-3853	1-520-544-2912	support@projtech.com
英国内営業	+44(0)8705-673056	+44(0)8705-673057	ukinfo@projtech.com
英国内サポート	+44(0)8705-673058	+44(0)8705-673057	uksupport@projtech.com
日本内営業	+81-(0)3-3279-0771	+81-(0)3-3246-0645	ss_sales@toyo.co.jp
日本内サポート	+81-(0)3-3279-0771	+81-(0)3-3246-0645	ss_support@toyo.co.jp

web サイト（www.projtech.com.）にアクセスすることもできます。

2

概念の概要

2.1 はじめに

Executable Translatable UML ($^{x}_{T}$UML) は、リアルタイムの組み込みテクニカルソフトウェアシステムの開発を促進します。UML 表記法を利用した $^{x}_{T}$UML は、多くの実績を持ち、明確に定義されている、完全に自動化された方法論です。

$^{x}_{T}$UML はオブジェクト指向の手法に基づいており、これまでに 1400 件を超えるリアルタイムのテクニカルプロジェクトで使用されています。これらのプロジェクトには、生命に関わる移植用医療機器、米国防総省の飛行システム、年中無休で稼動するパフォーマンス重視のフォールトトレラントな通信システム、リソースが厳しく制限される一般消費者向けの電子機器、大規模な分散型の離散型事象のシミュレーションシステムなどが含まれます。

$^{x}_{T}$UML では、アプリケーションとソフトウェアアーキテクチャ設計の完全分離を中心にして、いくつかの基本概念を利用しています（図 2-1）。

- アプリケーションモデルは、アプリケーションの動作を、明確に、かつ正確に表します。これらのモデルは実行可能な形式であり、アプリケーション要件を早期に検証できます。アプリケーションモデルは、設計と実装の詳細にまったく依存しません。
- 設計パターン、設計規則、および実装テクノロジに関して定義されるソフトウェアアーキテクチャは、ターゲットシステムに適したコードを生成するトランスレータに組み込まれています。ソフトウェアアーキテクチャは、これがサポートするアプリケーションにまったく依存しません。
- トランスレータは、アプリケーションモデルを、適切な設計規則と設計パターンにマップすることによって、モデル化コンポーネントに対応する 100% 完全なコードを生成します。

xTUML および Project Technology 社の自動化ツールの直接的な利点として、プロジェクトスケジュールの短縮、ソフトウェア不良率の減少、システムパフォーマンスの最適化、きわめてコンパクトなコードなどが挙げられます。xTUML を使用すると、リリースが多数あったり、ターゲットや製品群が多かったり変化したりするような製品の開発を促進することもできます。xTUML を使用することで、価格、マーケットに参入するまでの期間、製品の品質の点で、顧客、契約、マーケットシェアを、より積極的に、かつ効果的に獲得できます。

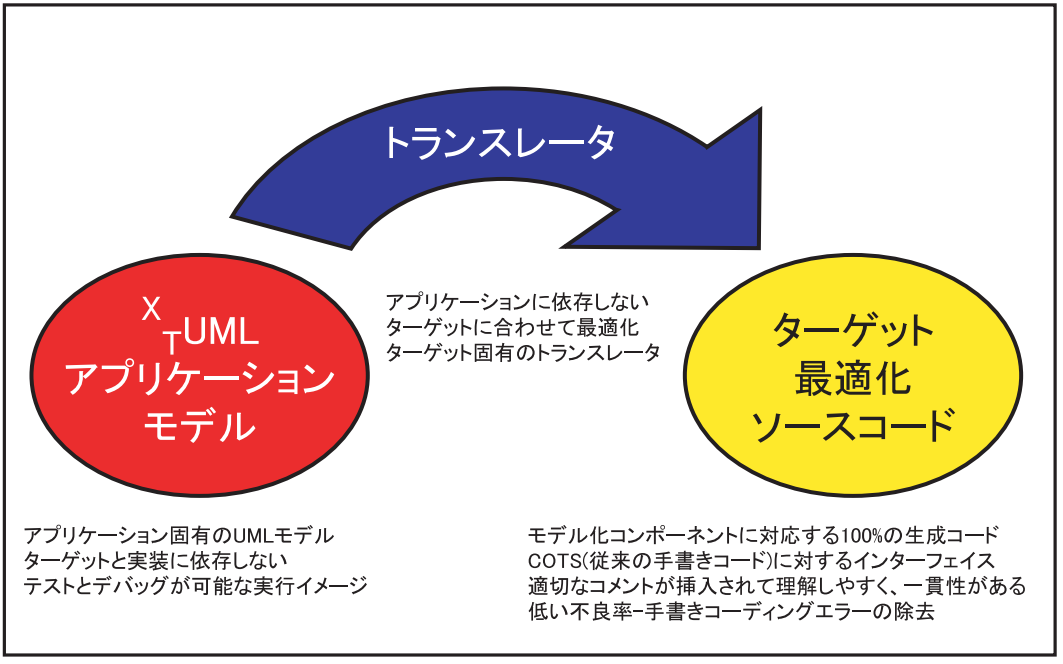

図 2-1: Executable Translatable UML による開発

2.2 Executable Translatable UML の基本

$^{x}_{T}$UML とその効果を理解するにあたって欠くことができない、いくつかの概念と特徴があります。以降の節では、これらの基本概念の概要を説明します。

- $^{x}_{T}$UML の開発プロセス
- $^{x}_{T}$UML の表記法
- プロジェクト分割
- Executable UML モデル
- 変換
- 統合、テスト、および保守

2.2.1 $^{x}_{T}$UML の開発プロセス

図2-1 に示されているアプリケーションモデルとトランスレータ（ソフトウェアアーキテクチャ設計）は、図 2-2 に示されている効率的で確実なプロセスを使って構成されます。

ソフトウェアアーキテクチャとアプリケーションが完全に分離されることによって、ソフトウェアアーキテクチャ設計とアプリケーション開発を並行して進めることができます。これらのアクティビティを順番に処理する他のプロセスと比べて、この並行性によって開発のライフサイクルが縮小され、プロジェクトスケジュールが短縮されます。

$^{x}_{T}$UML モデリングアクティビティは、各アクティビティに明白な開始ゲートと終了ゲートを指定することで、明確に定義されます。最終ゲートは、完成したアプリケーションモデルが実行するという条件であり、紛れもなく明白です。この明白な$^{x}_{T}$UML 終了ゲートによって、不完全なシステム定義や過剰なオーバーモデリング（" 解析の麻痺状態 " とも呼ばれます）といった問題が発生する可能性が取り除かれます。

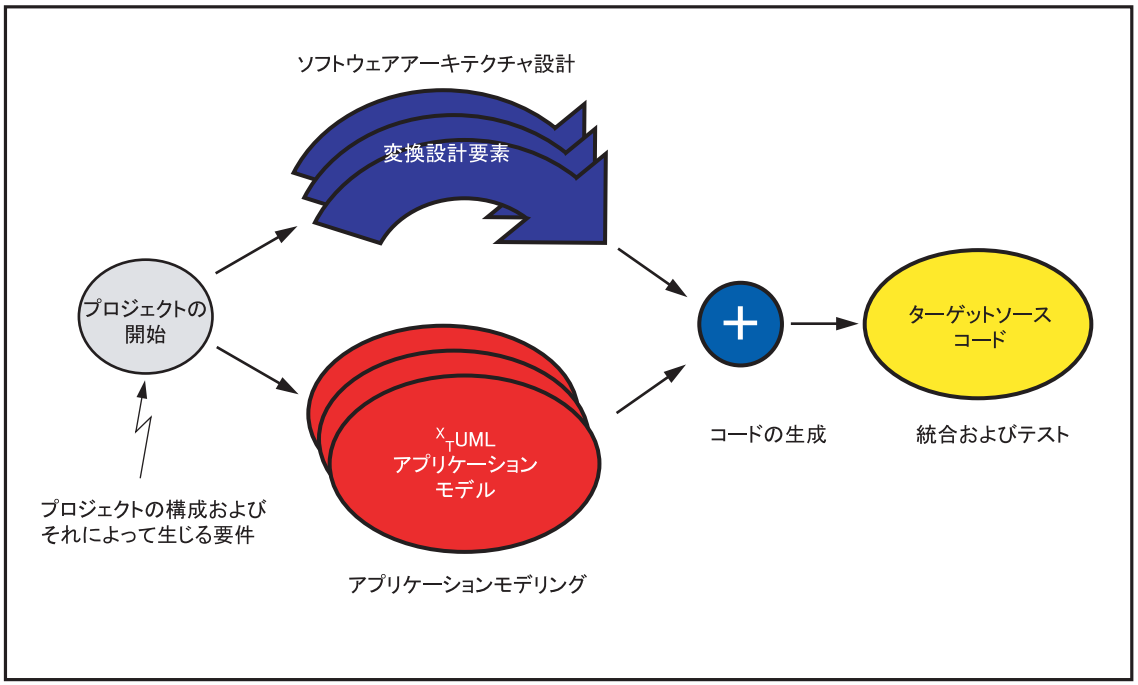

図 2-2: $^{x}_{T}$UML の開発プロセス

2.2.2 $^{x}_{T}$UML の表記法

UML (Unified Modeling Language™、統一モデリング言語）は、多数のモデリング構造体で構成される優れた表記法のセットです。開発者がこの広範な構造体の集合を理解するには、かなりの時間が必要です。これに加えて、何通りもの方法で同じ概念を表すことができます。このため、新規プロジェクトの開発チームは、最初に十分時間をかけて議論し、合意した上で条件に該当する UML サブセットを選択し、チームの方法論とモデリングプロセスを、選択したこの UML サブセットにマップする必要があります。

$^{x}_{T}$UML は、実行ベースおよび変換ベースの開発のニーズを、効率的かつ効果的にサポートできるように入念に選択された、合理的な UML のサブセットで構成されています。このサブセットには、クラス図、ステート図、およびプロシージャ仕様という 3 つの主要コンポーネントが含まれています。その他のダイアグラム（コラボレーション図など）は、3 つの主要コンポーネントで表現する情報から自動的に生

成されます。生成されるこれらのダイアグラムによって、モデル特性の総合的なビューが完成し、効果的な開発が促進されます。xTUML において事前に定義されている方法論および合理的な UML サブセットによって、UML サブセットを選択し、そのサブセットを方法論にマップするために必要な時間が省かれます。また、UML 表記法の習得に関する学習曲線も最小化されます。

2.2.3 プロジェクト分割

大きな問題は、これをもっと小さくて管理しやすい大きさに分割し、その小さな問題を個別に解決することで、解決しやすくなります。この分割ストラテジによって、大規模で複雑なプロジェクトにとって必要なスケーラビリティが実現されます。xTUML の場合、問題の分割は、問題領域の範囲（ドメイン）で発生します。このため、各分割部を、その問題領域に関連する語彙を使って個別に調査、検討、記述、およびモデル化できます。この分割手法には、次の利点もあります。

- プロジェクトチームおよび作業を構成できる、自然な分岐点の識別
- 相互に依存しない、大規模で再利用可能な分割部の定義

きわめて大きな問題領域は、もっと小さくて管理しやすい大きさ（サブシステム）に分割します。サブシステム分割によって、大きなプロジェクトであっても、その問題領域の解析とモデリングが簡単化され、効果的なチームのサイズを保つことができます。

2.2.4 Executable UML モデル

xTUML アプリケーションモデルには、設計や実装とは無関係に、アプリケーションモデルの確認と検証を実行するために必要な詳細情報が含まれています。正規のテストケースをモデルに対して実行し、アプリケーション要件が適切に満たされていることを確認できます。モデルを実行するために、設計の詳細情報やコードを開発したり追加したりする必要はありません。モデルは、タイミング関係を確認するために入念に定義された、タイミング規則のフレームワークの中で動作します。これによって、アプリケーションモデルは、設計とは無関係にテストできるだけでなく、モデルを実行するために必要な手間と時間を合理化し、簡単化できます。アプリケーションモデルの実行によって、わずかな手間とコストで早期にシステムエラーが取り除かれ、紛れもなく明確な終了ゲート、つまりモデルの終了条件が作成されます。

2.2.5 変換

$^{x}_{T}$UML の性能の大部分は、アプリケーションモデルを、ターゲットで実行可能な 100% 完全な最適化コードを自動的に変換できる機能に基づきます。

トランスレータは、3 つの部分で構成されています（図 2-3）。

- 設計パターンと変換規則（典型）のセット。変換規則では、1) コード生成時に適用されるパターンのセット、2) 指定のパターンを使用する条件、3) 設計パターンからコードを作成する場合の $^{x}_{T}$UML モデルコンポーネントの設定方法または利用方法を記述します。
- 変換エンジン。$^{x}_{T}$UML アプリケーションモデル情報を取得し、設計パターンおよび規則を理解し、モデルコンポーネントを設計パターンにマップして完全なコードを生成します。
- ランタイムライブラリ。生成されるコードモジュールをサポートするプリコンパイルルーチンで構成されます。

トランスレータが 3 つの部分に分かれていることによって、トランスレータのカスタマイズ、構成、および保守が能率的に行われます。変換エンジン自体の詳細情報やコードに対処することなく、設計パターン、変換規則、またはランタイムライブラリを変更したり追加したりできます。

トランスレータは、コード生成時に、$^{x}_{T}$UML アプリケーションモデルから情報を取得します。次に、変換するモデル要素に適した設計パターンを選択します。このモデルから取得された情報は、選択した設計パターンの " 空白を埋める " ために使用されます。この結果、完全にコード化されたモデル要素が得られます。

この単純な手法は、現実のアプリケーションでは実に強力となります。パターンを設定すると、通常は、別のパターンと規則の呼び出しが要求されます。場合によっては、新しく呼び出されたこのパターンと規則が、さらに別のパターンと規則を呼び出します。1 つのモデル要素のように見えても、コードを作成すると、最終的には複数のモデル要素に対応するパターンと規則が何層にもネストすることがあります。トランスレータによって、この作業が完全に自動化されます。

システムのパフォーマンスや制約の要件を満たすために、ソフトウェアプロジェクトでは、指定のモデル要素タイプに対して複数のパターンを実装することが、場合によっては要求されます。このような状況では、実装オプションごとに 1 つのパターンを定義できます。特定のモデル要素に対して使用されるデフォルトパターンを無効にすることで、特別な状況における最適なパターンを選択できます。

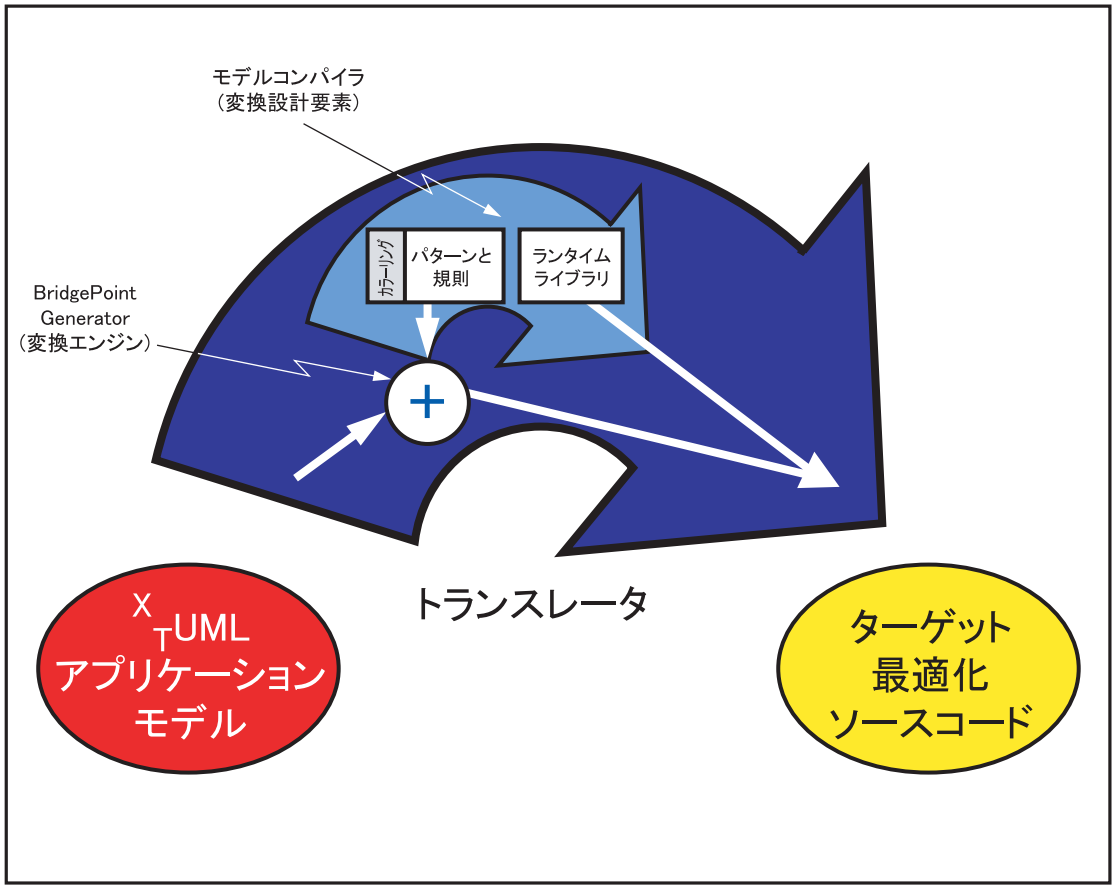

図 2-3: xTUML トランスレータ

このパターン選択は、カラーリングと呼ばれるプロセスを通して指定します。モデルカラーリングによって、特定のシステムの実装特性とパフォーマンスについて、詳細な付加の制御機能が設計エンジニアに与えられます。

xTUML の標準トランスレータを使用することで、モデル化システムを、コンパクトで理解しやすく、かつ適切に記述された、パフォーマンスに優れたソースコードに、自動的に 100% 変換できます。従来の手書きで記述され外部で生成されるコードの COTS は、トランスレータが生成するコードに簡単に統合できます。これらのトランスレータでは、パフォーマンスやサイズの最適化に関するオプションなど、各種の標準事前定義のカラーリングオプションをサポートしています。これらは、オープンでカスタマイズ可能な設計パターンおよび規則に基づきます。標準トランスレータが備えている、設計パターンおよび規則のオープンな特性によって、プロジェ

クト固有のニーズをサポートするための変更を簡単に加えることができます。

2.2.6 統合、テスト、および保守

xTUML では、ライフサイクルの早期において不良を取り除くことで、統合とテストに要する時間と手間を大きく減らしています。アプリケーションと設計のテストが独立していることに加えて、xTUML の不良削減チェックポイントによって、統合とテストの前に不良を取り除くことができます。この詳細については、「不良の削減」で説明します。

xTUML では、不良の修正も簡単化されており、新たな不良がシステムに間違って発生する可能性は最小限で済みます。設計や実装のエラーが、不適切なトランスレータのパターンや規則が原因である場合、エラーは頻繁に発生し、きわめて明らかです。このようなエラーについては、不良を引き起こしている特定のトランスレータ要素を簡単に追跡できます(Project Technology 社の標準トランスレータの場合、コードヘッダーには、生成元の特定のモデル要素が示されており、これが、不適切なトランスレータ要素を直接表しています)。不適切なトランスレータ要素を修正すると、システムが生成するコードすべてに、その修正内容が自動的に反映されます。

アプリケーションモデルのエラーは、不良なシステムの振る舞いを引き起こします。アプリケーションモデル要素と生成コードは直接マップされているため、このような不良が見つかった場合、問題の原因である特定のアプリケーションモデル要素は直ちに判明します(Project Technology 社の標準トランスレータを使用している場合、不良コードのヘッダーには、生成元の特定のモデル要素が含まれています)。不良なアプリケーションモデル要素を修正することで、エラーが解決します。

すべての変更と修正は、アプリケーションモデルとトランスレータのいずれかに対して行います。修正は不良の原因に対してのみ行われるので、1 つの問題を修正することで、新たな問題が発生することはまずありません。手書きのコーディングによる修正で副作用が生じる危険性は取り除かれます。

システムの統合、テスト、および保守に対する xTUML の手法では、長期のサポート、継続的な不良修正、機能の追加、およびパフォーマンスチューニングを行う効率的な方法が提供されています。xTUML の場合、これらのアクティビティによって、ますますもろいコードが生じることはありません。アプリケーションモデルやソフトウェアアーキテクチャ設計のレベルで行われる修正や変更は、常に、明白で簡潔であり、適切に記述および構成されたコードを生成します。コードレベルの修正だと生じることがある、複雑な構造や同期がとれていない記述がされることはありません。

2.3 Executable Translatable UML の影響

xTUML は、多数の反復や構成をサポートするプロジェクトに大きな恩恵をもたらします。これには、リリースが多数あるプロジェクト、要件が発展していくプロジェクト、最新の保守が必要なプロジェクト、製品ラインを持つプロジェクト、製品群を持つプロジェクト、ターゲットが移行するプロジェクト、プラットフォームが多数あるプロジェクトが含まれます。xTUML は、パフォーマンス重視のシステムやリソースに制約があるシステムの開発者にとっても大きな価値があります。

以降の節では、これらの分野における xTUML の影響と効果について説明します。具体的には、次の内容について説明します。

- プロジェクトの立ち上げ
- 反復型アプリケーション開発
- 複数アプリケーション開発
- パフォーマンスの最適化と設計の調整
- ターゲットの移行
- 不良の削減
- 設計とアプリケーションの再利用

2.3.1 プロジェクトの立ち上げ

迅速な立ち上げが、プロジェクト成功の鍵をしばしば握ります。自動化されている xTUML 手法の各種機能によって、迅速で生産的な立ち上げが実現されます（図 2-4）。

- xTUML の能率的な UML のサブセットによって、UML 表記法の学習曲線が大きく削減されます。
- 事前に定義されている方法論と UML サブセットによって、UML サブセットを選択および合意し、そのサブセットを方法論にマップするために必要な時間が省かれます。
- アプリケーションモデルを実行してテストおよび確認するにあたって、設計の詳細情報、実装の詳細情報、またはターゲットコードは必要あり

ません。アプリケーションの機能的な正確さは、実装、統合、およびテストに先立ち、ライフサイクルの早期に簡単に検証できます。

- トランスレータのテストとアプリケーションモデルの実行を、早期に、かつ互いに関係なく実行できることによって、テスト部門はその作業を開発ライフサイクルの早期に開始できます。
- 汎用のトランスレータや以前に開発したトランスレータを、そのまま利用したり、カラーリングオプションを指定して調整したり、設計やトランスレータを変更する上での開始点として使用したりして、最適なターゲットソースコードを生成できます。これによって、プロジェクトでは、ニーズに適したソフトウェアアーキテクチャ設計をすばやく定義したり、再利用したりできます。

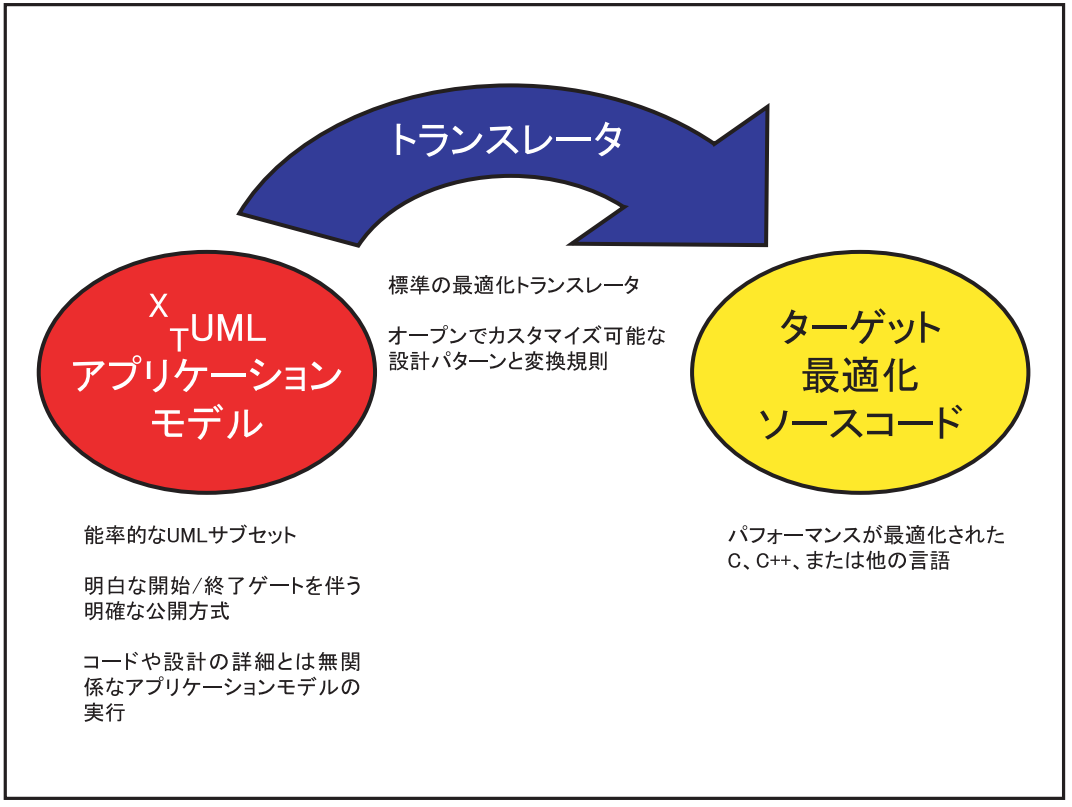

図 2-4: 迅速なプロジェクトの立ち上げを目的とする $^{X}_{T}$UML サポート

2.3.2 反復型アプリケーション開発

xTUML ではアプリケーションと設計が完全に分離されていることで、反復型の開発が大きく促進され、簡単化されます。アプリケーションモデルに加えられた変更は、設計やトランスレータに変更を及ぼすことなく、新しいアプリケーションコードに自動的に変換されます（図 2-5）。

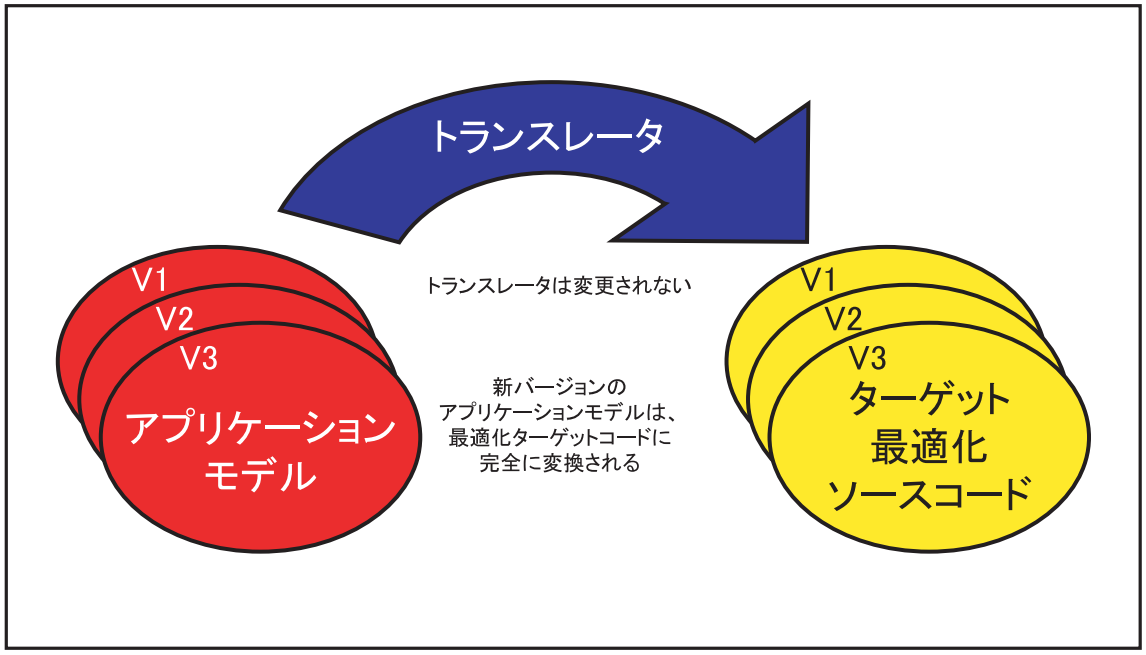

図 2-5: xTUML の反復型アプリケーション開発

2.3.3 複数アプリケーション開発

xTUML では、1 つのソフトウェアアーキテクチャ設計を実装する必要は 1 回だけなので、共通または類似のターゲットプラットフォームに対して複数のアプリケーションを開発することが、大きく促進されます。追加のアプリケーションモデルを開発すると、その新しいアプリケーションに対応するターゲット最適化コードに自動的に変換されます。このとき、既存のトランスレータを変更する必要はありません（図 2-6）。

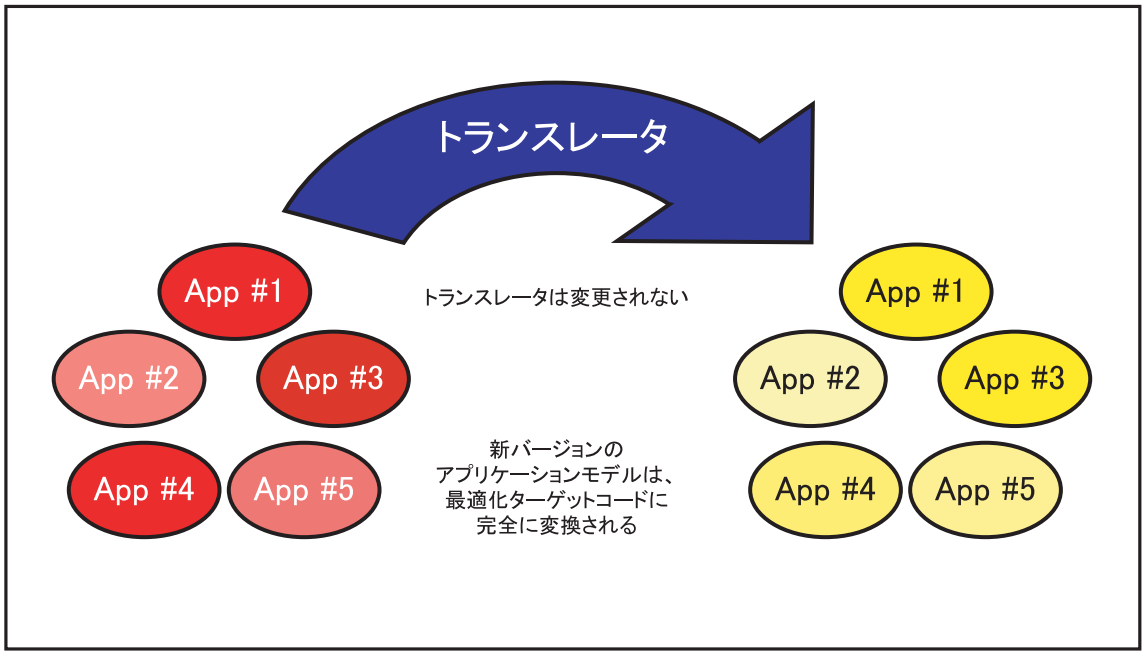

図 2-6: $^{x}_{T}$UML の複数アプリケーション開発

2.3.4 パフォーマンスの最適化と設計の調整

開発システムでパフォーマンスや制約の問題が発生した場合、コードを再作成したりアプリケーションモデルを修正することなく、問題を調整できます。$^{x}_{T}$UML 手法では、ある 1 つの場所で、システムパフォーマンスや制約特性を調整できます。その 1 つの場所とは、トランスレータです。アプリケーションモデルのサイズ、使用しているアプリケーションモデルの数、使用しているコードの量、または生成されるコードの量に一切関係なく、トランスレータでの調整が可能です。トランスレータに加えられる調整は、システムとそのすべてのコード全体に、自動的に反映されます（図 2-7）。

トランスレータに取得される設計パターンおよび規則によって、そのトランスレータが生成するソフトウェアのパフォーマンス特性が決まります。たとえば、$^{x}_{T}$UML のステートチャート図はさまざまな方法で実装できますが、その中の 2 つを以下に示します。

- 定義されているすべての遷移をリンクリストに配置する方法。リスト内の各ノードは、現在の状態、受信イベント、および遷移先の状態を表します。
- 遷移を 2 次元配列に配置する方法。1 つのインデックスは現在の状態に対応し、もう 1 つは受信イベントに対応します。配列のエントリは、指定されている現在の状態と受信イベントに対応して発生する遷移を表します。

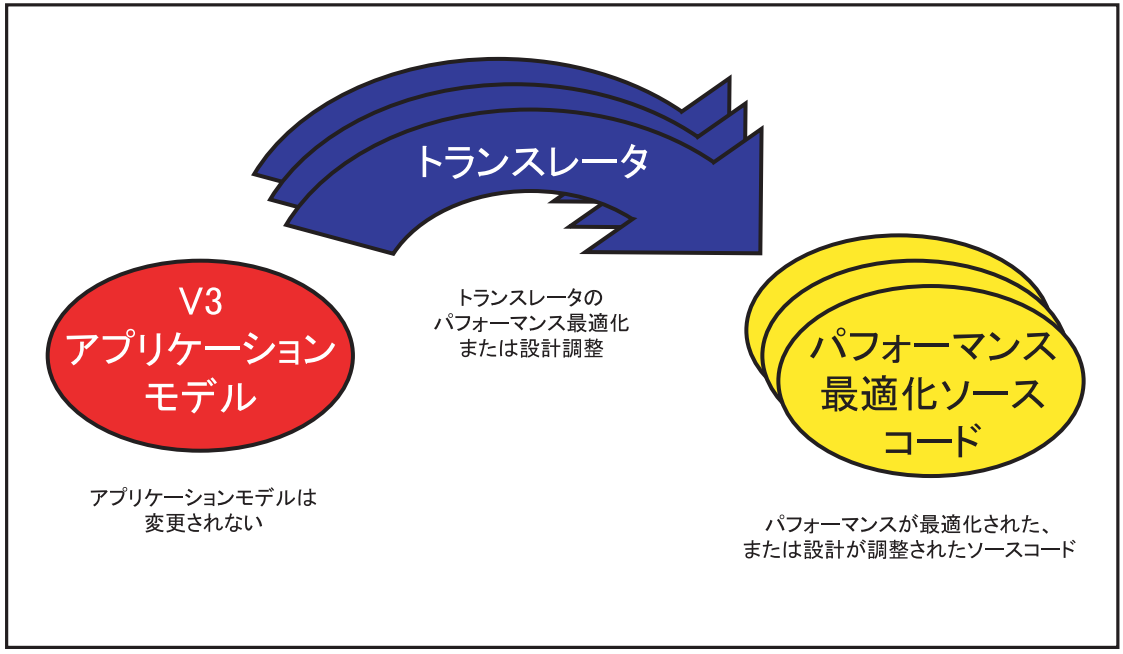

図 2-7: $^{x}_{T}$UML におけるパフォーマンスの最適化と設計の調整

2 次元配列を使用する後者の実装方法は、状態の変化が、きわめて高速な一定時間で計算されるために、前者の方法に比べて高速です。ただし、この方法の欠点は、配列のセルに生じる可能性がある空の容量です。速度よりも容量の方が重要な場合は、最初の方法の方が適切です。

場合によっては、あるモデル要素タイプについて、パフォーマンスを最大化したり特定の制約を満たしたりするために、いくつもの実装方法の候補が必要です。このような状況では、必要な実装オプションごとに 1 つのパターンが定義されます。「変換」で説明したように、設計エンジニアは、カラーリングを使って、特定のモデル要素および状況で使用する最適なパターンを指定できます。既存の要件や要件の変更に基づいて、トランスレータに新しい設計パターンおよび規則（典型）を追加して、パフォーマンスを向上したり、特定のシステム制約を満たすことができます。

アプリケーションモデルがすでに構築され、コードが生成されていても、カラーリングを使用すると、システムパフォーマンスと実装方法を、効果的に、かつきわめて詳細に、付加的に制御することができます。トランスレータは、コードとアプリケーションモデルとは無関係であるため、パフォーマンスの最適化が、コードの " ホットスポット " を再作成したり、システムを再設計するプロセスである必要はもはやありません。

2.3.5 ターゲットの移行

新しいターゲットに移行する場合、アプリケーションモデルの変更や修正は必要ありません。設計と実装のすべての問題はトランスレータに集約されるため、プラットフォームの移行に要するのは、既存のトランスレータの変更だけです。または、新しいターゲットが根本的に異なる場合は、新しいトランスレータの取得や作成が必要です（図 2-8）。どちらの場合も、アプリケーションモデルの変更は必要ありません。

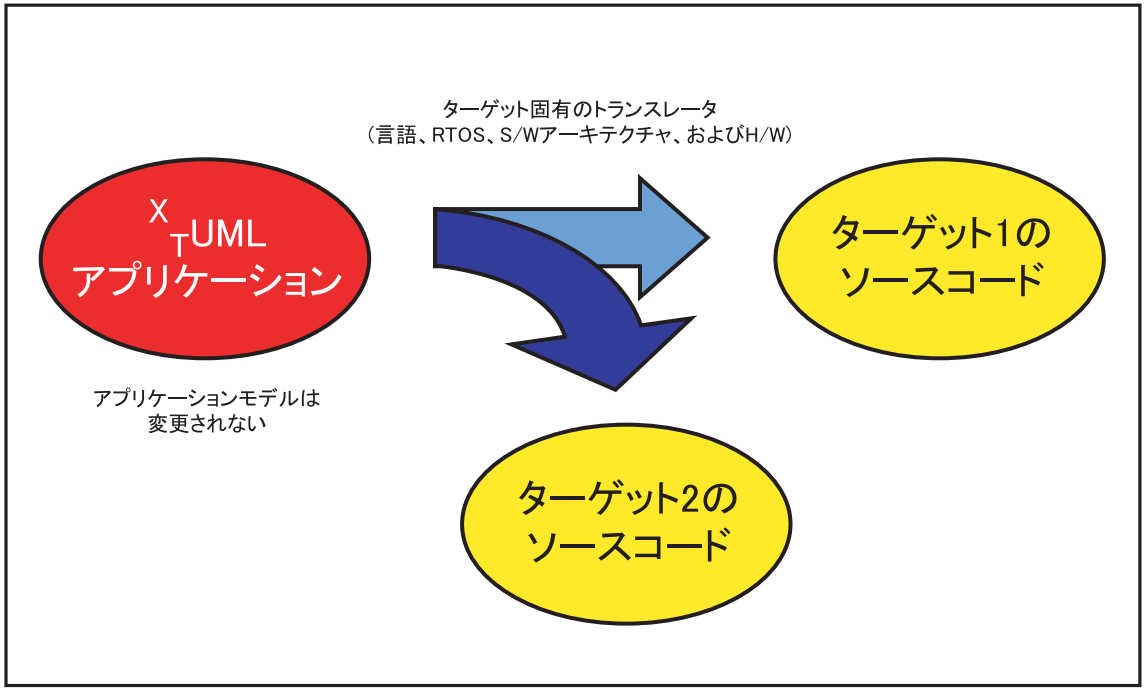

図 2-8: ターゲットの移行

2.3.6 不良の削減

xTUML を使用すると不良率は劇的に減少し、場合によっては桁の単位で異なります（10 倍以上）。この利点は、プロジェクトが大きくなればなるほど、そして複雑になればなるほど明らかです。

xTUML アプリケーションモデルは、設計やコードとは無関係に実行およびテストできるため、統合やテストのかなり前に、不良を取り除くことができます。xTUML には、不良箇所を識別して除去できる、多数のチェックポイントも組み込まれています（図 2-9）。

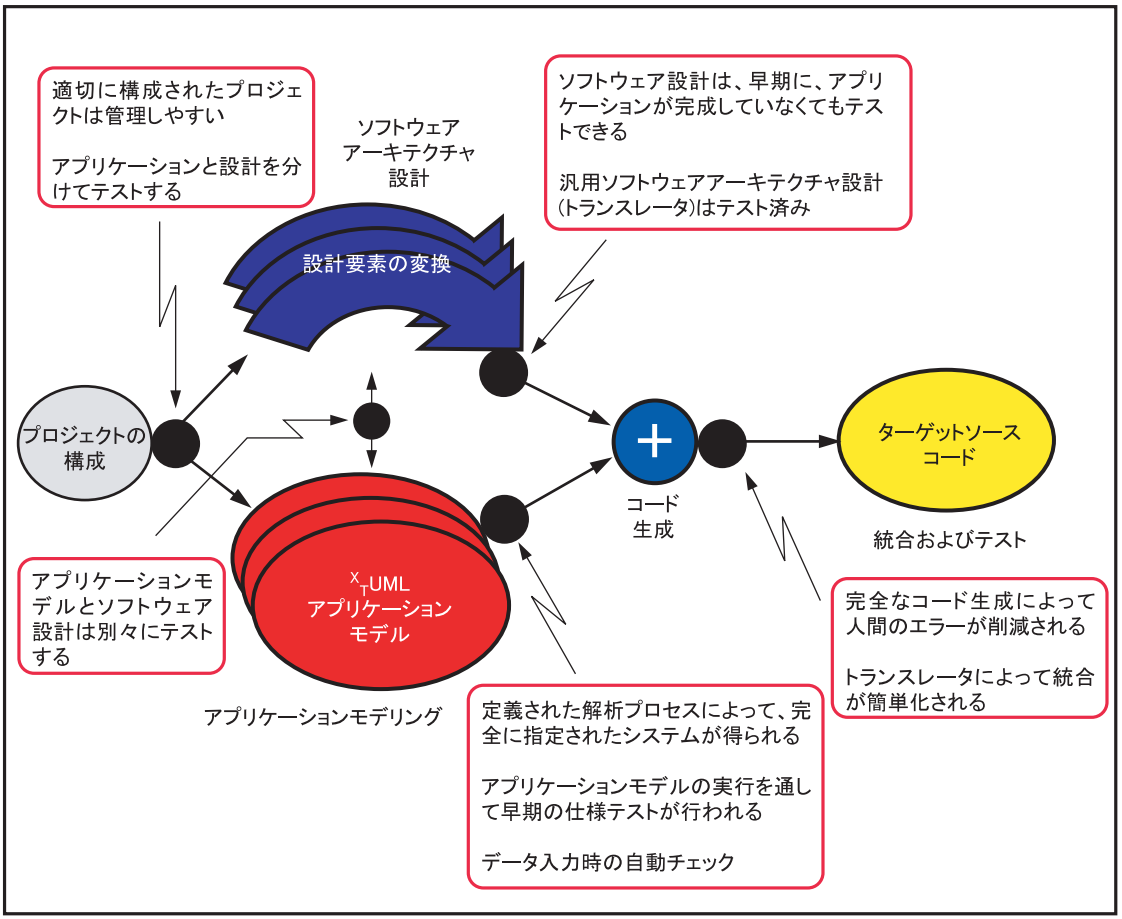

図 2-9: 不良削減ポイント

アプリケーションと設計を個別および独立してテストできることで、これらの要素を早期および継続してテストすることが、促進および簡単化されます。テスト部門は、より早い段階からもっと効果的に関わることができます。

アプリケーションロジックのエラーは、アプリケーションを表す $^{x}_{T}$UML モデルに限定されます。モデルの実行後、設計や実装の詳細とは無関係に、正規のモデルテストを介して、アプリケーションのロジックと取り決めの不良を検出し、除去します。トランスレータに組み込まれている設計パターンおよび規則を確認するには、標準のトランスレータテストスイートを使用し、アプリケーションとは無関係に設計と実装の不良を識別および除去します。アプリケーションモデルから 100% 完全なコードが自動的に生成されるため、" 手書きのコーディング " によるすべての不良は除去されます。

2.3.7 設計およびアプリケーションの再利用

図 2-10 は、再利用可能なアプリケーションモデルと再利用可能なトランスレータ (埋め込まれているソフトウェアアーキテクチャ設計を含みます) を強調した、大規模な再利用の全体像を示しています。この手法によって、企業は、これまでの開発の成果を企業の資産として活用できます。これによって、将来の開発工程が促進されるだけでなく、そのコストも大きく削減されます。テスト済みのアプリケーション、アプリケーションコンポーネント、および設計を再利用することで、品質も向上します。企業は、価格、マーケットに参入するまでの期間、製品の品質の点で、顧客、契約、マーケットシェアを、より積極的に、かつ効果的に獲得できます。

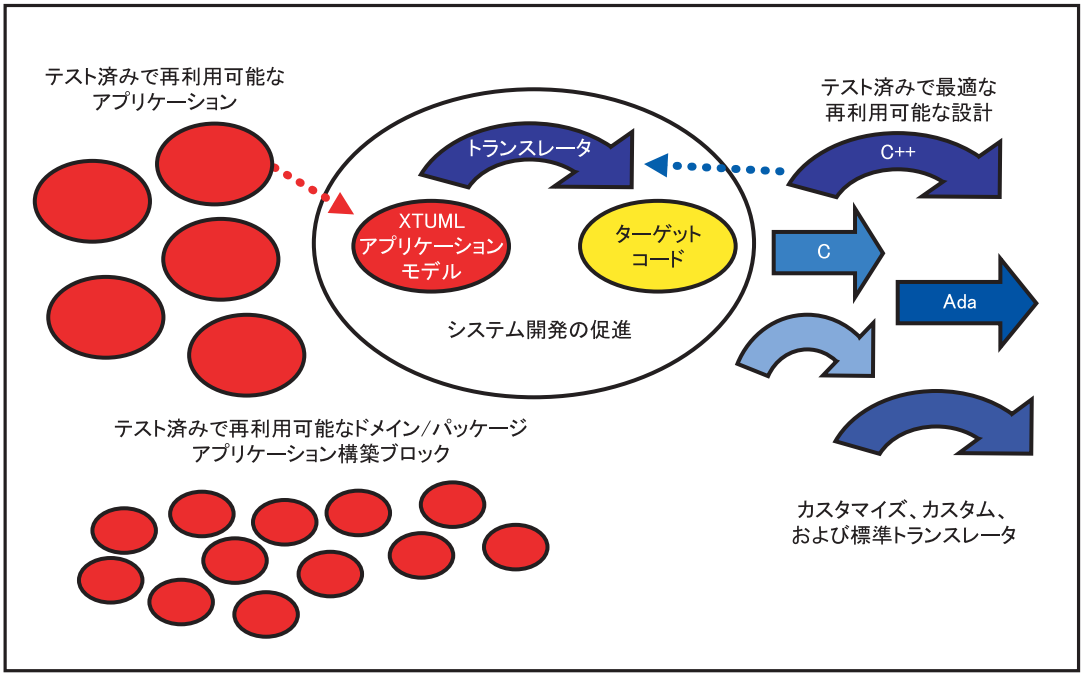

図 2-10: アプリケーションおよび設計の再利用

2.4 $^{x}_{T}$UML のまとめ

$^{x}_{T}$UML は、リアルタイムの組み込みテクニカルシステムの開発を促進するだけでなく、品質、パフォーマンス、およびリソースの活用を向上できる、これまでにない機能を備えています。

$^{x}_{T}$UMLではソフトウェアアーキテクチャ設計とアプリケーションモデルが分離されており、設計とアプリケーション開発を並行して進めることができます。$^{x}_{T}$UML 変換では、アプリケーションモデルを設計にマップすることによって、完全なターゲットコードが自動的に生成されます。システムのターゲットコードは、アプリケーションモデルとソフトウェアアーキテクチャ設計を、常に正確に反映します。

$^{x}_{T}$UML は、要件が頻繁に変わったり、要件が増えていくプロジェクトに対応するために必要な、反復的開発手法を促進します。アプリケーションの機能は、設計を変更したり、ターゲットコードを作成し直すことなく、追加または変更できます。実装と設計を変更するには、変換規則とパターンの修正だけが必要です。$^{x}_{T}$UML を使用すると、リリースが多数あり、要件の変更があり、ターゲットが多数あるか発展する可能性があり、場合によっては最新の保守も必要なプロダクトの開発を促進することができます。

設計とアプリケーションを分離することによって、ソフトウェアアーキテクチャ設計、アプリケーションモデル、およびアプリケーション分割（ドメイン）を効率よく再利用できます。アプリケーションは、ターゲットの特性、実装の詳細、または言語に関係なく再利用可能です。設計は複数のシステムにわたって再利用可能であり、各システムのアプリケーションの内容に依存しません。複数のプロダクトにわたっての再利用は、実用性と優れた生産性に適しています。

パフォーマンスチューニングは、トランスレータの規則とパターンを変更または選択（カラーリング）することによって実現されます。トランスレータに加えられた変更は、システムのサイズや生成されたコードの量に関わらず、すばやく、そして自動的に、システムコード全体に反映されます。$^{x}_{T}$UML を使用すると、パフォーマンス要件を満たすためにローカルコードを再作成する必要性や、システムを再開発するリスクが解消されます。

$^{x}_{T}$UML では、能率化された UML のサブセット、明確に定義された方法、およびアプリケーションモデルの早期検証を利用することで、プロジェクトを迅速に立ち上げることができます。プロジェクトでは、汎用の、または社内で開発したソフトウェアアーキテクチャ設計（トランスレータ）をそのまま利用したり、新しい設計のベー

スとして利用したりできます。これによってプロジェクトは設計が促進され、完全なターゲット最適化コードがアプリケーションモデルから直接、すばやく生成されます。

xTUML によって、不良率は劇的に減少します。変換を使用しているので、手書きのコーディングによる不良が発生することはありません。xTUML プロセス全体を通して行われる品質のチェックポイントに加えて、アプリケーションと設計に対して早期の個別テストが行われることによっても、不良は取り除かれます。

不良箇所はすばやく追跡され、その原因であるソフトウェアアーキテクチャ設計またはアプリケーションモデルにおいて修正されます。コードレベルで修正する必要はないので、新たなエラーが間違って発生する可能性は最小限で済みます。コードは、設計やアプリケーションモデルに加えられた最新の修正を常に反映しており、コードレベルで修正すると生じる複雑な構造が含まれることもありません。システムは増大しても、その保守が時間と共に困難になることはありません。システムは、生成されたコードの量、アプリケーションのサイズ、およびこれまでに実行された保守の範囲に関わらず、効率的に増大と発展を続けることができます。既存のシステムは、保守しなければならないものではなく、活用し再利用できる企業の資産となり得るのです。

xTUML は多くの実績を持ち、効果をいかんなく発揮しています。400 万行を超える C++ から成る大規模な通信システムで使用されている一方、メモリ量にすればわずか数キロバイトの移植用医療機器でも使用されています。xTUML は、パフォーマンスが重視されるシステム、飛行用システム、安全対策システム、生命に関わるシステム、リソースに制約があるシステム、およびコストに制約があるシステムで、その効果を大いに発揮しています。

3

演習 1: BridgePoint Model Builder の開始および Model Builder インデックスウィンドウについて

注意 : このチュートリアルのすべての演習は、BridgePoint がシステムにインストールされていることを前提としています。インストールがまだ済んでいない場合は、チュートリアルを続ける前にまずインストールしてください。

BridgePoint Model Builder は、実行可能形式の $^{X}_{T}$UML モデルを作成するためのツールです。このようにして作成したモデルは、BridgePoint Model Verifier を使ってシミュレートおよびデバッグでき、BridgePoint Generator 変換エンジンを使って自動的にコードに変換できます。指定のドメインに対するこのモデルには、ドメインパッケージ図、クラス図、ステートチャート、およびアクション記述が含まれます。他のモデル（パッケージ依存図 - 関係、パッケージ依存図 - 非同期、パッケージ依存図 - 同期、オブジェクトコラボレーション図 - 非同期、オブジェクトコラボレーション図 - 同期、状態遷移テーブル）は、分析するシステムをさらに深く理解するために、これらの基本モデルから自動的に作成されます。

この演習では、次の内容を学習します。

- BridgePoint Model Builder を開始する方法
- BridgePoint がリポジトリとワークスペースに情報を構成する方法
- 既存のリポジトリから新しいワークスペースを作成し、それを Model Builder にロードする方法
- Model Builder インデックスウィンドウの構成方法およびウィンドウで使用できる機能
- BridgePoint Model Builder を終了する方法

3.1 BridgePoint Model Builder の開始

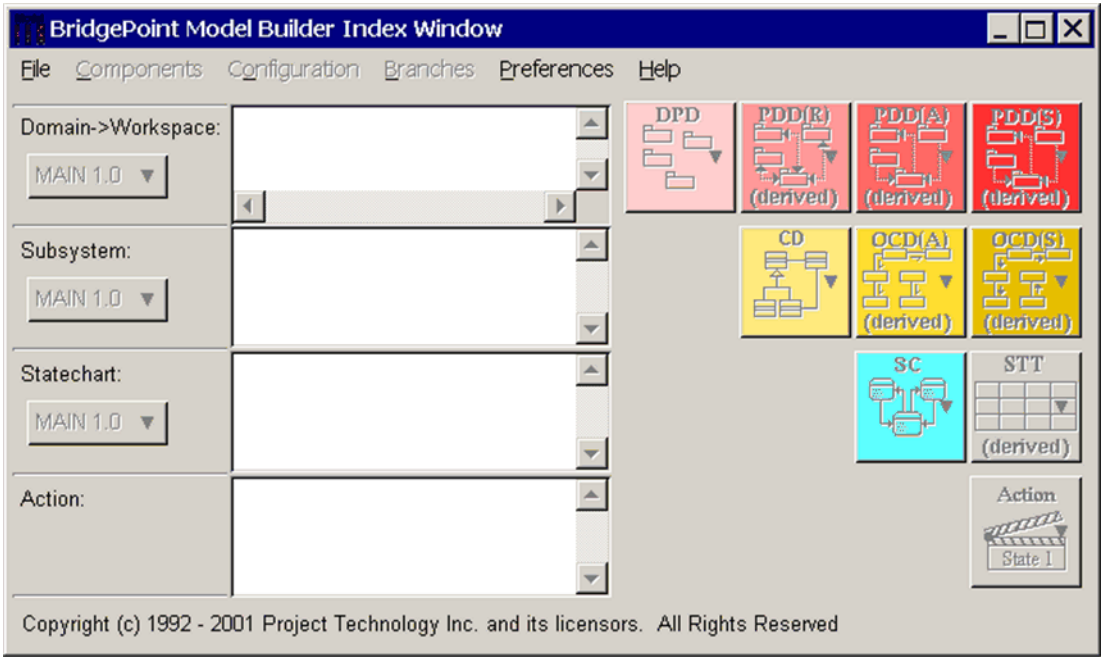

図 3-1: BridgePoint Model Builder インデックスウィンドウ

- BridgePoint Model Builder を開始する方法は、使用しているプラットフォームによって異なります。Windows 環境の場合は、タスクバーの [*スタート*] ボタンを使って Model Builder を開始します。

  [*スタート*]|[*プログラム*]|[*BridgePoint 6.0*]|[*Model Builder*]

  Unix 環境の場合は、コマンド行で次を入力して Model Builder を開始します。

  ```
  pt_builder
  ```

  注意 : BridgePoint `bin` ディレクトリがパスにある必要があります。BridgePoint のインストール位置については、システム管理者に確認してください。

図 3-1 の Model Builder インデックスウィンドウが表示されます。インデックスウィンドウは、作成するモデルの静的プロパティの設定とチェックに関する、すべてのアクティビティの開始点です。この演習の目的は、この初期ウィンドウの機能と、このウィンドウで実行できる操作について理解することです。

3.2 BridgePoint のリポジトリとモデルワークスペース

BridgePoint がモデルとデータをどのように保存し、どのようにアクセスするかを理解することは、ツールを効果的に使用する上で重要です。データの格納メカニズムには、リポジトリとモデルワークスペースの 2 つがあります。リポジトリは、多くの点で、パブリックライブラリに似ています。あるドメインに関連するモデルを表示するためには、最初に、そのドメイン情報をリポジトリから取得し、モデルワークスペース（場合によってはプライベートモデルワークスペースと呼ばれます）に配置する必要があります。

モデル要素の読み取り専用バージョン（自分が表示しているときに、他のユーザも、その要素に関する情報を同時に表示できます）を要求することも、編集可能バージョン（自分だけがアクセスできます）を要求することもできます。ある要素は読み取り専用として表示し、他の要素は編集可能としてチェックアウトできます。編集可能要素への変更が終了し、これをチェックしてリポジトリに戻すと、新しいバージョン番号が割り当てられ、他のユーザが使用できるようになります。

3.3 BridgePoint とドメインモデリング

BridgePoint では、複数のドメインの情報を、モデルワークスペースにロードできます。これによって、目的のドメインに関連する詳細情報に加えて、ドメイン間に存在する関係も定義できます。

3.4 BridgePoint へのドメインのロード

この演習では、microwave oven を記述するドメインを Model Builder にロードすることによって、このドメインに関連するモデルを調べます。この時点では、ドメインに関連するモデルを調べるだけなので、モデル要素のチェックアウト操作は必要ありません。

この時点では、プライベートなモデルワークスペースは定義されていません。このため、microwave ドメインモデルセットがロードされるワークスペースを作成する必要があります。

`[File]|[New]`

[*Create New Model Workspace*] ダイアログボックス（図 3-2）が表示されます。この図を見ると、パーソナルワークスペースに関連付けるファイルの位置を指定できることがわかります。ここではチュートリアルを簡単に進めるために、デフォルト値をそのまま使用します。

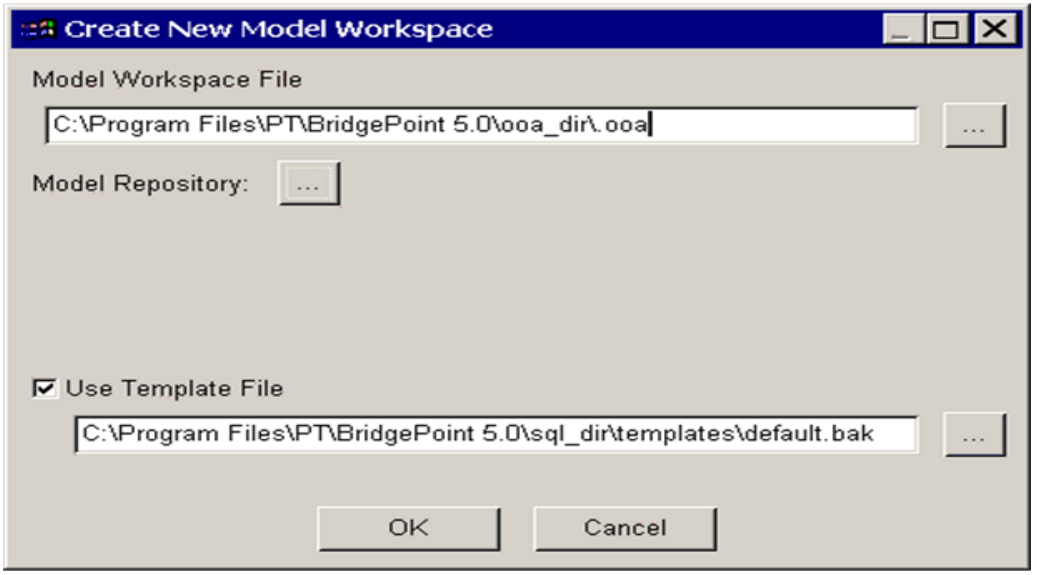

図 3-2: [Create New Model Workspace] ダイアログボックス

- [*Model Repository*] ラベルの横の [...] ボタンをクリックします。[*Select Domain Name*] ダイアログボックス（図 3-3）が表示されます。このダイアログボックスには既存のリポジトリの一覧が表示され、ここから microwave ドメインを選択できます。ドメイン名をダブルクリックするか、ドメイン名を 1 回クリックして、[*OK*] ボタンをクリックします。

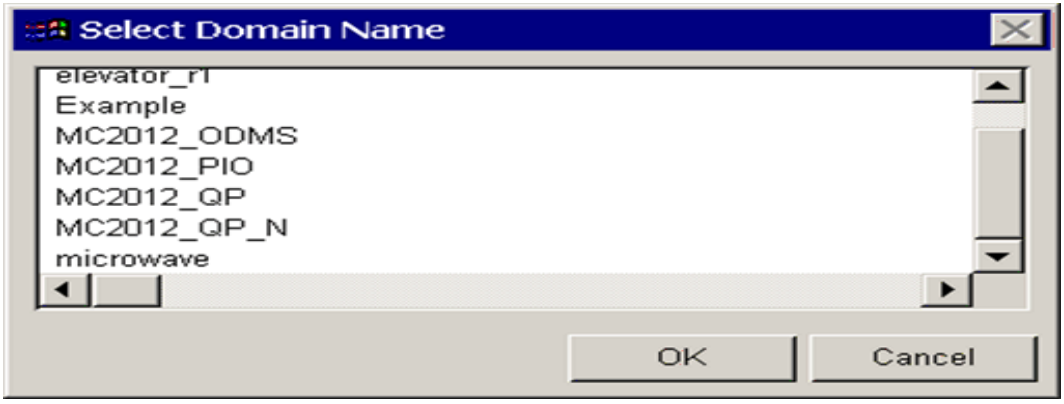

図 3-3: [Select Domain Name] ダイアログボックス

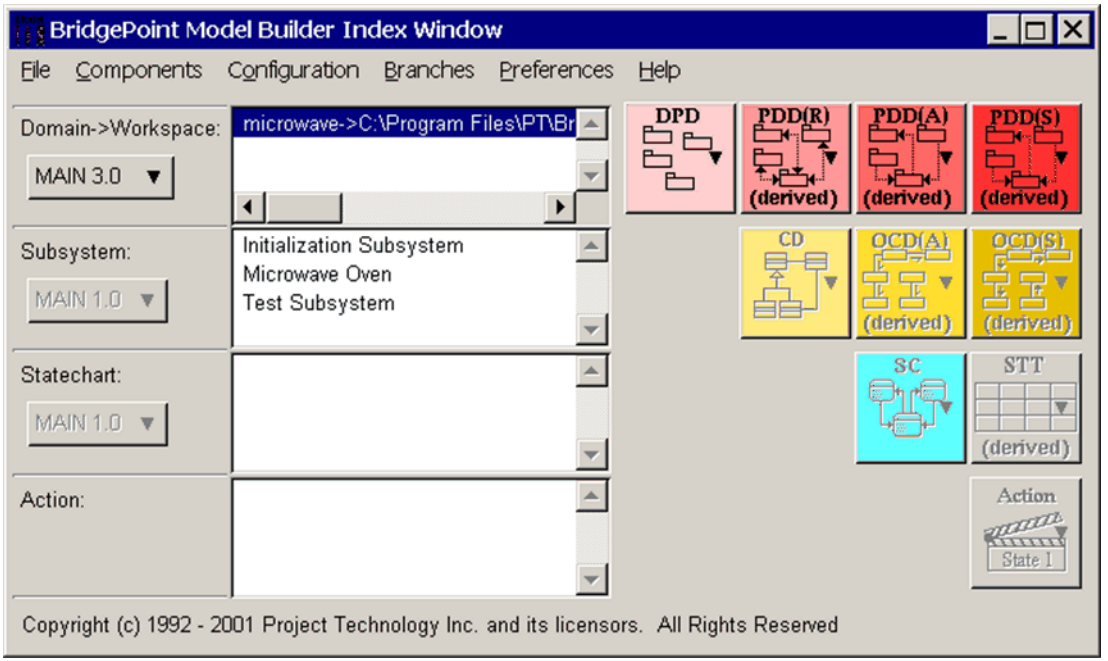

図 3-4: ドメインがロードされた状態の BridgePoint Model Builder インデックスウィンドウ

- [*Create New Model Workspace*] ダイアログボックスの [*OK*] ボタンをクリックします。図 3-4 のインデックスウィンドウが表示されます。

- Model Builder インデックスウィンドウは、1)[*Domain->Workspace*]、2)[*Subsystem*]、3)[*Statechart*]、および 4)[*Action*] という 4 つのセクションに分かれています。各セクションは横方向に配列されており、セクションラベル（例:[*Domain->Workspace*]）、チェックイン操作とチェックアウト操作にアクセスするためのボタン、選択できる項目が表示されるスクロールボックス、および選択した項目に対応するモデル（ダイアグラム）を表示するボタンのセットが関連付けられています。

  [*Domain->Workspace*] セクションには、現在、1 つのドメインだけがロードされていることに注意してください（microwave ドメイン）。このセクションのスクロールボックスで項目を選択すると、ウィンドウの右側にあるダイアグラムアクセスボタンがアクティブになり、ドメインに関連付けられているドメインサブシステムが [*Subsystem*] セクションに表示されます。表示されるサブシステム

の 1 つを選択すると、ウィンドウの右側にあるダイアグラムアクセスボタンがアクティブになり、そのサブシステムに関連付けられているステートチャートのリストが [*Statechart*] セクションに表示されます。ステートチャートを選択すると、ウィンドウの右側のダイアグラムアクセスボタンがアクティブになり、ステートチャートに関連付けられているアクションのリストが[*Action*]セクションに表示されます。アクションを選択すると、[*Action*] アクセスボタンがアクティブになります。

☐ スクロールボックス内で別の要素を選択し、隣接する下のウィンドウに表示される情報とアクティブ化されるダイアグラムアクセスボタンに注意してください。これらの間にある関係を理解するまで、この操作を続けます。ダイアグラムアクセスボタンはまだクリックしないでください（これについては次の演習で説明します)。

☐ BridgePoint Model Builder を終了します。

`[File]|[Quit]`

4

演習 2: 既存のモデルの表示

4.1 既存のモデルワークスペースを開く

Model Builder インデックスウィンドウについて理解できたところで、既存のモデルをいくつか見てみましょう。この演習では、BridgePoint Model Builder ツールがサポートする基本モデルと、これらのモデルを作成するときに使用する編集機能について説明します。表示するモデルは読み取り専用であるために、エディタは使用できません。

- BridgePoint Model Builder を開始します。
- 新しいワークスペースを作成するのではなく、前の演習で作成した既存のワークスペースを開きます。

  [*File*]|[*Open*]

  [*Open OOA DB File*] ダイアログボックス（図 4-1）が表示されます。[microwave.ooa] を選択して [*Open*] をクリックするか、[microwave.ooa] をダブルクリックすると、Model Builder にモデルワークスペースがロードされます。既存のモデルの表示を開始できる状態になります。
- [*Domain->Workspace*] スクロールボックスの右側にある 4 つのダイアグラムアクセスボタンに注意してください。

  このボタンを使用すると、microwave ドメインの最上位レベルに関連するダイアグラムにアクセスできます。アクセスできるダイアグラムは、ドメインパッケージ図 (DPD)、パッケージ依存図（関係）PDD(R)、パッケージ依存図（非同期）PDD(A)、およびパッケージ依存図（同期）PDD(S) です。

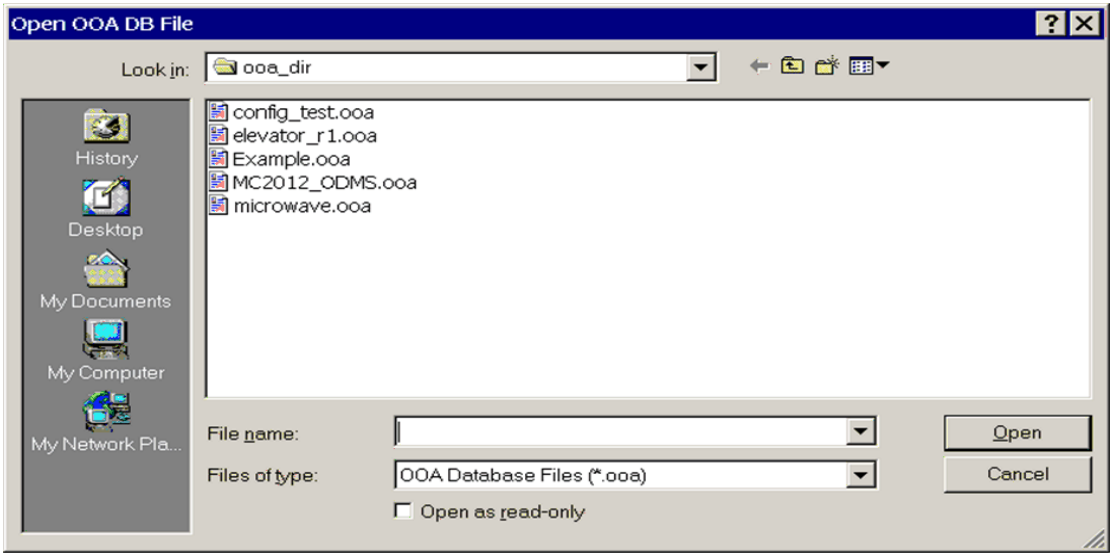

図 4-1: [*Open OOA DB File*] ダイアログボックス

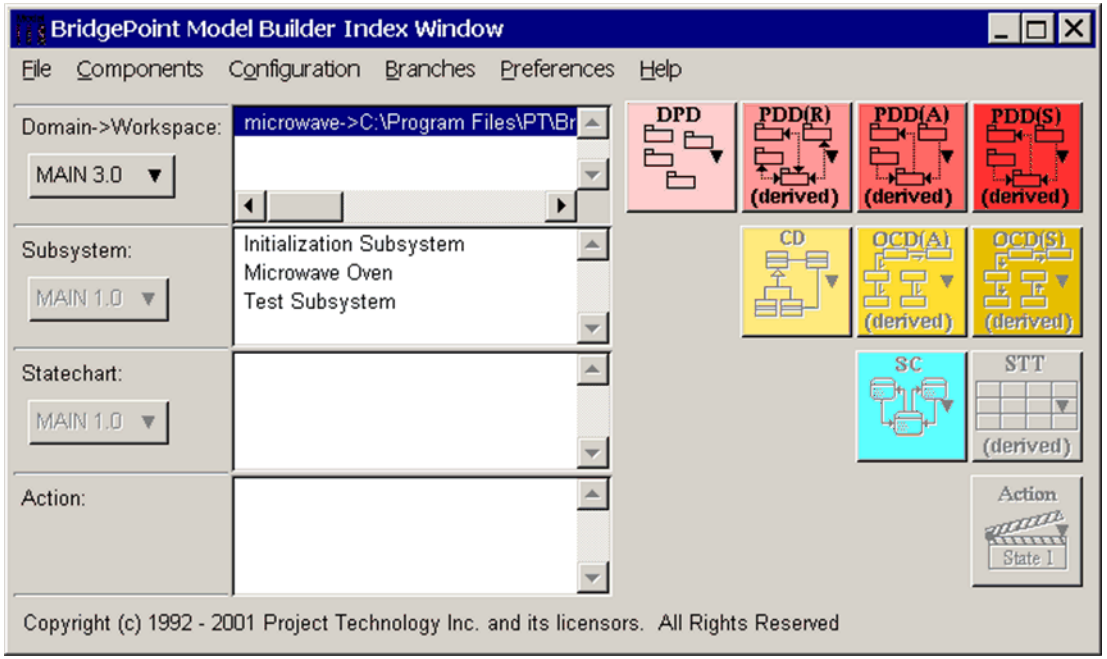

図 4-2: ドメインがロードされた状態の Model Builder インデックスウィンドウ

4.2 既存のドメインパッケージ図の表示

- [*DPD*] というラベルのボタンをクリックします。ドロップダウンメニューが表示されます。[*Show*] を選択します。ドメインパッケージ図が別のウィンドウに表示されます（図 4-3）。このダイアグラムでは、microwave ドメインを構成するすべてのサブシステムパッケージと、これらのサブシステムが関係を持つ外部エンティティパッケージ（他のドメイン）が図示されます。

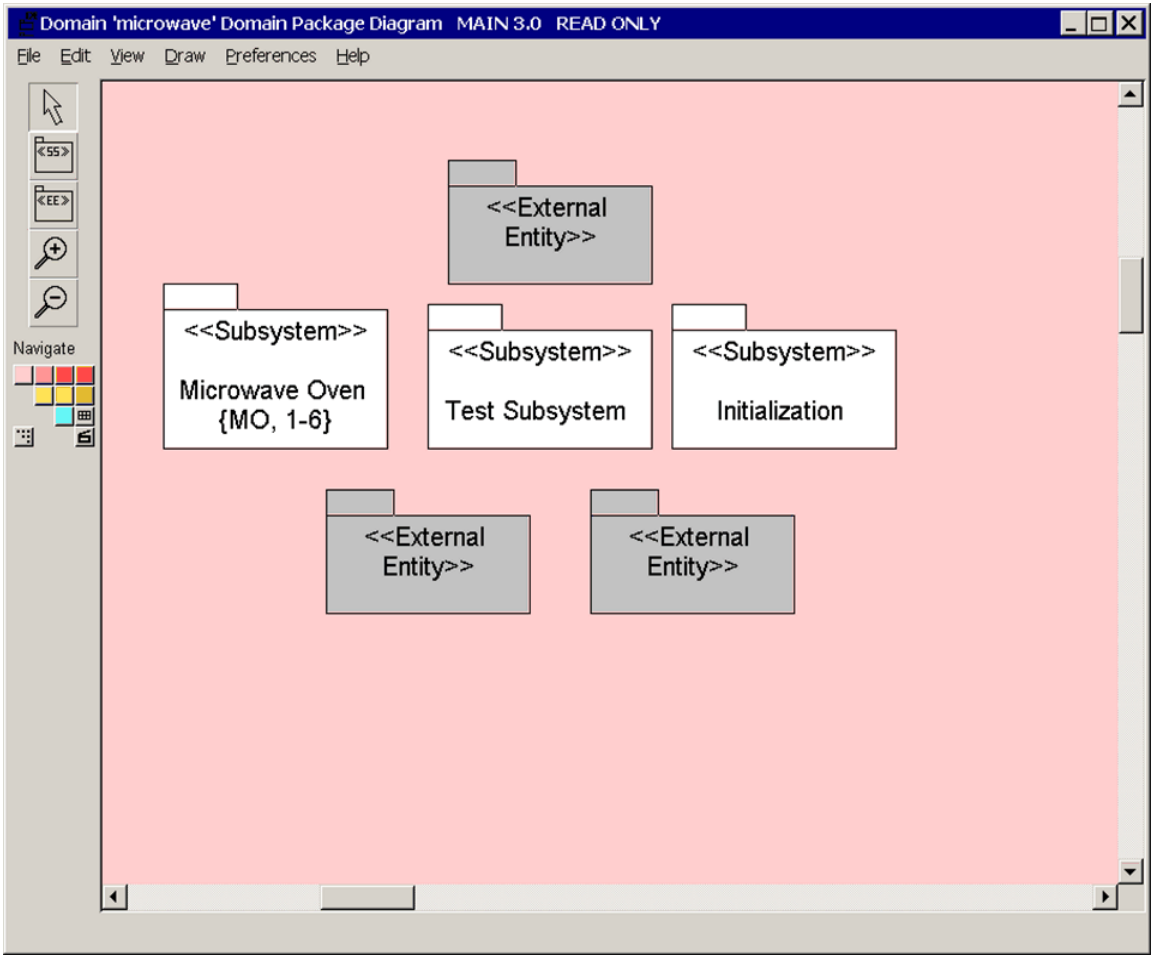

図 4-3: Microwave Oven の DPD （ドメインパッケージ図）

ドメインパッケージ図には、ドメインを分割して得られる各種のパッケージ（サブシステム）が表示されます。小さなドメインの場合は、パッケージが 1 つだけということもあります。大きなドメインの場合は、複数のパッケージが表示される可能性が高いでしょう。ドメインをいくつのサブシステムパッケージに分割するかを判断する基準は、何人のアナリストが、互いの作業に関わらずに 1 つのドメインについて作業できるか、という問題に大きく関わっています。これまでの経験から、この値は 3 ないし 4 付近の値です。

サブシステムパッケージは、データの関連性、使用依存性、通信パスのいずれかで強く関係するクラスの固まりで構成することが望まれます。ドメイン分析はまだ行われていないため、一般には、これらの要素は明確に定義されていません。そのため、ドメイン分割は、しばしば、プロジェクトのライフサイクルの初期に行われるブレーンストーミングセッションで生じます。分析が終了した後のドメイン分割は、クラスをどのように分けるかについて、経験に基づく推量の結果です。

4.3 ダイアグラム編集ウィンドウの外観と使い勝手

このチュートリアルの演習を進めていく中で、モデル編集ウィンドウの外観と使い勝手がどれも似ていることに注意してください。現在表示しているウィンドウには、編集ウィンドウの一般的な機能が示されています。

ドメインパッケージ図、パッケージ依存図（関係）、パッケージ依存図（非同期）、パッケージ依存図（同期）、クラス図、オブジェクトコラボレーション図（非同期）、オブジェクトコラボレーション図（同期）、およびステートチャートの編集ウィンドウは一連のボタン（描画ツール）を備えており、モデルのバージョンが編集用にチェックアウトされるとアクセスできます。ただし、読み取り専用の場合は使用できません。このダイアグラムタイプに関連する描画ツールは、[*SS*]（サブシステム）と [*EE*]（外部エンティティ）の 2 つだけです。この描画ツールは、モデル図を作成および変更する、この後の演習で使用します。

- 描画ツールの下には、虫眼鏡が描かれた 2 つのボタンで表される拡大ツールがあり、これは、ダイアグラムを拡大または縮小するメカニズムに相当します。いずれかのボタンをクリックし、表示されるカーソルをダイアグラムの上に置き、クリックおよびドラッグすると、現在の拡大レベルを変更できます。選択するボタンに応じて、ダイアグラムのサイズは大きくなるか、または小さくなります。拡大ツールをしばらく操作して、ダイアグラムのサイズを変更する方法、および特定のダイアグラムコンポーネントを表示の中心にする方法を学習します。

- 拡大ツールの下には、一連のボタンがあり（Model Builder インデックスウィンドウの場合と同じように配置されています）、これは、ナビゲーション機能に相当します。このナビゲーション機能を使用すると、上記のどのモデル編集ウィンドウからでも、関連するモデル（またはインデックスウィンドウ）にアクセスできます。

  現在、ダイアグラムのコンポーネントが選択されていない場合は、同じレベルまたは上のレベルにあるモデルコンポーネントにだけナビゲートできます。すぐ下のレベルへのナビゲーションは、該当する要素が選択されている場合のみ可能です。たとえば、現在、クラス図を操作していてクラスを選択すると、そのクラスのステートチャートにナビゲートできます。

  別のダイアグラムタイプに移動するには、ナビゲーションボタンを使用します。この時点では、どのダイアグラムを選択しても問題ありません。しばらくの間は、ツールの操作手順を学習することが目的に過ぎません。ナビゲートする各ダイアグラムに対応して、個々にウィンドウが開きます。下位レベルのダイアグラムに移動するには、ナビゲートを試みる前に、現在のダイアグラムにおいて項目を選択する必要があることを忘れないでください。アクティビティによって開いたウィンドウは、[*File*]|[*Close Editor*] の順に選択することによって、または編集ウィンドウの右上角にある [*X*] をクリックすることによって、個々に閉じることができます。

- Model Builder インデックスウィンドウに戻ります。現在開いているすべてのダイアグラム編集ウィンドウを閉じます。

4.4 既存のクラス図の表示

- [*Subsystem*]スクロールボックスの[Microwave Oven]サブシステムを選択します。ウィンドウの右側にあるダイアグラムアクセスボタンがアクティブになることに注意してください。

- クラス図を表す [*CD*] ボタンを選択します。ドロップダウンメニューが表示されます。メニューの [*Show*] を選択します。Microwave サブシステムに関連する $^{X}_{T}$UML クラス図が表示されます（図 4-4）。

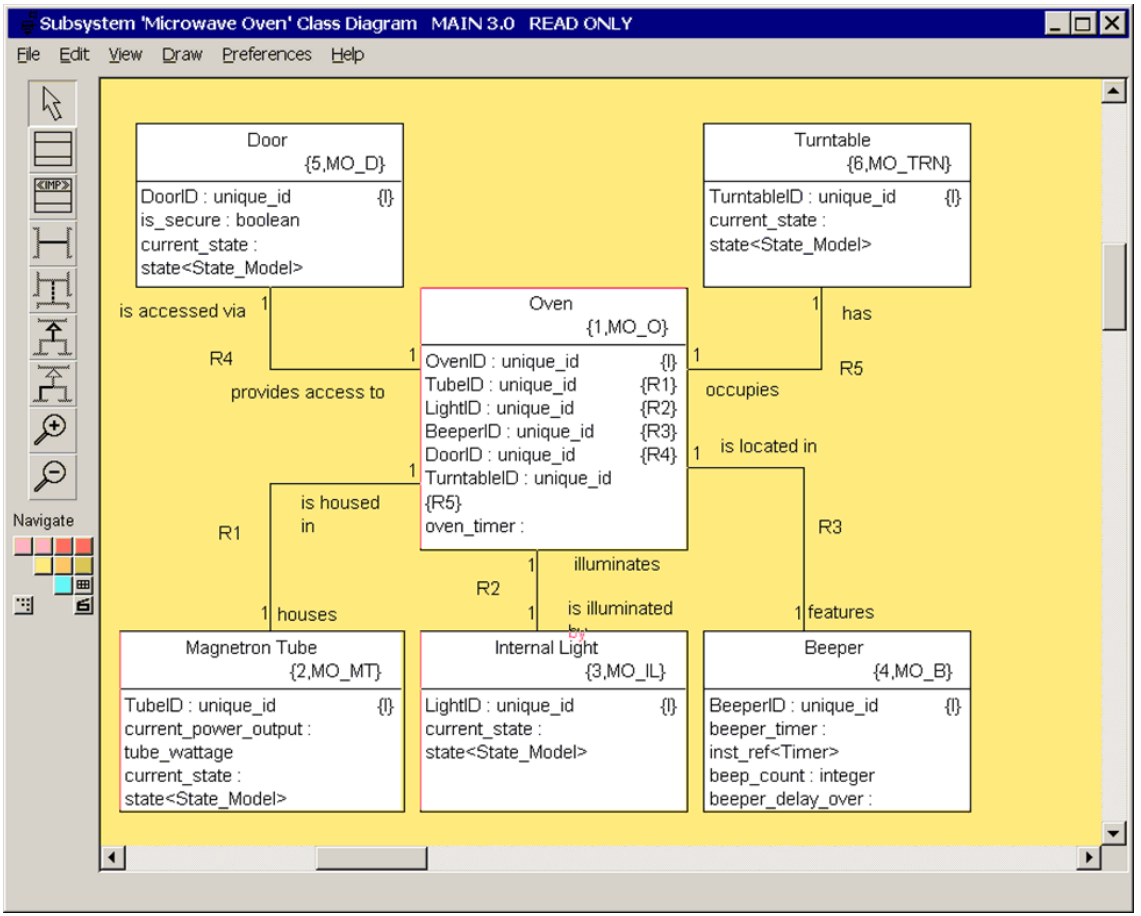

図 4-4: Microwave Oven クラス図

クラス図は、システムにおけるクラスのタイプと、そのクラス間に存在する静的な関係を記述するものです。クラス図は、すべてではないにしても、既存の大部分のオブジェクト指向手法の中心にあたります。ダイアグラム自体に直接表示されないこともありますが、クラス図には、実に多くの情報が関連付けられていることを意識することが重要です。この節では、BridgePoint Model Builder が、ダイアグラムと関連情報の両方をどのように表すかについて、簡単に説明します。クラス図の詳細については、『*Executable UML: A Foundation for Model-Driven Architecture*』（Stephen Mellor [1] 著）や『*UML Distille*』（Martin Fowler [2] 著）などの書籍を参照してください。

ダイアグラムは現在読み取り専用であるにも関わらず、このダイアグラムタイプの場合は、いくつかの描画ツールが使用可能であることに注意してください。これらのツールを使用すると、クラス図にクラスを追加したり、異なるクラス間に関係を作成したりできます。これらのツールについては、後で行うダイアグラム編集の演習の際に説明します。

- [Oven] クラスの上にカーソルを置き、1 回クリックして、クラスを選択します。
- 右クリックします。選択したクラスに対して実行できる操作のポップアップメニューが表示されます。
- [*Show Class Description*] をクリックします。Oven クラスのすべての属性、クラスの一意な識別子、およびテキスト記述が新しいウィンドウに表示されます（図 4-5）。テキスト記述は、モデルの実行やコードの生成にとって必須ではありませんが、分析のドキュメントという点できわめて重要です。この記述は、モデルを参照したり検討したりする人達に、分析にクラスが含まれる理由と、分析しているシステムにおけるその実際のロールに関する情報を提供します。意味のあるクラス記述の構成について、多くの書籍があります[3],[4]。

1. Stephen Mellor、Marc Balcer 著『Executable UML: A Foundation for Model-Driven Architecture』Reading, MA: Addison-Wesley 発行（2002 年）
2. M. Fowler、K. Scott 著『UML Distilled』Reading, MA: Addison-Wesley 発行（2000 年）
3. Leon Starr 著『How to Build Shlaer-Mellor Object Models』Upper Saddle River, NJ: Yourdon Press 発行（1996 年）
4. S. Shlaer、S. Mellor 著『Object-Oriented Systems Analysis: Modeling the World in Data』Englewood Cliffs: NJ: Yourdon Press 発行（1988 年）

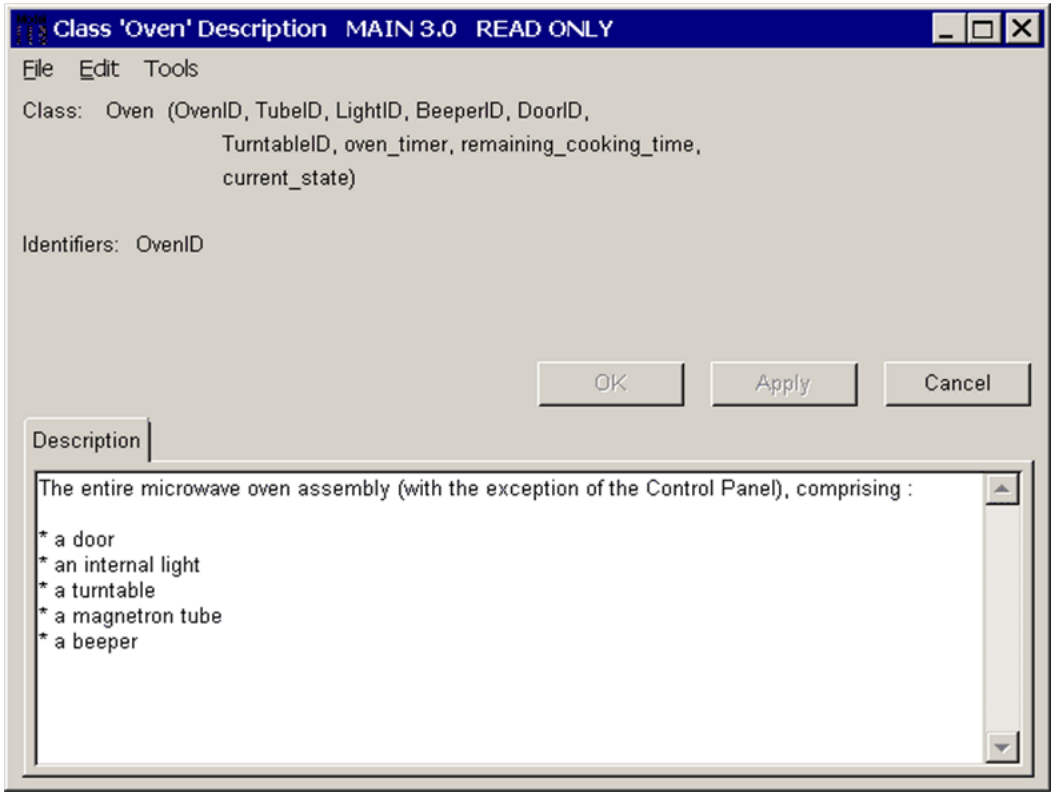

図 4-5: [Class Description] ウィンドウ

- [Class Description] ウィンドウの右上角にある [*X*] をクリックして、ウィンドウを閉じます。

- クラス図で `[Oven]` クラスが引き続き選択されていることを確認します。選択されていない場合は、目的のクラスボックスを 1 回クリックします。

- 右クリックして、選択したクラスに対して実行できるオプションのポップアップメニューを表示します。

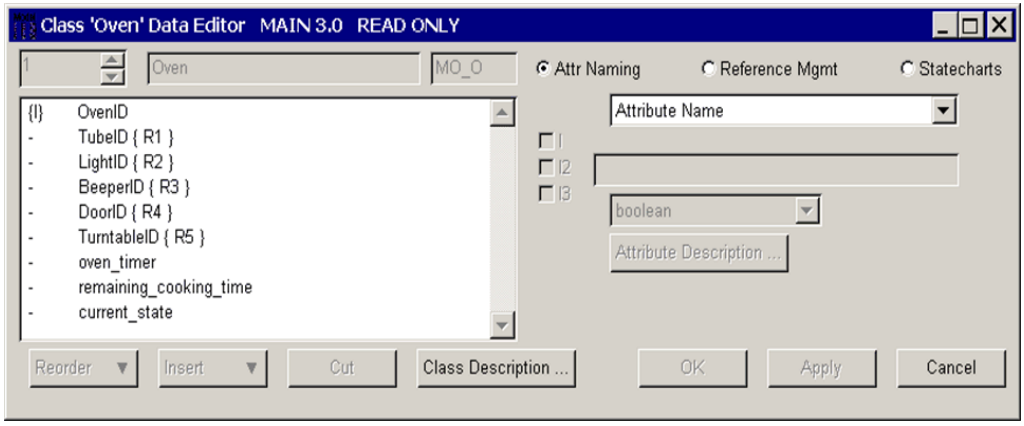

図 4-6: Oven クラスが表示されているクラスデータエディタ

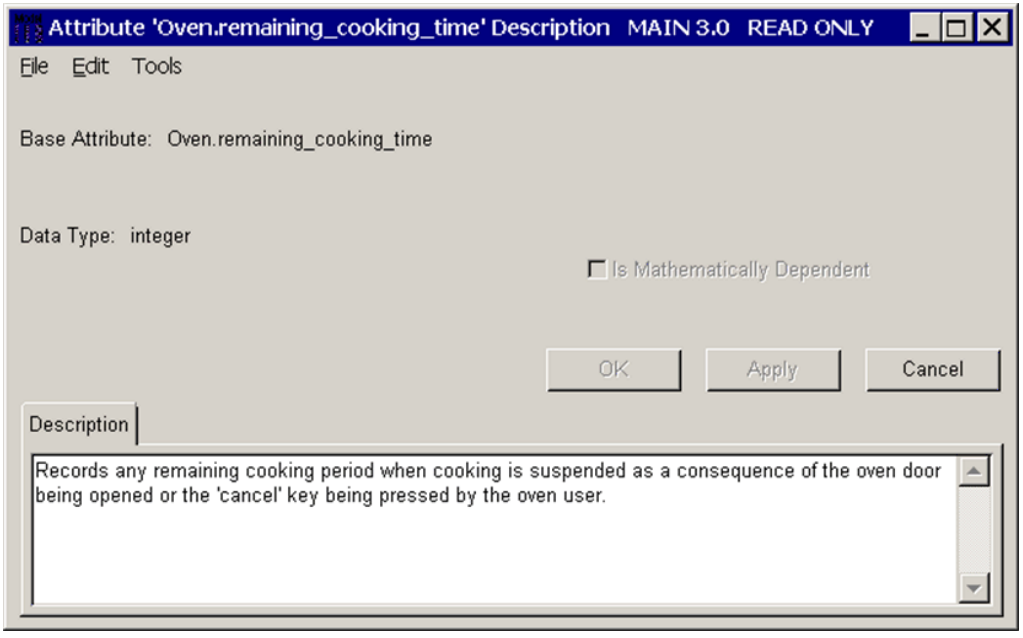

図 4-7: 属性の記述

- [*Show Class Data Editor*] をクリックします。クラスデータエディタが新しいウィンドウに表示されます（図 4-6）。このデータエディタを使用すると、クラス図のクラスに表示されるクラス属性を定義できます。このエディタでは、属性名に加えて、クラスの属性を識別する属性タイプと、クラス図に表示される関係の定式化に伴う属性を特定できます。

- クラスデータエディタウィンドウの左側にあるスクロールボックスで、属性の 1 つをクリックします。編集ウィンドウの右側にある [*Attribute Description*] ボタンがアクティブになります。選択した属性（名前、タイプなど）に関する情報も表示されることに注意してください。クラス図は読み取り専用モードであるため、現時点ではこの情報を変更できません。

- [*Attribute Description*] ボタンをクリックします。属性のテキスト記述が図 4-7 に示すように表示されます。クラスの記述と同様、属性の記述も、モデルを参照したり検討したりする人達にとって重要なドキュメントです。属性の目的と共に、属性が取ることができる値の制約が記述されます。

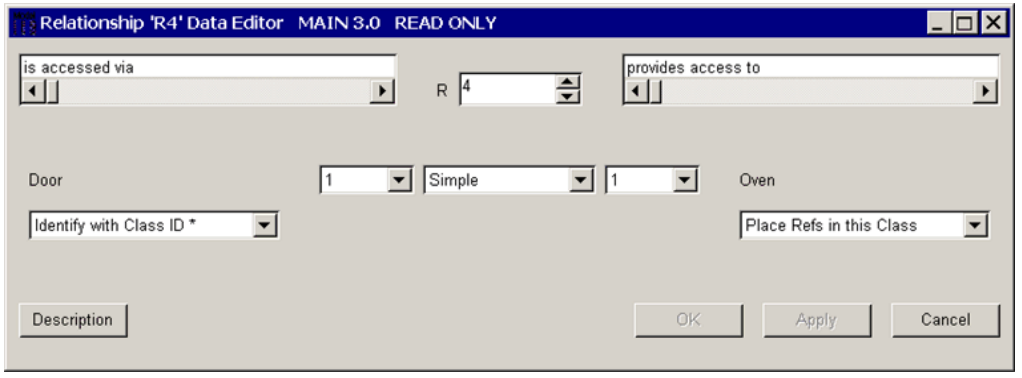

図 4-8: 関係データエディタ

- [Attribute Description] ウィンドウの右上角にある [*X*] を右クリックして、ウィンドウを閉じます。

- クラス図上の関係を選択します。

- 右クリックして、選択した関係に対して実行できる操作のメニューを表示します。

- ポップアップメニューの [*Show Relationship Data Editor*] を選択します。関係データエディタが別のウィンドウに表示されます（図 4-8）。このエディタを使用すると、クラス図に表示される関係の特性を定義できます。

  表示される情報のタイプに注意してください。これらは、関係名、関係先の数、一意な関係識別子（R4 など）、および識別属性を介して関係を定式化する情報です。

- 関係データエディタウィンドウの右上角にある [*X*] を右クリックして、関連するウィンドウを閉じます。

- クラスデータエディタウィンドウの右上角にある [*X*] を右クリックして、ウィンドウを閉じます。

4.5 既存のステートチャートの表示

この時点では、インデックスウィンドウが、現在表示されている唯一のウィンドウです。

- [ ] [Microwave Oven] サブシステムが、現在選択されているサブシステムであることを確認します。サブシステムに関連するステートチャートのリストが、インデックスウィンドウの [*Statecharts*] セクションに表示されます。

注意 : このモデルでは、ステートチャートは、" 目的の " 動的振る舞いを持つ個々のクラスに関連付けられています。

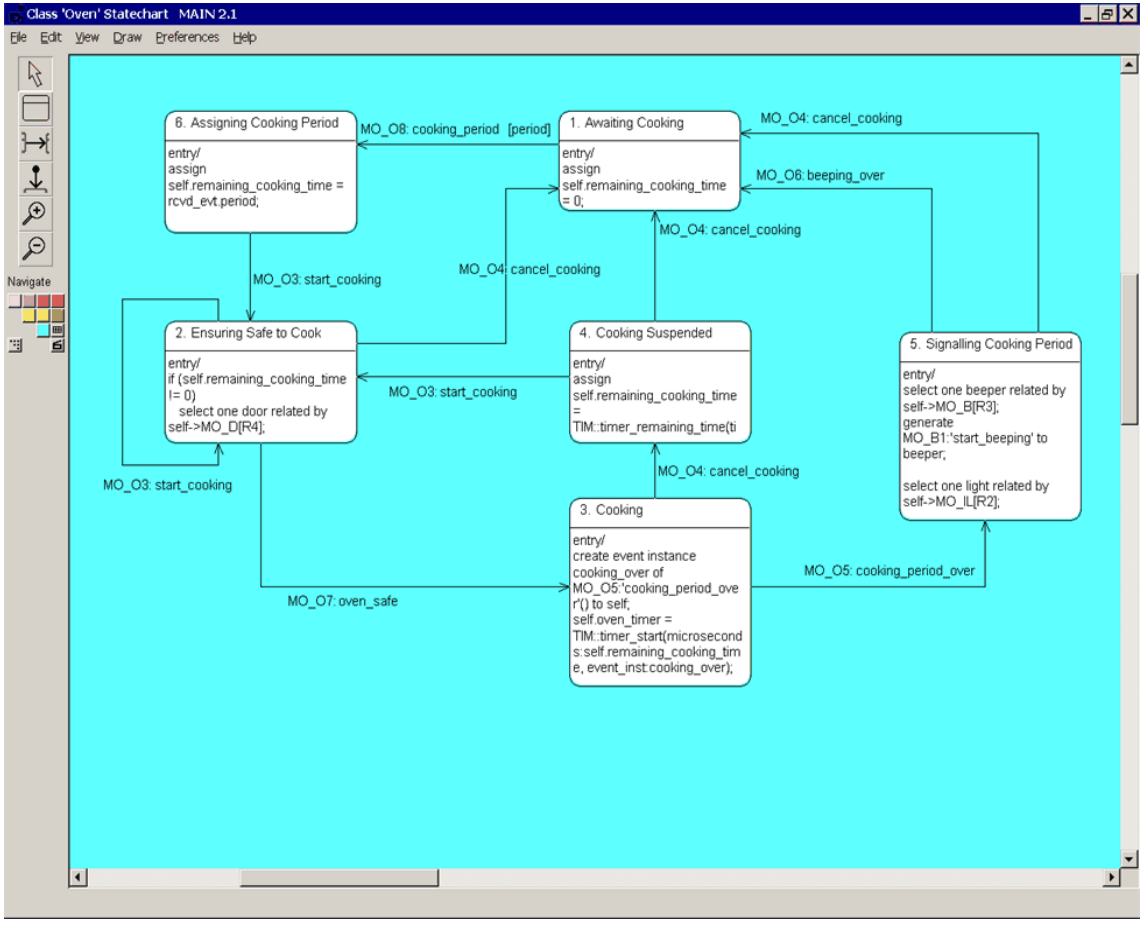

図 4-9: クラスのステートチャートエディタ

- スクロールバーを使って [Oven] ステートチャートを検索し、選択します。選択すると、ウィンドウの右側にある、[*SC*] というラベルが付いた青色のボタンがアクティブになります。

- 青色の[*SC*]ボタンをクリックして､有効なステートチャート操作のメニューを表示し、[*Show*] を選択します。図 4-9 に示すように、Oven ステートチャートが別のウィンドウに表示されます。Microwave Oven クラス図で、[Oven] クラスを選択し、ナビゲーションツールの一部を構成している青色のボタンをクリックすることによって、クラス図から Oven ステートチャートにアクセスすることも可能です。

ステートチャートは、クラスに関連する動的振る舞いを表します。この振る舞いは、クラスが実行できるアクション、クラスがそのアクションを実行する条件、および指定のアクションのパフォーマンスを開始する条件で表されます。クラスの各インスタンスは、ステートチャートに適合する状態機械のインスタンスに割り当てられます。Oven ステートチャートに表示される各状態は、Oven クラスのライフサイクル内のフェーズに対応します。アクションは、厳密なアクション記述言語（次の節で詳しく説明します）を使って、状態（Moore 状態機械概念）ごとに定義されます。異なる状態間の遷移は矢印を使って表し、矢印の向きが遷移の方向を表します。各遷移上に示されるイベントラベルは、目的の遷移の発生を引き起こすイベントを表します。

- ステートチャートの [Cooking] 状態を、左クリックして選択します。

- 右クリックして、Cooking　状態に対して実行できる操作のメニューを表示します。[*State Data Editor*] を選択します。状態データエディタが別のウィンドウに表示されます（図 4-10）。

  状態データエディタを使用すると、状態に名前を割り当てたり、デフォルトの番号が適切でない状態に対して一意な番号（識別を目的とします）を割り当てたりすることができます。また、この特定の状態が現在のところ応答していない、定義済みのイベントのリストも表示されます。このようなイベントは、無視されるイベント、または発生できないイベント（アプリケーションの視点から見た場合）として、自動的に作成される状態遷移テーブルに入力されます。

- ウィンドウの右上角にある [*X*] をクリックして、ウィンドウを閉じます。

- [Cooking Suspended] 状態と [Ensuring Safe to Cook] 状態を結ぶ遷移（矢印）を、左クリックして選択します。

- 右クリックして、選択した遷移に対して実行できる操作のメニューを表示します。[*Show Transition Data Editor*] を選択します。遷移データエディタが別のウィンドウに表示されます（図 4-11）。

  このエディタの目的は 1 つだけであり、事前定義イベントを遷移に関連付けることです。この操作を実行すると、ステートチャート図の遷移の上にイベントラベルが設定されます。Model Verifier を使ってシミュレーションしたり、Generator を使ってコードを生成する前に、すべての遷移をイベントに関連付ける必要があります。

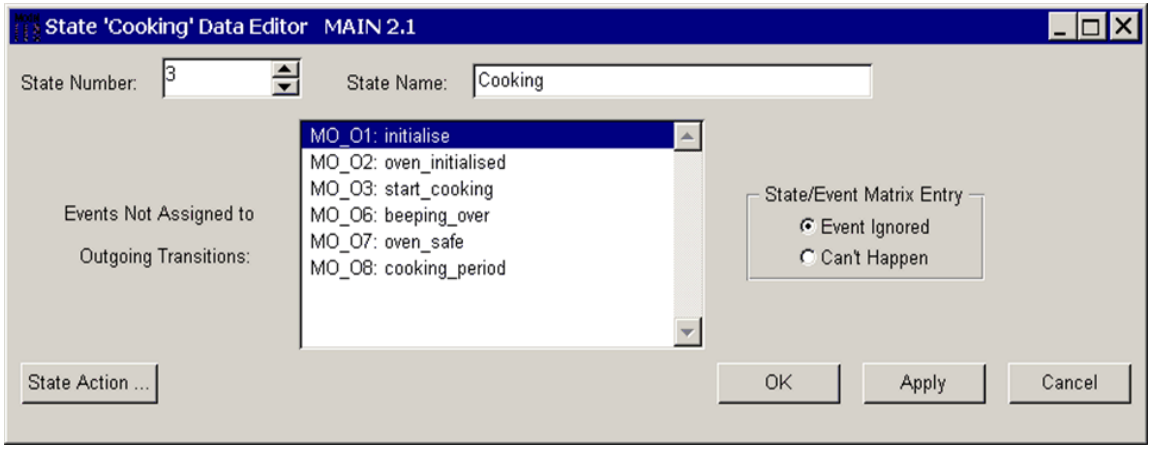

図 4-10: 状態データエディタ

- 遷移データエディタウィンドウの右上角にある [*X*] をクリックして、ウィンドウを閉じます。ステートチャートエディタには、引き続き Oven ステートチャートが表示されています。

状態データエディタと遷移データエディタの説明の中で、事前定義イベントという用語が出てきました。このイベントは、イベントデータエディタを使って定義します。

- ステートチャートキャンバスの何も表示されていない部分をクリックして、現在選択されているステートチャート要素の選択を解除します。

- 右クリックして、ダイアグラム要素が何も表示されていないときに実行できるアクションのメニューを表示します。[*Event Data Editor*] を選択します。イベントデータエディタが別のウィンドウに表示されます（図 4-12）。

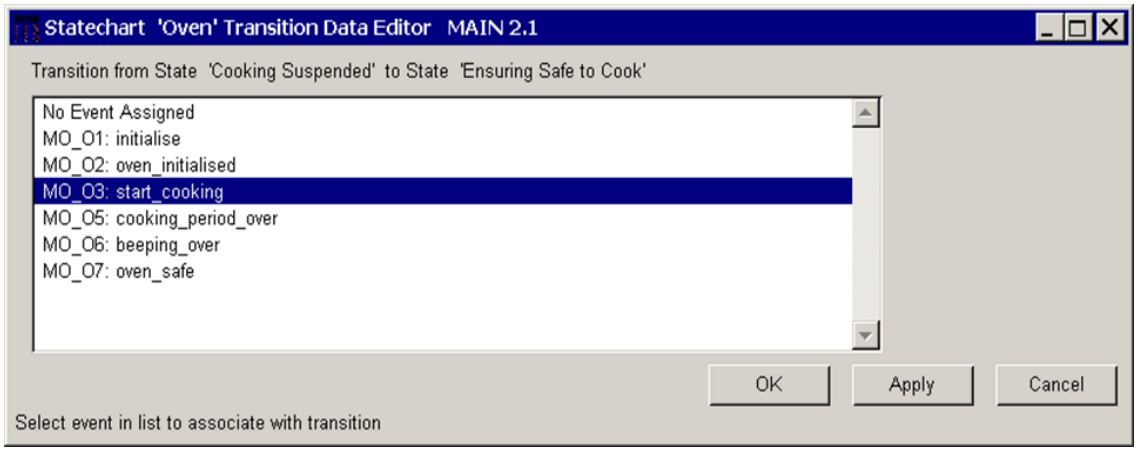

図 4-11: ステートチャートの遷移データエディタ

状態データエディタを使用すると、イベント定義を作成し（[*Insert*] ボタンを使用）、イベントに伴って発生し、ステートチャートに送られるデータ項目を指定できます。発生するデータ項目は、標準のプログラミング言語において、引数を関数に渡すのとほとんど同じように、状態アクションに渡されます。

このエディタウィンドウの左側にはスクロールボックスがあり、現在定義されているすべてのイベントが列挙されます。いずれかのイベントを選択すると、そのイベントのイベント番号と意味が、ダイアログボックスの対応する領域に表示されます。イベントがイベントリストに表示されているというだけでは、ステートチャート上の遷移に割り当てられているとは限りません。この状態は、イベントが定義されており、遷移に割り当てることができることを単に意味しています。

[Event Parameters] ボックスに表示される項目は、現在選択されているイベントに関連付けられているパラメータではありません。これらの項目は、イベントパラメータとして割り当てることができる、定義済みのデータ要素を表します。すでに割り当てられているパラメータを表すのは、イベントリストです（例： `MO_08: cooking_period` の `period`）。新しいイベントパラメータの候補は、[*Edit Data Items*] ボタンを使って定義できます。

- イベントデータエディタウィンドウの右上角にある [*X*] を右クリックして（または [*Cancel*] ボタンをクリックして）、ウィンドウを閉じます。

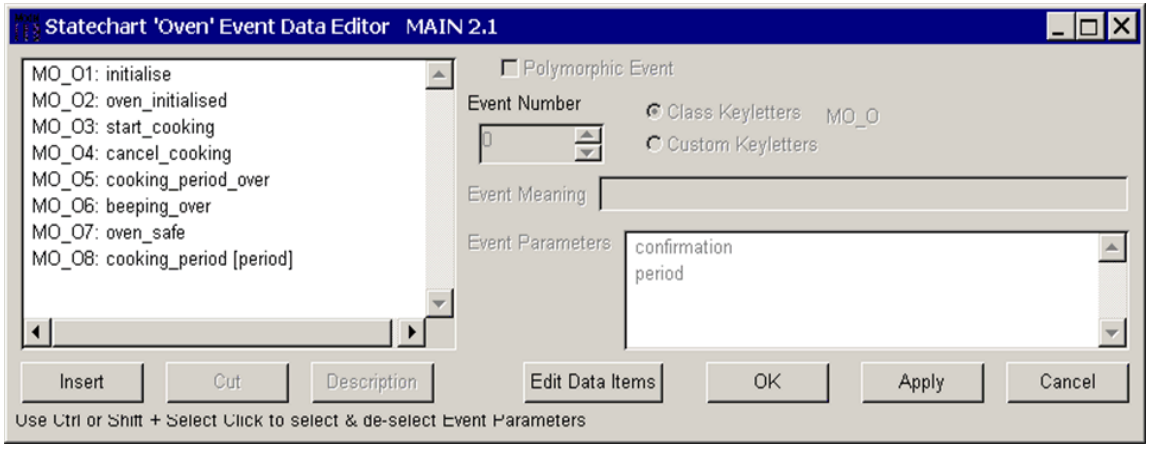

図 4-12: ステートチャートのイベントデータエディタ

4.6 事前定義アクションの表示

- ステートチャートの [Cooking] 状態をクリックして選択します。
- ナビゲーション機能の中で、[Action] ボタン（最終列の一番下のボタン）を選択します。Cooking 状態に関連付けられているアクションの仕様が、BridgePoint Object Action Language で表示されます（図 4-13）。

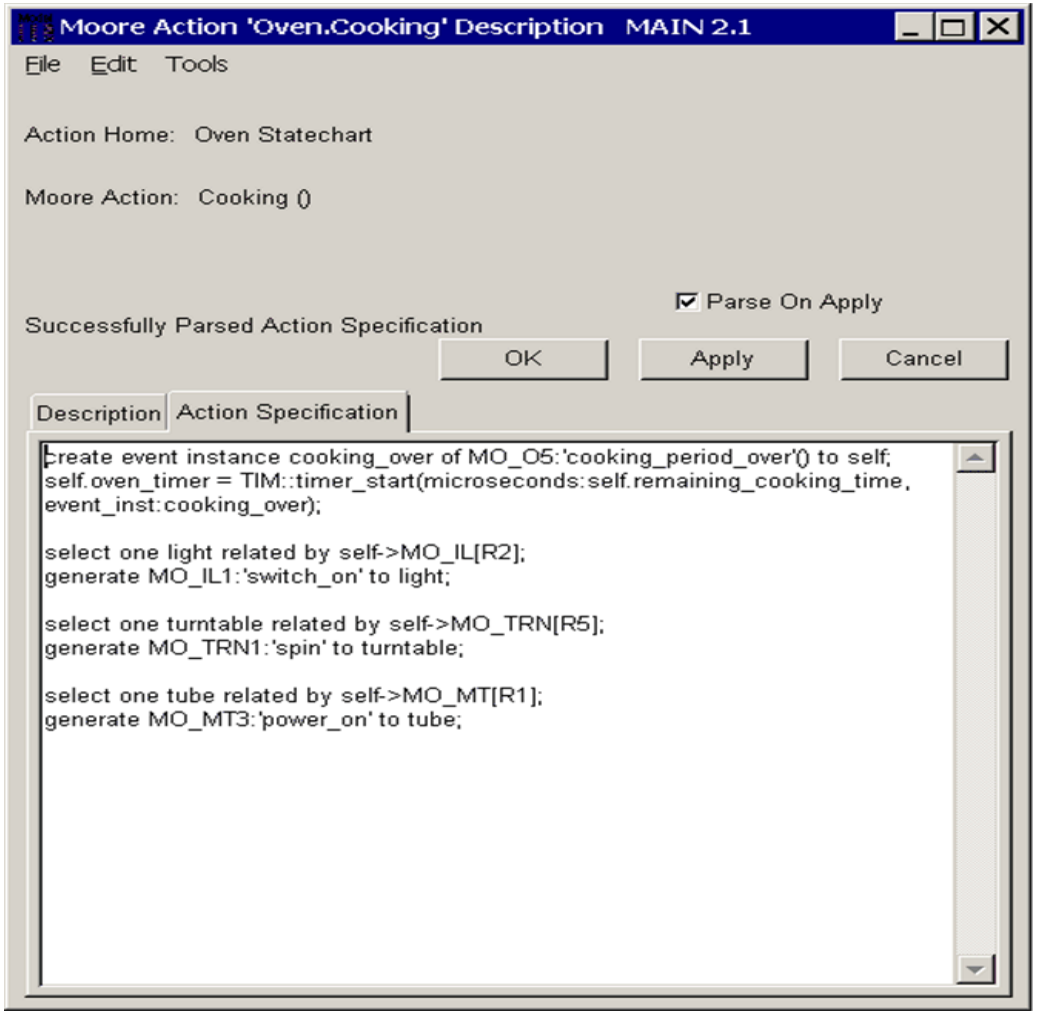

図 4-13: [Moore Action Description] ウィンドウ /[Actoin Specification] タブ

BridgePoint Object Action Language （『BridgePoint Object Action Language Manual』を参照）は、ステートチャートの状態に関連するアクションを含めて、各種の異なるモデリング要素タイプの処理要件を定義するために使用されます。この言語は、プログラミング言語ではなく、モデル操作言語と考える方が適切です。言語を構成するステートメントは、目的のモデルを構成するダイアグラムに表示される、モデリング構造体の操作に適応しています。クラス図の関係の追跡、クラス図に表示される属性への値の割り当て、各種モデルクラスに関連するステートチャートに対してのイベントの生成などが、その操作例です。

言語は厳密に定義されており、実行可能であるため、モデルのシミュレーションとコード生成の両方をサポートできます。高度なモデル操作言語の使用は、ターゲットとなるプログラミング言語の性質に関して一切の前提が不要であるため、コード生成に好都合であり、実装の詳細を気にせずに分析し、アプリケーションの問題に集中できます。

- このエディタには、BridgePoint Object Action Language を使って正しく記述されていることを確認する機能が、[*Parse On Apply*] チェックボックスを介して用意されています。

エディタウィンドウの右上角にある [*X*] をクリックして、ウインドウを閉じます。

- 開いているすべてのエディタウィンドウを閉じます。

- Model Builder インデックスウィンドウを閉じます。

4.7 まとめ

この演習では、BridgePoint Model Builder の各種機能を使用して、さまざまな種類のダイアグラムとアクション記述を表示しました。この後の演習では、これらの機能を使って、モデルダイアグラムとアクション仕様を作成します。

5

演習 3: クラス図の作成

5.1 リポジトリとワークスペースの新規作成

これまでの 2 つの演習では、既存のリポジトリからワークスペースを作成する方法と、既存のワークスペースを開く方法を学習しました。この演習では、新しいワークスペースだけでなく、新しいリポジトリを作成します。

- Model Builderインデックスウィンドウのメニューバーから、[*File*]|[*New*]の順に選択して、リポジトリとワークスペースの新規作成を開始します。

  この操作によって、[*Create New Model Workspace*] ダイアログボックスが表示されます。

- 図 5-1 に示すように、新しいモデルワークスペース名を「`mo.ooa`」として指定します。

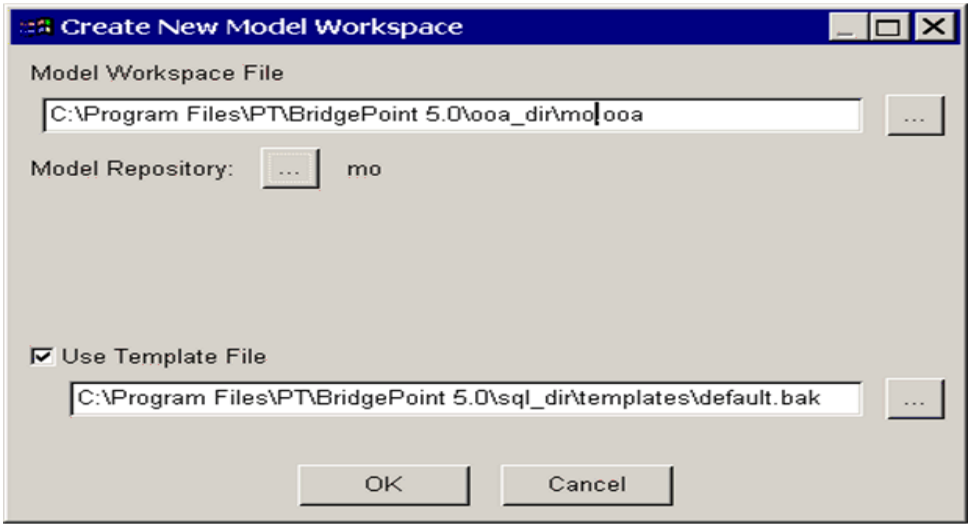

図 5-1: [Create New Model Workspace] ダイアログボックスにおける「mo.ooa」の入力

mo ワークスペースの派生元であるモデルリポジトリが mo として指定されたことに注意してください。リポジトリは存在していないので、[*OK*] ボタンをクリックすると作成されます。

- [*OK*] をクリックします。

  Model Builder インデックスウィンドウの [*Subsystem*] セクションにあるスクロールボックスに、1 つのサブシステム（[Unnamed Subsystem] という名前）が表示されます。

  Model Builder インデックスウィンドウの [*Domain->Workspace*] セクションに、新しく作成したワークスペースが表示されます（図 5-2）。これは、編集用に自動的にチェックアウトされています。このため、このモデリングレベルにおいて、ダイアグラムに変更を加えることができます。

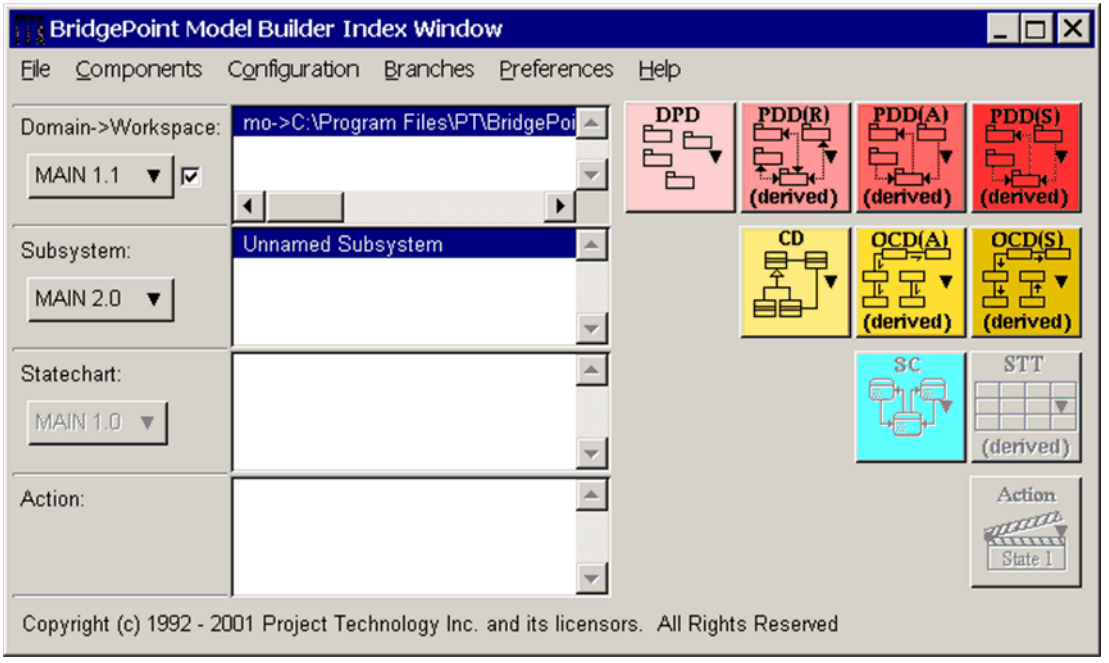

図 5-2: Model Builder インデックスウィンドウにおける [Unnamed Subsystem]

- *ドメインパッケージ図*を表す [*DPD*] ボタンをクリックし、ダイアグラムにアクセスします。ドロップダウンメニューが表示されます。メニューの [*Show*] を選択します。

DPD エディタが新しいダイアグラムとして開きます。新しい DPD には、すでに 2 つのパッケージが配置されています。[Unnamed Subsystem] という、名前のないサブシステムと、[Time] という外部エンティティです（図 5-3）。ここでは、サブシステムパッケージだけを扱います。

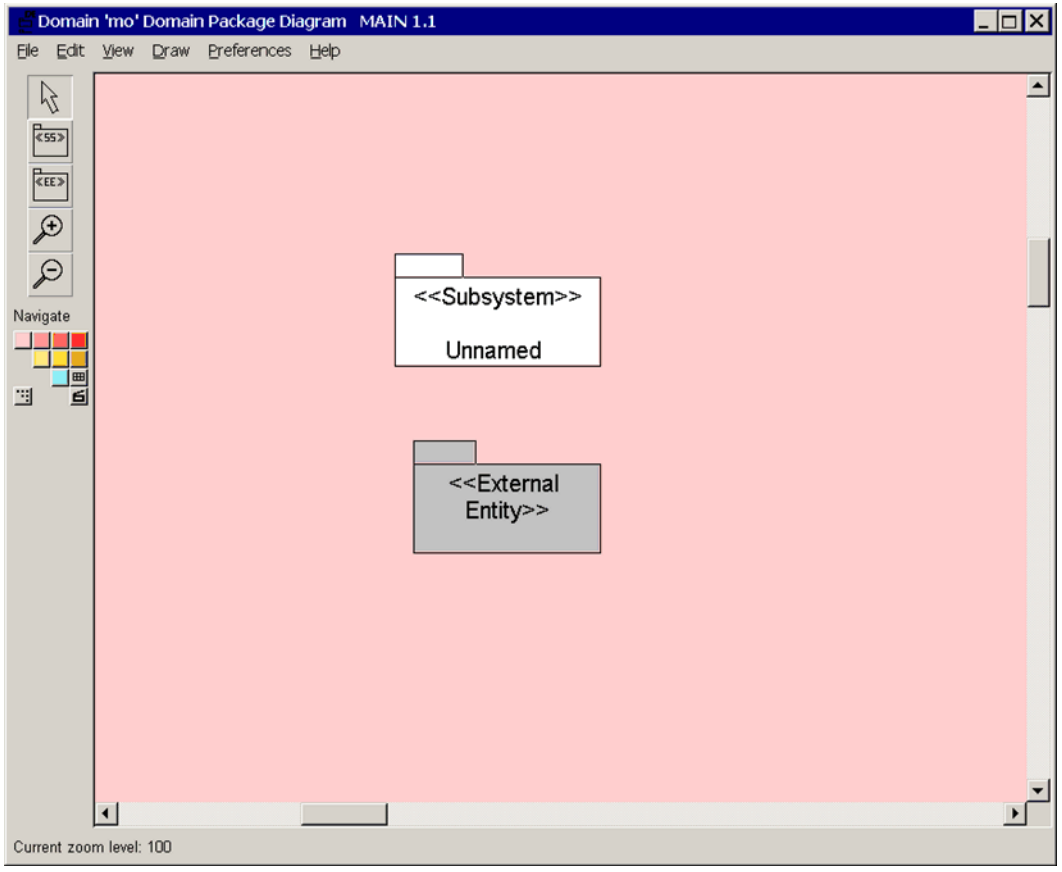

図 5-3: DPD エディタにおける新しいダイアグラム

- [Unnamed Subsystem] パッケージをクリックして選択します。
- 右クリックします。
- [*Show Subsystem Data Editor*] を選択します。
- 図 5-4 に示すように、サブシステムの名前を "Microwave Oven" に変更します。

[*Auto Numbering Start Value*] は、サブシステムのクラス図のクラスに、自動的に割り当てられる最初の番号を指定します。

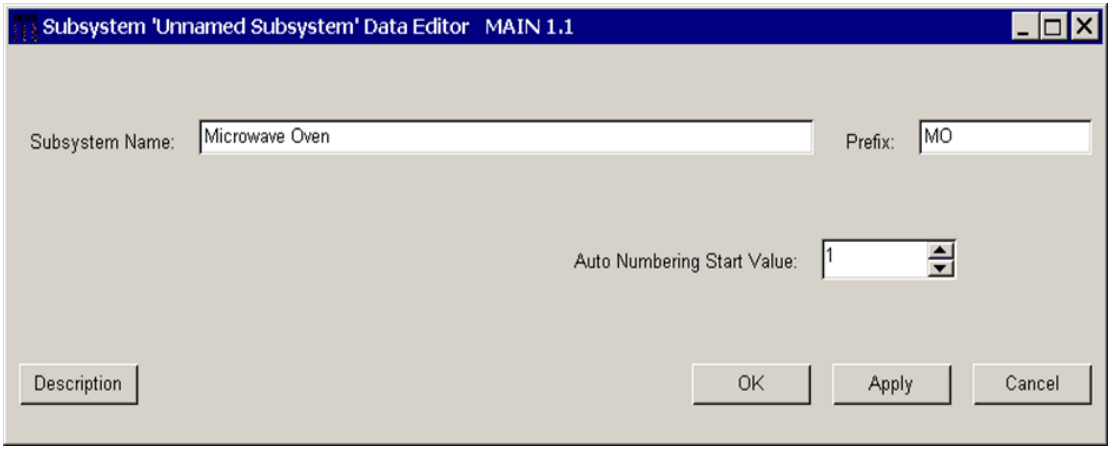

図 5-4: サブシステムデータエディタを使った Microwave Oven サブシステムの編集

- [*Apply*] をクリックします。ダイアグラムが更新され、新しいサブシステム名を反映します。

- 図 5-4 に示すように、`Microwave Oven` サブシステムの接頭辞を、[Prefix] ボックスに「`MO`」と入力して設定します。この接頭辞によって、異なるサブシステムにある同じ名前のクラスが区別されます。

- [*OK*] をクリックします。指定の接頭辞が、後で使用できるようにモデルワークスペースに保存され、サブシステムデータエディタが閉じます。

ここで、クラス図を作成する中で、クラスに属性を追加するときに使用する新しいデータ型を定義しましょう。新しいデータ型は、通常、必要に応じて追加しますが、DPD エディタを使っていつでもドメインに追加できます。便宜上、新しいデータ型をここで追加します。この操作によって、この後の演習の中で、クラス図エディタからドメインパッケージ図エディタへのジャンプが回避されます。

- DPD エディタの描画キャンバス上で、何も表示されていない部分にカーソルを移動します。

- 右クリックします。ポップアップメニューが表示されます。

- ポップアップメニューの [*Show Data Type Data Editor*] を選択します。

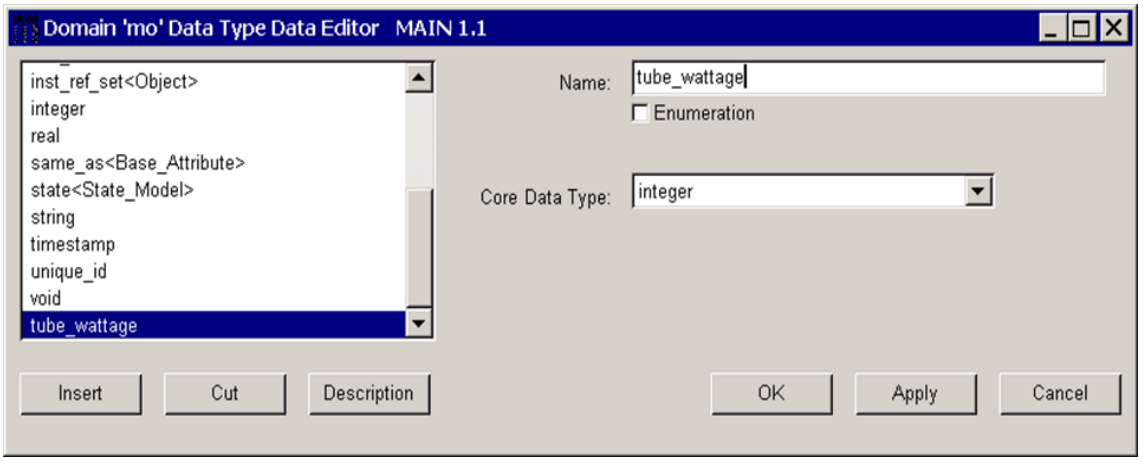

図 5-5: ドメインデータ型データエディタを使ったデータ型の追加

*データ型データエディタ*を使用すると、モデル化しているドメインに対して現在定義されているデータ型を表示したり、新しいデータ型を作成したりできます。アクセスを制御する目的から、このエディタは、DPD エディタからのみアクセスできます。

- データ型データエディタウィンドウの左下角にある [*Insert*] ボタンを押します。
- [Name] ボックスに「`tube_wattage`」と入力します。
- [*Core Data Type*] ドロップダウンメニューの [*integer*] を選択します。データ型データエディタウィンドウは図 5-5 のように表示されます。

  [*OK*] をクリックします。新しいデータ型がドメインに追加され、データ型データエディタが閉じます。

- DPD を閉じます。

Model Builder インデックスウィンドウが表示され、[Unnamed Subsystem] と表示されていた部分が、図 5-6 に示すように `[Microwave Oven]` に変更されて表示されます。この変更は、DPD に対して加えたばかりの変更を直接反映しています。DPD に表示される、名前を持つすべてのサブシステムパッケージも、Model Builder インデックスウィンドウに表示されます。

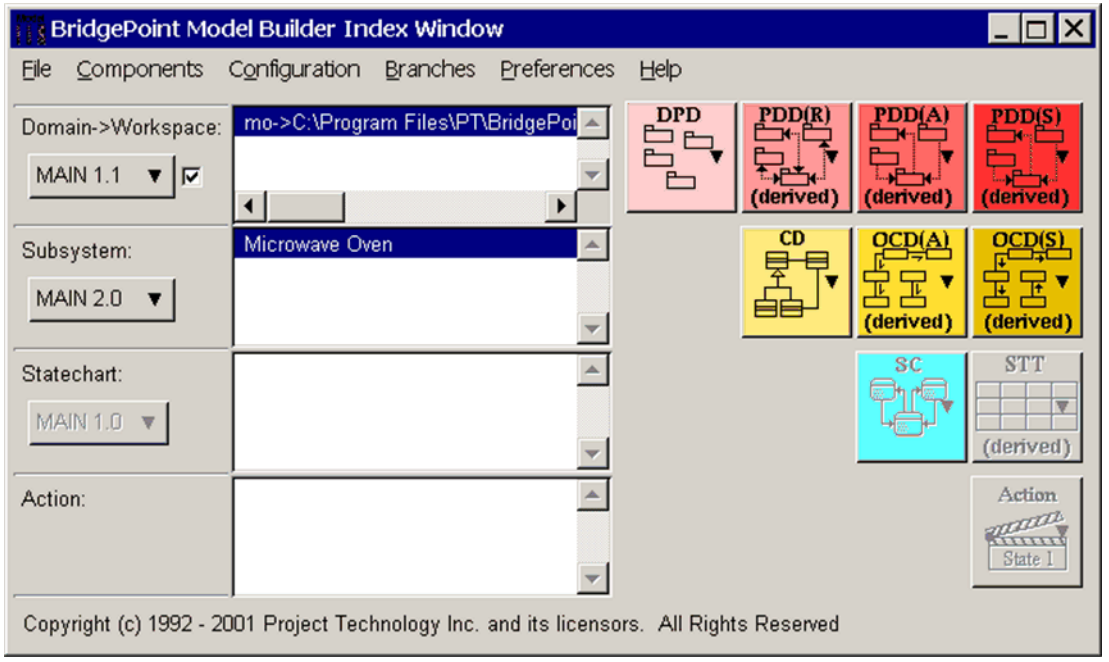

図 5-6: DPD を編集した後の Model Builder インデックスウィンドウ

5.2 クラス図の作成

- クラス図を編集できるためには、その前に、編集可能な形式でクラス図をチェックアウトする必要があります。

- Model Builder インデックスウィンドウの [Subsystem] セクションで、[*Main 2.0*] ボタンを押します。ドロップダウンメニューの [*Check-Out Latest from Domain Repository*] を選択します。[Component Check-Out] ダイアログボックスが表示されます（図 5-7）。

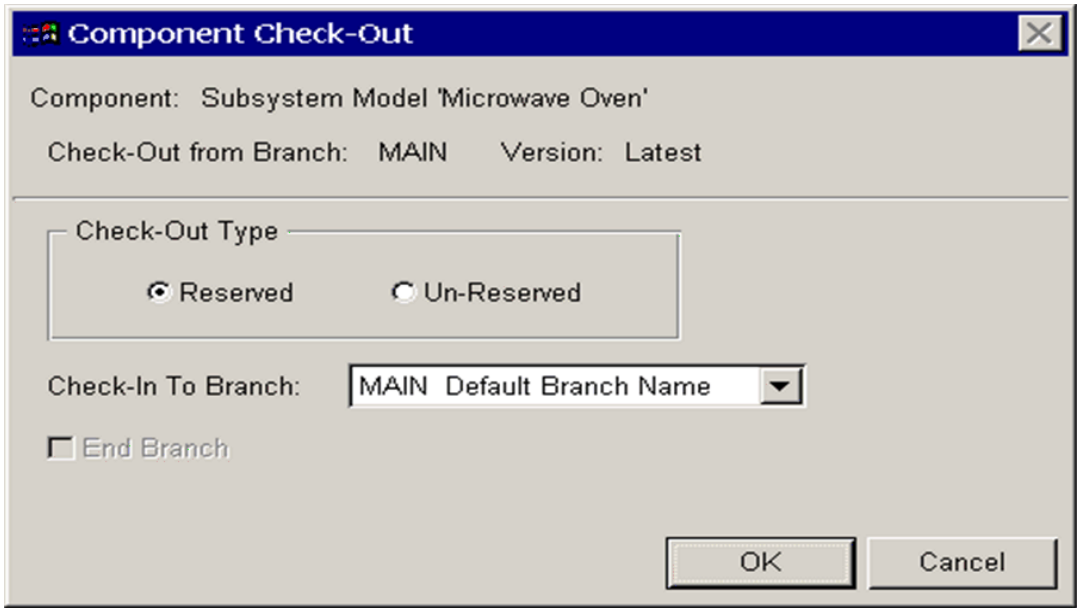

図 5-7: [Component Check-Out] ダイアログボックス

- [*OK*] をクリックして、デフォルト値をすべてそのまま使用します。Model Builder インデックスウィンドウが更新され、チェックアウト操作が反映されます（ボタンの名前が新しくなり、バージョンが 2.1 に変わります。また、ボタンの右側のチェックボックスがオンになります）。

- [*CD*] ボタンをクリックし、[*Show*] を選択して、`Microwave Oven` サブシステムのクラス図を開きます。空のクラス図が表示されます（図 5-8）。

これから、5 つのクラスで構成されるクラス図を作成します。

- クラス描画ツール（上から 2 番目にあり、カーソルが描かれているボタンの下にあるボタン）をクリックします。カーソルを描画キャンバスの上に移動すると、鉛筆の形に変化します。

- クラスボックスの左上角の位置にカーソルを置きます。

- 左クリックし、ボタンを押したまま、クラスボックスの右下角の位置までカーソルをドラッグします。マウスボタンを離します。

  名前が `Unknown Class`、番号が 1、キー文字列が `MO_KEY` であるクラスが作成されます。

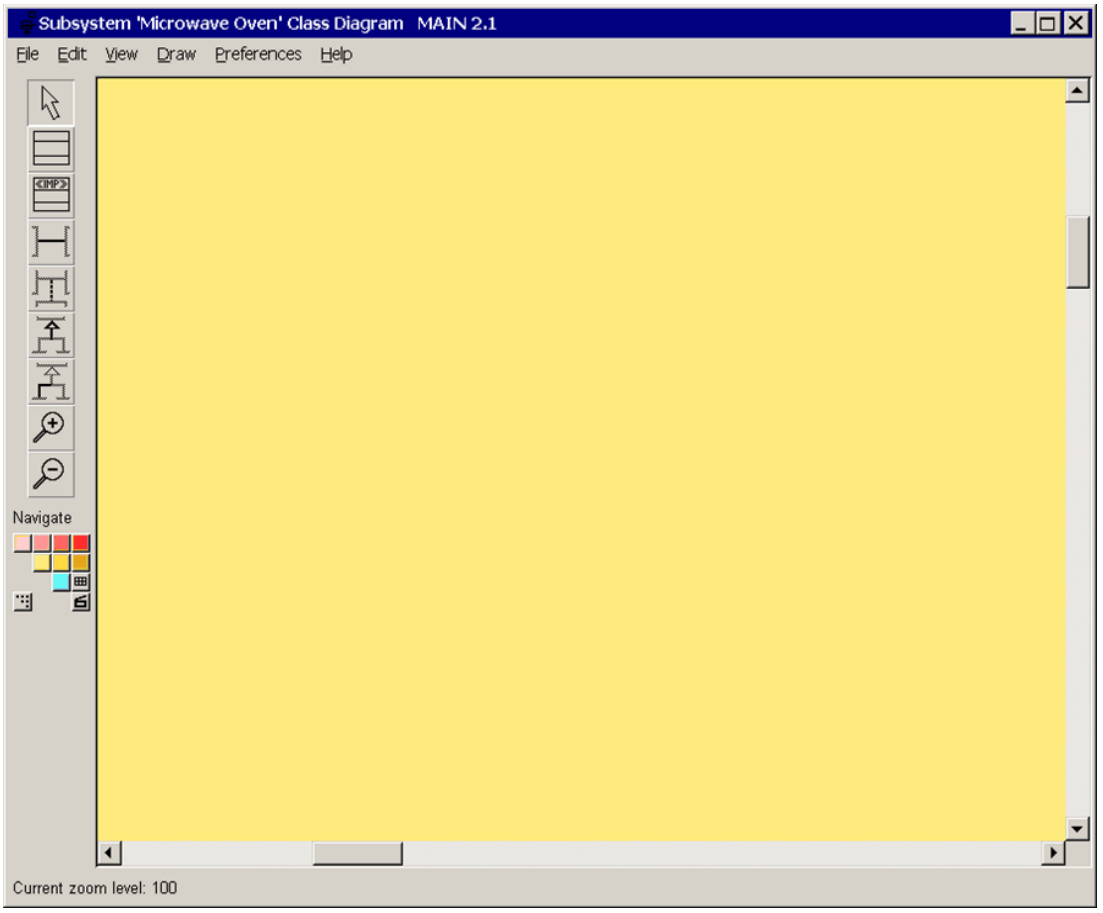

図 5-8: 空のクラス図

- カーソルツール（パレットの一番上にある矢印）をクリックします。
- 作成したばかりのクラスボックスをクリックして選択します。
- 右クリックし、選択したクラスに対して有効なオプションのポップアップメニューを表示します。
- [*Show Class Data Editor*] を選択します。
- クラス名を `Unknown Class` から `Oven` に変更します。
- クラスのキー文字列を `MO_KEY` から `MO_O` に変更します。クラスデータエディタウィンドウは図 5-9 の状態です。

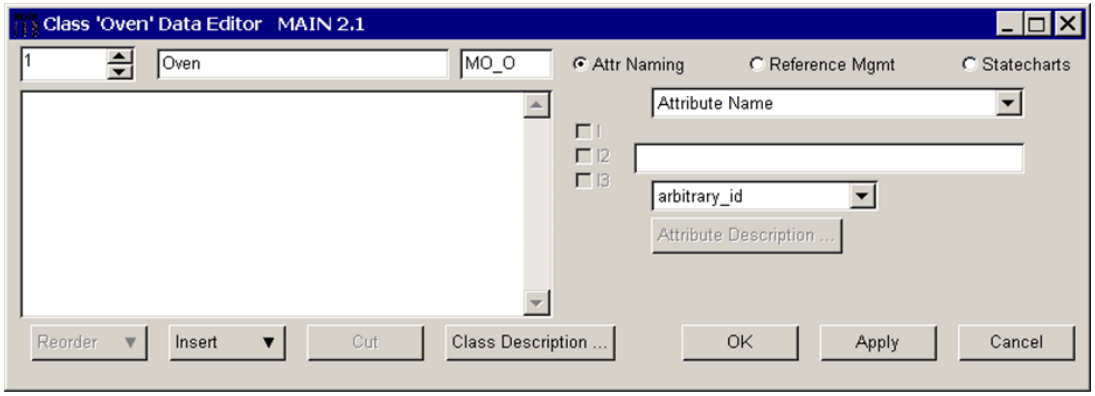

図 5-9: クラスデータエディタを使った Oven クラスの編集

- [*Apply*] ボタンをクリックします。ダイアグラムが更新され、加えた変更が反映されます。
- [*Class Description*] ボタンをクリックし、クラス記述を追加します。クラス記述エディタが表示されます。
- 次のテキストを入力します。

```
This class represents the entire microwave oven assembly (with the
exception of the control panel) comprising:

  • a door
  • an internal light
  • a turntable
  • a magnetron tube
  • a beeper
```

□ [*OK*] をクリックしてクラス記述エディタを閉じます。

5.3 クラス図への属性の追加

クラスデータエディタウィンドウの左側にある大きなスクロールボックスには、クラスの属性が表示されます。Oven クラスの場合、ボックスはまだ空です。ここでは、作成したクラスに属性を割り当てます。

- [*Insert*] ボタンをクリックします。[*Top*] を選択します。属性リストに、名前のない属性が挿入されます。

- クラスデータエディタウィンドウの右側にある空の編集ボックスに属性名（OvenID）を入力します。

- [*I*] ボックスをクリックして、この属性が識別属性であることを指定します。

- 属性名を入力したボックスのすぐ下にあるドロップダウンボックスを使って、属性のタイプを [unique_id] に設定します。

- [*Apply*] ボタンをクリックします。クラス図が更新され、加えたばかりの変更が反映されます。

場合によっては、ダイアグラム上のクラスボックスのサイズを変更する必要があります。この操作は、ボックスを選択し、四隅に表示されているサイズ変更ハンドルの 1 つをクリックし、ボックスが目的のサイズになるまでドラッグすることによって行います。

- 学習したばかりの手順を使って、次の属性を Oven クラスに追加します。

  remaining_cooking_time: integer （これを識別属性にする必要はありません）

- クラスに目的の振る舞いがある場合は、ステートチャートを使ってそれをモデル化します。クラスの現在の状態値を追跡するには、特別な属性がなければなりません。クラスデータエディタウィンドウの右上に、3 つのラジオボタンがあります。現在選択されているボタンは [*Attr Naming*] であり、この場合は、これまで操作してきたようにクラスデータエディタを使って属性を追加できます。現在の状態を表す属性（current_state 属性）を追加するには、[Statecharts] ラジオボタンをクリックする必要があります。

- [*Statecharts*] ラジオボタンをクリックします。[*Instance Statechart*] チェックボックスをオンにして、[*Apply*] をクリックします。クラスデータエディタウィンドウは図 5-10 の状態です。

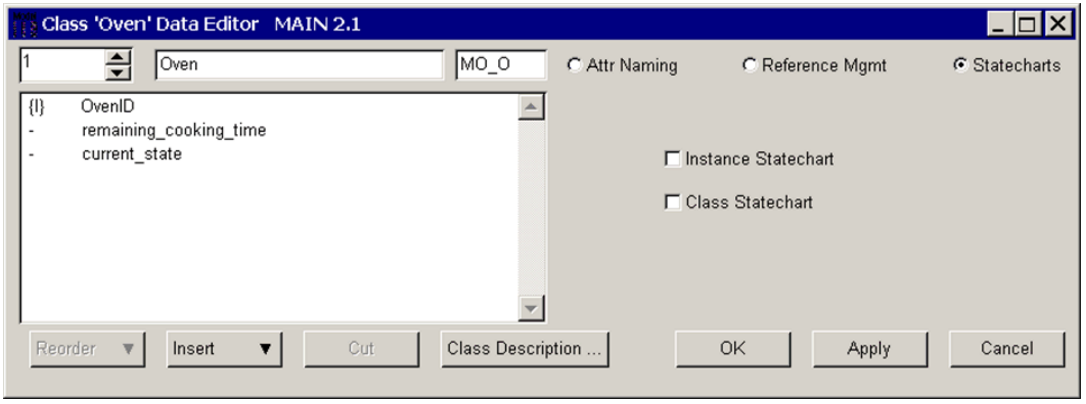

図 5-10: クラスデータエディタを使った current_state 属性の追加

- 以下に示されている残りのクラスと属性を追加します。図 5-11 は、指定がすべて終了した後のクラス図の状態を示しています。必ず、クラスのキー文字列を指定してください。最終的なクラス番号は、新しいクラスをダイアグラムに追加するときの順番によって、変わる可能性があります。クラス、属性、およびプロパティは次のとおりです。

1. Magnetron Tube

- キー文字列 : MO_MT
- クラス番号 : 2
- TubeID: unique_id （識別属性として指定）
- current_power_output: tube_wattage
- [*Instance Statechart*] チェックボックスをオン

2. Internal Light

- キー文字列 : MO_IL
- クラス番号 : 3
- LightID: unique_id （識別属性として指定）
- [*Instance Statechart*] チェックボックスをオン

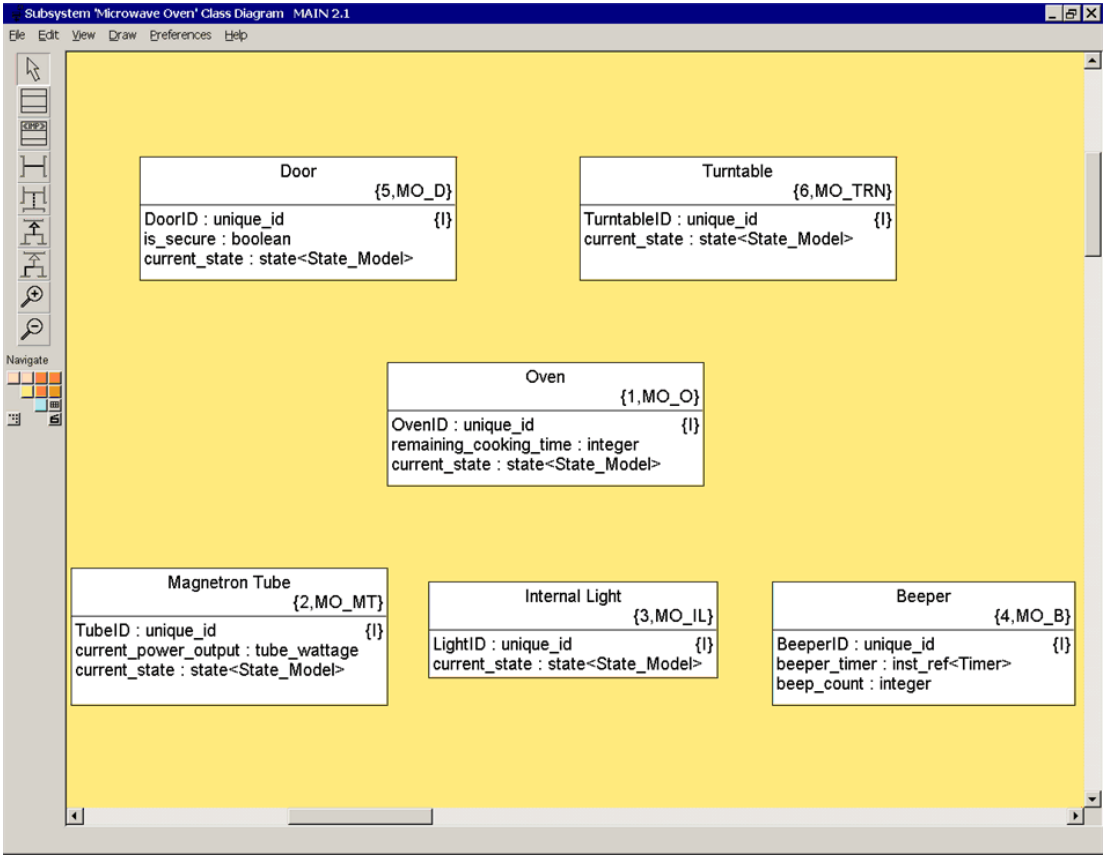

図 5-11: クラスが定義されたクラス図（関係なし）

3. Beeper

- キー文字列 : MO_B
- クラス番号 : 4
- BeeperID: unique_id （識別属性として指定）
- beeper_timer: inst_ref<Timer>
- beep_count: integer
- [*Instance Statechart*] チェックボックスをオン

4. Door

- キー文字列 : MO_D
- クラス番号 : 5
- DoorID: unique_id （識別属性として指定）
- is_secure: boolean
- [*Instance Statechart*] チェックボックスをオン

5. Turntable

- キー文字列 : MO_TRN
- クラス番号 : 6
- TurntableID: unique_id （識別属性として指定）
- [*Instance Statechart*] チェックボックスをオン

5.4 クラス図への関係の追加

この節では、まず関係を描画し、これを消去し、そしてアングルポイントを使って再描画します。直線の関係線と、アングルポイントを含む関係線の両方について、その描画方法を学習します。

- 関係描画ツール（描画ツールパレットの上から 4 番目のボタン）を選択し、カーソルを描画キャンバスに移動します。カーソルが鉛筆に変化します。
- カーソルを [Magnetron Tube] クラスボックスの内側に置きます。左クリックし、ボタンを押したままにします。
- 鉛筆を [Oven] クラスボックスの内側までドラッグし、マウスボタンを離します。2 つのクラスの間に、R1 という名前の 1 対 1 の関係が作成されます。
- カーソルツールをクリックして、関係描画ツールを無効にします。
- 関係線を左クリックして、関係を選択します。
- 右クリックし、ポップアップメニューの [*Delete*] を選択します。確認のプロンプトに対して [*OK*] をクリックします。
- 関係描画ツールを選択します。
- カーソルを [Magnetron Tube] クラスボックスの内側に置きます。左クリックし、ボタンを押したままにします。どのクラスにも触れない位置まで鉛筆カーソルをドラッグし、マウスボタンを離します。関係のアングルポイントが作成されます。
- 左クリックし、ボタンを押したままにします。[Oven] クラスボックスの内側までアングルポイントをドラッグします。マウスボタンを離します。2 つのクラスの間に R1 関係が再び作成されます（図 5-12）。
- カーソルツールをクリックして、関係描画ツールを無効にします。
- 関係線を左クリックして、新しく作成した関係を選択します。
- 右クリックし、ポップアップメニューの [*Show Relationship Data Editor*] を選択します。

2 つのクラス間の関係の性質を明らかにするために、関係にラベルを付けることができます。以下に例を示します。

- `Magnetron Tube` ***is housed by*** `Oven`
- `Oven` ***houses*** `Magnetron Tube`

🗀 図 5-13 に示すように、関係ラベルはテキスト編集ボックスに入力します。

この関係は単純な 1 対 1 のタイプであるため、関係データエディタウィンドウの中央にあるボックスを使って、関係先の数を変更する必要はありません。

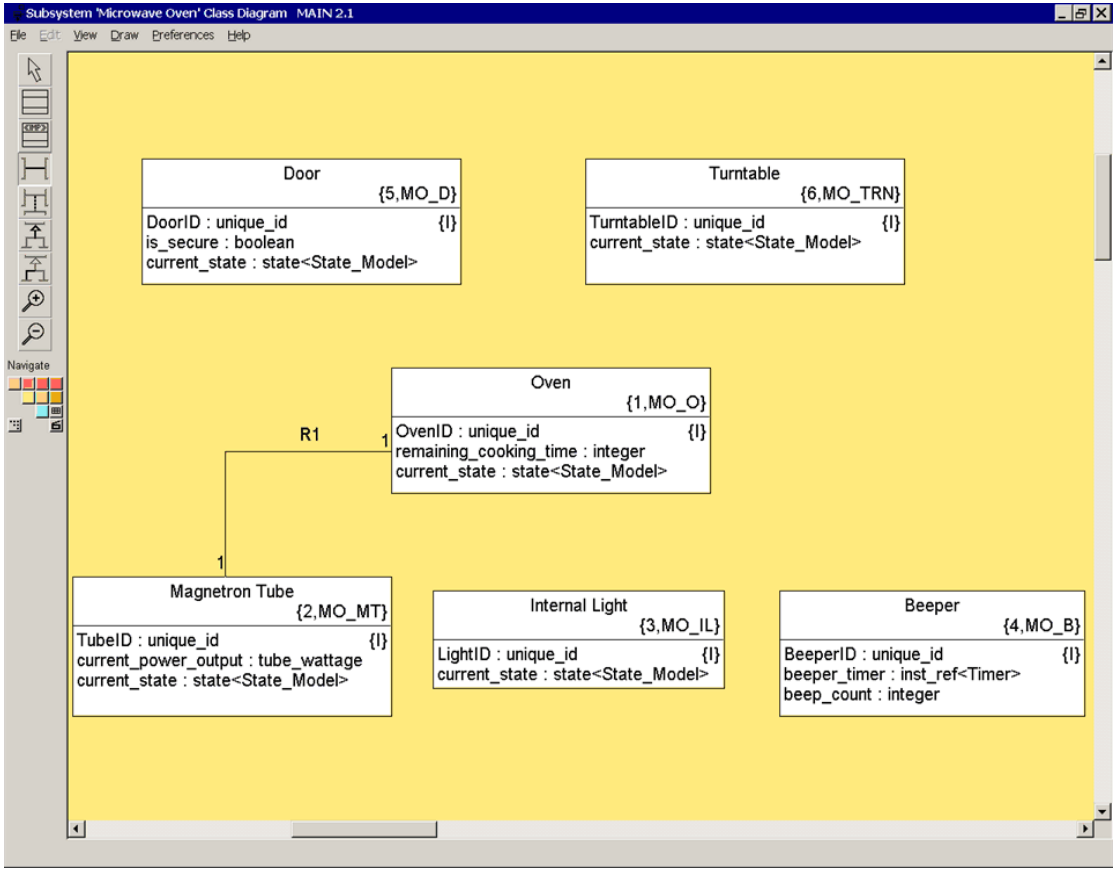

図 5-12: 関係が追加されたクラス図

- エディタウィンドウの [Oven] の側にあるドロップダウンボックスを使って [*Place Refs in this Class*] を選択することによって、Oven クラスと Magnetron Tube クラスの間の関係を定式化します（図 5-13）。

- [*OK*] をクリックします。指定した変更がクラス図の関係に対して実行され、関係データエディタが閉じます。

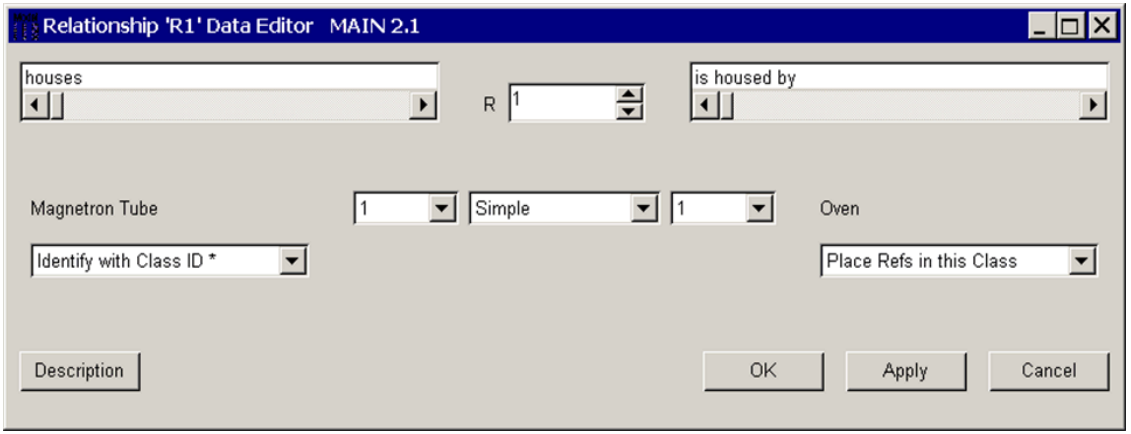

図 5-13: 関係データエディタを使った関係の定式化

クラス図では、R1 関係が引き続き選択されています。関連するテキスト要素をクリックし、適切だと思われる任意の位置へドラッグできることに注意してください。

- 描画キャンバス上で何も表示されていない位置をクリックし、R1 関係の選択を解除します。

- 学習したばかりの方法を使って、次の関係を追加します。

1. R2（*Internal Light と Oven の間*）

- Oven *is illuminated by* Internal Light
- Internal Light *illuminates* Oven
- 関係のタイプ : 1 対 1
- Oven クラスに参照を設定することによって関係を定式化

2. R3（*Oven と Beeper の間*）

- Oven *features* Beeper
- Beeper *is located in* Oven
- 関係のタイプ : 1 対 1
- Oven クラスに参照を設定することによって関係を定式化

3. R4（*Oven と Door の間*）

- Oven *is accessed via* Door
- Door *provides access to* Oven
- 関係のタイプ : 1 対 1
- Oven クラスに参照を設定することによって関係を定式化

4. R5（*Oven と Turntable の間*）

- Oven *has* Turntable
- Turntable *occupies* Oven
- 関係のタイプ : 1 対 1
- Oven クラスに参照を設定することによって関係を定式化

🗀 関係の定式化のために設定した参照を表示するために、[Oven] クラスボックスのサイズを変更します。

🗀 [*View*]|[*Snap Grid On/Off*] の順に選択して、スナップグリッドをオンにします。

🗀 クラス図のレイアウトを好きなように変更します。

🗀 [*View*]|[*Snap Grid On/Off*] の順に選択して、スナップグリッドをオフにします。最終的なクラス図は、図 5-14 に示されているクラス図のようになります。

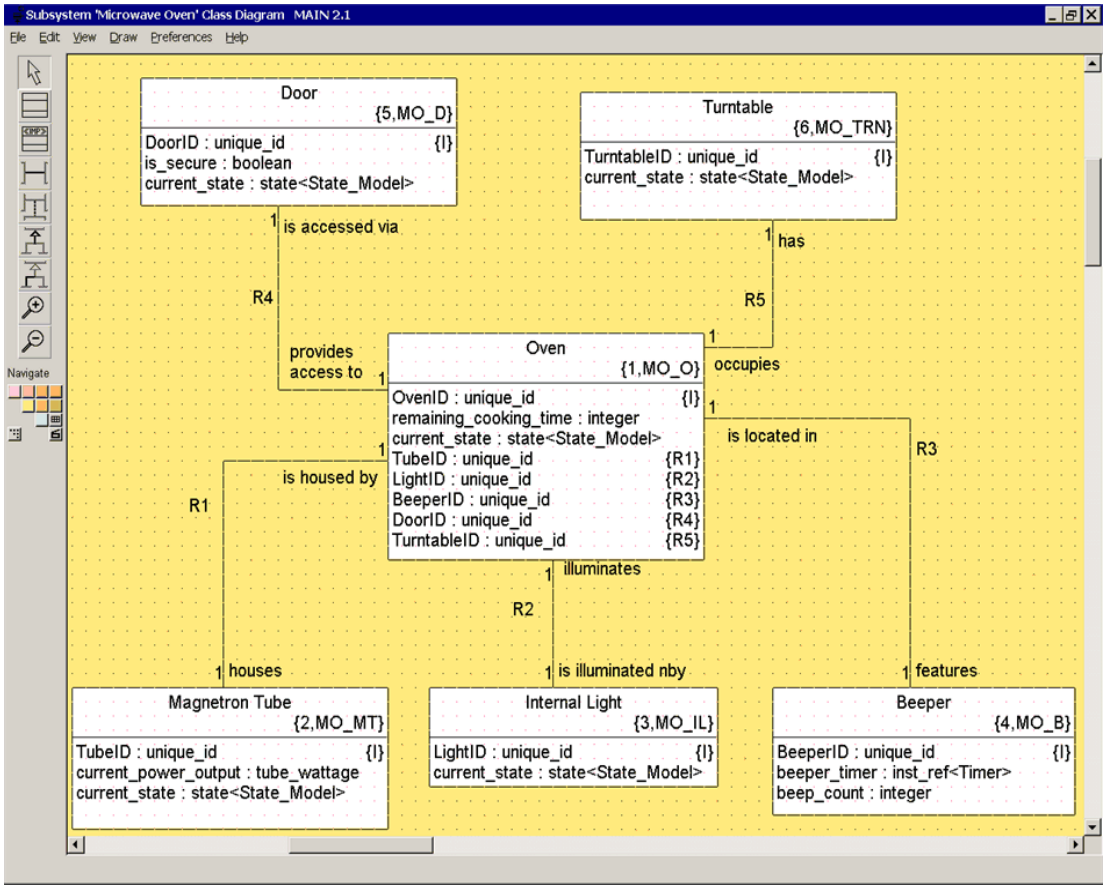

図 5-14: 最終的なクラス図

- クラス図エディタを閉じます。
- Model Builder インデックスウィンドウを閉じます。

5.5 演習 3 のまとめ

この演習では、新しいリポジトリとモデルワークスペースを作成することによって、モデリングプロジェクトを開始しました。また、DPD エディタを使ってドメインパッケージ図を作成し、Microwave Oven サブシステムパッケージを識別してラベルを付け、ドメインに新しいデータ型を追加しました。最も重要なことは、クラス、属性、および関係から成る、Microwave Oven サブシステムのクラス図を作成したことです。次の演習では、Oven クラスの振る舞いを記述するステートチャートの作成について学習します。

6

演習 4: ステートチャートの作成

6.1 ステートチャートエディタを開く

この演習では、前の演習で作成した Oven クラスのステートチャートを作成します。クラスデータエディタウィンドウの [*Instance Statechart*] チェックボックスをオンにすると、クラスの current_state 属性が作成され、関連するステートチャートコンポーネントの作成が Model Builder に対して指示されることを思い出してください。作成されたステートチャートは、Model Builderインデックスウィンドウの[*Statechart*] スクロールボックスから使用できます。各クラスの [*Instance Statechart*] プロパティの指定は、この時点まで延ばすことができますが、便宜上、先に行いました。

- Model Builder インデックスウィンドウを表示します。

- mo.ooa モデルワークスペースを開くか、[*Recent Workspaces*] リストの [mo.ooa] を選択します。既存のワークスペースを開くために必要な手順については、35 ページの「既存のモデルワークスペースを開く」を参照してください。

[Microwave Oven] サブシステムが選択されると、ステートチャートを持つことが指定されているすべてのクラスのリストが（クラス図上で定義されている current_state 属性を介して）、Model Builder インデックスウィンドウの [*Statechart*] セクションに表示されます（図 6-1）。

- [Microwave Oven] サブシステムが選択されていることを確認します。

- Model Builder インデックスウィンドウの [*Statechart*] セクションにあるスクロールバーを使って、[Oven] クラスを検索し、選択します。ダイアグラムに変更を加える前に、ダイアグラムの編集可能なバージョンをチェックアウトする必要があります。

- Model Builderインデックスウィンドウの [*Statechart*] セクションにある [*Main 1.0*] という構成管理ボタンをクリックして、Oven クラスの編集可能バージョンをチェックアウトします。

[Main 1.0]|[Check-Out Latest From Domain Repository]

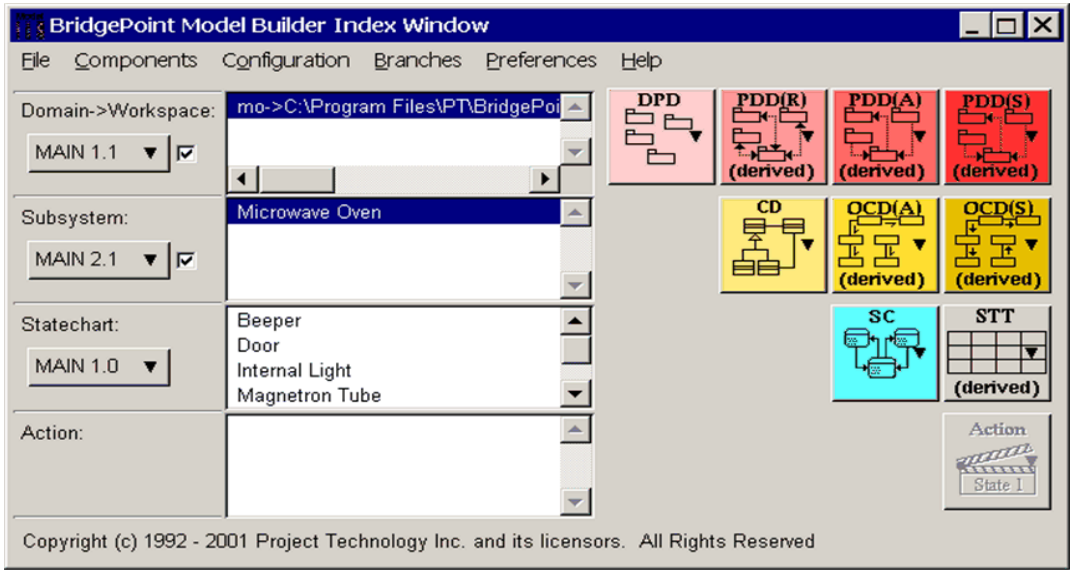

図 6-1: Model Builder インデックスウィンドウ

6.2 ステートチャートへの状態の追加

- [Component Check-Out] ダイアログボックスが表示されたら、[*OK*] をクリックして、デフォルト値をすべてそのまま使用します。

- ステートチャートがチェックアウトされたので、これを開いて編集できます。Model Builder インデックスウィンドウの [Statechart] セクションで、スクロールボックスの右側にある [*SC*] ボタンを左クリックします。ポップアップメニューが表示されます。ポップアップメニューの [*Show*] を選択します。空のステートチャートエディタが図 6-2 のように表示されます。

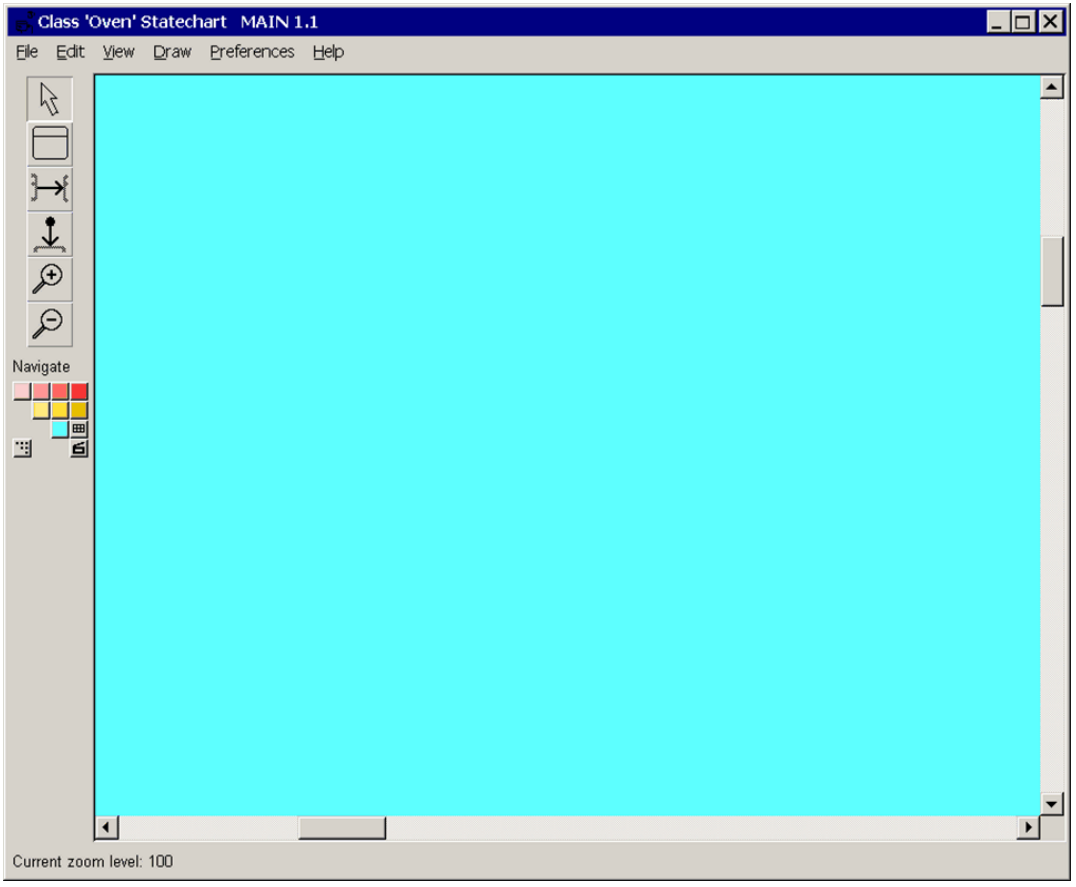

図 6-2: 空のクラスステートチャート

- 状態描画ツール（描画ツールパレットの上から 2 番目のボタン）をクリックし、描画キャンバスにカーソルを移動します。カーソルが矢印から鉛筆に変化します。これは、ステートチャートに状態シンボルを追加できる状態であることを表します。

- 表示する状態ボックスの左上角の位置にカーソルを置き、左クリックし、ボタンを押したまま、状態ボックスの右下角の位置までカーソルをドラッグします。マウスボタンを離します。ステートチャートに状態シンボルが作成されます。

- 矢印描画ツールをクリックし、ステートチャートへの状態の配置を終了します。

- 状態シンボルをダブルクリックします。状態データエディタが表示されます。

- [*State Name*]テキストボックスにおいて、状態名を`Awaiting Cooking`に変更し（図 6-3 を参照）、[*OK*] をクリックします。ステートチャートが更新され、加えたばかりの変更が反映されます。

- 最初の状態 (`Awaiting Cooking`) で使用した手順と同じ手順を使って、次の状態をステートチャート図に追加します。

  - `Ensuring Safe to Cook` （状態番号 2）
  - `Cooking` （状態番号 3）
  - `Cooking Suspended` （状態番号 4）
  - `Signaling Cooking Period` （状態番号 5）
  - `Assigning Cooking Period` （状態番号 6）

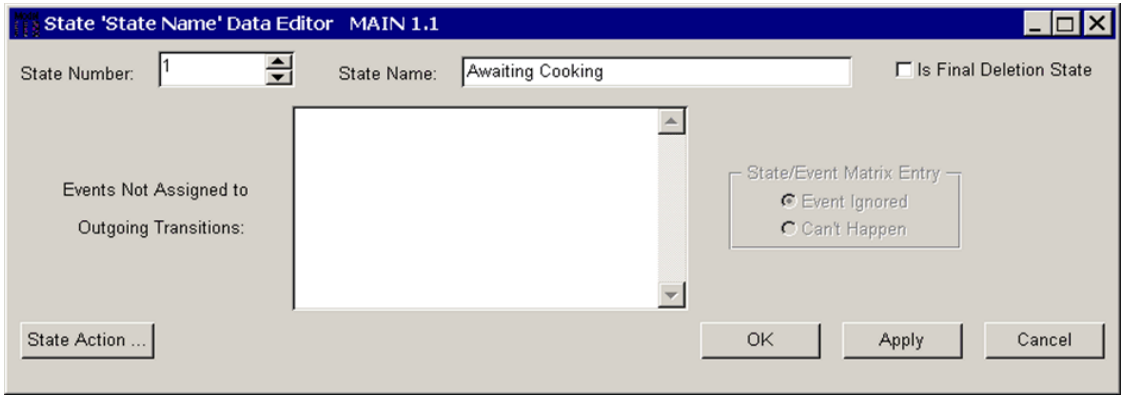

図 6-3: 状態データエディタ - [Awaiting Cooking] 状態

これらの状態をステートチャートに配置すると、ステートチャートは図 6-4 のようになります。

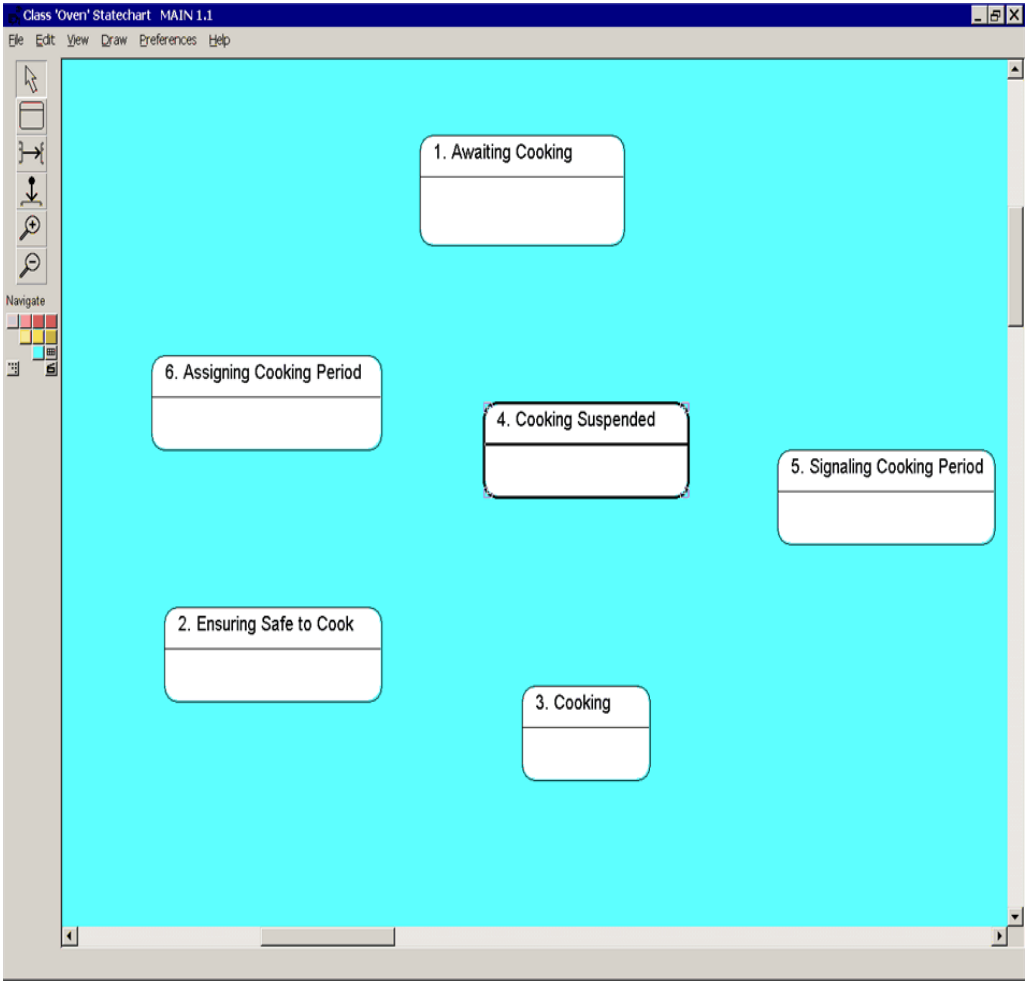

図 6-4: Oven クラスのステートチャート（状態のみ）

6.3 遷移の追加

次に、ダイアグラムに遷移を追加します。最初に、状態 1（Awaiting Cooking）と状態 6 （Assigning Cooking Period）の間に遷移を作成します。

- 遷移描画ツール（描画ツールパレットの上から 3 番目のボタン）を選択します。
- 状態 1 （Awaiting Cooking）の内側にカーソルを置きます。
- 左クリックし、ボタンを押したままにします。
- アングルポイントの位置へカーソルを移動します。
- マウスボタンを離します。
- 左クリックし、ボタンを押したままにします。
- 状態 6 （Assigning Cooking）の内側にカーソルを移動します。
- マウスボタンを離します。
- カーソルツールをクリックして、遷移描画ツールを無効にします。

遷移の追加が終了すると、新しく作成した遷移に [No Event Assigned] というラベルが付けられます（図 6-5）。イベントを遷移に割り当てる前に、イベントを定義する必要があります (注意 : ステートチャートに状態と遷移が設定される前を含めて、イベントはいつでも定義できます)。

- 描画キャンバス（状態ボックスや遷移以外の場所）を右クリックします。ポップアップメニューが表示されます。
- [*Show Event Data Editor*] を選択します。ステートチャートのイベントデータエディタが表示されます。
- エディタウィンドウの左下にある [*Insert*] ボタンをクリックします。エディタウィンドウの左側にある空のスクロールボックスに、新しいイベントが定義済みのイベントリストとして表示されます。

- [*Event Meaning*] テキストボックスに「cooking_period」と入力し、[*Apply*] をクリックします。定義済みのイベントリストのイベントエントリが更新され、このアクションが反映されます。

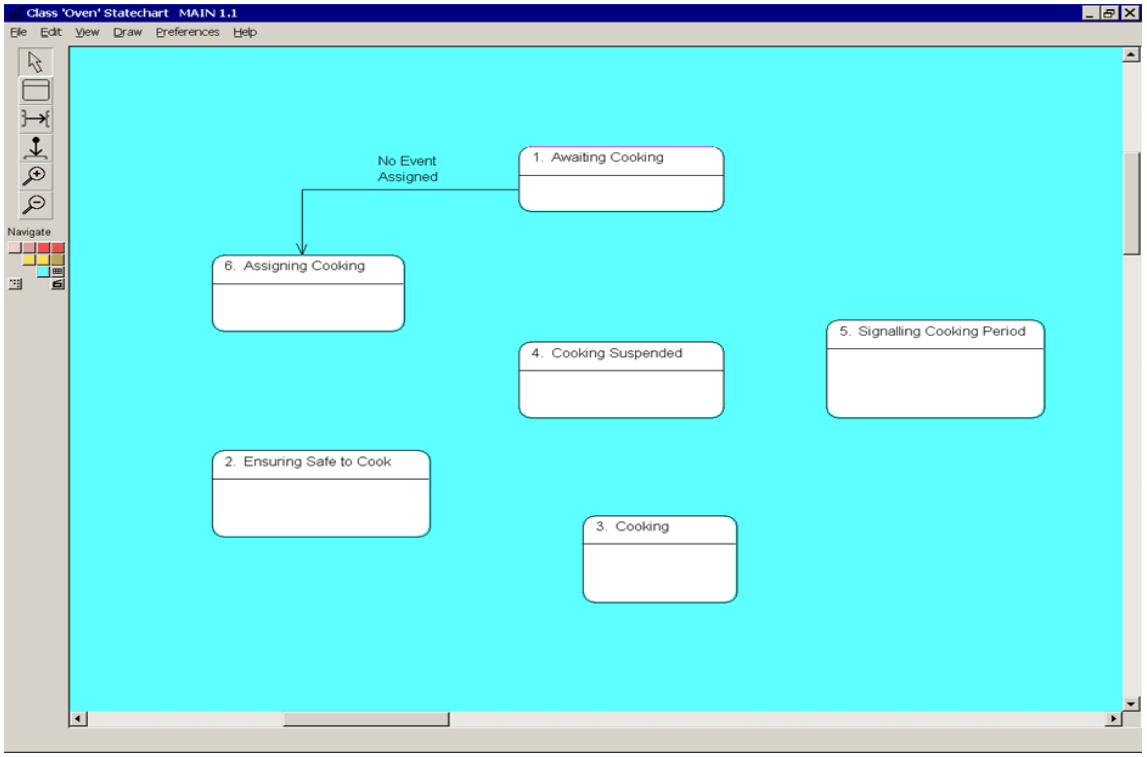

図 6-5: ステートチャート上の最初の遷移（イベントが割り当てられていない状態）

このイベントは、パラメータが関数に渡されるのとほとんど同じように、状態アクションに渡されるデータ項目を保持しています。このデータ項目の名前は period です。データ項目をイベントに関連付ける前に、データ項目を定義する必要があります。

- [*Edit Data Items*] ボタンをクリックします。

ステートチャートのイベントデータ項目エディタが表示されます。エディタウィンドウの左側にある、定義済みのデータ項目が表示されるスクロールボックスには、現在、定義されているデータ項目はありません。

- [*Insert*] ボタンをクリックします。
- [*Name*] テキストボックスに「period」と入力します。
- ドロップダウンメニューを使って、データ項目のデータ型を [integer] に設定します。
- [*Apply*] をクリックします。
- [*Edit Events*] ボタンをクリックします。

[*Event Parameters*] リストにエントリが表示されます。これが、定義済みのイベントデータ項目です。リストにエントリがあっても、これがイベントに関連付けられているわけではありません。エントリが存在し、イベントに関連付けられることを単に意味しています。

- 定義済みのイベントデータリストの [MO_01:cooking_period] イベントを選択します。
- [period] イベントパラメータをクリックし、これを選択したイベントに関連付けます（注意 : Ctrl キーを押しながらクリックすると、選択したイベントとイベントパラメータとの関連を解除できます。試してみてください）。
- イベントデータ項目 (period) が、MO_01:cooking_periodのパラメータとして、イベントリストに表示されます。
- [*OK*] ボタンをクリックします。ステートチャートのイベントデータエディタが閉じます。定義済みのイベントを、前に作成した遷移に関連付けられる状態です。
- 遷移を左クリックして、選択します。
- 右クリックしてポップアップメニューを表示します。
- ポップアップメニューの [*Show Transition Data Editor*] を選択します。
- 表示されたイベントリストから、[MO-01:cooking_period[period] イベントを選択します。

- [*OK*] をクリックします。遷移データエディタが閉じて、目的のイベントのラベルが遷移に付けられます。

イベントラベルは、体裁を整えるために、場合によっては調整が必要です。このような調整を行うには、次の手順を実行します。

- 遷移を選択します。イベントラベルが、サイズ変更ハンドルを含むボックスで強調表示されます。

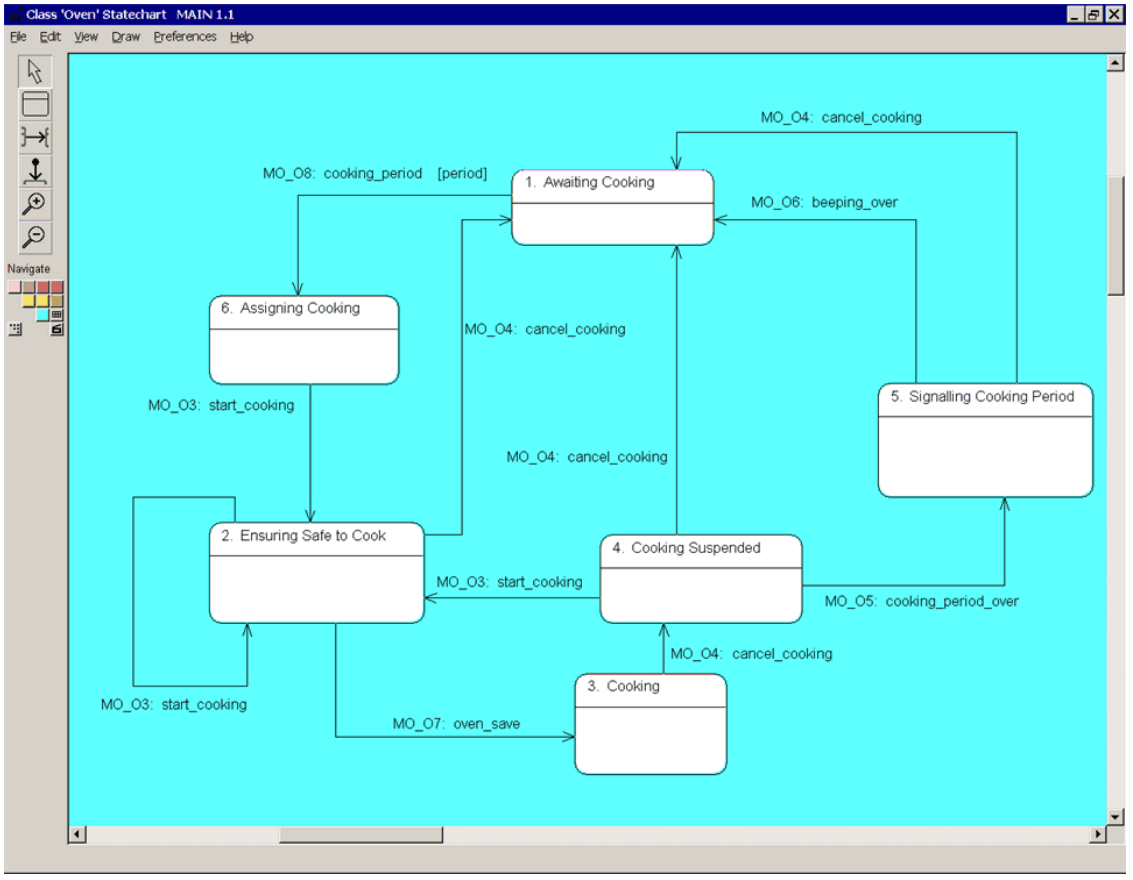

図 6-6: すべての遷移が定義された状態

- サイズ変更ハンドルの 1 つをクリックし、ボタンを押したまま、ラベルの目的の外観に合わせてドラッグします。目的の外観になったところで左マウスボタンを離します。

- テキストボックスの内側を左クリックします（ボタンは押したままにします)。
- ラベルを目的の位置までドラッグし、ボタンを離します。
- 描画キャンバスの任意の位置をクリックし、遷移の選択を解除します。
- イベントデータエディタを使って次のイベントを定義します。
  - MO-03:start_cooking
  - MO-04:cancel_cooking
  - MO-05:cooking_period_over
  - MO-06:beeping_over
  - MO-07:oven_safe
- イベントデータエディタを使って､イベントの番号を MO_01 から MO-08 に変更します。
- イベントデータエディタを閉じます。
- 図 6-6 に示すように､状態の間に遷移を作成し､該当するイベントを各遷移に割り当てます。操作が終了すると、ステートチャートはこの図のようになります。

6.4 アクションの記述

状態アクションは、BridgePoint Object Action Language を使って記述します。この言語はモデル操作言語であり（プログラミング言語ではありません）、『Bridgepoint Object Action Language Manual』に詳細な説明があります。言語は、モデルのシミュレーションと、アプリケーションモデルからのコードの自動生成を、両方とも直接サポートします。

状態アクションは、入力された時点でモデルの他の部分に対して検査して、すべての参照モデルコンポーネントが定義されていることを確認できます。これは、入力されたアクション言語を解析することによって行われ、入力されたアクションの記述が、モデル要素と比較されます。

- [Ensuring Safe to Cook] という名前の状態 2 を選択します。
- 右クリックして、選択した状態に対して実行できる操作のポップアップメニューを表示します。
- [Show State Data Editor] を選択します。図 6-7 に示す状態データエディタが表示されます。
- 状態データエディタウィンドウの左下角にある [*State Action*] ボタンをクリックします。

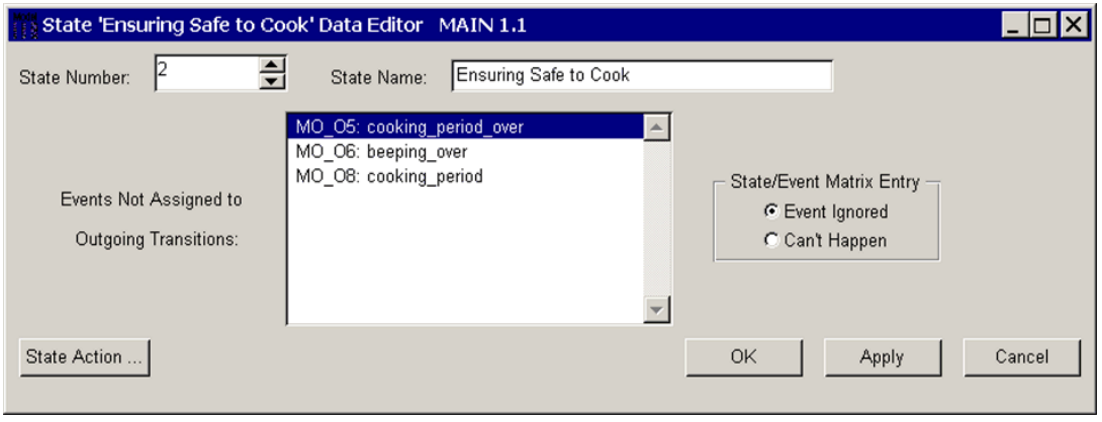

図 6-7: 状態データエディタ

- すぐに表示されない場合は、[*Action Specification*] タブを選択します。

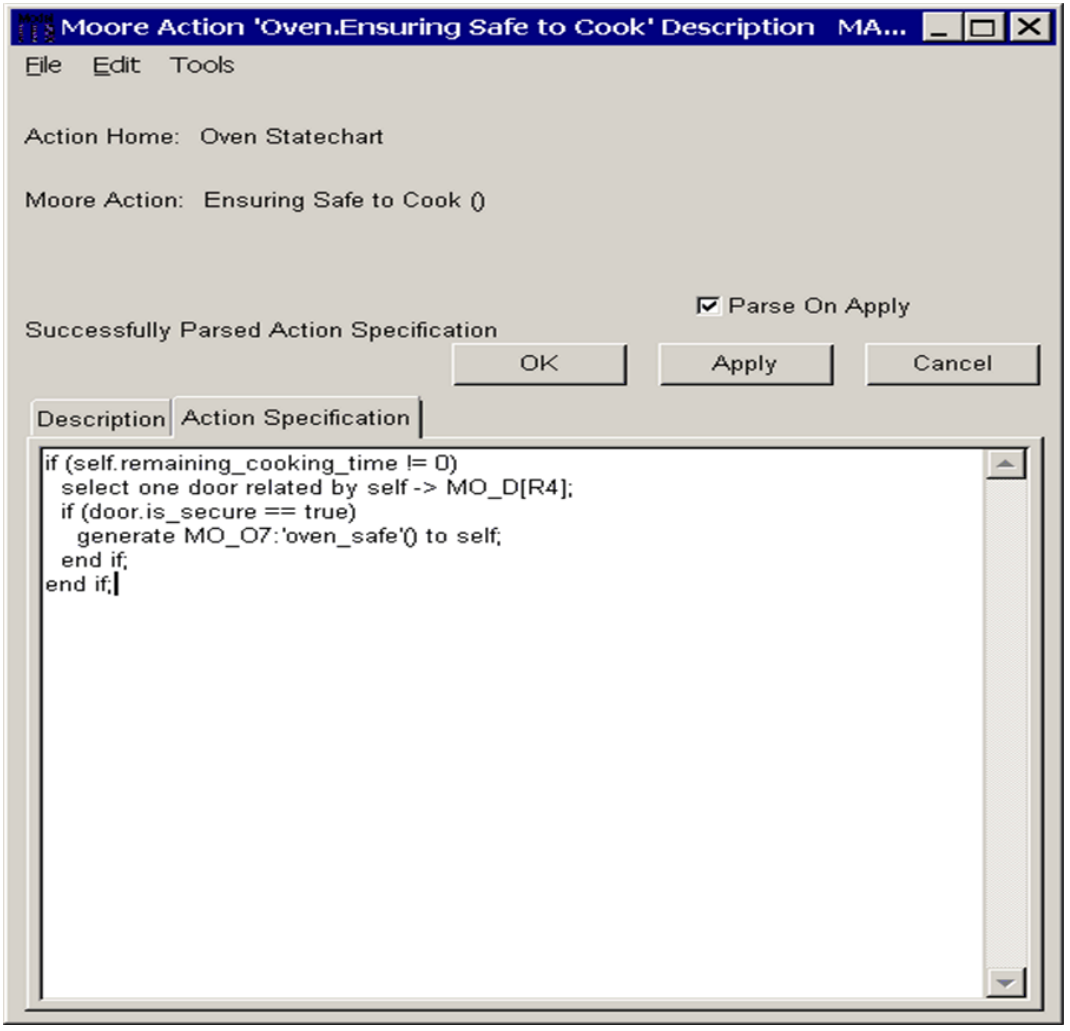

図 6-8: Moore アクション記述エディタ

- 図 6-8 に示すように、次のアクション仕様を入力します。

```
if (self.remaining_cooking_time != 0)
  select one door related by self -> MO_D[R4];
  if (door.is_secure == true)
    generate MO_O7:'oven_safe'() to self;
  end if;
end if;
```

- [*Parse On Apply*] チェックボックスをオンにします（図 6-8 を参照）。

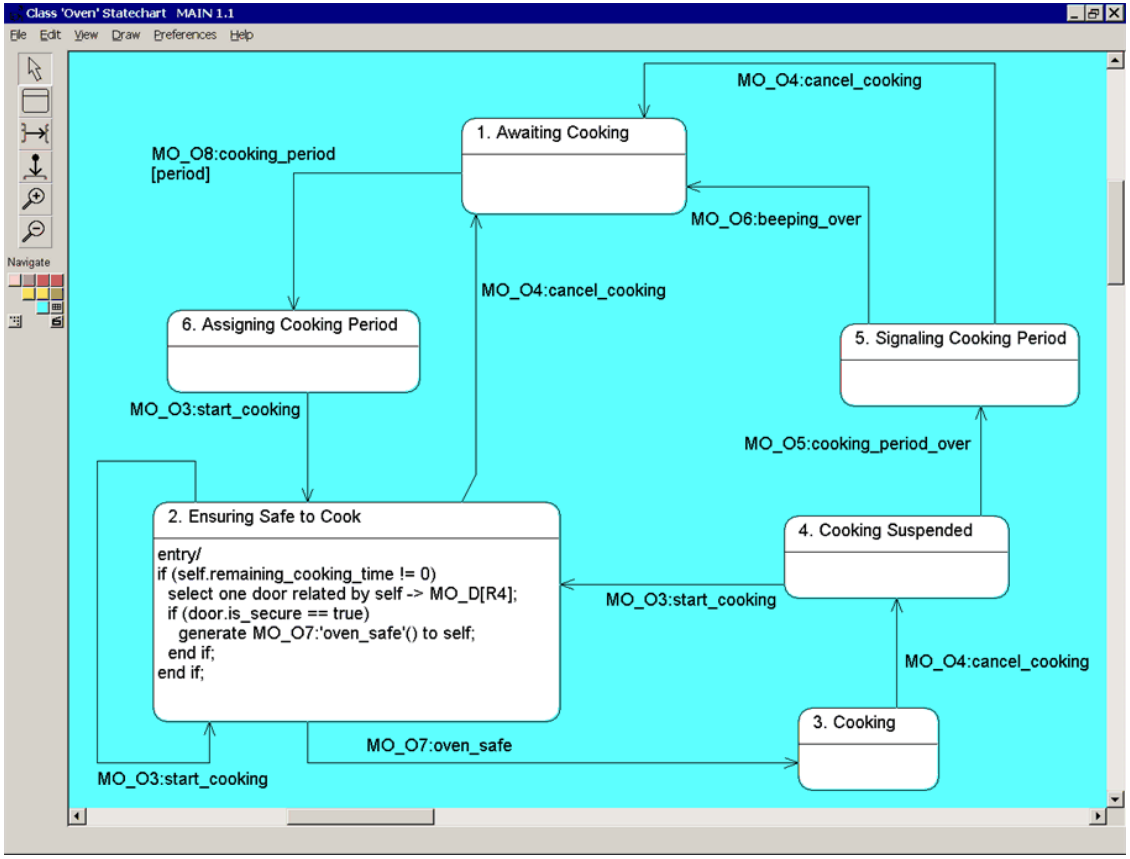

図 6-9: 正しく入力された状態アクション - [Ensuring Safe to Cook]

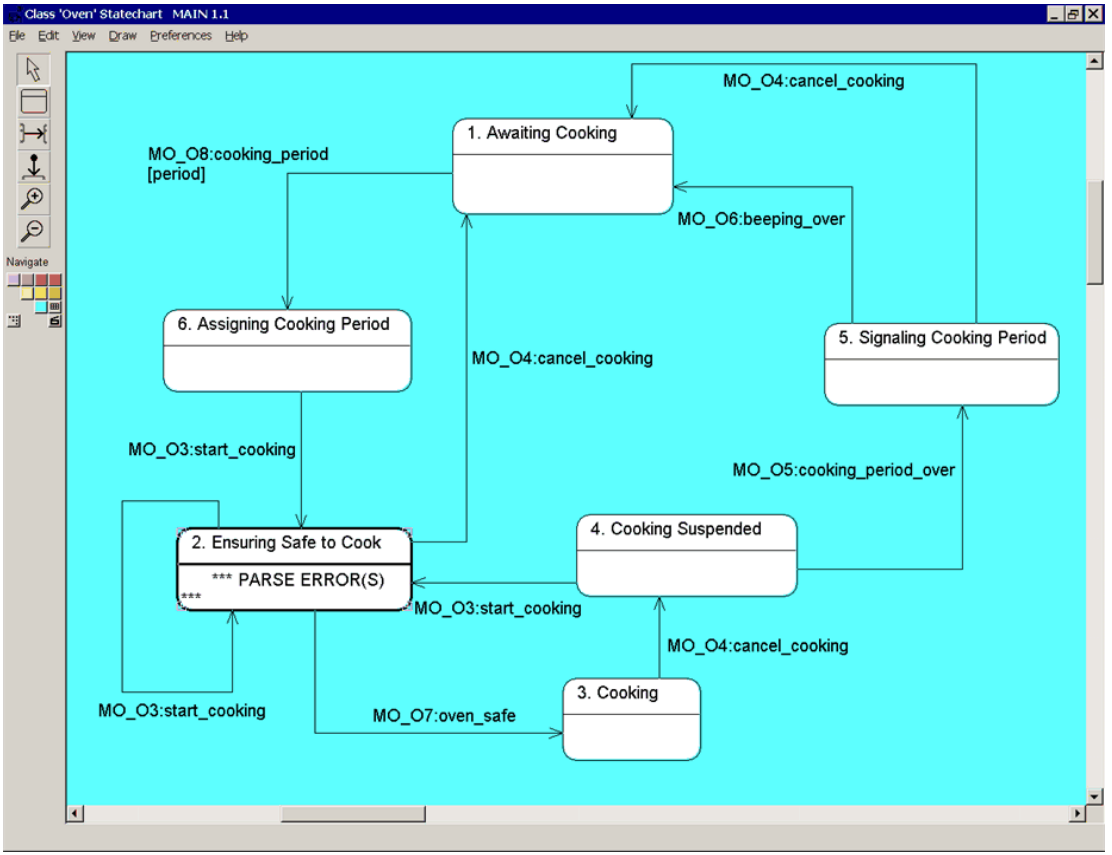

図 6-10: 正しく入力されていない状態アクション

- [*Apply*] をクリックします。アクションが正しく入力されている場合は、データベースに入力され、ステートチャート上で `[Ensuring Safe to Cook]` 状態に関連する 2 番目の領域にその内容が表示されます (図 6-9)。入力時にエラーがあった場合は、エラーメッセージが表示され、ステートチャート上の状態ボックスの2番目の領域に`"***PARSE ERRORS***"`という文字列が表示されます(図6-10)。

- アクションの解析が正しく行われたら、[*OK*] をクリックしてアクション記述エディタを閉じます。

- ステートチャート編集ウィンドウ、およびそれ以外のデータウィンドウやキャンバスウィンドウを閉じます。Model Builder インデックスウィンドウから BridgePoint Model Builder をシャットダウンします。

このチュートリアルにおいて、モデリングの演習はこれで終わりです。次の章では、BridgePoint Modeling Suite に付属している microwave oven サンプルモデルの完成品を元に、モデルの検証と変換について説明します。

もちろん、完成モデル (`microwave.ooa`) を参考にして、モデルの残りを `mo.ooa` ワークスペースに入力することもできます。[*DPD*]|[*Notebook*] の順に選択すると、プロジェクトノートブック機能を使って、完全な microwave ドメインのノートブックを印刷できます。この演習を行うと、ツール、異なるモデル要素、およびアクション言語に対する理解が深まります。

7

演習 5: アプリケーションモデルの検証

7.1 アプリケーションモデルのテスト

モデルを作成し、監査機能（付録 A を参照）を使って静的な検証を行った後、モデルを実行してその動的なプロパティを調べることができます。モデル実行の説明に先立ち、いくつかの用語を定義しておく必要があります。

制御スレッド
システムが特定の状態にあるときに、特定の任意イベントの到着に応答して発生するアクションとイベントのシーケンス。制御スレッドは、システムモデル全体の処理パスを定義します。このようなスレッドは、任意イベントによって開始され、スレッドに関連して処理が必要なイベントがこれ以上なくなると終了します。

任意イベント
システムで管理される外部ソースからのイベントで、前のシステムアクションに応答して生成されたものではありません。

応答型イベント
システムで管理される外部ソースからのイベントで、前のシステムアクションに対する応答としてシステムに要求されます。

決定属性
アクションによって生成されるイベントが、current_state 属性以外の属性値に基づくというロジックが、アクションに含まれることがあります。このような属性によって、アクションを通しての制御スレッドの展開が決まるために、これらは決定属性と呼ばれます。

シナリオ
分析対象のシステムのモデルを調べるために定義された、任意イベントのシーケンス (およびその発生時間)。シナリオによっては、アプリケーションモデルを通して複数の制御スレッドを開始します。

モデルを単に実行するというだけでは、あまり意味がありません。テストプランのコンテキストで実行することによって、正常な状態と異常な状態の両方において、モデルがどのような振る舞いを示すかを、系統的に調べる必要があります。モデルテストプランの詳細はこのチュートリアルの範囲を超えますが、目的のモデルテストを定義する場合に守らなければならない基本方針があります。この方針は、次の 3 つのステップから成ります。

- システムの初期状態を設定します。

  ここでは、必要なすべての初期インスタンスの作成、インスタンス属性の適切な値の設定、および作成したインスタンス間で必要な関係の確立が要求されます。目的のテストに対して定義された初期状態は、テストの事前条件とも呼ばれます。

- シミュレーションを実行します。

  1 つ以上の任意外部イベントのシーケンスとその発生時間を定義し、モデル実行を行うシナリオを提供します。シミュレーションの際、これらのイベントは指定された時間に処理されます。シミュレーションの進行に伴い、内部生成されたイベントはイベントキューに設定され、$^{x}_{T}$UML 時間規則に従って処理されます。テストしている制御スレッドに関するイベントが、イベントキューにこれ以上残っていない状態になると、シミュレーションは終了します。

- 実行の結果を評価します。

  テストの終了時、モデルの状態は、テストが開始したときの状態とは異なります。結果として得られるこの状態は、テスト終了時に存在するクラスインスタンス、そのインスタンスの属性値、およびその間の関係として定義されます。

  結果の評価には、1) 最終的な属性値が相互にすべて一貫していること、2) 元の任意イベントを受け取った結果として、目的の応答がシステムで生成されていることの検証も含まれます。

7.2 BridgePoint Model Verifier

この演習のポイントは、BridgePoint Model Verifier が備えている機能を認識することで、これをモデルテストプランのコンテキスト内で効果的に使用できるようになることです。前に定義した 3 つのステップから成るストラテジに照らし合わせた場合、Model Verifier は、初期システム状態の確立、任意イベントの受信、およびドメイン内のアクションとイベントのシーケンスで定義される制御スレッドの実行と記録を、直接サポートします。シミュレーション結果の評価は、モデルテストを定義したソフトウェアエンジニアまたはシステムエンジニアの仕事です。

BridgePoint Model Verifier には、『Object Lifecycles: Modeling the World in States [1]』に記載されている時間規則が組み込まれています。この規則は、状態アクションを処理する方法と条件、アプリケーションモデル内でのデータの一貫性の保持、およびイベントの生成方法と処理方法に対処するものです。この演習を実行するにあたって時間規則を理解していることは前提条件ではありませんが、Model Verifier のシミュレーション機能を実際のプロジェクトで使用する場合は、あらかじめ規則を理解することが推奨されます。

Model Verifier では、一度に 1 つのドメインのテストと検証がサポートされます。シナリオの実行結果は、2 つのファイルに記録されます。2 つのファイルのうち、1 番目のファイルの拡張子は `.sim` です。このファイルを管理することで、シミュレーションの開始、休止（Model Verifier からの実際の終了を使用）、および実行の継続が可能です。2 番目のファイル（拡張子は `.sim_log`）はテキストファイルであり、シミュレーションの結果を記録するために使用されます。このファイルには、シミュレーションの際に発生するイベントの詳細レコードが ASCII 形式で含まれており、後処理することで、目的のテストケースの結果を評価するために必要な情報を取り出すことができます。

1. Sally Shlaer、Stephen J. Mellor 著『Object Lifecycles: Modeling the World in States』Englewood Cliffs, NJ: Yourdon Press 発行、1992 年

7.3 初期システム状態の確立 - Initialization Subsystem

今回の演習では、すでに完成し、検証済みの microwave oven モデルのバージョンを使用します。この例の場合、初期システム状態の確立は、`Initialization` Subsystem によって行います。これは、初期システム状態を定義するいくつかの方法の中の 1 つです。シナリオやテストケースの中でシステムの初期状態を直接定義する方法もあります。

- Model Builder を使って、`microwave.ooa` モデルを開きます（[*File*]|[*Open*]）。図 7-1 の Model Builder インデックスウィンドウが表示されます。

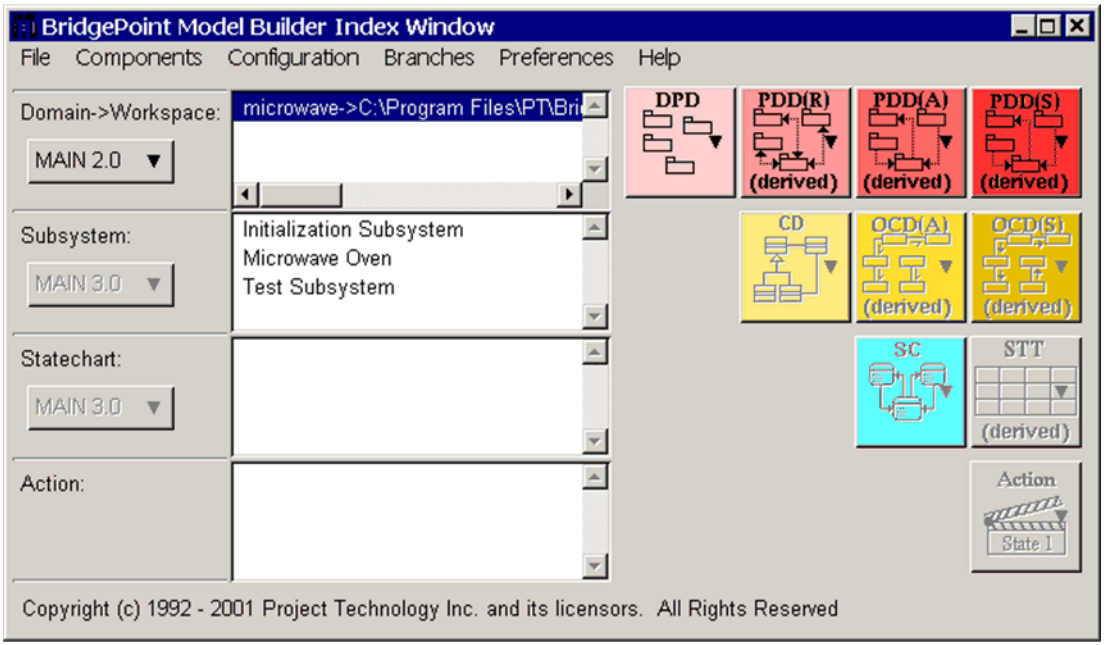

図 7-1: microwave.ooa が Model Builder にロードされた状態

`Microwave Oven`モデルの[*Subsystem*]セクションを見ると、3つのサブシステムがあることがわかります。その 1 つは`Microwave Oven` サブシステムであり、このチュートリアルの以前の演習でその一部を作成したサブシステムに似ています。このサブシステムは、Microwave Oven アプリケーションそのもののモデルです。

この他に、2 つのサブシステムがあります。1 つのサブシステムは Initialization Subsystemであり、シミュレーションアクティビティをサポートする上で必要な初期クラスインスタンスを確立するために使用します。もう 1 つは Test Subsystem であり、各種テストケースが記述されます。

- Model Builderインデックスウィンドウで、[Initialization Subsystem]を選択します。
- Initialization Subsystem のクラス図を開きます。ダイアグラムには、[Initialization Subsystem init]という名前のクラスが1つだけ表示されます。
- クラス図に表示されているクラスに関連するステートチャートを開きます。ステートチャートに表示されている 1 つの状態を選択します。

```
// Create the instances in the system.
create object instance mo of MO_O;

assign mo.remaining_cooking_time = 0;

create object instance tube of MO_MT;
relate mo to tube across R1;
assign tube.current_power_output = tube_wattage::high;

create object instance light of MO_IL;
relate mo to light across R2;

create object instance beeper of MO_B;
relate mo to beeper across R3;

create object instance door of MO_D;
relate mo to door across R4;
assign door.is_secure = false;

create object instance turntable of MO_TRN;
relate mo to turntable across R5;

// create an instance of test sequence, then generate the event that starts the test scenario.

create object instance ts of MO_TS;
generate  MO_TS1:'initialize'() to ts;
```

図 7-2: BridgePoint アクション言語による初期クラスインスタンスの作成

□ ステートチャートに表示されている 1 つの状態に関連するアクションを開きます。図 7-2 に示すアクション言語が表示されます。

Initialization Subsystem は、1 つのクラス（Initialization Subsystem init）とその関連するステートチャートで構成されます。ステートチャートは 1 つの状態で構成されており、その状態に関連するアクションによって（図 7-2）、この演習のシミュレーションアクティビティをサポートするために必要なMicrowave Ovenサブシステムのクラスインスタンスが作成されます。この状態に対応するオブジェクトアクション言語を調べてみると、アクション言語の create ステートメントを使って次のクラスインスタンスが作成されていることがわかります。

- Microwave Oven クラスの 1 つのインスタンス
- Magnetron Tube クラスの 1 つのインスタンス
- Internal Light クラスの 1 つのインスタンス
- Beeper クラスの 1 つのインスタンス
- Door クラスの 1 つのインスタンス
- Turntable クラスの 1 つのインスタンス

これらのクラスインスタンスの作成に加えて、これらの間に存在する関係は relate ステートメントで定義されており、これによって完全な Microwave Oven 部品が定義されます。最後に、この演習におけるシミュレーションアクティビティをサポートするために、いくつかのインスタンス属性に値が代入されます。

クラスインスタンスの作成が終了すると、Test Sequence クラスのインスタンスが作成され、テストケースを自動的に開始するためにイベント（MO_TS1）がそのインスタンスに送られます。この方法によるクラスインスタンスの作成と、そのインスタンスに対するイベントの生成は、通常、初期システムインスタンスの作成時に行われることはありませんが、ツールを紹介する目的でここに含まれています。

□ Model Builder インデックスウィンドウを除いて、開いているすべてのウィンドウを閉じます。

7.4 実行シナリオの確立（任意外部イベントの定義）

システムの初期状態が確立されると、その状態に基づくテストケース（またはシナリオ）を定義できます。

- Model Builder インデックスウィンドウの [*Subsystem*] セクションで、[Test Subsystem] を選択します。
- Test Subsystem に関連するクラス図を開きます。
- クラス図に表示されている 1 つのクラスのステートチャートを開きます。
- ステートチャート上の [State 2 - Performing Test Sequence 1] に関連するアクションを開きます。図 7-3 は、表示されるアクション言語を示しています。

この状態に関連するアクション言語は、Microwave Oven モデルが応答する必要がある外部イベントのシーケンスと、そのイベントが発生する時間を定義します。システムの初期状態と、少なくとも 1 回のテストケースの両方を定義することで、Microwave Oven モデルのシミュレーションを実行できます。

- Model Builder インデックスウィンドウを除いて、開いているすべてのウィンドウを閉じます。

```
// Step 1. At T+2000000us, open the oven door to insert dish
select any door from instances of MO_D;
if (not_empty door)
  create event instance release_door of MO_D1:'release' to door;
  step1_timer=TIM::timer_start(microseconds:2000000,event_inst:release_door);
end if;

// Step 2. At T+3000000us, lower the output power setting to Med_High
select any tube from instances of MO_MT;
if (not_empty tube)
  create event instance lower_power of MO_MT2:'decrease_power' to tube;
  step2_timer =TIM::timer_start(microseconds:3000000,event_inst:lower_power);
end if;

// Step 3. At T+4000000us, close oven door
select any door from instances of MO_D;
if (not_empty door)
  create event instance close_door of MO_D2:'close' to door;
  step3_timer =TIM::timer_start(microseconds:4000000,event_inst:close_door);
end if;

// Step 4. At T+5000000us, assign cooking period of 10 seconds and start cooking
select any oven from instances of MO_O;
if (not_empty oven)
  create event instance cooking_time of MO_O8:'cooking_period'(period:10000000) to oven;
  step4a_timer =TIM::timer_start(microseconds:5000000,event_inst:cooking_time);
  create event instance start of MO_O3:'start_cooking' to oven;
  step4b_timer =TIM::timer_start(microseconds:5000001,event_inst:start);
end if;

// Step 5. At T+15secs, open the oven door to remove cooked dish
select any door from instances of MO_D;
if (not_empty door)
  create event instance release_door of MO_D1:'release' to door;
  step5_timer =TIM::timer_start(microseconds:15000000,event_inst:release_door);
end if;

// For codegen: Testing complete.  After 30 seconds, terminate the system
// otherwise, executable will not terminate.
create event instance finished of MO_TS4:'test_seq_complete' to self;
terminate_timer=TIM::timer_start(microseconds:30000000,event_inst:finished);
```

図 7-3: オブジェクトアクション言語で記述されたテストシナリオ

7.5 Model Verifier の開始

Model Builder の場合と同様、Model Verifier を開始する方法は、使用しているプラットフォームによって異なります。Windows 環境の場合は、タスクバーの [スタート] ボタンを使って Model Verifier を開始します。

[*スタート*]|[*プログラム*]|[*BridgePoint 6.0*]|[*Model Verifier*]

Unix 環境の場合は、コマンド行で次を入力して Model Verifier を開始します。

```
pt_verifier
```

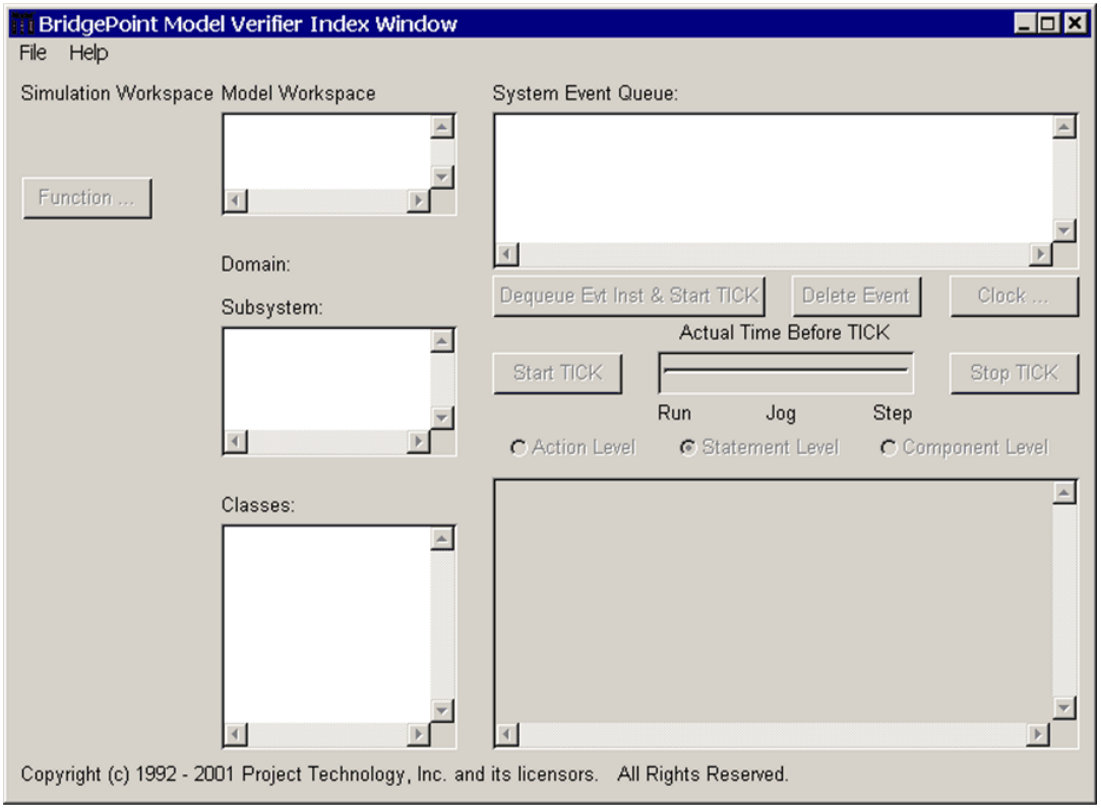

図 7-4: Model Verifier インデックスウィンドウ

図 7-4 の空の Model Verifier インデックスウィンドウが表示されます。

Model Verifier インデックスウィンドウには、シナリオの実行を開始、制御、および追跡するために適切に定義されているいくつかの領域が含まれています。ウィンドウの左上角にある表示領域は、現在使用しているモデル（.ooa ファイル）とシミュレーション（.sim ファイル）のワークスペースを表示するために使用します。ワークスペース情報の下は [*Domain*] フィールドであり、シミュレーションを介して現在分析しているドメインを示します。ドメインに関連するサブシステムは、[*Domain*] フィールドのすぐ下にあるスクロールボックスに表示されます。表示されたサブシステムのいずれかを選択すると、そのサブシステムのクラスが [*Classes*] ボックスに表示されます。

Model Verifier インデックスウィンドウの右上角にある表示領域には、シミュレーションによる処理を現在待っている、システムイベントキュー上のイベントが表示されます。右下角の表示領域は、シミュレーションのトレース情報を表示するために使用されるものであり、シミュレーションの展開に伴う、その進行を確認できます。システムイベントキューとシミュレーショントレースウィンドウの間にあるボタン、スライダ、およびラジオボタンを使用すると、イベントキューから取得した各イベントの処理の間でシミュレーションが休止するかどうか、シミュレーショントレースに表示される詳細情報のレベル、およびシミュレーションの実行速度を制御できます。

- シミュレーションの対象となる（このケースでは`microwave.ooa`モデル）をModel Verifier にロードします。.

  `[File]|[Create New Simulation Workspace ...]`

  シミュレーションの発生元であるモデルワークスペースファイル名と、シミュレーションワークスペースファイル名の両方を指定する必要があります（目的のモデルワークスペースに関連するシミュレーションワークスペースが、複数ある場合があります）。シミュレーションワークスペースファイル名のデフォルト値は、モデルワークスペース名と同じです（ファイル拡張子は異なります）。この演習では、シミュレーションワークスペースファイル名のデフォルト値を使用します。

  モデルワークスペースファイルは、[*Model Workspace File*] テキストボックスにワークスペース名を入力することによって、または省略記号が記されたボタンをクリックし、目的のモデルワークスペースを選択することによって、指定できます。

- 省略記号が記されたボタンをクリックします。

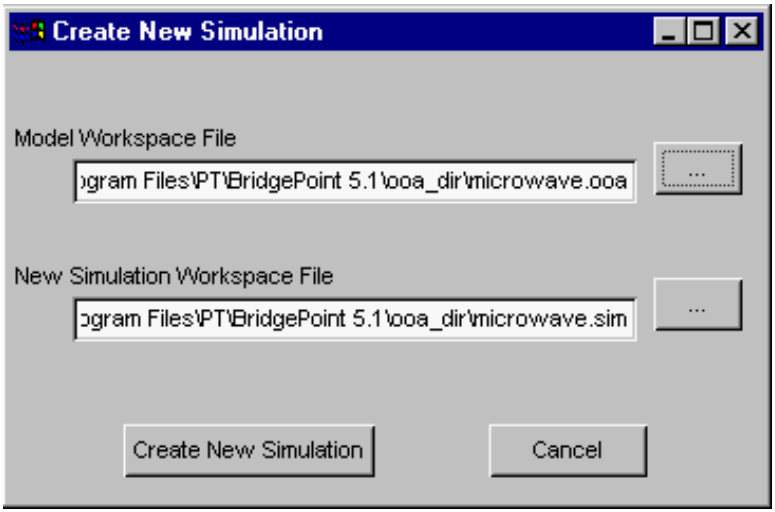

図 7-5: [Create New Simulation] ダイアログボックス

- 表示されたダイアログボックスで、`[microwave.ooa]` ワークスペースを選択します。

- [*Open*] ボタンをクリックします。この時点で、図 7-5 の [Create New Simulation] ダイアログボックスが表示されます。

- [Create New Simulation] ダイアログボックスの [*Create New Simulation*] ボタンをクリックします。

  Model Verifier に `microwave.ooa` がロードされた状態の Model Verifier インデックスウィンドウが、図 7-6 のように表示されます。

- [*Subsystem*] スクロールボックスの `[Initialization Subsystem]` をクリックします。

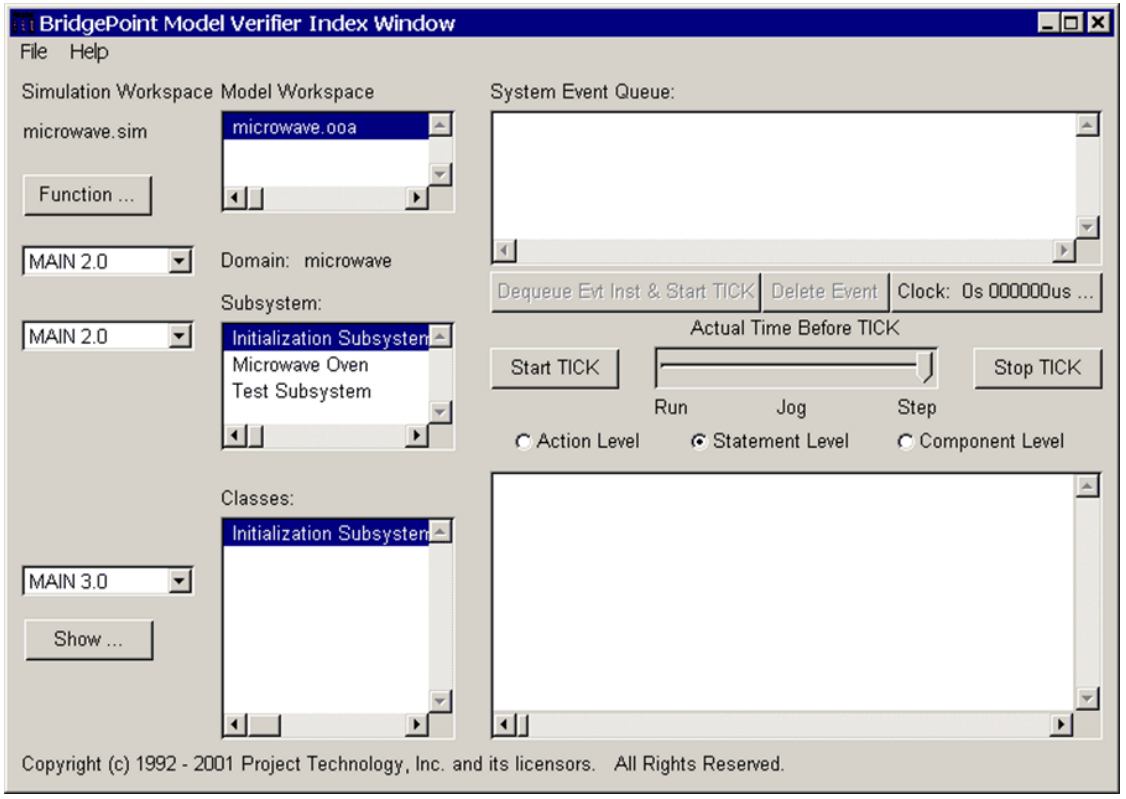

図 7-6: microwave.ooa が Model Verifier にロードされた状態

- Model Verifier インデックスウィンドウの左下角にある [*Show*] ボタンをクリックします。Initialization Subsystem init クラスのための [Class Instances] ウィンドウが表示されます。

- [*Create*]ボタンをクリックして、`Initialization Subsystem init`のインスタンスを作成します。作成されたインスタンスが [Class Instances] ウィンドウに表示されます（図 7-7 を参照）。

- [Class Instances] ウィンドウの下部にある [*Add Event*] ボタンをクリックし、`[MO-I1:'start']` イベント（クラスインスタンスに割り当てることができる唯一のイベント）を選択します。これによって、新しく作成された `Initialization`

Subsystem init のインスタンスに、イベントがキューイングされます。キューイングされたイベントは、Model Verifierインデックスウィンドウの[System Event Queue] ボックスに表示されます（図 7-8 を参照）。

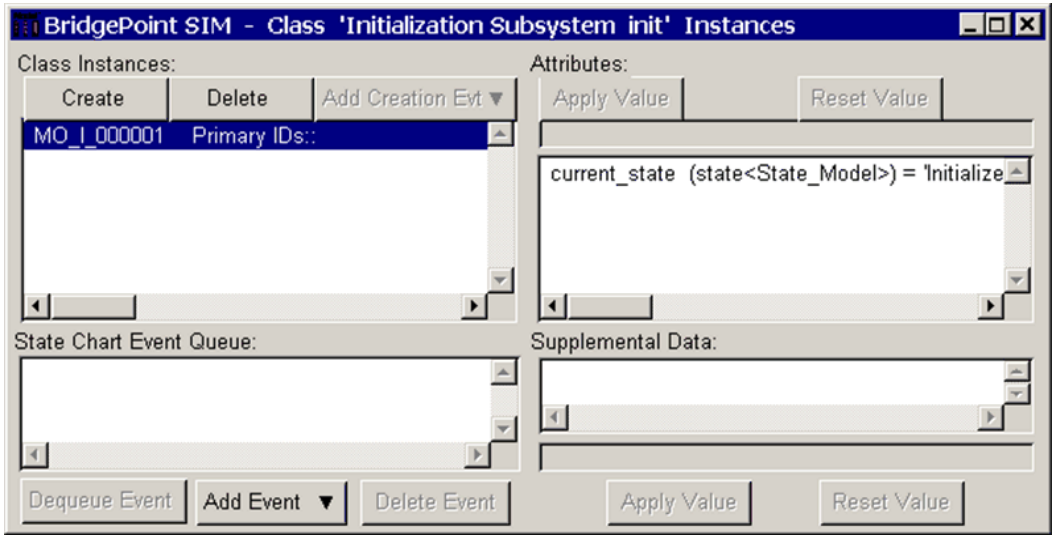

図 7-7: [Class Instances] ウィンドウ

- [Class Instances] ウィンドウの右上角にある [*X*] をクリックして、ウィンドウを閉じます。

- [System Event Queue] ボックスに現在表示されている 1 つのイベントを、左クリックして選択します。[System Event Queue] ボックスの下のボタンがアクティブになります。

- [*Dequeue Evt Inst & Start TICK*] をクリックします。以前に示した初期化のアクション言語が実行されます。

  Model Verifier インデックスウィンドウの右下角にあるシミュレーショントレースウィンドウに、実行されたステートメントが表示されます。

- Model Verifierインデックスウィンドウに表示される、[*Subsystem*] スクロールボックスの `[Microwave Oven]` を選択し、[*Classes*] スクロールボックスの `[Oven]` クラスを選択します。

- Model Verifier インデックスウィンドウの左下角にある [*Show*] ボタンをクリックして、Oven クラスのための [Class Instances] ウィンドウを開きます。スクロールボックスから直接、[Oven] クラスをダブルクリックすることもできます。図 7-9 は、[Class Instance] ウィンドウを示しています。

  Oven クラスの 1 つのインスタンスが存在することが確認できます。このインスタンスの属性とその値は、[*Attributes*] セクションに表示されます。[*StateChart Event Queue*] ボックスには、クラスインスタンスのイベントインスタンスがないことに注意してください。

- [Class Instances] ウィンドウを閉じます。

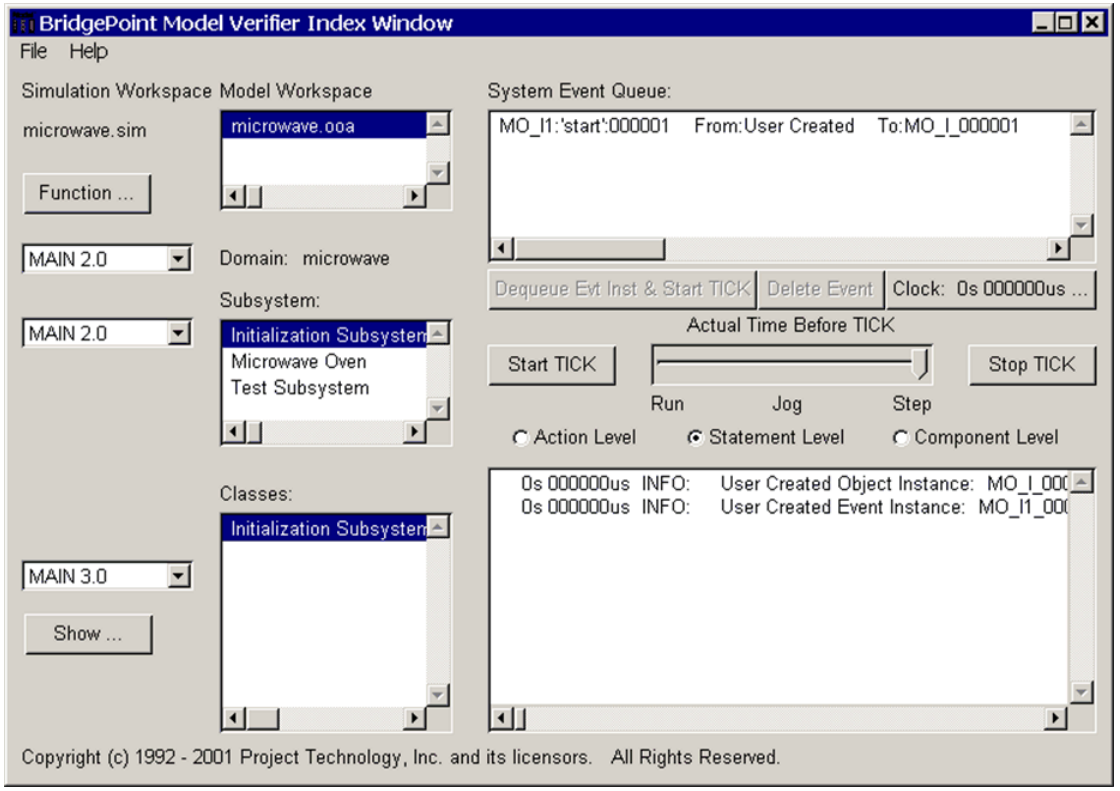

図 7-8: システムイベントキュー上の初期化イベント

システムイベントキューには、現在、1 つのイベントがあります。

- [System Event Queue] ボックスの [MO_TS1] イベントを選択します。

- [*Dequeue Event Inst & Start TICK*] をクリックします。このイベントの処理結果がシミュレーショントレースウィンドウに表示され、新しいイベントが [System Event Queue] ボックスに設定されます。

- `[MO_TS2:'perform_test_seq_1']` イベントを選択します。

このイベントが処理されると、図 7-3 のアクション言語で記述されるテストケースを定義する、アクション言語が実行されます。

- [*Dequeue Event Inst & Start TICK*]ボタンをクリックし､テストシーケンス 1 の実行を開始します。

テスト目的の遅延イベントとしてモデル化されているすべての任意イベントが、[System Event Queue] ボックスに表示されます。

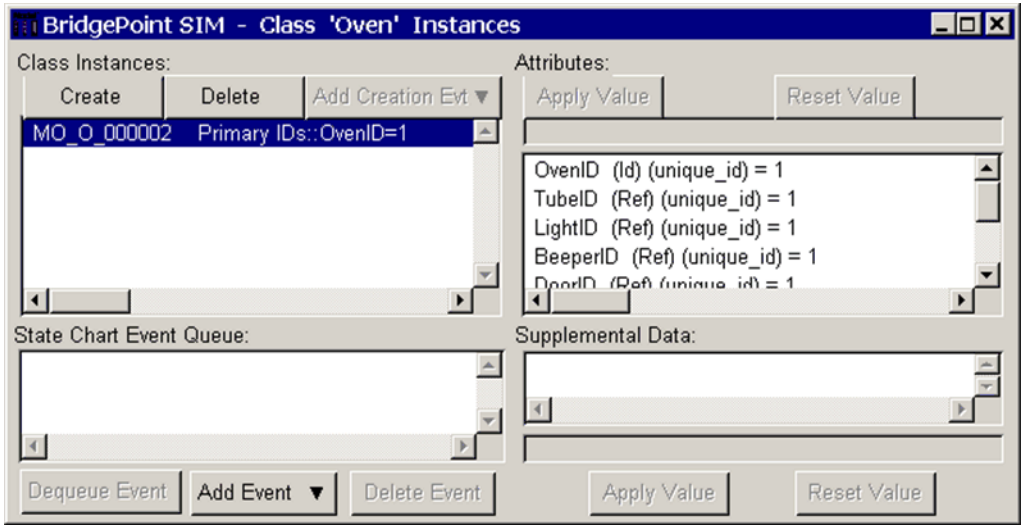

図 7-9: [Class Instances] ウィンドウにおける Oven クラスのインスタンス

システムイベントキューのイベントは、キューイングを解除して、これまで行ってきたように、1 つずつ処理することもできます。しかし、シナリオのシミュレーションを、できるだけ高速に実行するという決定が成されたものと仮定してください。

- スライダを [*Step*] から [*Run*] に移動します。

- [*Start TICK*] ボタンをクリックします。クロックが定期的に更新され、イベントが処理されます。
- システムイベントキューのイベントがなくなったら、[*Stop TICK*] をクリックします。
- Model Verifier インデックスウィンドウを閉じて、ツールの実行を終了します。.

8

演習 6: モデルからコードへの変換

8.1 MC-2020 アーキテクチャ

検証のアクティビティの後、Executable Translatable UML ($^{X}_{T}$UML) の手法における次のステップでは、検証したモデルから変換を介してコードを生成します。多くの変換が可能ですが、ここではその中の 1 つだけを扱います。この演習では、MC-2020 モデルコンパイラを使って、このような変換の実行方法を説明します。既存のアプリケーションモデルのセットを利用し、変更を加えることなく、各種プラットフォームに適したコードを生成できる機能は、変換手法の長所であり、際だった特徴であることを認識することが重要です。アプリケーションモデルとソフトウェアアーキテクチャの大規模な再利用を可能とするのが、まさにこの技術であり、ソフトウェア開発のライフサイクルの並列性と短縮の両方を実現するのも、まさにこの技術です。

MC-2020 は、Unified Modeling Language (UML) モデルから、リアルタイム、最適化、効率的という特徴を備えたコードを生成します。モデルは、各種のターゲットプラットフォーム上で実行できる C++ に変換されます。メモリ管理とイベントディスパッチの方針から、ソフトウェアのコーディング規則とファイル名にいたるまで、ソフトウェアの設計上のすべての決定は、MC-2020 によって提供されるソフトウェアアーキテクチャに含まれる、規則のセットで確立されます。MC-2020 はオープンソースのモデルコンパイラであり、目的のプロジェクトのニーズに合わせて簡単に変更できます。

注意：チュートリアルのこの章を実行するためには、MKS Toolkit 6.0 以上がシステムにインストールされている必要があります。これに加えて、Windows 開発ホスト上で実行するイメージを作成するには、Microsoft Visual C++ が必要です。

8.2 変換プロセスのステップ

Windows/MKS 環境におけるモデルからコードへの変換は、8 ステップから成るプロセスです。

ステップ 1: Korn Shell コマンドウィンドウを表示する

MC-2020 を使って Windows 環境のコードを生成するには、MKS Toolkit がインストールされている必要があります。[*スタート*]|[*プログラム*]|[*MKS Toolkit*]|[*Korn Shell*] の順に選択して、Korn Shell ウィンドウを表示します。この節でこの後に示すすべてのコマンドは、このウィンドウから実行します。次の Korn Shell プロンプト、または類似のプロンプトが、ウィンドウに表示されます。

```
$
```

ステップ 2: 変換ワークスペースを作成する

MC-2020 インストールの bin ディレクトリには、変換ワークスペースを作成するためのスクリプトが含まれています。インストールディレクトリ <install_dir> は、MC-2020 のインストール時に指定されています。デフォルトのインストールディレクトリを選択した場合、bin ディレクトリは次の位置にあります。

```
c:/PT/DesignPoint/MC-2020/bin
```

ディレクトリをこのディレクトリに変更し、次のコマンドを入力します。

```
$ ./ptc_init_workspace
```

このコマンドを入力するとプログラムが実行され、変換ワークスペースディレクトリ <workspace_dir> の作成を確認するプロンプトが、最初に表示されます。モデルの変換を行うディレクトリの名前を入力します。たとえば、「c:/PT/DesignPoint/wksp/microwave」と入力します。

次に、ビルドのコンパイルおよびリンクフェーズで使用されるビルド仕様について、入力が求められます。各種のビルド仕様が提供されており、リストで示されます。「vc_debug」と入力します。

指定したディレクトリに変換ワークスペースが作成されます。

ステップ 3: ディレクトリを変換ワークスペースのディレクトリに変更する

ディレクトリを変換ワークスペースのディレクトリに変更し、「make help」と入力します。help make target によって、使用可能なすべてのターゲットのリストが表示されます。

```
$ cd <workspace_dir>
$ make help
```

ステップ 4: 実行イメージの名前を変更する

任意のテキストエディタを使って Makefile.user ファイルを編集します。PROGRAM 変数で定義される実行ファイルの接頭辞を、ptc から microwave に変更します。

```
PROGRAM = ptc${BUILD_EXT}
```

上の行を、次のように変更します。

```
PROGRAM = microwave${BUILD_EXT}
```

ファイルを保存し、次のステップに進みます。

ステップ 5: Microwave ドメインを変換ワークスペースに追加する

ドメインを変換ワークスペースに追加する make コマンドを実行します。

```
$ make domain=microwave dom_node
```

ドメインがワークスペースに追加されます。

ステップ 6: カラーリング情報を指定する

変換を実行する前に、変換を管理するモデルコンパイラに対して、情報を指定する必要があります。最初に、モデルコンパイラの実行エンジンの動作を確立する、システムレベルのカラーリング情報を追加します。

ディレクトリをシステムレベルのカラーリングディレクトリに変更します。

```
$ cd system/color
```

任意のテキストエディタを使って system.clr ファイルを編集します。ファイルに次の行を追加して、保存します。先頭のドット "." を忘れずに指定してください。

```
.Invoke TagOoaThreadConcurrency(2)
```

次に、ドメインレベルのカラーリング情報を追加します。以下の情報によって、分析モデル内のオブジェクトの 1 つが、初期化オブジェクトとして設定されます。

ディレクトリを、microwave ドメインのドメインレベルのカラーリングディレクトリに変更します。

```
$ cd ../../microwave/color
```

任意のテキストエディタを使って domain.clr ファイルを編集します。ファイルに次の行を追加して、保存します。先頭のドット "." を忘れずに指定してください。

```
.Invoke TagInitializationObject("MO_I")
```

ドメインレベルのカラーリングが完成しました。次のステップに進み、モデルをコードに変換できます。

ステップ 7: モデルをコードに変換する

すべての make コマンドは、最上位レベルの変換ワークスペースディレクトリから実行する必要があります。ディレクトリを変更した上で、変換、コンパイル、および実行イメージへのコードのリンクをすべて行う make コマンドを入力します。

```
$ cd ../..
$ make all
```

make コマンドと各種のシェルスクリプトによって、モデルからコードへの変換が調整されます。高いレベルの視点で捉えた、変換の際に行われる処理を以下に示します。これは、出力で確認できます。

- アプリケーションモデル全体が分析され、その結果が生成データベースに保存されて、現在の変換における後の段階で使用されます。情報は、モデルに関する統計情報を提供するレポートに書き込まれます。レポートは、reports ディレクトリに作成されます。
- カラーリング情報が読み込まれて、適用または拒否されます。この時点で、可能であれば、モデル内の警告やエラーがレポートされます。エラーと警告は reports ディレクトリに保存されます。
- モデルの変換が開始されます。mircrowave oven モデル内のオブジェクト、イベント、およびアクションのクラスが生成され、microwave/gen/include および microwave/gen/source に保存されます。
- 初期化オブジェクトが特別な実装に変換されます。microwave/gen/include/microwave_dom_init.hpp ファイルおよび microwave/gen/source/microwave_dom_init.cpp ファイルを確認してください。
- システムレベルファイルが system/gen/include および system/gen/source に作成されます。
- コードがコンパイルされ、1 つのイメージにリンクされます。

変換プロセスの終了には数分間かかります。make all コマンドが終了すると、実行イメージとして、bin ディレクトリに microwave.exe ファイルが作成されます。

ステップ 8: 実行ファイルを実行する

前のステップで作成された実行ファイルは bin ディレクトリに保存され、microwave.exe という名前が付けられています。microwave プログラムを実行するには、次を入力します。

```
$ cd bin

$ ./microwave
```

microwave アプリケーションが、分析モデルの指定に従って実行します。`microwave/color/debug.clr`内のカラーリング情報によって、アクション言語ステートメントのトレースをオンにすることが MC-2020 に指示されます。実行イメージが実行している間、モデルの実行に伴い何が起きているかを示す出力を確認できます。

付録 A

Brigedpoint の監査機能

A.1 xTUML モデルの完全性と一貫性

xTUML のモデルは、シミュレーションや変換を実行できるように、あらかじめ密接に統合し、完全かつ一貫していなければなりません。BridgePoint Model Builder は、この完全性と一貫性の検査を実行できる監査機能を備えています。また、すべてのモデルコンポーネントについて、ドキュメント（要素記述の形式）が記述されていることも確認できます。このようなドキュメントは、シミュレーションや変換には必要ありませんが、これらが提供されない限り、モデリングアクティビティは完全とは見なされません。

アプリケーションモデルに対して、全部で 35 の検査を実行できます。どの検査も、ユーザが個別に有効または無効にできます。Model Builder インデックスウィンドウから [*User Preferences*] メニューにアクセスすると、これらの検査のリストが表示されます。

- [*Preferences*]|[*User*]
- ドロップダウンリストの [*Audits*] を選択

図 A-1 のダイアログボックスが表示されます。このダイアログボックスでは、提供されている検査のいずれかを有効または無効にできます。このリストに変更を加えた場合は、[*OK*] ボタンまたは [*Apply*] ボタンをクリックして、その変更を反映する必要があります。

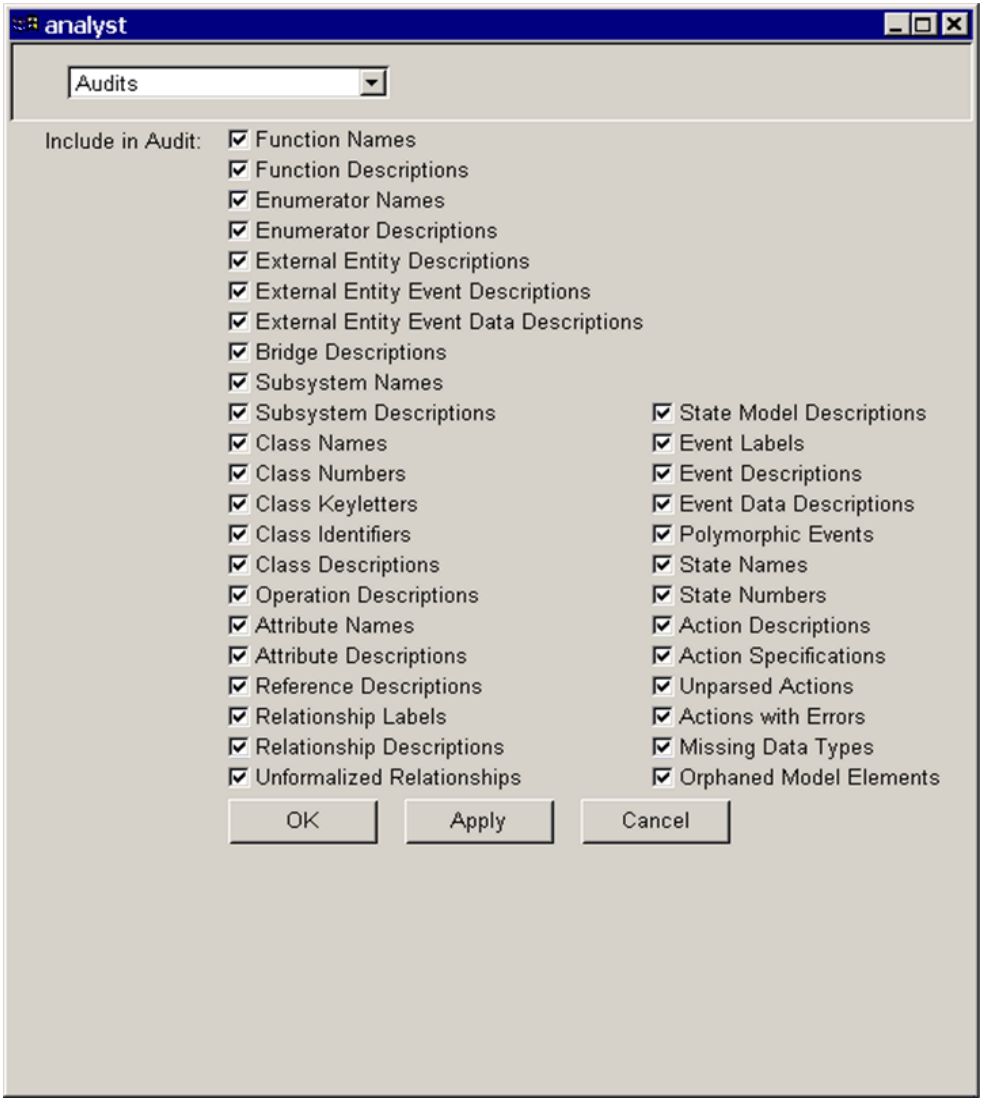

図 A-1: Model Builder の監査機能を通して使用できる検査

A.2 必須の検査

図 A-2 は、シミュレーションや変換を実行する前にパスする必要がある検査（チェックボックスがオン）を示しています。

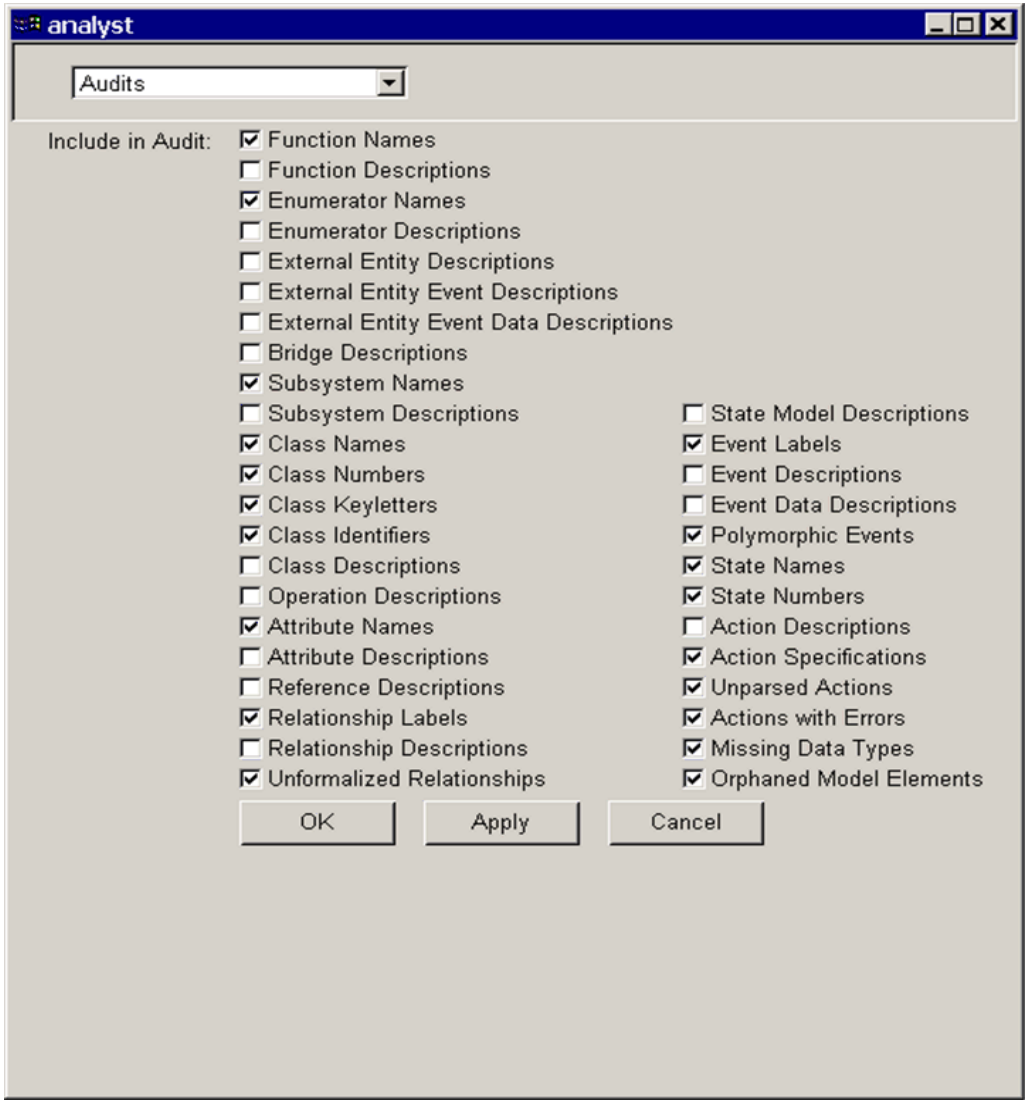

図 A-2: シミュレーションや変換に必要な検査

A.3 ドキュメント検査

図 A-3 は、すべてのモデルコンポーネントが記述されていることを確認する検査を示しています。

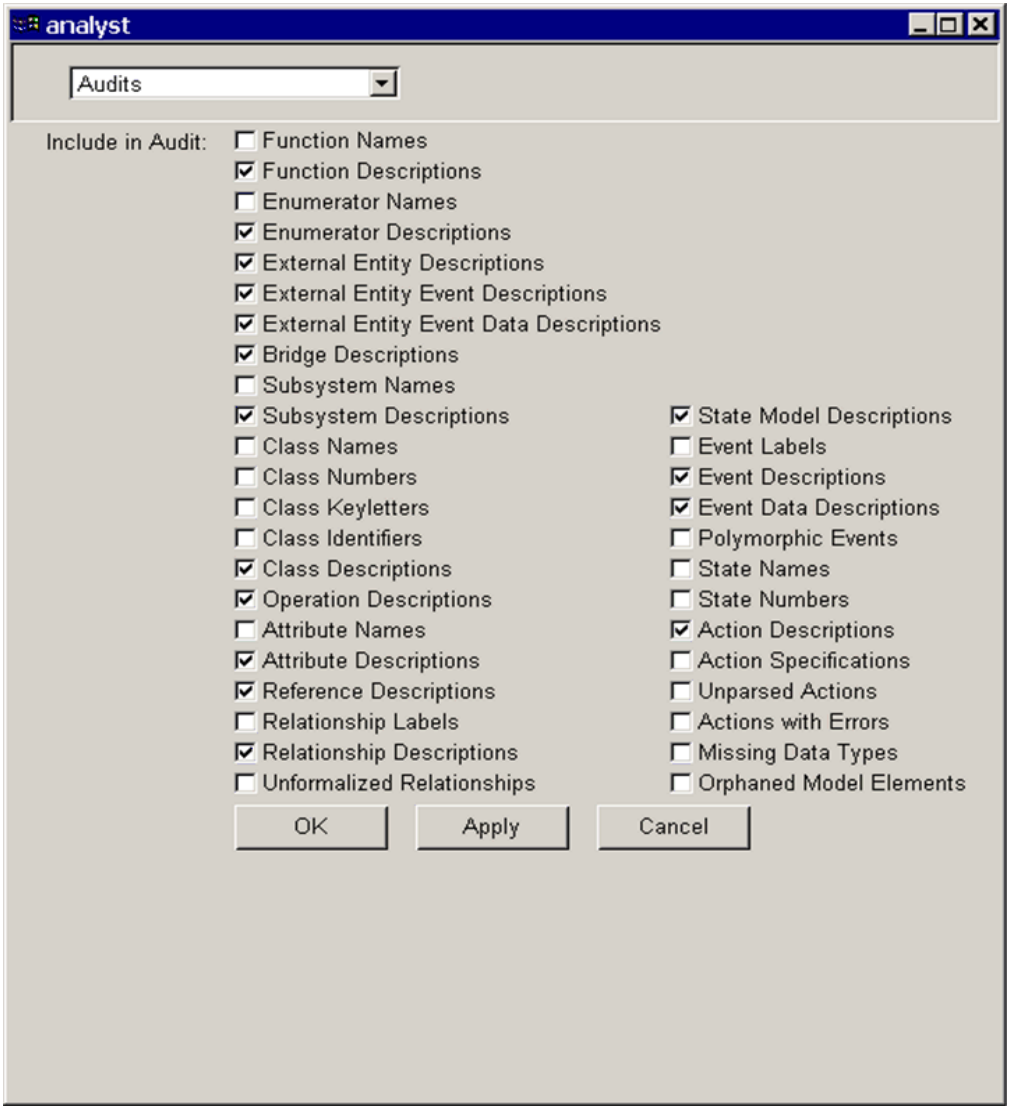

図 A-3: ドキュメントが適切であることを確認する検査

A.4 Model Builder インデックスウィンドウからの監査機能の起動

BridgePoint のモデル監査機能は、Model Builder のさまざまな場所から起動できます。監査機能が起動された位置が、監査の範囲（検査されるモデル）に影響します。

Model Builder インデックスウィンドウからの監査機能の起動は、次のいずれかのボタンに対応するポップアップメニューから、監査機能を表すメニュー項目を選択することによって実現されます。

- [*DPD*]
- [*CD*]
- [*SC*]

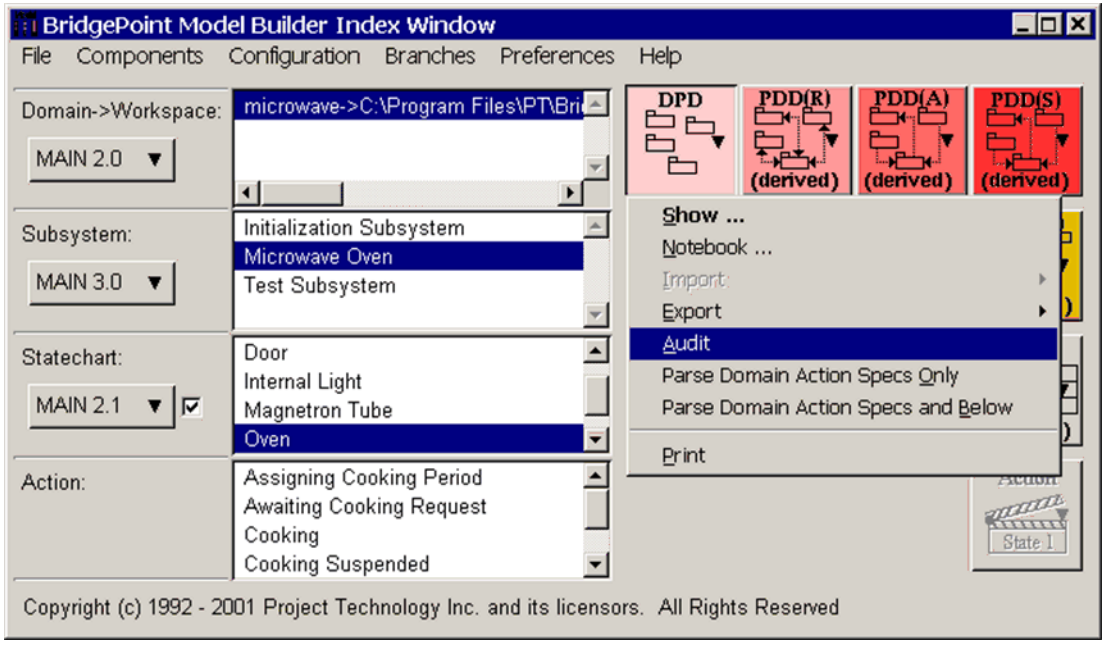

図 A-4: [DPD] ボタンからの監査機能の起動

図 A-4 は、[DPD] ボタンからの起動を示しています。

[*DPD*] ボタンから監査機能を起動すると、モデルセット全体（ドメイン、サブシステム、ステートチャート、およびアクション）が検査されます。これに対して、他のボタンから監査機能を起動した場合は、限定的な検査が実行されます。[*CD*] ボタンから起動される監査では、クラス図、ステートチャート、およびアクションを検査する、サブシステム全体の監査が実行されます。ドメインレベルでの検査は実行されません。[*SC*] ボタンから起動される監査では、現在選択されているステートチャートと関連するすべてのアクションが検査されます。

A.5 ダイアグラムエディタからの監査の起動

監査機能は、Model Builder インデックスウィンドウ以外の場所からも起動できます。具体的には、任意のダイアグラムエディタウィンドウの [File] メニューから起動できます。図 A-5 は、クラス図エディタからの起動例を示しています。

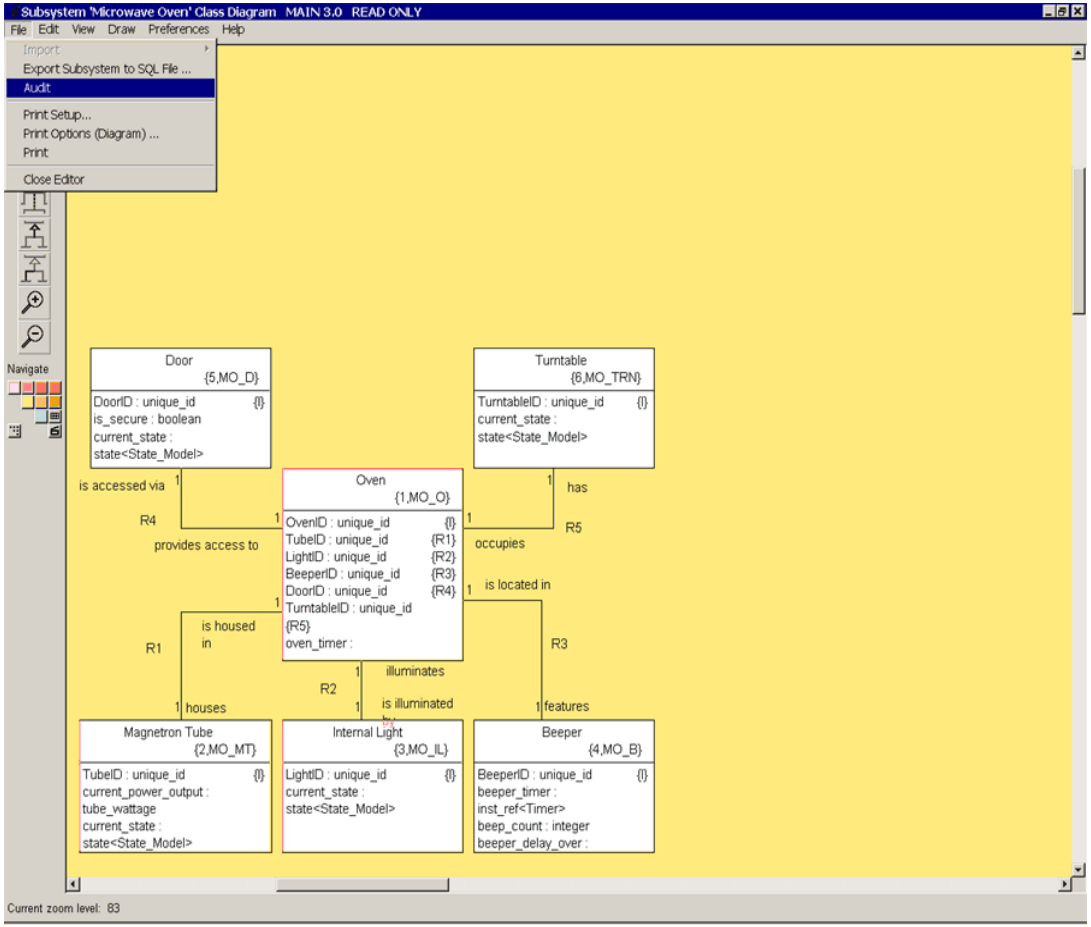

図 A-5: クラス図エディタからの監査機能の起動

ダイアグラムエディタウィンドウから起動される監査機能の範囲は、Model Builder インデックスウィンドウから起動される監査機能の範囲に比べて限定されています。この場合の監査機能は、ダイアグラムエディタに関連するダイアグラムがその対象であり、他のモデルコンポーネントに対しては検査を実行しません。