

BridgePoint 版 鹿威し 解説書

FirstRelease Apr12 2004

V1.0

(株)リコー 斎藤 賢一

目次

1	はじめに	3
2	MDA 概要	3
3	開発プロセス概要	4
4	要求仕様	5
5	モデル解説	6
5.1	ユースケース	6
5.1.1	ユースケース図	6
5.1.2	ユースケース記述	6
5.2	ドメインとブリッジ	8
5.2.3	ドメインチャート	8
5.2.4	ドメイン記述とブリッジ記述	8
5.3	クラス	9
5.3.5	クラス図	9
5.3.6	クラス説明	9
5.3.7	関連の説明	10
5.4	ステートマシン図とアクション記述	12
5.5	コラボレーション図	13
5.6	同期サービス	14
6	モデルコンパイラによる自動コード生成	16
6.1	コード生成概略	16
6.2	色づけ概要	16
6.3	コード生成	17
6.3.1	環境設定	17
6.3.2	システムノードの作成	17
6.3.3	OO ドメインフォルダの作成	17
6.3.4	フォルダ構成	18
6.3.5	色づけ	18

6.3.6	コード生成	22
7	設計指針	24
7.1	モデルコンパイラ的设计指針	24
7.2	モデルコンパイラのカスタマイズ	24
7.3	タスク関関図	24
8	手書きコードとモデルとのインタフェース	25
9	実装環境	26
9.1	ビルド	26
9.2	デバッグ	28
9.3	ROM 化	29
9.4	実行	29

1 はじめに

近年、組み込みソフトウェアの業界において、UML を用いたオブジェクト指向手法は急速に広まり、また、最近では MDA (Model Driven Architecture : モデル駆動型アーキテクチャ) 技術を用いた開発手法も実用段階に入り、組み込みソフトウェアの開発手法が変わりつつある。

このような背景から、SESSAME (組み込みソフトウェア管理者・技術者育成研究会) で紹介された “ 鹿威し (ししおどし) ” の仕様 ((株) 東陽テクニカ 二上氏作 「セサミ電気 Sozex002 ソフトウェア仕様書 v1.2」) をもとに、TOPPERS/JSP カーネルを利用し、かつ、MDA 技術を用いて “ 鹿威し ” の開発を行ってみた。

MDA 技術には組み込み業界で実績のある Executable UML の概念を適用し、CASE ツールには BridgePoint 6.1D (Project Technology 社) を使用した。また、モデルをコードに変換するツールとして、同じ Project Technology 社のモデルコンパイラ (MC-3020 v3.3) を使用した。

実機環境としては、オークス電子 (株) の OAKS16-MINI キットのボードを使用し、統合開発環境として、キットに付属の TM V.3 を使用した。

TOPPERS/JSP カーネルは jsp-1.4 を使用した。

2 MDA 概要

MDA は UML によるモデルベースのオブジェクト指向開発をさらに進化させる、モデリング主導のシステム開発、およびライフサイクル管理を実現するための参照アーキテクチャである。

「駆動」という言葉には「まずモデルありき、モデルが定義されることで特定言語や製品へのマッピングは自動的に決定し、さらにシステム管理、インテグレーションもモデルを中心として行うことを可能にする」という意味が込められている。

MDA は対象となるアプリケーションの機能要求による仕様と実装技術 (プログラミング言語、OS、ミドルウェアなどのプラットフォーム) を分離して開発する方法を提供する。モデルを各種のプラットフォーム技術へとマッピングすることによって、プラットフォーム技術の変化に対応することのできる堅牢なフレームワークを提供する。仕様と実装技術を分離することによって、高度な自動化を可能にし、従来に比べて高い生産性を実現する。

MDA では、プラットフォームに対して、実装技術 (プラットフォーム) に依存しないモデルと依存するモデルを定義する。

- ・ PIM (Platform Independent Model) = プラットフォーム独立モデル : 実装技術非依存
- ・ PSM (Platform Specific Model) = プラットフォーム固有モデル : 実装技術依存

PIM はシステムの本質 (機能要求) を表す分析モデルであり、PSM は実装技術の情報が付加された設計モデルである。

MDA によるシステム開発は、機能要求による仕様を表現する分析モデル (PIM) を作成し、非機能要求を考慮した設計モデル (PSM) へ変換 (マッピング) した後コードを生成する。

3 開発プロセス概要

Executable UML の開発プロセスから実装、ROM 化までのプロセスを以下に示す。

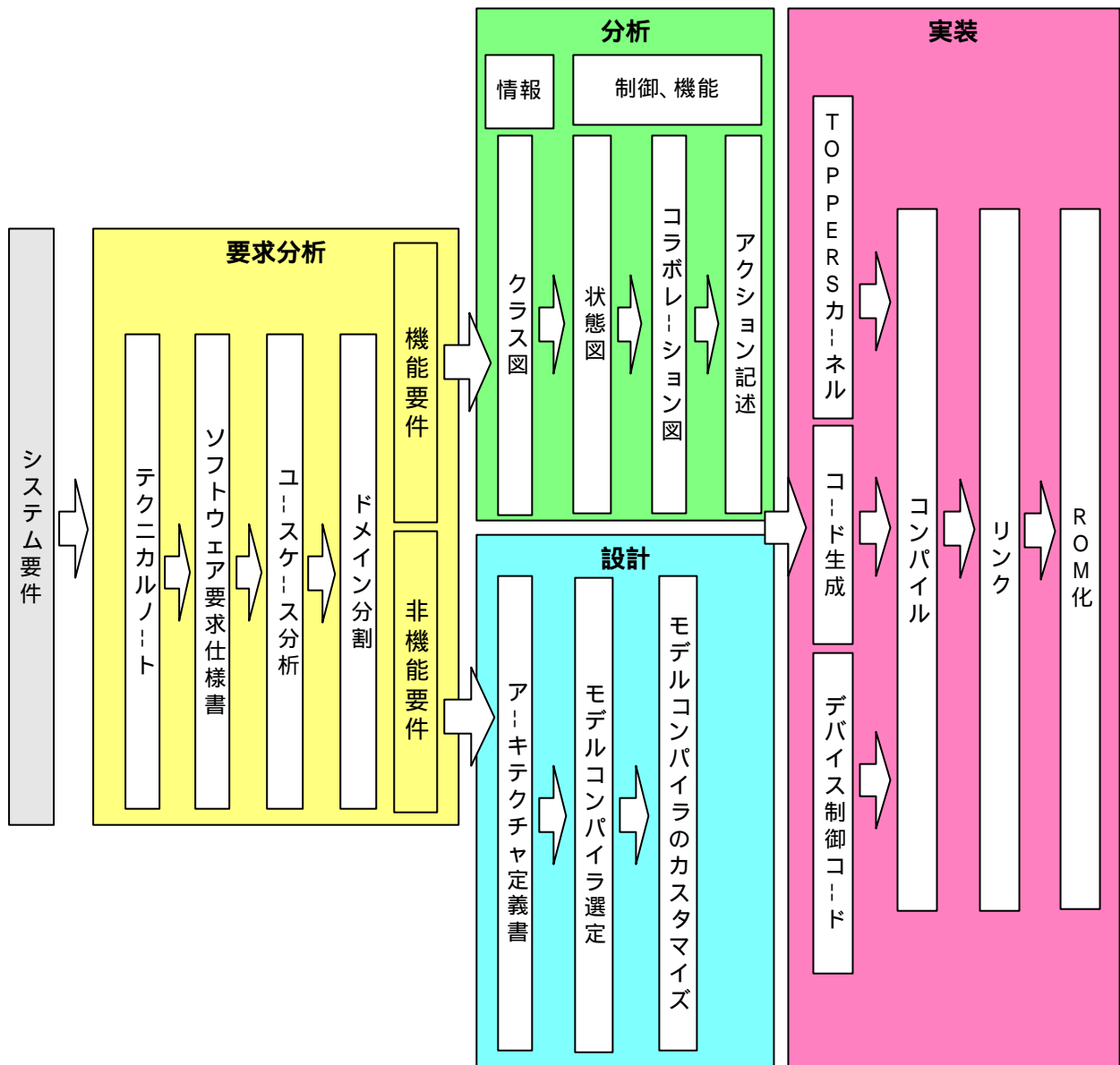


図 3.1 開発プロセス概要図

各フェーズの解説

要求分析フェーズ

要求分析フェーズでは、システム要件を基に技術調査や要求の整理を行う。機能要求はユースケース図とユースケース記述でまとめ、非機能要件は、別途記述を行う。要求を基にシステムを問題領域毎に分割したドメインチャートを作成し、以降、ドメイン毎に並行作業を行う。

分析フェーズ

ドメイン単位にクラスを抽出し、クラス図を作成する。クラスには振る舞いを持つものと持たないものがあり、振る舞いを持つクラスは状態図を作成する。また、各状態で行うべきプロセス

をアクション言語で記述する。

状態を持つクラスにおけるクラス間のイベントのやり取りをコラボレーション図で表現する。

アクションまで記述すれば、モデルの段階でシミュレーションが可能であり、実装に依存しないモデルを論理検証することができる。

設計フェーズ

非機能要件を実現するための設計戦略を記述したアーキテクチャ定義書を作成する。また、システム要件に最適なモデルコンパイラの選定を行い、必要であればモデルコンパイラのカスタマイズを行う。また、タイマーなどのメカニズムライブラリを用意しておく。

実装フェーズ

デバイス制御用コードを用意しておく。モデルコンパイラによりコード生成を行い、他のソースと一緒にコンパイル、リンクを行う。実行ファイルができたなら ROM へ書き込み、動作検証を行う。

4 要求仕様

冒頭でも述べたように、ソフトウェア仕様書として、

- ・ (株)東陽テクニカ 二上氏作 「セサミ電気 Sozex002 ソフトウェア仕様書 v1.2」

また、企画書として、

- ・ (株)東陽テクニカ 二上氏作 「セサミ電気 鹿威し 開発企画書 v4.1」

を基にした。

5 モデル解説

5.1 ユースケース

ソフトウェア要求仕様書から、アクターを抽出し、アクターに価値を与えるシステム要件をユースケースとして抽出した。

5.1.1 ユースケース図

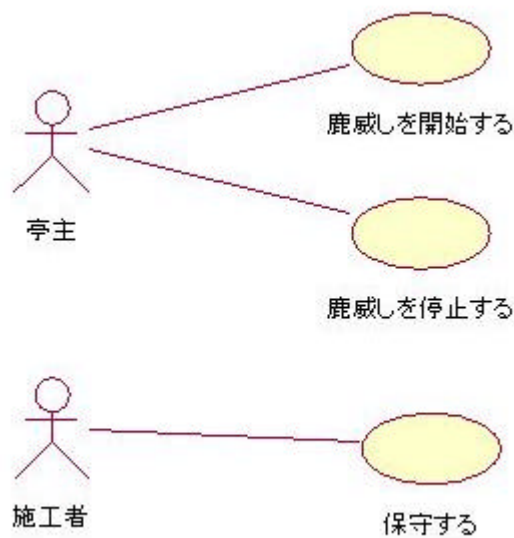


図 4.1 ユースケース図

5.1.2 ユースケース記述

UC-01：鹿威しを開始する

概要

システムが始動し、鹿威しが始動される。

アクター

亭主

事前条件

なし

事後条件

定期的にコーン音響が響いている。

基本系列

1. アクターは、システム開始を要求する。
2. システムは、ポンプを作動させ、竹に水をためる。
3. システムは、竹に水が満水になり、竹が離床し、反転したことを知る。
4. システムは、水はねを防止するため、ポンプを一時停止させる。
5. アクターは、コーン音響を楽しむ。

5. システムは、竹の水が空になり、竹が着床し、反転したことを知る。

6. 2 から 5 を繰り返す。

代替系列

3.1 水をため始めてから 80 秒間離床しなければ異常と判断し、LED ランプを点滅させる。

5.1 離床してから 20 秒間着床しなければ異常と判断し、LED ランプを点滅させる。

6.1 システムを開始してから 180 分間何も操作がなければ、消し忘れのためにシステムを停止する。

備考

なし

UC-02：鹿威しを停止する

概要

システムが停止する。

アクター

亭主

事前条件

システムが開始されている。

事後条件

システムが停止している。

基本系列

1. アクターは、システム停止を要求する。

2. システムは、竹に水が満水になり、竹が反転したことを知る。

3. システムは、竹筒に残っている水をすべて吐き出した後、ポンプを停止させる。

代替系列

なし

備考

なし

VMUC-02：保守する

概要

T.B.D

5.2 ドメインとブリッジ

5.2.3 ドメインチャート

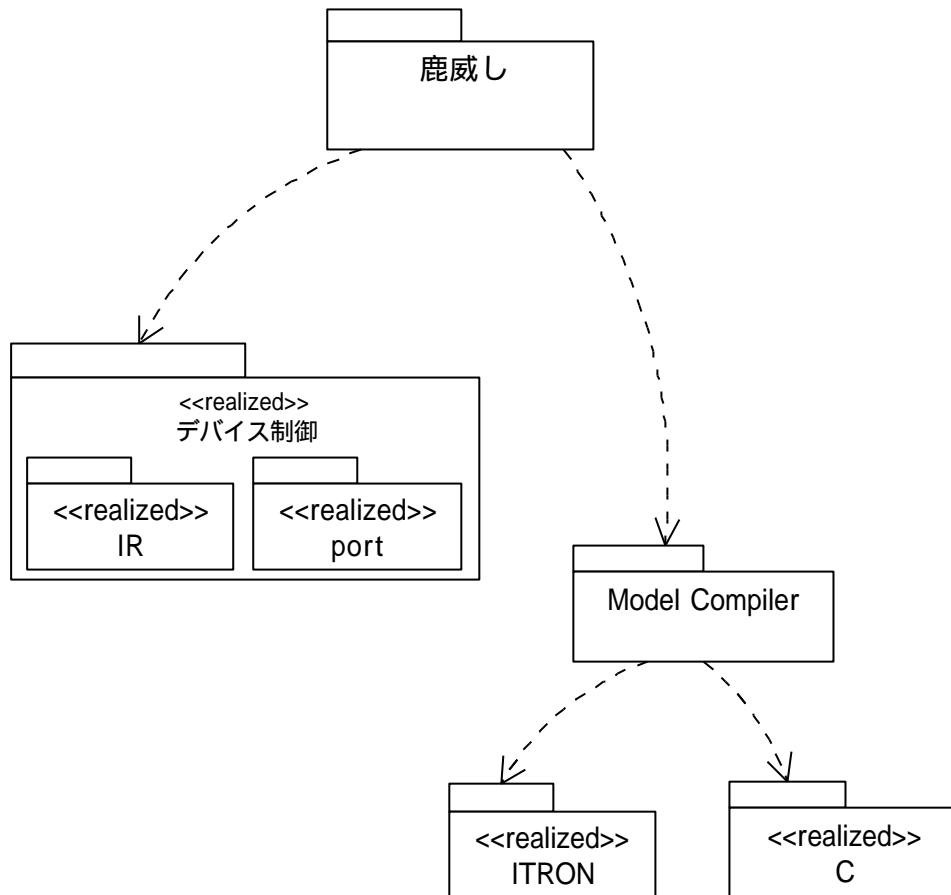


図 4.2 ドメインチャート

5.2.4 ドメイン記述とブリッジ記述

ドメインの使命

鹿威し domain

水流を制御し、快適なコーン音響をユーザーに提供する。

デバイス制御 domain

スイッチ、ランプ、リモコンセンサ (IR) を制御する手段を提供する。

Model Compiler domain

C 言語によるコンパクトな設計思想を反映した、変換による実装を提供する。

ブリッジ記述

鹿威し?? デバイス制御

・鹿威しドメインは、デバイスに情報を伝えるためにもしくは、デバイスの情報を伝えてもらうためにデバイス制御ドメインを使う。

鹿威し?? Model Compiler、UI?? Model Compiler

鹿威しドメイン、UI ドメインは、システム的设计思想を利用するために Model Compiler ドメインを使う。

5.3 クラス

5.3.5 クラス図

ししおどしドメインのクラス図を以下に示す。

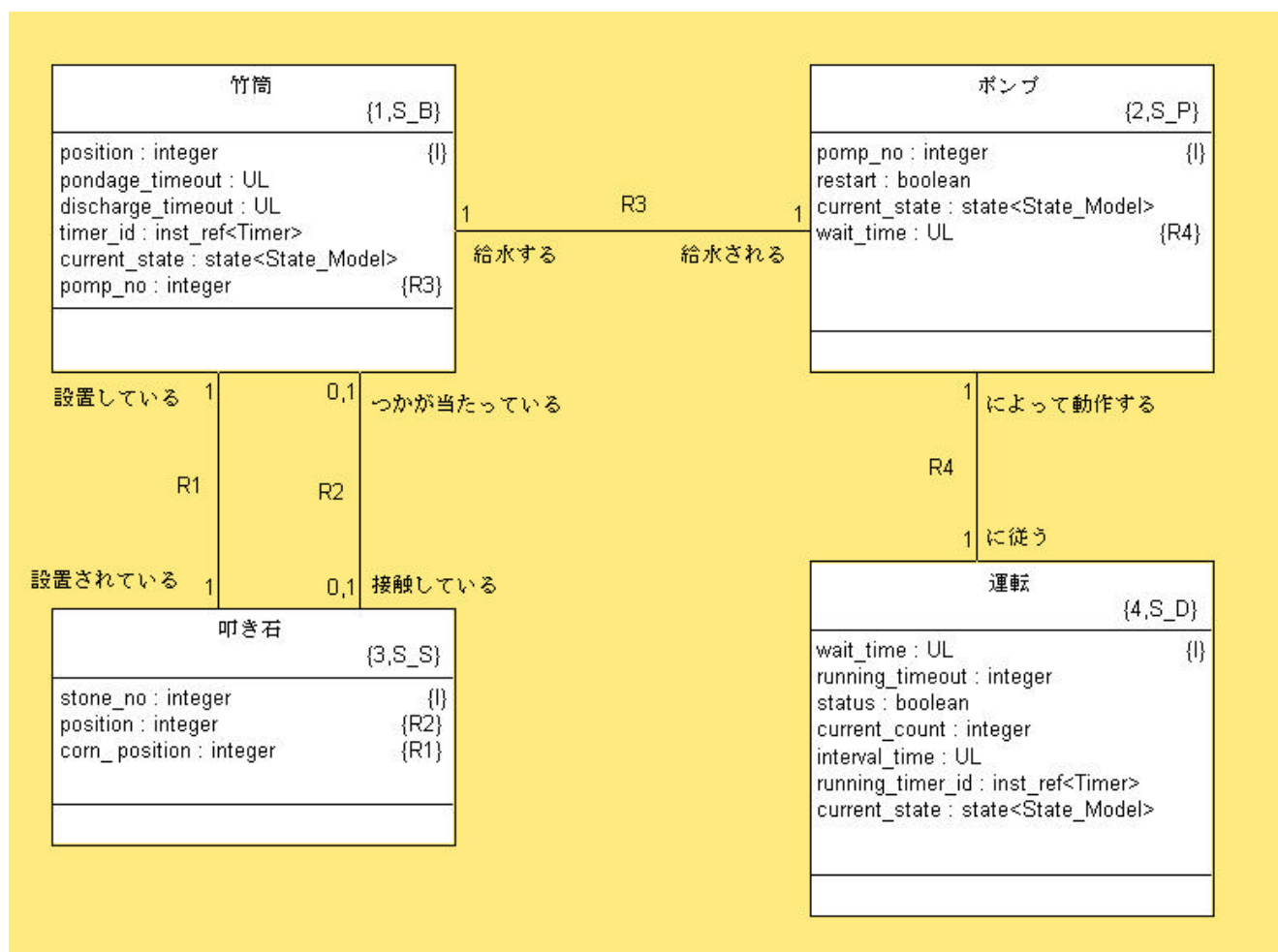


図 4.3 クラス図

5.3.6 クラス説明

各クラスに関する解説を行う。解説の表で使われている記号、略語を以下に示す。

* ----- インスタンスを特定することができる属性（識別子）

UL ----- ユーザー定義型（6.3.5章のデータ型の指定参照）

竹筒 (S_B)

竹筒そのもの表したクラス。

	属性	型	記述
*	position	integer	竹筒がどこにあるかをあらわす属性。竹筒が複数あった場合でも、この属性で特定することができる。
	pondage_timeout	UL	水が貯まりだしてから離床するまでの最大時間。この時間を超えると異常と判断する。単位：μsec
	discharge_timeout	UL	水がなくなってから着床するまでの最大時間。この時間を超えると異常と判断する。単位：μsec

timer_id		inst_ref	タイマーをスタートさせたときのハンドル
pomp_no	R3	(参照属性)	(ポンプ.pomp_no と同様)

ポンプ (S_P)

給水ポンプを抽象化したクラス。

	属性		型	記述
*	pomp_no		integer	ポンプを特定する属性。ポンプが複数あった場合にこの属性で特定できる。
	restart		boolean	再給水されたことを示す TRUE：再スタート開始 FALSE：再スタート開始しない
	wait_time	R4	(参照属性)	(運転.wait_time と同様)

叩き石 (S_S)

竹筒の土台となる石。竹筒がこの石から離れ、戻ってきたときにコーン音響が発生する。

	属性		型	記述
*	stone_no		integer	叩き石を特定する値。
	position	R2	(参照属性)	(竹筒.position と同様)
	corn_position	R1	(参照属性)	(竹筒.position と同様)

運転 (S_D)

鹿威しの運転に関する責務を持つ。

	属性		型	記述
*	wait_time		UL	停止指示から、一定時間遅らせて停止させるための TIME 値。
	running_timeout		integer	消し忘れ防止の為の最大運転時間、すなわち操作しない時間を表す。単位：分
	status		boolean	運転の状態を表す。 TRUE：ON FALSE：OFF
	current_count		integer	操作しない時間がどれだけ続いているかを表す。単位：分
	interval_time		UL	180 分をカウントするためのインターバルタイム。 今回は 60 秒とする。
	running_timer_id		inst_ref	タイマーをスタートさせたときのハンドル

5.3.7 関連の説明

R1： 竹筒 (1:1) 叩き石

1つの竹筒は1つの叩き石に設置されているという関係を表す。

R2： 竹筒 (1c:1c) 叩き石

竹筒のつかが叩き石に接触しているか、離れているかを表す関係。

R3： ポンプ (1:1) 竹筒

1つのポンプが1つの竹筒に給水するという関係を表す。

R4： 運転(1:1)ポンプ

あるポンプにおける、水はね防止のタイミング値を定義するという関係を表す。

5.4 ステートマシン図とアクション記述

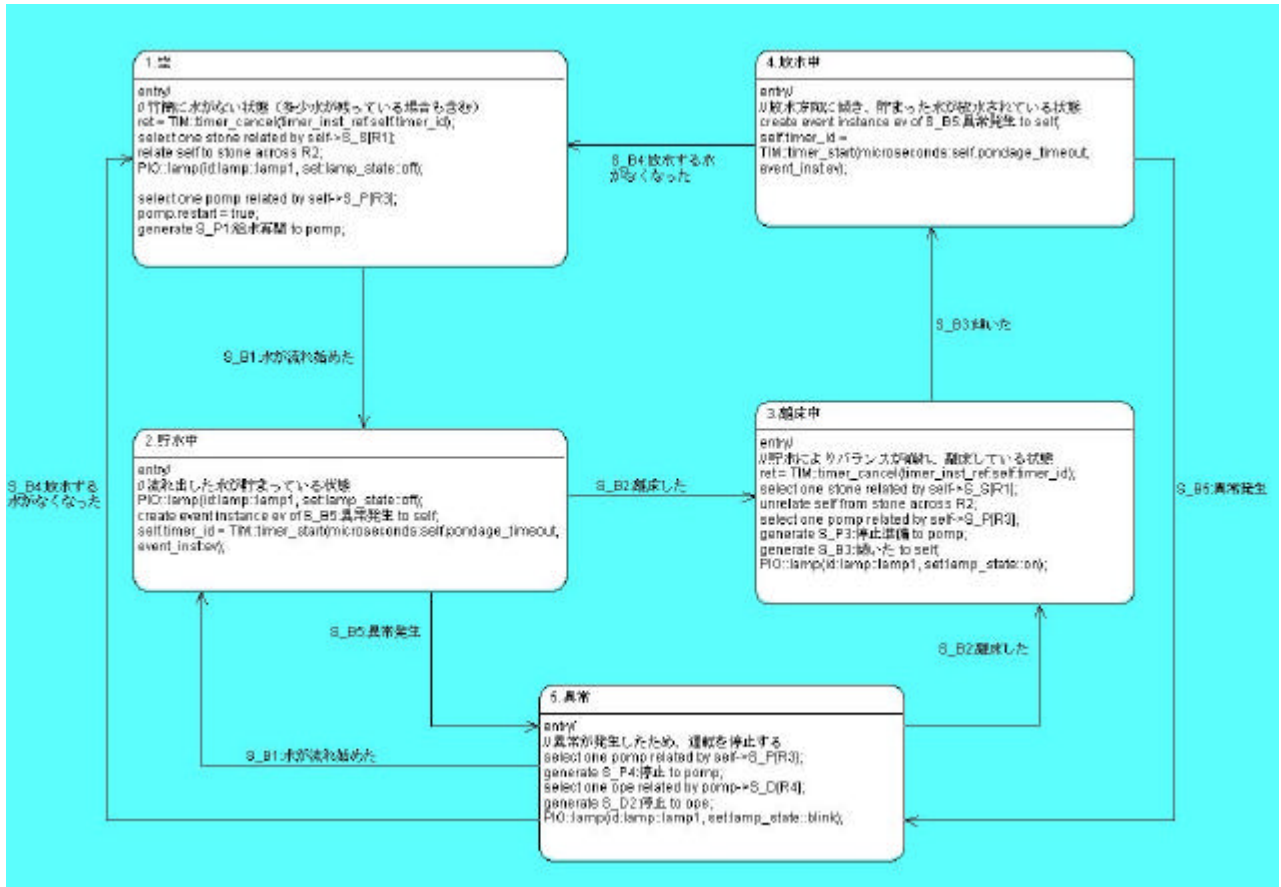


図 4.1 竹筒クラスの状態遷移図とアクション記述

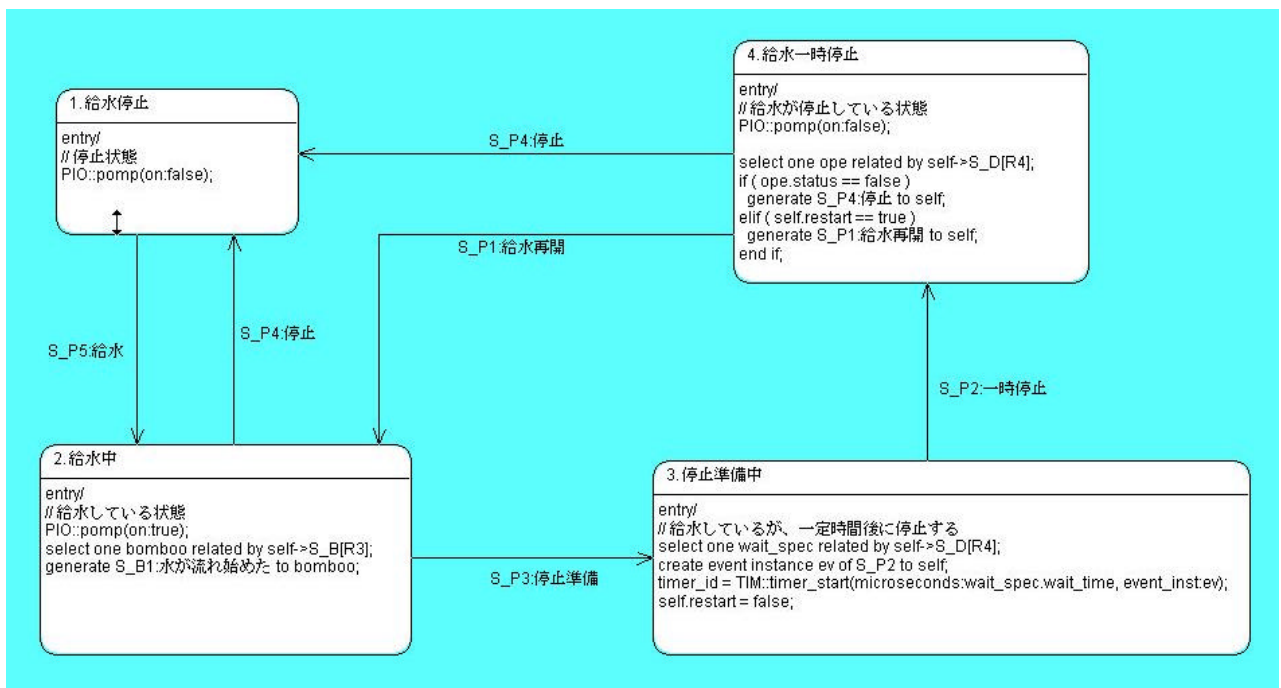


図 4.2 ポンプクラスの状態遷移図とアクション記述

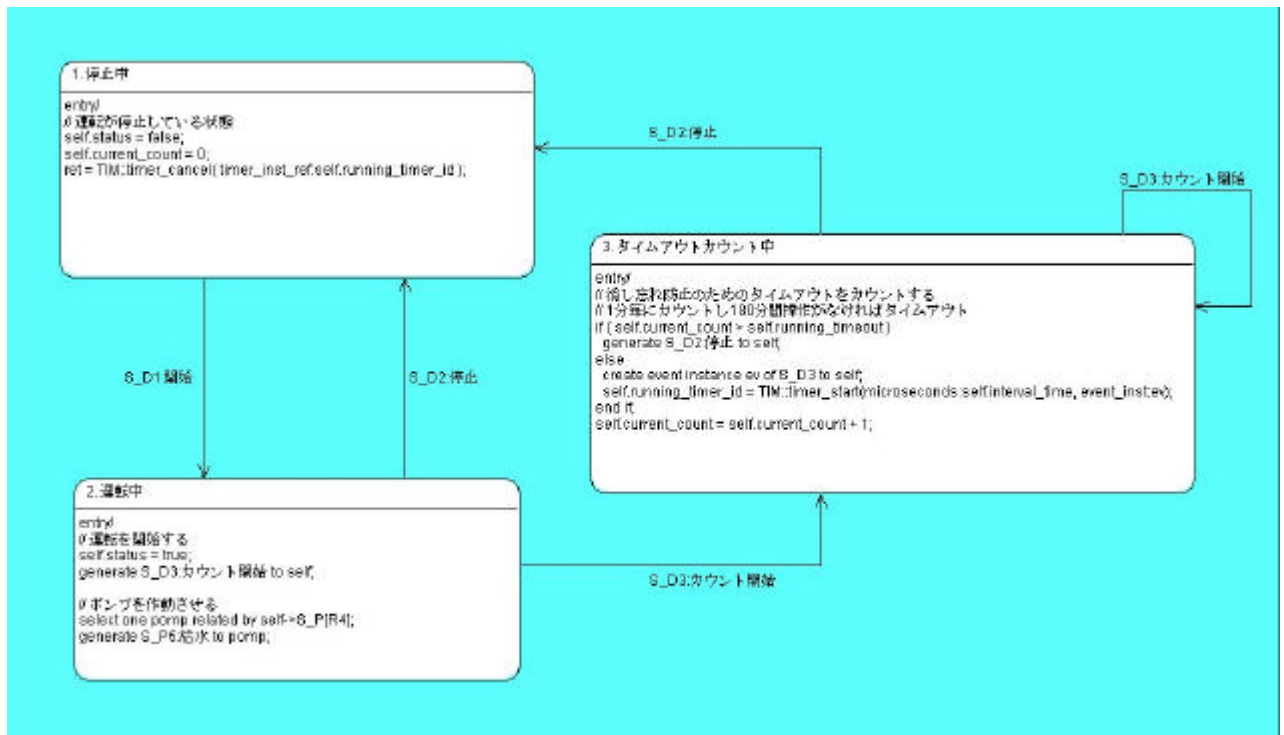


図 4.3 運転クラスの状態遷移図とアクション記述

5.5 コラボレーション図

コラボレーション図では、状態を持つクラス（アクティブクラス）間のイベントのやり取りを表現する。今回のモデルでは、ポンプクラス、竹筒クラス、運転クラスの3つがアクティブクラスである。<<External Entity>>は外部ドメインを表している。外部ドメインからの要求をどのクラスが受け取り、どのクラスにイベントを発行しているかを、ある1つのシナリオについてのみ表現するのではなく、全シナリオを1つの図で表現する。

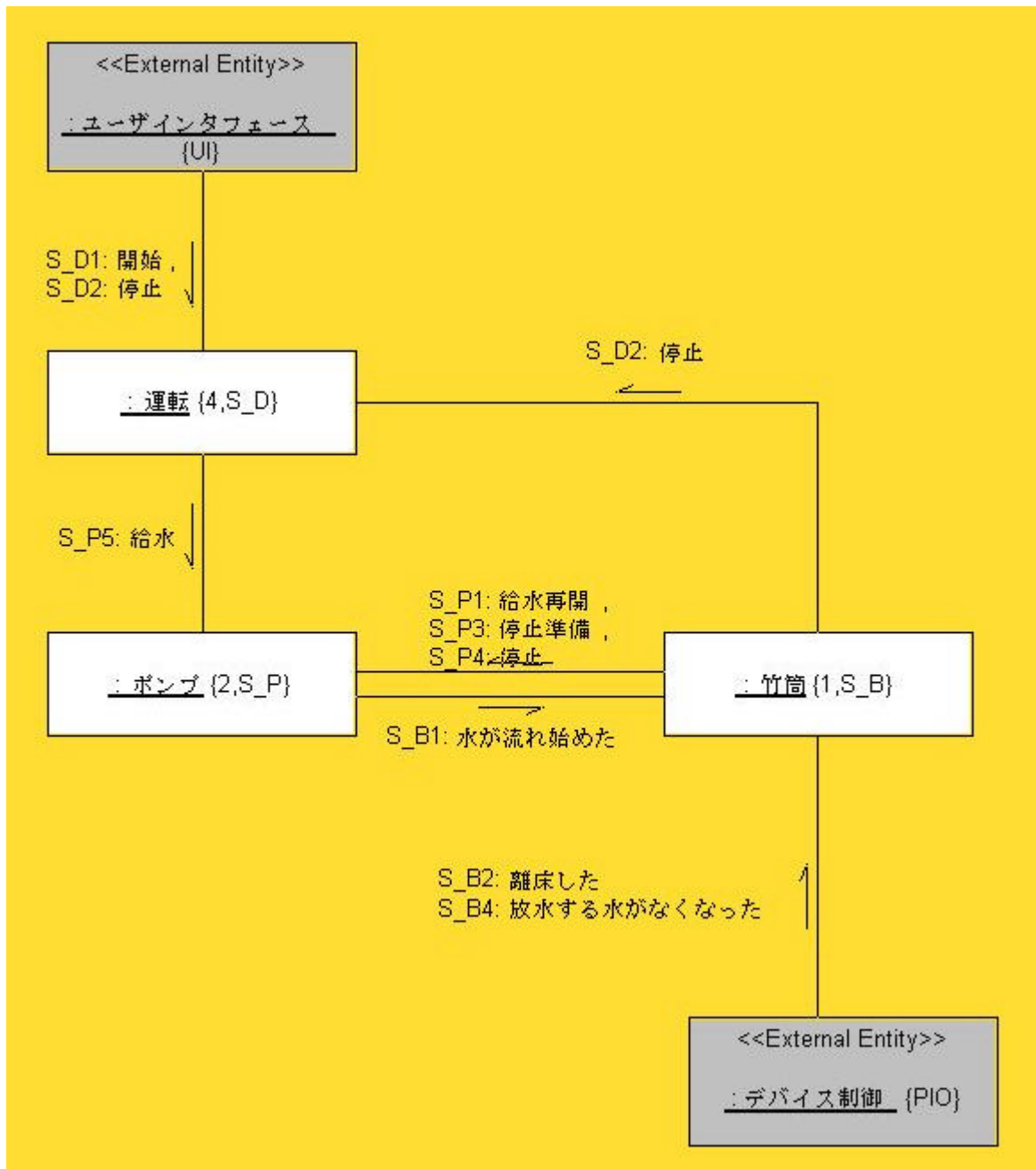


図 4.4 コラボレーション図

5.6 同期サービス

同期サービスとは、他ドメインが自ドメインに対して通知や要求を行う際の手段として、自ドメインが用意しておくサービスのことである。このサービスを他ドメインが呼び出すことで、自ドメインに情報が伝わり、自ドメインが動き出すきっかけとなる。今回定義した同期サービスは以下のとおりである。

- ・ Init ドメインの初期化を行う。必要なインスタンスの生成や関連を張っておく。
- ・ Start 運転が開始されたことを知らせてもらう。
- ・ Stop 運転が停止されたことを知らせてもらう。

-
- ・ Touch 叩き石に竹筒が接触したことを知らせてもらう。
 - ・ Leave 叩き石から竹筒が離れたことを知らせてもらう。

Init はシステム初期化時に呼ばれる。Init 以外の同期サービスがどのドメインから呼ばれるかは、8 章で解説する。

6 モデルコンパイラによる自動コード生成

MDA の基本概念である PIM (Platform Independent Models) から特定の技術、言語へのマッピングを実現させる手段として、変換による実装がある。モデルコンパイラを利用すると変換作業が自動化される。変換による実装とは、分析モデルの構成要素をどのようなコードで実現させるかを決定し、分析モデルからコード (実装) への変換をツールによって実現することである。

モデルコンパイラは以下の 3 つの要素で構成されている。

- ・ **アーキタイプ** (変換ルール)
- ・ **メタモデル** (分析モデルの構成要素をモデル化したもの。OMG では MOF(Meta-Object Facility) として標準化されており、ここでいうメタモデルは MOF の M2 に相当する。)
- ・ **メカニズム** (プラットフォーム依存のランタイムライブラリ)

6.1 コード生成概略

分析モデルからコードへ変換を行うには、アーキタイプ(テンプレートともいう)を使用する。このアーキタイプには、メタモデルを基に、分析モデルの情報を取り出し、コードのどの部分に情報を埋め込むかといった内容になっている。また、ソフトウェアを実現するための共通ライブラリ (リスト、キューなど) をメカニズムと呼び、変換時に利用する。1 通りの変換方法では要件を満たせない場合 (例えばあるクラスは永続クラスにしたいなど) には、分析者が別の変換ルールを指定することができる。これを色づけ (OMG における MDA の規格ではマークと呼ばれている) と呼ぶ。これらアーキタイプ、メカニズム、色づけは、モデルコンパイラによって実現され、モデルコンパイラを利用することで変換による実装が可能となる。

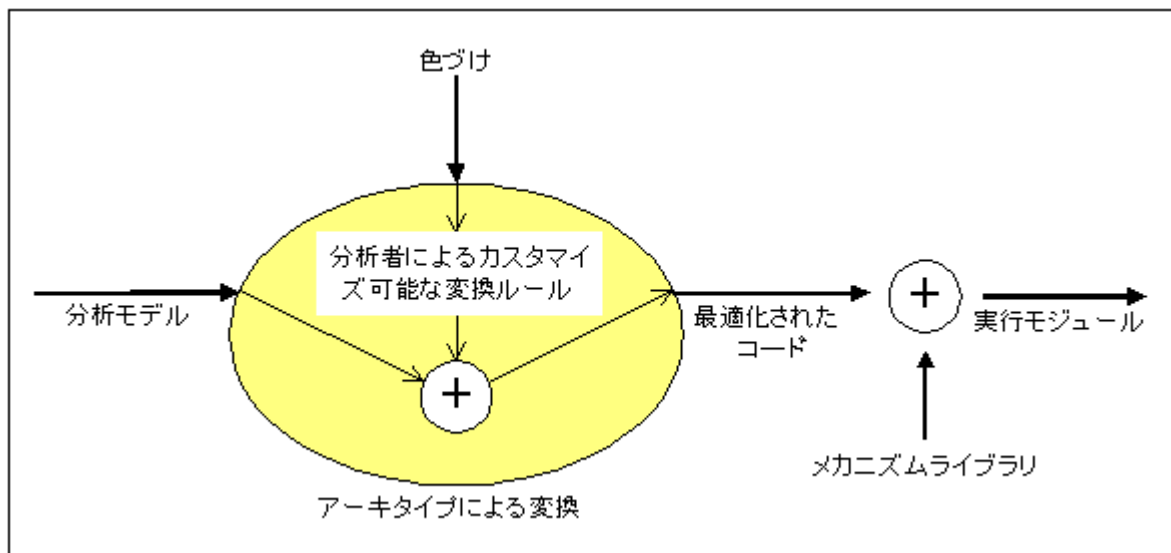


図 6.1 モデルコンパイラによる変換

6.2 色づけ概要

色づけとは、メタモデルからの一様な変換に異なる制約を与える変換オプションである。色付けにより、同じ分析モデルから異なる実装制約を満足するシステムが構築可能となる。

色づけ例

- ・クラス A は、EEPROM 上に存在する永続クラス
- ・ドメイン B は別タスクで実行する
- ・クラス B のインスタンスはヒープ領域から動的に確保し、クラス C,D は配列で静的に確保する
- ・クラス E は使用しない (コード生成しない)

6.3 コード生成

6.3.1 環境設定

コード生成を行うには Cygwin を使用する。Cygwin での環境設定において、以下のようにモデルコンパイラの環境変数の設定とパスの設定が必要となる。

```
#####  
# MC3020 #  
#####  
setenv ROX_MC_ROOT_DIR C:/mc3020 ? モデルコンパイラをインストールしたフォルダ  
setenv ROX_MC_BIN_DIR $ROX_MC_ROOT_DIR/bin  
  
set path=($ROX_MC_BIN_DIR $path)
```

6.3.2 システムノードの作成

コード生成するフォルダ (システムノード) を作成する。作成したら、Cygwin のシェルを起動し、システムノードへ移動する。

ビルドツール `rox_init_node` を実行して、システムノード下にコード生成用のフォルダを作成する。システムノードのパス入力を求められるので、システムノードの場所を指定する。

```
someone[3]% rox_init_node  
Enter the pathname to the directory in which you wish to translate:  
c:/arch_test/soze_md システムノード  
Upgrading application node: c:/arch_test/soze_md  
  
Application translation node successfully installed!  
  
(1) Change directory to: c:/arch_test/soze_md  
and enter 'make help' for a list of available targets.  
(2) Use 'make dom_node' target to add a domain to the system.  
'make dom_node' without further parameters will provide examples.  
(3) Edit Makefile.user to provide specific commands and options  
related to your C compiler, assembler, linker, etc.  
someone[4]%
```

6.3.3 OO ドメインフォルダの作成

BridgePoint で分析モデルを作成した各ドメインについて、`dom_node` オプションを付けて `make` を実行する。

```
someone[4]% make dom_node domain=soze  
rox_get_dom_sql: INFO: 'c:/arch_test/soze_md/soze/schema/sql/pathfinder.sql' CREATED.  
'rox_get_dom_sql: INFO: Domain 'soze' using version '18  
on_branch 'MAIN'.
```

```

rox_dom_init: 'c:/arch_test/soze_md/soze/Makefile' CREATED.

Application domain node successfully installed!

(1) Before continuing with translation, you must register domain 'hw'
    in system coloring file: system/color/registry.clr.
(2) Domain coloring files have been placed in: hw/color.
    You may wish to edit these at this point before continuing with translation.

someone[5]%

```

6.3.4 フォルダ構成

以下にフォルダ構成を示す。

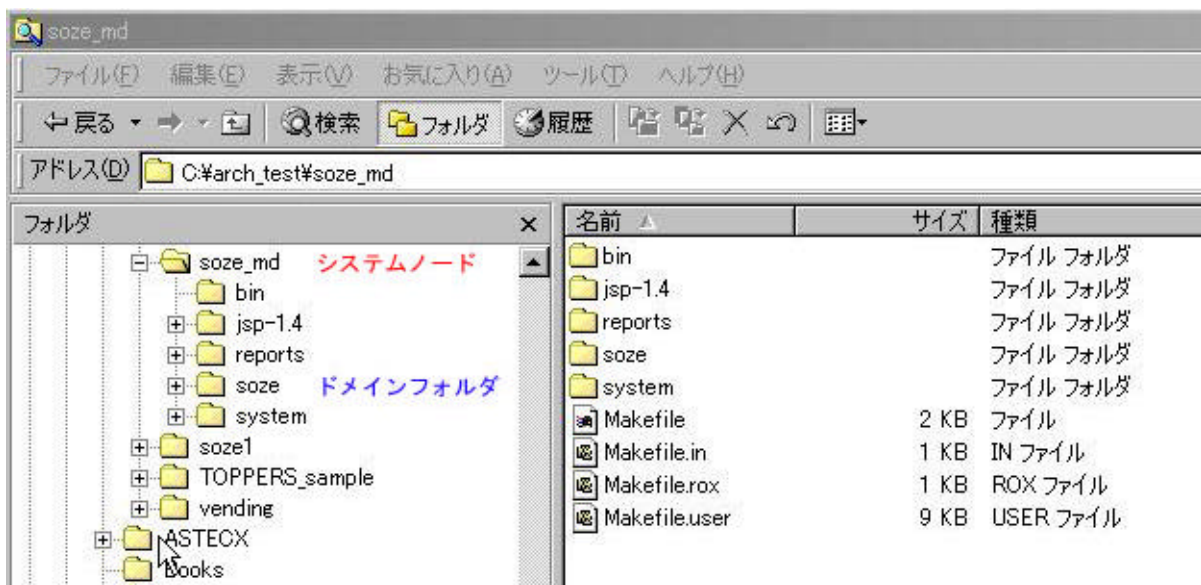


図 6.2 フォルダ構成

bin は使用しない。

jsp-1.4 は TOPPERS カーネルが格納されているフォルダ。

reports は生成時のログ情報ファイルが格納されている。

system は特定のドメインに依存しないシステム全体に関わるコードが格納されている。

6.3.5 色づけ

6.3.5.1 システム全体の色づけ

システム全体に関係する色付けは、`${システムノード}/system/color` ディレクトリ下のカラーリングファイル (*.clr) に対して行う。

OOA ドメインの登録

MDA のモデルで作成されているドメインを OOA ドメインと呼ぶ。OOA ドメインの名称と、キーレーター、ドメイン番号を設定する。

`${システムノード}/system/color/registry.clr`

注意: repository_name のところは、BridgePoint の OOA ファイル xxx.oaa の xxx を入れる。

```

.//=====
.// Register OOA domain(s):
.//
.// RegisterOoaDomain( "repository_name", "registered_name", registered_id )
.//=====
.invoke RegisterOoaDomain( "soze", "SOZE", 1 )

```

手書きドメインの登録

モデル以外で作成されたドメインの説明文と、キーレター、ドメイン番号を設定する。

```

.//=====
.// Register Realized domain(s):
.//
.// RegisterRealizedDomain( "description", "registered_name", registered_id)
.//=====
.invoke RegisterRealizedDomain( "", "PIO", 100 )
.invoke RegisterRealizedDomain( "", "UI", 200 )

```

ブリッジの登録

5.2章で定義したドメインチャートを実現するためにドメインとブリッジの結びつけを行う。

色づけファイル: \${システムノード}/system/color/bridge.clr

・ OOA ドメインどうしの連結

ブリッジ関数を呼ぶ側のドメイン、外部ドメイン、呼ばれる側のドメインを、全てキーレターで設定する。OOA ドメインが1つしかない場合は、設定する必要はない。

```

.//=====
.// Wire Bridge Between OOA Domains using Synchronous Services:
.//
.// WireSynchServiceOoaBridge( "initiating_dom", "ee_key_letters", "recipient_dom" )
.//=====

```

・ OOA ドメインから手書きドメインへの連結

OOA ドメイン、外部ドメイン、手書きドメインのキーレター、手書きドメインのブリッジ関数の接頭文字とヘッダファイル名を設定する。今回は、UI を IR(リモコンセンサ) PIO を port (ポート制御) にマッピングした(図 6.3 参照)

```

.//=====
.// Wire Bridge Between OOA domain and Realized Domain:
.//
.// WireRealizedExternalEntity( "ooa_domain_name", "ee_key_letters",
.// "realized_domain_name", "method_prefix", "include_file" )
.//=====
.invoke WireRealizedExternalEntity( "SOZE", "UI", "UI", "UI", "UI_bridge.h" )
.invoke WireRealizedExternalEntity( "SOZE", "PIO", "PIO", "PIO", "PIO_bridge.h" )

```

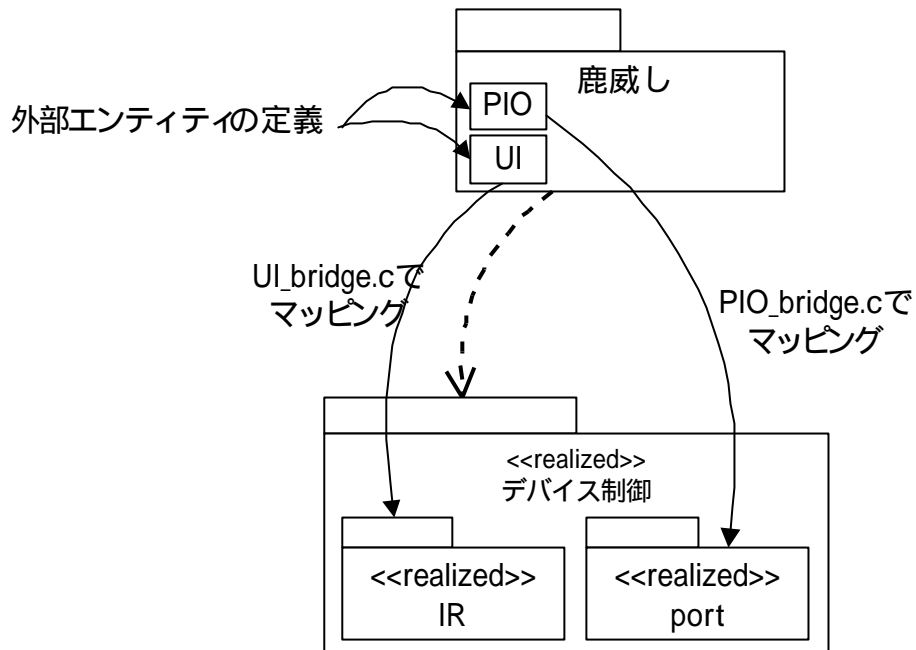


図 6.3 ブリッジのマッピング

他プロセスから呼ばれる同期サービス

他プロセスから呼ばれる同期サービスを登録する。

```

//=====
// Mark Synchronous Service as Safe for Interrupt Execution:
//
// TagSyncServiceSafeForInterrupts( "domain", "sync_service" )
//=====
.invoke TagSyncServiceSafeForInterrupts( "SOZE", "Start" )
.invoke TagSyncServiceSafeForInterrupts( "SOZE", "Stop" )
.invoke TagSyncServiceSafeForInterrupts( "SOZE", "touch" )
.invoke TagSyncServiceSafeForInterrupts( "SOZE", "leave" )

```

データ型の指定

モデルでデータ型を定義した場合、その型が実際の言語の型とどういう関係かを設定する。

`${システムノード}/system/color/datatype.clr`

・データ型

```

//=====
// Specify User Data Type Precision:
//
// TagDataTypePrecision( "domain", "dt_name", "tagged_name", "initial_value" )
//=====
.invoke TagDataTypePrecision( "*", "UL", "unsigned long", "0" )

```

・ポインタ型

```

//=====
// Mapping User Data Type to an Implementation Pointer:
//
// MapDataTypeAsPointer( "domain", "dt_name", "pointer_type", "include_file" )
//=====

```

システム全体の色づけ

必要に応じて設定する。

`${システムノード}/system/color/system.clr`

- 文字列の最大長

```
./=====
./ TagMaximumStringLength( max_len )
./=====
./invoke TagMaximumStringLength( 1 )
```

- 関連インスタンス領域の最大数

```
./=====
./ TagMaximumRelationshipExtentSize( value )
./=====
```

- 選択インスタンス領域の最大数

```
./=====
./ TagMaximumSelectionExtentSize( value )
./=====
```

- イベントキューのサイズ

```
./=====
./ TagMaximumSelfDirectedEvents( value )
./ TagMaximumNonSelfDirectedEvents( value )
./=====
./invoke TagMaximumSelfDirectedEvents( 2 )
./invoke TagMaximumNonSelfDirectedEvents( 5 )
```

- 同時に使用している OOA Timer の数の定義

```
./=====
./ TagMaximumPendingOoaTimers( value )
./=====
./invoke TagMaximumPendingOoaTimers( 4 )
```

- InterleavedBridge の最大値の定義

```
./=====
./ TagMaximumInterleavedBridges( value )
./=====
./invoke TagMaximumInterleavedBridges( 4 )
```

- InterleavedBridgeDataSize の定義

```
./=====
./ TagInterleavedBridgeDataSize( value )
./=====
./invoke TagInterleavedBridgeDataSize( 2 )
```

6.3.5.2 OOA ドメインの色づけ

各ドメインに対する色付けは、`${システムノード}/{ドメイン名}/color` フォルダ下のカラーリングファイル (`*.clr`) に対して行う。

初期化関数の指定

`${システムノード}/{ドメイン名}/color/domain.clr`

ドメインの初期化関数の関数名を設定する。

ドメインの初期化が不要な場合は、設定する必要はない。

```
./=====
./ Tag Initialization Function:
./
./ TagInitializationFunction( "function_name" )
./=====
./invoke TagInitializationFunction( "Init" )
```

クラスの色づけ

`${システムノード}/{ドメイン名}/color/object.clr`

- ・インスタンスの最大数 (デフォルト)

次項で、クラス別最大数を設定しなかったクラスは、この項での設定数の領域を確保される。

```
./=====
./ TagSystemObjectDefaultExtentSize( value )
./=====
./invoke TagSystemObjectDefaultExtentSize( 1 )
```

- ・インスタンスの最大数 (クラス別)

クラスごとに、確保するインスタンス領域を設定できる。

```
./=====
./ TagObjectExtentSize( "key_letters", value )
./=====
```

- ・静的インスタンスの生成

```
./=====
./ Mark Class as Having a Static Instance Population
./
./ TagStaticInstancePopulation( "ss_name", "class_key_letters" )
./=====
```

- ・オペレーションのコード生成解除

```
./=====
./ Disable Object Operation Semantics Translation
./
./ TagClassOperationTranslationOff( "object_key_letters", "op_name" )
./=====
```

イベントの優先度

`${システムノード}/{ドメイン名}/color/event.clr`

```
./=====
./ Tag Priority Event(s):
./
./ TagPriorityEvent( "event_label", value )
./=====
```

6.3.6 コード生成

全てのドメインのコード生成を一度にまとめて行う場合は、(A)を実行する。OOA ドメイン

を個別にコード生成する場合は、(B)を実行する。

(A) 全ドメインを一度にコード生成

gen_all オプションを付けて make を実行する。

まとめてコード生成を行うと、生成エラーの原因がどのドメインにあるのかが探しにくくなる。

設定やモデルに問題がある可能性が高い場合は、ドメイン別にコード生成したほうがよい。

```
someone[5]% make gen_all
rox_get_dom_sql: INFO: 'c:/arch_test/soze_md/soze/schema/sql/pathfinder.sql' UNCHANGED.
'rox_get_dom_sql: INFO: Domain 'soze' using version '18
on branch 'MAIN'.
(ログ省略)
someone[6]%
```

(B) OOA ドメインを個別にコード生成

各ドメインについて、gen_dom オプションを付けて make を実行する。

```
someone[6]% make gen_dom domain=soze
rox_get_dom_sql: INFO: 'c:/arch_test/soze_md/soze/schema/sql/pathfinder.sql' CREATED.
'rox_get_dom_sql: INFO: Domain 'soze' using version '18
on branch 'MAIN'.
(ログ省略)
someone[7]%
```

システム全体に関係するソースコードを生成するために、gen_sys オプションを付けて make を実行する。

```
someone[7]% make gen_sys
rox_sys_init: 'c:/arch_test/soze_md/system/schema/sql/sys_root.sql' CREATED.
rox_app_init: 'init_seq.c' CREATED.
(ログ省略)
someone[8]%
```

7 設計指針

本 7 章をアーキテクチャ定義書とする。

7.1 モデルコンパイラ的设计指針

今回使用した PT 社のモデルコンパイラ MC-3020 は C 言語を生成する。特徴を以下に述べる。

- ・ パフォーマンスとリソースにおいて最適なコードに変換
- ・ オープンなアーキテクチャでカスタマイズ可能
- ・ コードとモデルのトレーサビリティが高い
- ・ RTOS を必要としない
- ・ 8 ビット、16 ビット、32 ビット CPU 対応
- ・ モデルレベルのトレース機能
- ・ ドキュメント生成

以上のように、コンパクトなコードを生成するため、メモリ制約の強い組み込みでも十分対応できるモデルコンパイラである。

7.2 モデルコンパイラのカスタマイズ

MC-3020 は RTOS を必要としないため、メインルーチンを OS のタスクとして生成するように変更する。詳しくは、TOPPERS ホームページの「JSP カーネル上でのモデルベース開発」参照。

(<http://www.toppers.jp/mda.html>)

7.3 タスク相関図

今回は 3 つのタスクに分割した。1 つはモデル部を、その他にタイマータスク、デバイス制御タスクとした。タスク相関図を以下に示す。

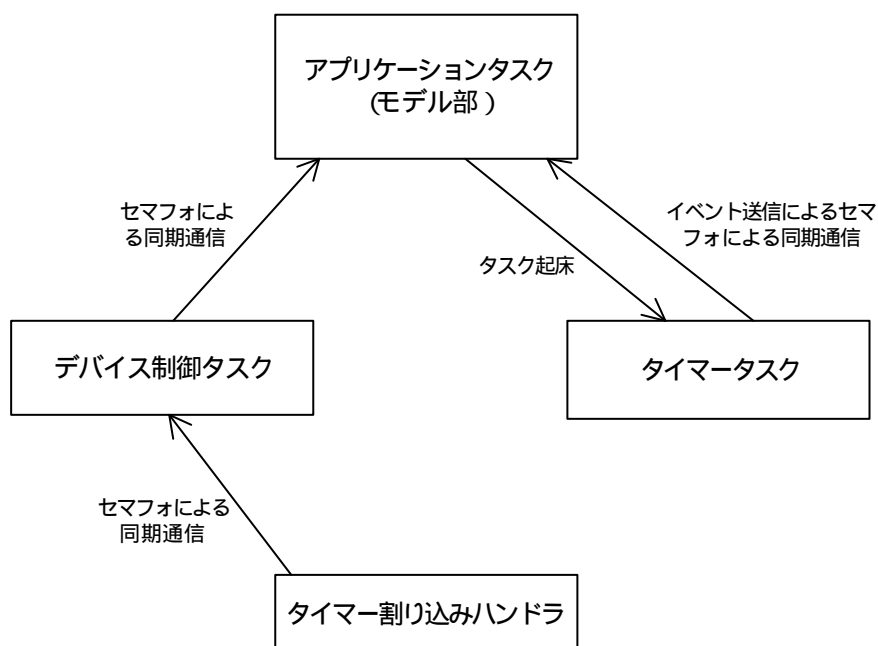


図 7.1 タスク相関図

8 手書きコードとモデルとのインタフェース

ドメインチャートに基づき、ブリッジ部分、デバイス制御部分を手書きコードとした。

- UI_beidge IR 情報をししおどしドメインに通知するための I/F
- PIO_Bridge ポンプの制御とランプの制御を行うための I/F
- TIM_bridge タイマー機能
- Port デバイスのポート設定とポートセンス、ポーリングタスク
- Ir リモコン受光部デバイスドライバ

モデルとのインタフェースにおいては、モデル側に通知する部分は 5.6 章で説明した同期サービスを用い、モデル側が呼び出す部分についてはデバイス制御側で用意している関数をブリッジを解して呼び出す。

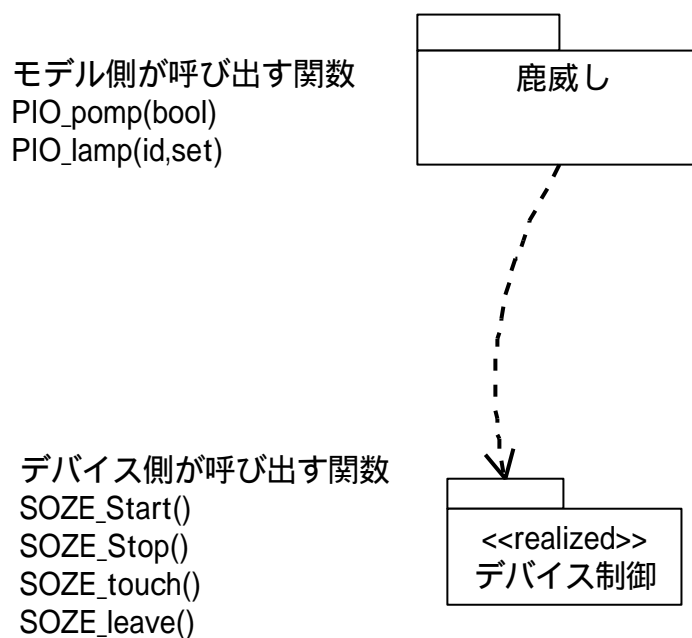


図 8.1 モデルとデバイス制御とのインタフェース

9 実装環境

開発環境として、OAKS16-MINI に付属の統合開発環境 TM を使用する。TM ではホストとターゲットをシリアル接続することで通信が可能である（図 9.1 参照）。

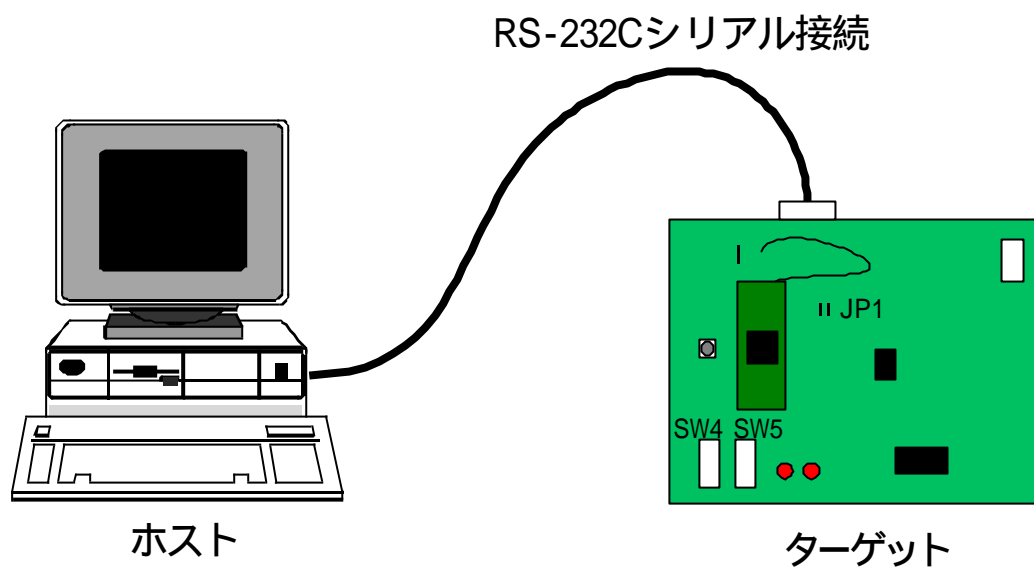


図 9.1 開発環境

9.1 ビルド

TM に今回の使用するプロジェクトファイルである Sozem.tmk を読み込ませ、ビルドを行う。

ビルドに必要なファイルはシステムノードの下の jsp-1.4¥WORK¥OO_SOZE に格納されている（図 9.1 参照）。

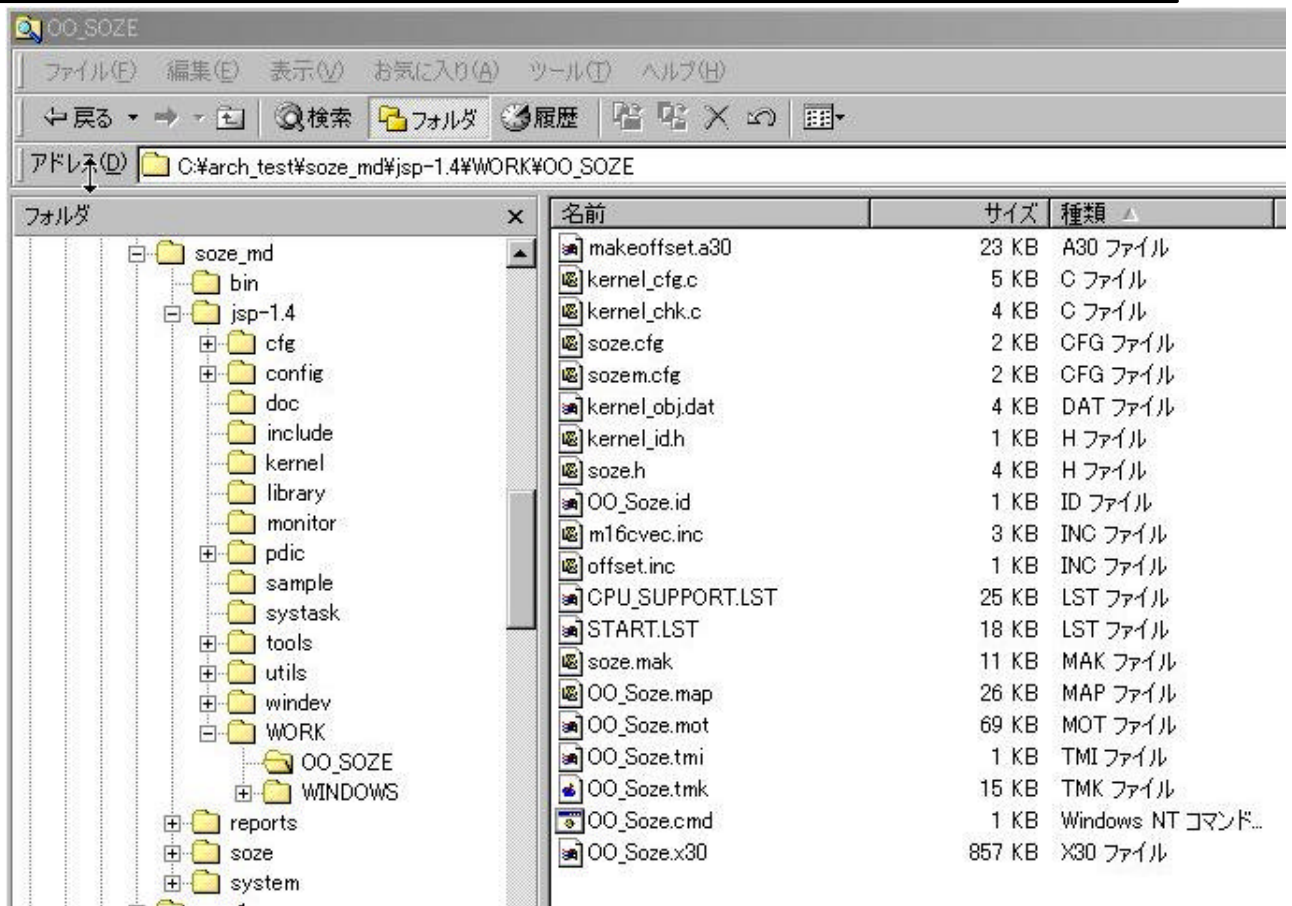


図 9.2 ビルドに必要なファイル群

TM のプロジェクトには以下のファイルが登録されている。

- ・ モデルから自動生成されたコード
- ・ デバイス制御コード
- ・ TOPPERS/Jsp カーネル、コンフィグ

図 9.2 にプロジェクトに必要なファイルを追加したところを示す。これらのファイルをビルドすることで、実行ファイル、マップファイル、ROM 化用ファイルが作成される。

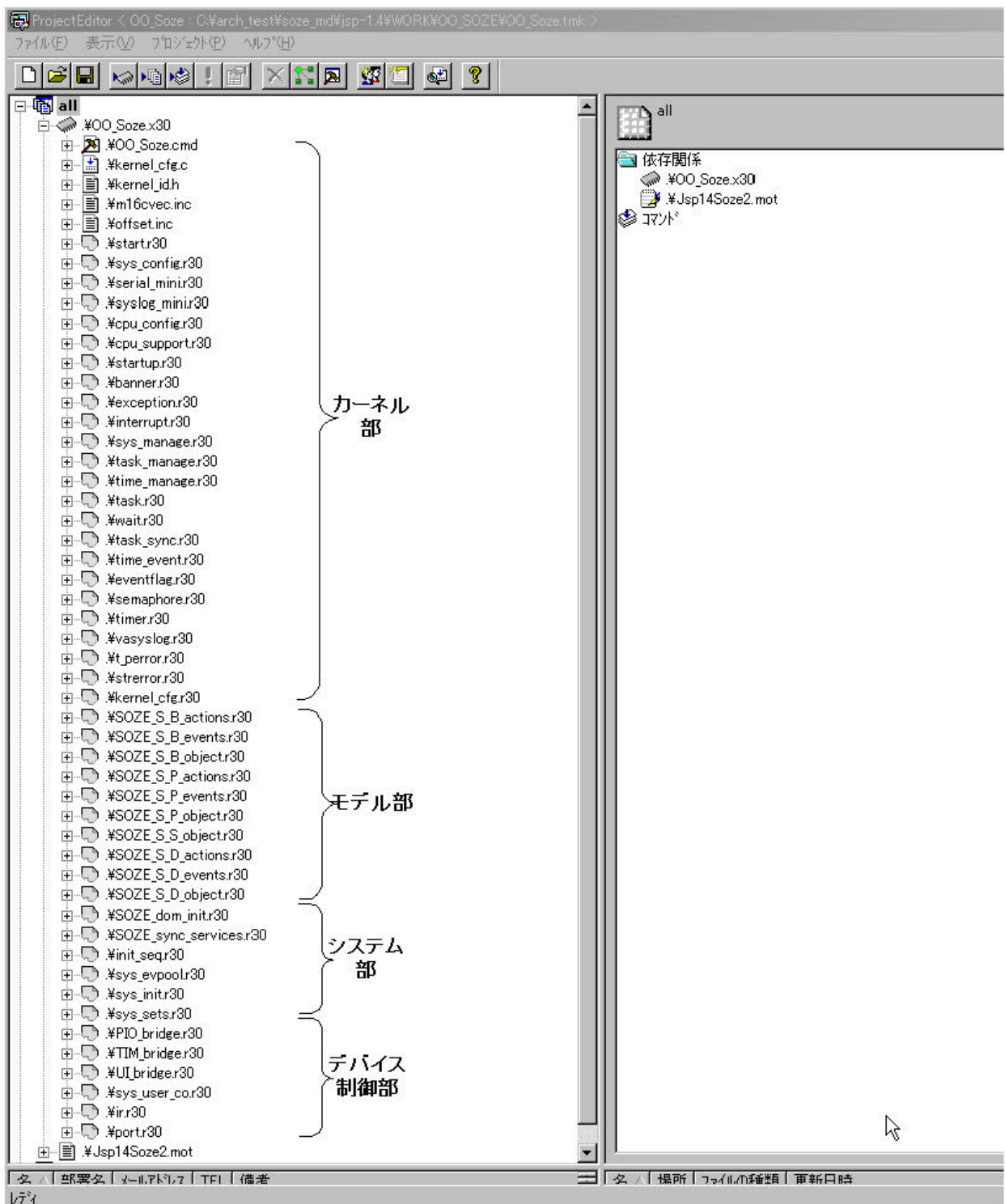


図 9.3 TM のプロジェクト内容

9.2 デバッグ

OAKS16-MINI のデバッグ環境はボード、ホスト間をシリアル通信によりデバッガでデバッグが可能である。しかし、今回はログ出力用にシリアルを使用するため、デバッガは使用しなかった。

9.3 ROM化

ROM化を行うには、OAKS16-MINI 付属の FlashSta を使用する。

- 1) 図9.1のターゲットにおけるジャンパJP1をショートさせ、ターゲットの電源を入れる。
- 2) flashSta を起動する。
- 3) Select Program メニューでシリアルポートを選択する。
- 4) Refer ボタンから、書き込むファイルを選択し(OO_Soze.mot)、OK を押す。
- 5) M16C Flash Start メニューで、Erase を選択し、フラッシュの中身をクリアします。
- 6) Program..ボタンで、書き込みを行う。
- 7) 書き込みが終了したら、Exit で終了する。
- 8) ターゲットの電源を切り、ジャンパを元に戻す。

9.4 実行

書き込み終了後は、電源 ON で鹿威しがスタートする。SW4 でポンプが始動する。また、リモコンは以下の機種に対応している。

- ・ RMT-H4
- ・ RMT-V305

他の機種を追加したい場合は、ir.c の変換テーブルを拡張することで対応できる。

ログ出力を行う場合は、ターミナルソフトを立ち上げ、以下のパラメータをセットする。

Baudrate : 19200

Data : 8bit

Parity : none

Stop : 1bit

Flow : none