MDD ロボットチャレンジ 2005（チーム FSEL）：参加報告と開発を通して得られた見識

丸山勝久[†] 山本哲男[†] 筏谷浩樹[†]
石塚聖啓[†] 原田彬直[†] 平井孝[†]

本稿では，MDD ロボットチャレンジ 2005 において筆者らが開発した飛行船制御システムのモデル，開発における活動，飛行結果を述べる．さらに，開発を通して得られた見識を議論する．

MDD Robot Challenge 2005 (Team FSEL): Report on the Challenge and Insight Obtained from the Development

Katsuhisa Maruyama,[†] Tetsuo Yamamoto,[†] Hiroki Ikatani,[†]
Masahiro Ishizuka,[†] Akinao Harada[†] and Takashi Hirai[†]

This paper describes our software models for a blimp control system, activities of its development, and the results of the actual flight in the MDD Robot Challenge 2005. It also discusses insight obtained from the develoment.

1. はじめに

モデル駆動開発 (MDD: Model-Driven Development)[1] を適用して自動飛行可能な飛行船制御システムを設計・実装する MDD ロボットチャレンジ（以下，MDD2005 と呼ぶ）が第 2 回目を迎えるという．そこで，大学に所属する者（どちらかというと実践よりも理論や概念を重要視する人々）がどの程度通用するのかを試してみたくなり，MDD2005 に教員 2 名（丸山，山本）と学部 4 回生 4 名（筏谷，石塚，原田，平井）で参加した．

本稿では，まず筆者らのチームが開発したソフトウェアに関して，いくつかのモデルを取り上げて開発活動を簡単に説明し，チャレンジにおける飛行結果を紹介する．その後，開発における問題意識を述べ，筆者らの採用したアプローチに関してソフトウェア工学的観点から考察する．最後に，参加者の所感を含む結論を述べる．

2. 開発活動とモデル一覧

筆者らのチームは，飛行船制御システムにおいて，**図 1** の基地局 PC 上で飛行船を制御するソフトウェア（図 1 の飛行船システム）のみを持参し，当日のチャレンジに臨んだ．飛行船やセンサなどのハードウェア，および，飛行船やセンサに搭載されるソフトウェアに関しては主催者側で用意したものを使用した（ただし，開発ソフトウェアのテストのため，研究室ではハードウェアの一部やセンサ用ドライバも作成している）．

2.1 開発チーム

開発に当たっては，以下のようにチームを 2 つの班に分けた．

設計班（丸山，石塚，平井） 分析モデルと設計モデルの作成，モデルの変換，ソースコードスケルトンの生成

実装班（山本，筏谷，原田） ソースコードの作成，ビルドとテスト，テスト用ハードウェアの組立て，センサ用ドライバの作成，提供されるソフトウェアの動作確認

開発当初は，モデル変換器を作成する班の編成も視野に入れていたが，研究開発に時間がとれなかったため，実際のモデル変換は手動で行うこととした．

参加学生に関しては，実際のソフトウェア開発の経験がないものの，C および Java 言語に関する（一部の学生は C++言語に関しても）プログラミング技術を有する．教員 2 名は大学において研究室を共同で運営しており，さらに参加学生はすべて同研究室所属ということもあり，互いのスキルや専門知識の学習度を

† 立命館大学情報理工学部情報システム学科
Dept. of Computer Science, Ritsumeikan University

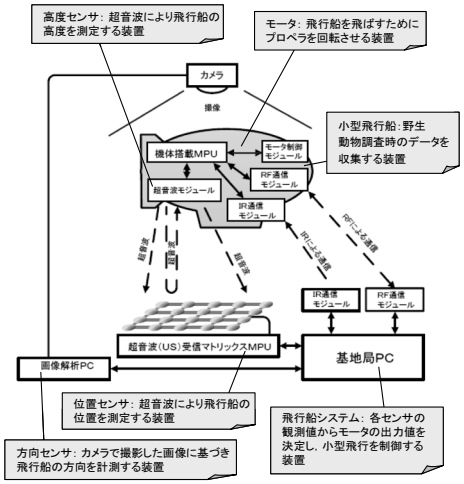

図 1　飛行船制御システム構成図[5)]

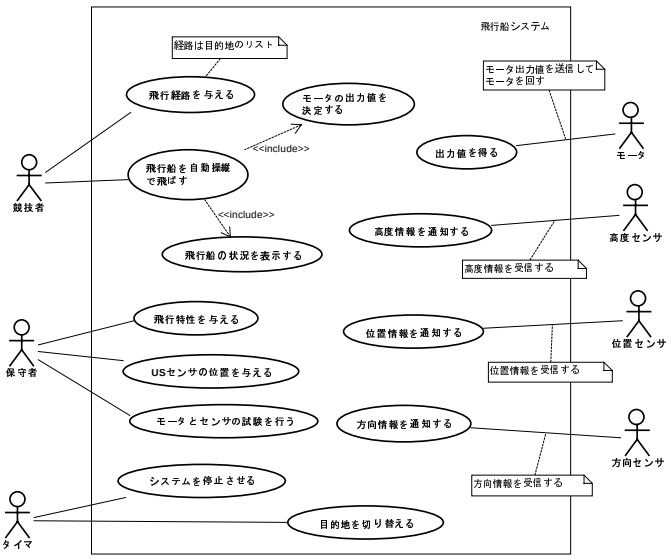

図 2　飛行船システムの要求モデル

ある程度知っている．

2.2　開 発 工 程

今回の開発における各工程，および，それぞれの工程において作成したモデル（ダイアグラム），ソフトウェア，ハードウェアを以下に示す．

(1)　要求の分析： 要求モデル (2 つのユースケース図)，分析モデル (10 のアクティビティ図)

(2)　PIM (Platform Independent Model) モデルの作成： 設計モデル (1 つのクラス図，11 のシーケンス図，3 つの状態図)

(3)　PSM (Platform Specific Model) モデルの作成： モデル変換図 (4 つのクラス図)，詳細設計モデル (1 つのクラス図，12 のシーケンス図)

(4)　設計モデルの洗練： フレームワーク図 (1 つのクラス図)，最終版設計モデル (1 つのクラス図)

(5)　実装とテスト：C++ソースコード，実行用および動作確認用スクリプト，飛行船搭載用・各センサ情報取得用ハードウェア (OAKS16[2)])，飛行シミュレート用センサ情報（テストケース）

モデリングには，UML モデリングツール JUDE[3)] を用いた．上記のダイアグラムとソースコードの一覧と入手方法を付録 **A.1**, **A.2** に示す．

繰り返し型ソフトウェア開発プロセスを採用し，上記の各工程を相互に行き来してソフトウェアを完成させた．実際には，モデルの洗練（PIM モデルや PSM モデルの再作成）に関して 11 回の繰り返しが行われた．ソースコードに関しても，PSM モデルの変更に応じて，5 回以上の書き直しが行われている．

各工程における作業と作成したモデルの説明を **2.2.1**∼**2.2.5** に示す．

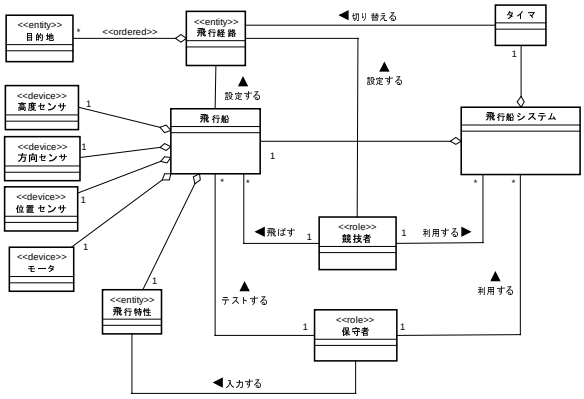

図 3　飛行船システムの概念モデル

2.2.1　要求の分析

MDD ロボットチャレンジ 2004「チャレンジャへの司令」[4)]，MDD ロボットチャレンジ 2005 競技仕様書[5)] に基づき，要求モデルを作成した．与えられた飛行船制御システムの構成と主な構成要素の役割を図 1 に示す．

要求をまとめるにあたり，2 つのシステムのユースケース図を作成した．1 つは，小型飛行船を用いて野生動物観測を実施する調査システム全体のユースケース図（付録 A.1 の**要求モデル**1/2）である．今回の開発では，飛行船制御システムのプロトタイプのみを作成する．開発する飛行船システムのユースケース図を**図 2**（付録 A.1 の**要求モデル**2/2）に示す．

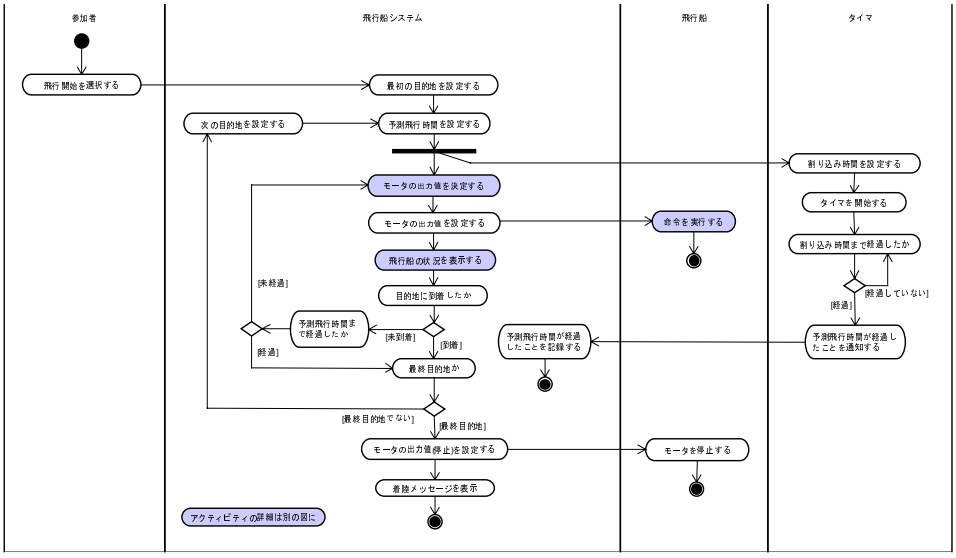

図 4　分析モデル（アクティビティ図：「飛行船を自動操縦で飛ばす」）

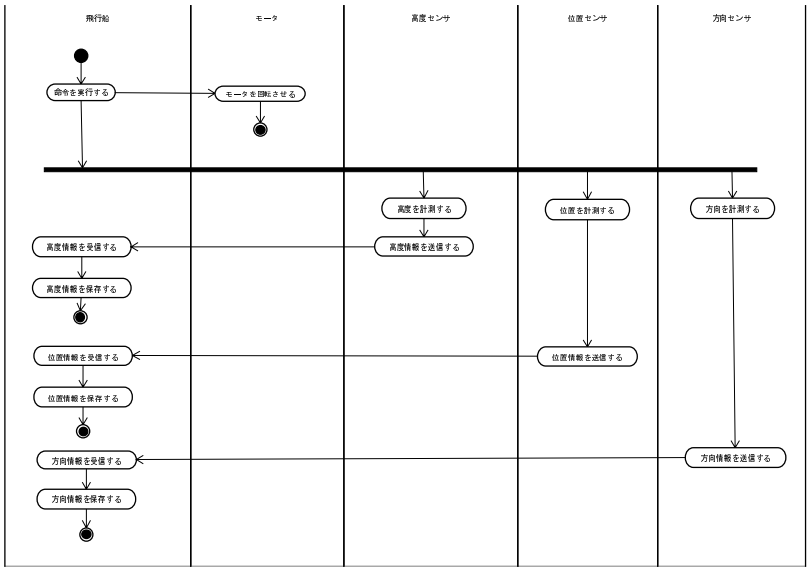

図 5　分析モデル（アクティビティ図：「命令を実行する」）

ユースケースを作成する際には，チャレンジ前日および当日において動作確認がしやすいように，競技者だけでなく，保守者から見た機能も検討した．図 1 のシステム構成図と図 2 のユースケース図に基づき作成した飛行船システムの概念モデルを**図 3**（付録 A.1 の**分析モデル**1/10）に示す．概念モデルにおいて，`<<entity>>`タイプクラスはシステム内に保持されるデータを指す．また，`<<device>>`タイプクラスは，ハードウェアデバイスとの境界あるいはデバイスを仮想化したものを指す．`<<role>>`タイプクラスは，システムの利用者を指す．図 3 において，システム化の対象は競技者と保守者を除くクラスである．

図 2 の各ユースケースに対して，アクティビティ図を記述することで，要求分析を行った．今回の開発では，ユースケース図においてシナリオを記述せず，アクティビティ図により利用手順を明確化した．ユースケース「飛行船を自動操縦で飛ばす」とその内部アクティビティ「命令を実行する」に対応するアクティビ

現象		障害	検出	対策
飛行船が一定時間以内に目的地に到着しない	飛行船の制御不能	基地局から指令送れず	対処しない	
		モータ回らず		
	センサ情報が間違い	高度情報が得られない	対処しない	
		位置情報が得られない	妥当な位置情報が取得できない	旋回飛行を実行し，制御可能になるまで待つ
		方向情報が得られない	対処しない	
	原因不明複合要因	—	タイマにより検出	目的地を切り替える

図 6　非正常処理の検討

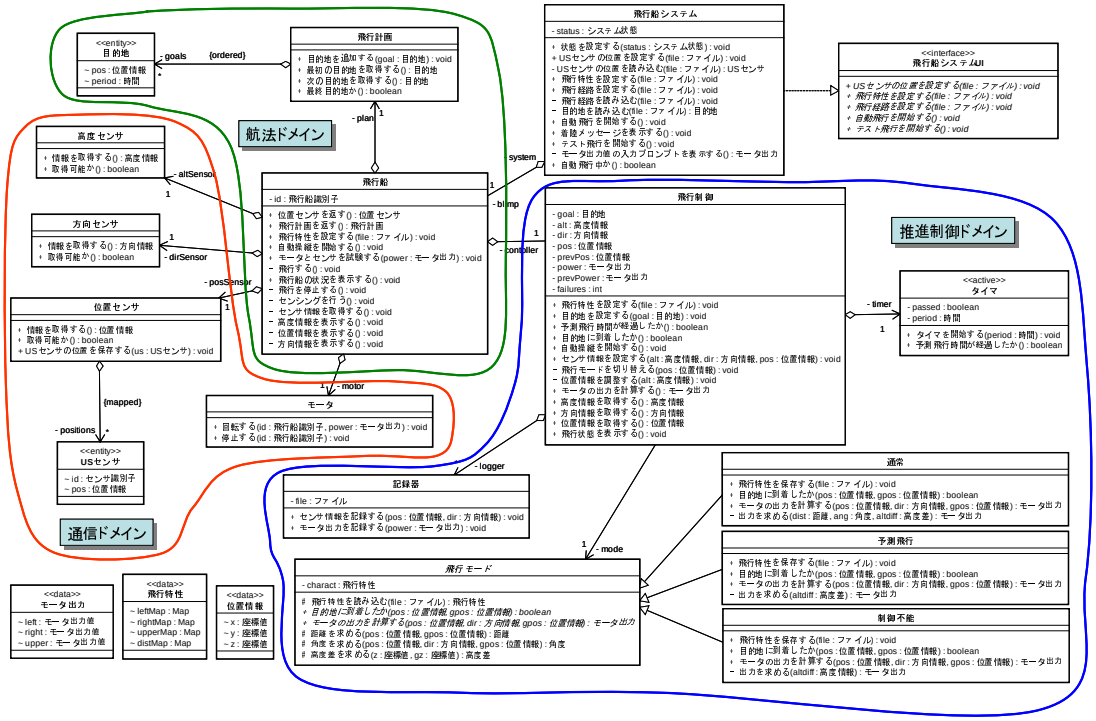

図 7　設計モデル（クラス図）とドメイン分割の様子

ティ図を**図 4** と**図 5** に示す．モータ，高度センサ，位置センサ，方向センサ，タイマ，それぞれのクラスのユースケースに関しては，機能が明確であるという理由から，単独でアクティビティ図を記述せず，別のアクティビティ図に含む形をとった．

この分析モデルでは，自動操縦に関する計算のほとんどを飛行船システム（基地局 PC）で行うこととしている．また，システムの主な責任がモータの回転を制御することであるため，モータ出力値の計算とモータへの出力値の送信を独立に実行する必要はないと判断し，図 5 においてモータへの送信は同期的とした．高度センサ，位置センサ，方向センサに関してはその情報が必ず規定時間で取得できるとは限らないため，それらのアクティビティは非同期実行とした．

要求分析においては，ユースケースにおける振る舞いの定義と同時に，非正常処理の検討も行った．**図 6** に検討した非正常処理を示す．検出が困難なものを対象外とし，位置情報が得られない場合と原因不明で飛行船が目的地（中間地点）に到着しない場合にのみ対処することとした．

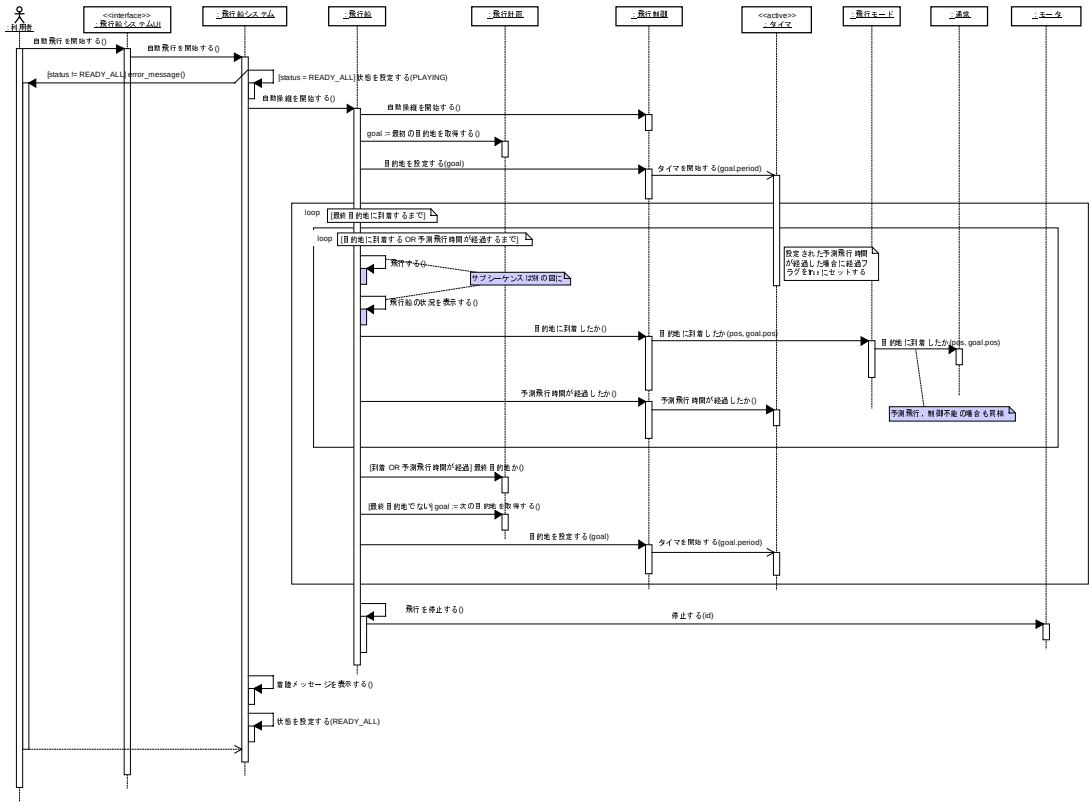

図 8　設計モデル（シーケンス図）

2.2.2　PIM モデルの作成

PIM モデルとして，クラス図，シーケンス図，状態図を作成した．クラス図ではシステムの静的構造を，シーケンス図と状態図では動的側面を表現する．図 3 の概念モデルに基づき，主要なクラスを抽出し，シーケンス図と状態図を作成する過程でクラス図を洗練した．

洗練後の設計モデルのクラス図とドメイン分割の様子を**図 7**（付録 A.1 の**設計モデル**1/15）に示す．概念モデルにおける飛行経路クラスの処理は，飛行計画クラスで管理することとした．目的地の管理を飛行計画に任せることにしたため，その作業に関わる飛行船システムの責任の一部を飛行船クラスに移動した．飛行船クラスと飛行計画クラスは，航法ドメインとし，最終目的地への到着に対して，途中経路（通過点）の選択と各センサおよびモータとのやり取りを管理する．機体の飛行特性に応じたモータ出力を決定する責任は飛行船クラスから分離し，推進制御ドメインに属する飛行制御クラスで行うこととした．飛行制御クラスは，現在の飛行船の状態を管理し，各センサから取得した情報に基づきモータ出力値を計算する．さらに，各センサクラスとモータクラスは，実際のデバイス（ハードウェア）との境界であり，開発システムでは無線あるいは赤外線による通信により情報の取得と命令の実行を行うため，これらのクラスを通信ドメインに分類した．

飛行船システムクラスは，飛行船に関する情報の設定や作業の種類（自動操縦かテスト飛行か）を管理する．競技者および保守者が実行可能な振る舞いは飛行船システム UI インタフェースで与えられ，それぞれの振る舞いに対してシーケンス図を作成した（付録 A.1 の**設計モデル**2/15 ～ 12/15）．**図 8** に，飛行船システム UI インタフェースのメソッド「自動飛行を開始する」に対応するシーケンス図（付録 A.1 の**設計モデル**5/15）を示す．

作成した状態図は，飛行船システム，飛行制御，タイマに関する 3 つである（付録 A.1 の**設計モデル**13/15 ～ 15/15）．**図 9** に，飛行制御クラスの状態図（付録 A.1 の**設計モデル**14/15）を示す．この状態図に基づき，Strategy パターン[6] を採用し，飛行船の制御アルゴリズムを飛行モード（飛行船の状態）により切り替える．このため，各状態に応じてクラス（図 7 の通

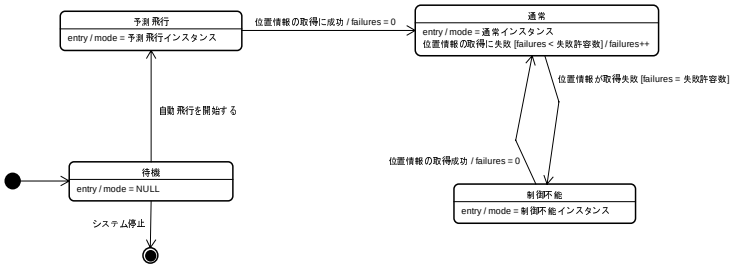

図 9　設計モデル（飛行制御クラスに関する状態図）

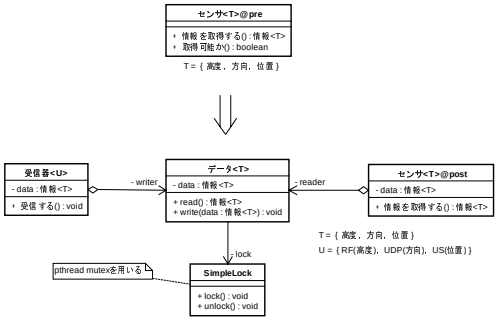

図 10　モデル変換図（センサに関する変換）

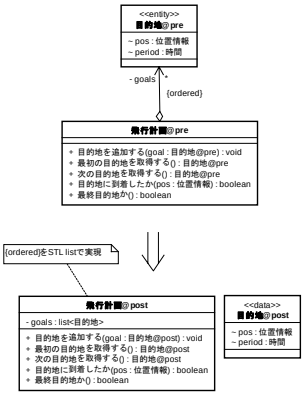

図 11　モデル変換図（<<ordered>>の変換）

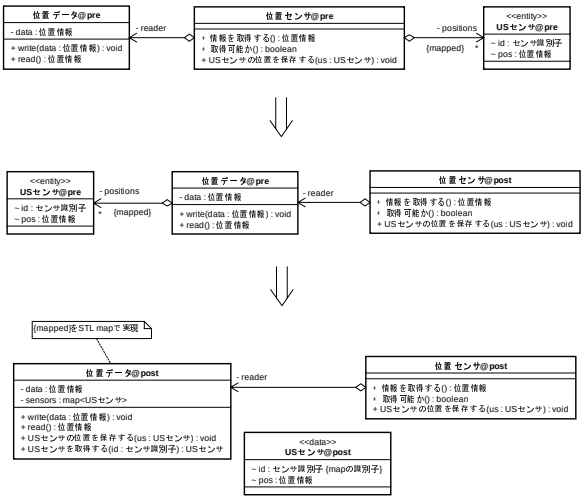

図 12　モデル変換図（<<mapped>>の変換）

常，予測飛行，制御不能）が用意されている．

2.2.3　PSM モデルの作成

動作環境や開発言語を意識し，PIM モデルから PSM モデルに変換する．ここで，飛行船システムの動作環境は，UNIX (FreeBSD および Linux)，開発言語は C++とした．変換は主に以下に示す 2 つの観点で行った．

(a)　通信処理のスレッド化

高度情報，位置情報，方向情報を取得するオブジェクトをメインスレッドと別のスレッドで実行するように，高度センサクラス，位置センサクラス，方向センサクラスに対して，**図 10**（付録 A.1 の**モデル変換図**1/4）に示す変換を適用した．<T>はセンサから受け取るデータの種類を表すパラメータ，<U>はセンサの種類を表すパラメータである．受信器<U>が外部デバイスからデータを受け取る部分であり，この処理は POSIX thread (pthread) を用いて生成されたスレッドで独立に動作する．

通信処理のスレッド化に伴い，送受信において共有されるデータのロックが必要になる．このロックには，Read-Write Lock パターン[7] を用い，SimpleLock クラスの提供する lock メソッドおよび unlock メソッドで実現する．データの書き込みおよび読み込み前に lock メソッドを呼び出し，データの書き込みおよび読み込み後に unlock メソッドを呼び出す．これらのシーケンスを付録 A.1 の**詳細設計モデル**8/13, 13/13 に示す．今回の開発では，比較的単純なロック器（SimpleLock クラス）を作成したが，このクラスを置き換えることで複雑なロック機能を採用することもできる．送信に関しては，要求分析において独立スレッドで実行する必要性はないとしたため，同期呼び出しで実現した．

(b)　クラスから属性への変換

C++による実装を考慮し，<<entity>>タイプクラスを属性（メンバ変数）に変換する．用意した変換は，順序つき集約<<ordered>> と識別子による検索つき集約<<mapped>>に関する 2 種類である．それぞれの変換を**図 11**（付録 A.1 の**モデル変換図**3/4）と**図 12**（付録 A.1 の**モデル変換図**4/4）に示す．<<ordered>>および<<mapped>> はそれぞれ C++ STL (Standard Tem-

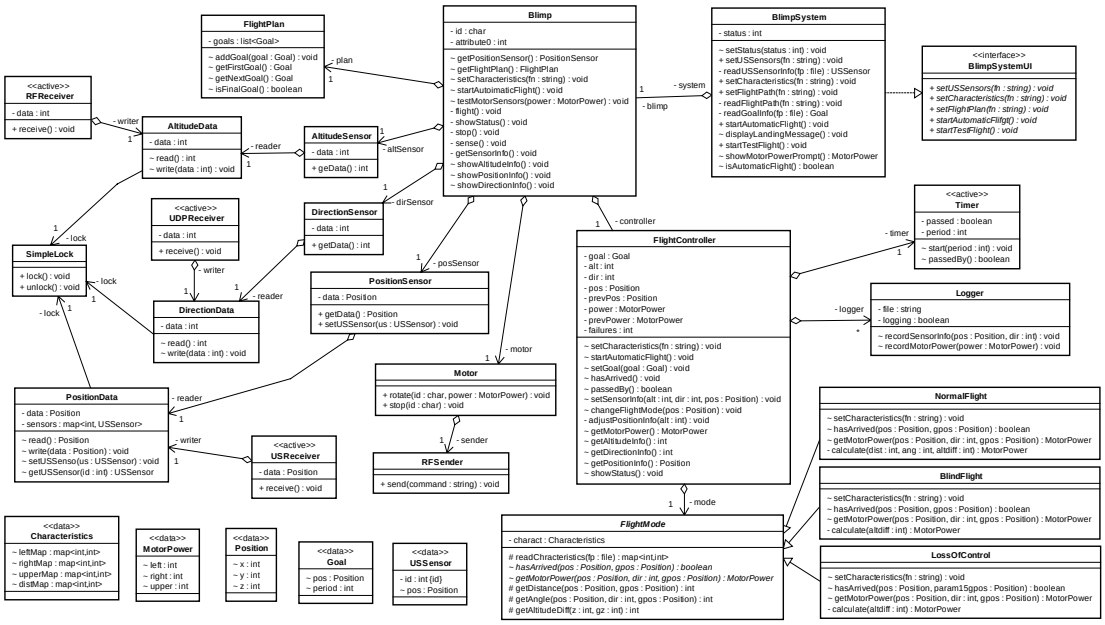

図 13　詳細設計モデル（クラス図）

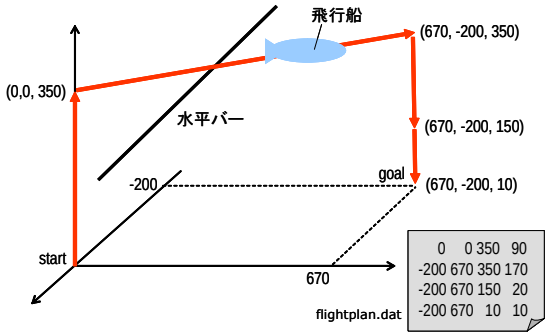

図 14　通過点と飛行経路

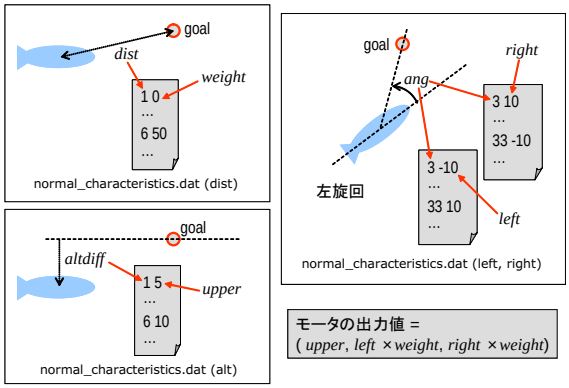

図 15　モータ出力値の決定方法

plate Library) の list および map で実現した．また，<<entity>>タイプクラスは，STL のコレクションに格納される具体的なデータタイプを表す<<data>>に変換した．

上記の変換 (a) と (b) を適用した後のクラスを，**図 13**（付録 A.1 の**詳細設計モデル**1/13）に示す．この時点で，名前の置換表を用意し，クラス名，フィールド名，メソッド名をアルファベット表記に置換した．さらに，ループ終了条件と分岐条件の詳細化を行い，PSM モデルのシーケンス図を作成した（付録 A.1 の**詳細設計モデル**2/13 ～ 13/13）．

ここで，推進制御アルゴリズムに関して実現方法を述べておく．飛行船システムは，最終目的地までの経路を複数の通過点のリストで表現する．**図 14** に総合飛行における通過点および目的地（最終通過点）と，（理想的な）飛行経路を示す．通過点は flightplan.dat ファイルで与えられる．データの第 1~3 列は通過点の座標，第 4 列は通過地点までの予測経過時間（秒）を指す（タイマ割り込みによる通過点切り替えに使用する）．ただし，実際のチャレンジでは，出発地付近では水平方向の位置情報の取得はできないため，予測飛行 (Blind) モードで飛行を開始し，通常 (Normal) モードに推移するまで左右のモータに関して一定の出力を与える指針を選択した（このため，離陸直後に垂直には上昇せず，前進上昇となる）．また，センサの感度が予想よりも大幅に悪かったため，予測経過時間として比較的大きな値を設定した．

開発した飛行船システムでは，機体の飛行特性として，次の 4 つのマップをそれぞれのモードに対して用意し，モータの出力を決定する．各マップにおける値の組は，X_characteristics.dat ファイルで与えられる（X は normal, loss, blind のどれか）．**図 15** にモータ出力値を決定する方法を示す．

距離マップ Characteristics クラスの distMap フィールドに格納される．distMap = (現在地と目的地との距離を正規化した値 $dist$，左右モータの出力への重み $weight$)

高度差マップ Characteristics クラスの upperMap フィールドに格納される．upperMap = (高度差を正規化した値 $altdiff$，上昇モータの出力値 $upper$)

角度マップ（左モータ用） Characteristics クラスの leftMap フィールドに格納される．leftMap = (目的地方向と進行方向の角度を正規化した値 ang，左モータの出力値 $left$)

角度マップ（右モータ用） Characteristics クラスの rightMap フィールドに格納される．rightMap = (目的地方向と進行方向の角度を正規化した値 ang，右モータの出力値 $right$)

これらのマップを検索して得られる 4 つの値から，モータに送信する出力値を以下の 3 つ組で定義する．

($upper$, $left$ $\times weight$, $right$ $\times$ $weight$)

第 1 パラメータは上昇モータへの出力値，第 2 パラメータは左モータへの出力値，第 3 パラメータは右モータへの出力値である．

2.2.4 設計モデルの洗練

詳細設計モデルに対して可変部分を検討し，フレームワーク化を行った．導入する可変部分を**図 16**（付録 A.1 の**フレームワーク図**1/1）に示す．このフレームワークでは，main 関数（あるいは main から最初に呼ばれる関数）で 3 つの受信器用のスレッドを生成し，それらをそれぞれの受信器オブジェクトに割り付ける．その後，BlimpSystemUI クラスのメソッドを呼び出すことで，ユースケースに基づく動作を実行する．Blimp クラスのメソッドの一部の可視性を protected に変更することで，飛行船の状況の表示方法に関するメソッドを再定義できるようになっている．新しい飛行モードを追加する場合は，それを実現したクラス（図では LowLevelFlight）を FlightMode から派生させ，同時に FlightController クラスの changeFlightMode メソッドを派生クラス（図では MyFlightController）で再定義する．最終的に完成した設計モデルのクラス図は付録 A.1 の**最終版設計モデル**1/1）のようになる．

2.2.5 実装とテスト

設計モデルの実装においては，JUDE のスケルトンコード生成機能を用いて，クラス図（付録 A.1 の**最終版設計モデル**1/1）から Java ソースコードを生成し，それを C++コードに手動で変換した．各クラスの振る舞いは，シーケンス図（付録 A.1 の**詳細設計モデル**2/13 ～ 13/13）に基づき，コードを記述した．

今回の開発では，次の 4 つの実装を作成した．

- fsel: 自動飛行用プログラム（チャレンジ本番用）
- fsel-manual: モータ出力を手動で入力し，飛行させるプログラム（テスト用）
- fsel-random: センサ情報をランダムに生成し，送受信器および飛行アルゴリズムの動作確認を行うプログラム（テスト用）
- fsel-comm, client: センサ情報（テストデータ）を TCP/IP ネットワークを通して入手し，自動飛行させるプログラム（飛行シミュレート用）

C++のファイル数（GUI 表示用コードも含む）はそれぞれ*.cpp が 30 個，*.h が 27 個，*.c が 1 個である．コード行数は 2469 行 (NCNB☆) であった．

テストに関しては，実際に飛行船搭載用および各センサ情報取得用ハードウェアの一部を作成し，センサ情報の送受信を確認した．また，適当なセンサ情報の時系列データをあらかじめ計算により求めておき，それをテストケースとして飛行シミュレートを行うことで動作確認を行った．ただし，テストデータの計算には非常に時間がかかるため，シミュレートによる十分な動作確認はできなかった．

2.3 飛行結果

筆者らのチームが実際の飛行船ハードウェアを見たのは，チャレンジの前日であった．前日の試験飛行で実環境での動作テストおよび飛行特性の調整を行い，当日のチャレンジに臨んだ．成績だけを見ると総合 2 位という高い順位ではあったが，反省も多いチャレンジであった．以下に，当日の飛行結果を述べる．

2.3.1 コンパルソリー

垂直方向に順調に上昇したものの着地に失敗した．あらかじめ設定した最上点に到着する直前に高度情報が取得できなくなり，モータが停止したため下降し始めた．この時点で課題として与えられた高度までは上昇していたため得点が加算されたが，今度は地表近く（地面に衝突しはね上がった後）において高度情報が取得できたため，再度上昇を始めた．センサ情報に関する記録を見る限り，飛行船に搭載されている赤外線

☆ Non-comment, Non-blank

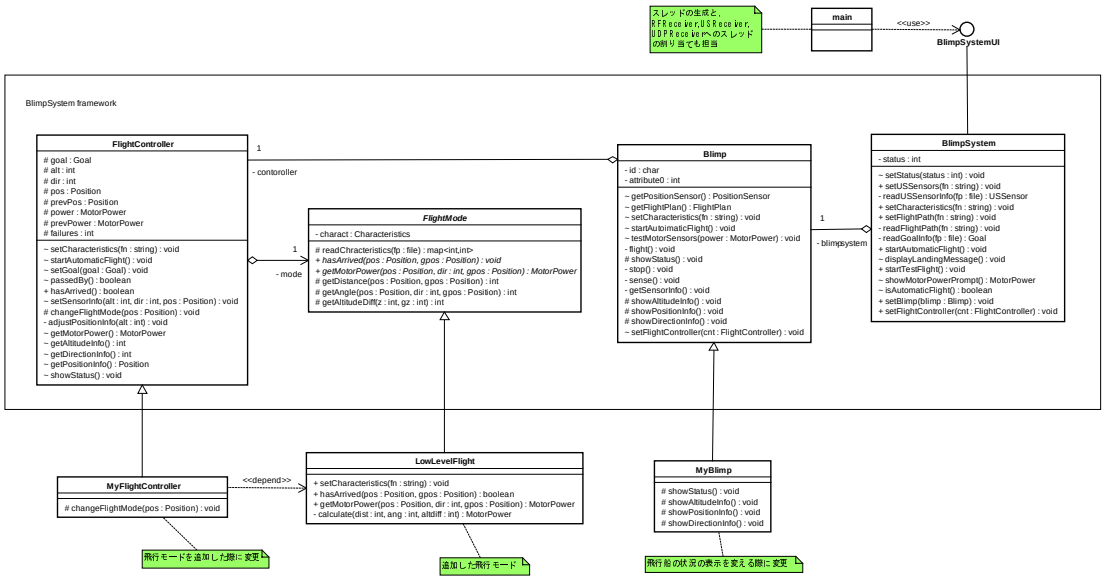

図 16　フレームワーク化（クラス図）

高度センサは，地表からの距離が遠くなる（飛行船の高度が高くなる）と，その値の精度が極端に悪くなり，高度制御に使えるような値の範囲になかった．このため，最上点への到達判定に失敗し，何度も上昇を試みるという飛行結果になった．最終的には，予測経過時間に達し，通過点が最終目的地（地表）に切り替わり，出発地点から少し離れて着陸した．

2.3.2　総 合 飛 行

午前のコンパルソリーの結果を受けて，飛行特性の調整とプログラムの修正を行い，総合飛行に臨んだ．水平バーは越えたものの，水平方向および垂直方向とも現在位置を失い，観客席上空で蛇行した．通常飛行への復帰見込みがないため，航行不能との判断より，モータ出力を停止し，飛行船を回収した．

ここで，総合飛行前のプログラム修正に関して簡単に触れておく．コンパルソリー競技時に高度センサと位置センサの情報を記録しておいたが，位置センサに関する情報がほとんど取得できていなかった（4 地点のセンサ情報が揃うどころか 1 地点のセンサ情報が取得できるかどうかという状況であった）．当初のアルゴリズムでは，4 つのセンサの値から連立方程式を解くことで飛行船の現在位置を求めていたが，このような状況では現在位置が確定しないと判断し，現在位置の特定アルゴリズムを急遽変更した．変更後のアルゴリズムでは，取得できたセンサ情報のうち一番飛行船に近いと判断できた（値が一番小さい）ものを採用し，飛行船はその地点の上空に存在すると仮定して，モータの出力値を決定する．

総合飛行時のセンサ情報を分析した結果でも，位置センサの情報は取得できていなかった．また，高度センサに関しても，コンパルソリー競技と同様に高度が高いところでは，妥当な値を示してはいなかった．コンパルソリー競技と異なり，飛行船が非常に高く上昇した原因は，飛行船の浮力に関するキャリブレーションの誤りにある．筆者らは，コンパルソリー競技時の飛行結果（モータ出力値がゼロで飛行船はかなり速い速度で落下する）を受け，飛行特性データを上昇方向に微調整した（上昇方向の出力にバイアスをかけた）．このような飛行特性の設定において，正確な高度情報が取得できなかったため，飛行船は下降できず一方的に上昇した．

実際の飛行船ハードウェアに触れたのが前日であったこと，センサ情報の精度が期待から大きくはずれたことが，このような飛行結果になってしまった原因の一部であるものの，このような状況はモデリング時に想定できたはずである．ハードウェア特性を十分に取り込んだ設計・実装になっていなかったこと，信頼性や冗長性にする検討が十分でなかったことが，筆者らのチームの最大の反省点である．

3. ソフトウェア工学的観点からの考察

本節では，筆者の丸山がソフトウェア工学研究者あるいは教育者の立場から，開発を通して得られた見識を述べる．

ソフトウェア工学に基づきソフトウェアを開発するにあたり，ISO9126 の品質特性を意識することは非常に重要である．よって，MDD2005 のモデル審査においても，この視点で評価が行われ，モデル講評が公開されると思われる☆．そこで，本稿では，筆者の専門分野に関わる特性の中から，実際の開発を通して重要であると感じた特性に焦点を当て，考察を述べる☆☆．

3.1 問題意識

今回のようなハードウェアとソフトウェアが混在したシステムを開発する場合，ハードウェアとソフトウェアの特性を意識しておくことは重要である．両者を比較した場合，ソフトウェアでは導入後の修正が可能である点が大きく異なる．少し補足しておくと，ソフトウェアは修正が容易というわけではなく，量産・回収・流通コストを考えると，ソフトウェア修正で済ます方がたいていの場合に有利である．このような状況では，顧客の要求変更，将来の（短期的な）機能拡張，ハードウェアにおける誤りへの対応の多くを，ソフトウェアが引き受けると言っても過言ではない．

また，今回の筆者らの参加形態と同様に，現実の開発においても，ハードウェアを自分の組織で開発しないことは多い．つまり，ハードウェアは別組織から供給される，あるいは，市販品を利用するという状況である．ハードウェア仕様や動作が事前に確認できるという利点を考えると，ソフトウェアのモデリング前にハードウェア実機が手にはいることが望ましい．しかしながら，システム開発の短期化に伴い，ハードウェアとソフトウェアの並行開発は当然となり，前提としたハードウェアの仕様が突然変更されたり，ソフトウェアに対する要求や制約が変わることも珍しくない．さらには，組込みシステムの場合，さまざまな種類のハードウェア環境が想定され，異なるハードウェア上でそれぞれのソフトウェアが（ほぼ）同じ機能を提供すること（クロス開発）が要求される．

このように，近年のシステム開発では，ソフトウェアに対する要求（あるいは仕様）は開発中に変化することが多いと考えるのが自然である．実際，今回の開発においても，想定していたハードウェア仕様や競技ルールと，最終的に決定された仕様やルールとの違いを何度も経験した．よって，今回のような開発形態におけるソフトウェアモデリングでは，ハードウェア仕様や競技ルールの変更にどの程度対応できるかが重要である．言いかえると，ソフトウェアモデルがハードウェア仕様やルールの差異を吸収できるような設計，および，吸収できない場合を想定した対策が開発を成功させる鍵となる．

これまでの観点は主にソフトウェア変更に関するものであったが，実際の開発現場への展開を考えた場合，もう一つ重要な観点として再利用がある．開発時間の短縮化とプロダクトの種類の多さを考えると，再利用を意識して作成していないモデルは実用的でない．ここで注意しておかなければならないこととして，再利用を意識するとは，既存のプロダクト（あるいはその一部）を単に利用することではない．新規にソフトウェアを開発する際，既存ソフトウェアで利用できる部分がないかどうか探すことは開発コストを削減する一つの方法ではあるが，現実的には再利用を意識して作成されていないプロダクトを再利用することは困難である．より大きな粒度かつ有用な再利用を期待するのであれば，オブジェクト指向設計の原則 (The Principles of Object-Oriented Design)[10] などを参考とし，再利用は設計段階から組み込んでおくべきである[11]．

3.2 アプローチと展望

今回の開発において，ソフトウェア変更と再利用に対する筆者らのアプローチと展望を議論する．

3.2.1 変更を取り込んだモデルを目指して

今回の開発において，あらかじめ予測できる変化に関しては，できる限り開発の初期段階で認識し，モデルで吸収しておくことを試みた．以下に，筆者らの採用した 4 つのアプローチを示す．

(1) 開発早期でのハードウェアデバイスの抽象化
(2) PIM モデル構築時におけるドメイン分割
(3) 抽象クラスを用いた飛行モードの切り替え
(4) テストを考慮した要求分析とフレームワーク化

まず，(1) について説明する．設計モデルにおいて，ハードウェアの差異によりモデルの違いが現れないように，概念モデル構築時にハードウェアデバイスを認識し，`<<device>>`タイプクラスとして抽象化しておいた．設計モデルでは，このタイプクラスのインタフェースだけを規定し，全体のモデリングを行った．最終的な実装において，`<<device>>`タイプのクラスと実際のデバイスを接続するデバイスドライバの役割を果たす（設計モデルには登場しない）クラス（`COM.cpp`）を別に作成することで，実際のハードウェアを制御している．このようにすることで，ハードウェア仕様が定義あるいは変更された際には，COM クラスの実装だけを書き換えることで対応できた．

☆ 文献 9) によると，2004 年の審査は合目的性，正確性，標準適合性，理解容易性，管理容易性，再利用性，変更性・拡張性，テスト容易性でモデル評価が行われている．

☆☆ 他の品質特性が重要でないといっているわけではない．合目的性や正確性はソフトウェアを開発する上で明らかに重要である．

(2) に関しては，図 7 に示すドメイン分割が該当する．飛行船の飛行制御を，航法ドメイン，推進制御，通信ドメインに分離した大きな理由は，それぞれのドメインが要求する知識が大きく異なることである．航法ドメインでは，個々の通過点に飛行船が確実に到着することを前提とした上で，どのような経路を選択すれば高得点を狙えるか（本来の目的では，動物の生体調査に有利か）に重点をおく．よって，このドメインに関わるクラスを実装するには，経路探索問題（さらには調査対象の動物の行動特性）の知識が必要である．これに対して，推進制御ドメインでは，飛行船をいかに正確かつ早く指定地点に誘導するかが問われる．このドメインに関わるクラスを実装するには，飛行船の空力特性や制御工学に関する知識が求められる．通信ドメインに関わるクラスの実装には，それぞれのデバイスに関する知識と通信プロトコルに関する知識が必要である．

このようなドメインの分離は，将来の変更要求（およびその影響）を各ドメインに閉じ込める可能性が高い．例えば，チャレンジのルール（調査対象の動物）が変更された場合，飛行計画に関わるものであれば航法ドメイン，推進制御アルゴリズムに関するものであれば推進制御ドメインで対処する．また，飛行船ハードウェアが変更された場合，それがセンサやモータなど個々のデバイスに関するものであれば通信ドメイン，飛行特性に関するものであれば推進制御ドメインで対処するという具合である．将来の変更要求を完全に予測することはできないが，今回の開発途中および来年度以降のチャレンジを想定した場合，ルール変更，飛行船の機体変更，デバイス変更の可能性が高いと考えた．

(3) に関しては，モデリング開始時点では，競技ルールの詳細とセンサ情報が取得できる範囲が未確定であったため，推進制御アルゴリズムにおけるセンサ情報の利用方法を隠蔽する目的で，飛行モードという抽象概念を導入した．これにより，各モードにおける具体的なアルゴリズムの決定時期をできる限り後ろの工程に引き延ばすことができ，さらには，アルゴリズムを独立に定義することができるようになる．実際，具体的アルゴリズムが未決定のまま，センサ情報により飛行モードが切り替わることを試験した．また，今回の開発では，図 6 で検討した非正常的な飛行はすべて制御不能（モード）で対処するように設計したが，それぞれの現象に対して特殊なモードを導入する対処も十分に考えられる．以上より，飛行モードの実現には Strategy パターンを採用した．これにより，さまざまなモード（着陸付近，離陸直後，低空飛行，旋回飛行など）を将来において容易に追加，あるいは，既存のモードを容易に削除可能である．今回の開発でも，その当初においてコンパルソリー競技専用の飛行モードを導入していたが，開発の終盤でそれを削除した．この作業は非常に容易であった．

(4) のテスト容易性に関して説明する．筆者らのチームはハードウェアおよびドライバは主催者側から提供されたものを使用する．そこで，図 2 に示すように，競技者だけでなく保守者から見た機能も意図的に明示した．開発時に，保守者のアクティビティに関しても検討し，設計および実装に組み入れておくことで，テストにおけるそれぞれのアクティビティが，設計モデルや実装コードにおいてどの部分に対応するのかが明確になる．例えば，図 2 における「モータとセンサの試験を行う」は，図 7 における飛行船システムクラスのメソッド「テスト飛行を開始する (startTestFligh)」で実現され，そのシーケンス図（付録 A.1 の**設計モデル**6/15 あるいは**詳細設計モデル**6/15）を見ることで，どのようなテストが行われているか理解しすくなり，誤りの発見が容易になる．

さらに，設計モデル洗練の際のフレームワーク化において，各センサのデータ受信処理用スレッドの生成と，それぞれのスレッドの主処理を外部から指定できるようなフレームワークを構築した（図 16）．これにより，1 つのフレームワークから，**2.2.5** に示す 4 つの実装を容易に作成できた．テストまで含めて設計および実装を行っておくことで，チャレンジ前日および当日において，テストの実施自体も効率的に行うことができた．

ここで述べたアプローチ (1) ～ (4) を採用することにより，変更を取り込んだモデル構築に関して，ある程度の成果は得られた．しかしながら，今回の開発は 1 回目のチャレンジということもあり，変更要求の検討が十分であったとは言えない．機会があれば，昨年度と今年度の競技仕様の差異や筆者らのチームと他チームとのモデルの差異を調査し，変更を分析しておきたいと考えている．

3.2.2 変更を意識した開発プロセスを目指して

今回の開発では，ハードウェア仕様あるいは要求（競技ルール）がチャレンジ直前に変更されるということが予測できた．そこで，これらの変更に迅速に対応できるように，直線的開発☆ を採用せず，繰り返し型ソ

☆ ウォーターフォール開発は手戻りを禁止しているわけではないため，ここではあえて直線的開発とした．

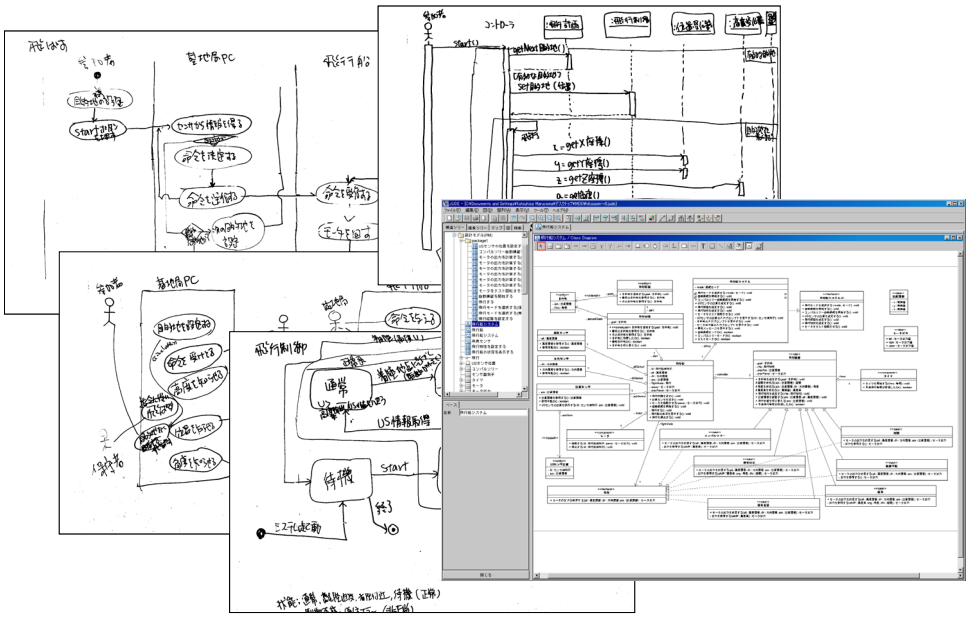

図 17　ホワイトボードを用いたスケッチ（一部）と JUDE を用いたモデリング

フトウェア開発プロセスを採用した．分析段階あるいは設計段階でモデルを十分に吟味し，変更に対する耐性の高いソフトウェアを作成しておけば反復は避けられという意見もあるが，ソフトウェアの開発当初においてあらゆる変更を予想しておくことは不可能である．よって，仕様や要求の変更はある程度当たり前と割り切り，**3.2.1** で述べた抽象化やフレームワーク化などだけで対応できない部分に関しては，変化への対応の早さで補うことを考えるべきである．その際，たび重なる変更は大量の生成物（人工物）を発生させるため，それらの間（モデル，ソースコード，テストなど）の追跡可能性は重要である☆．今回の開発では，モデルとソースコードとの追跡可能性，および，MDD の採用を強く意識し，ソースコードに対する直接的な修正あるいはリファクタリング[8] は避け，必ずモデル上で設計の変更あるいはリファクタリング（UML リファクタリング）を行い，これをソースコードに反映させるプラクティスを徹底した．ただし，モデル変換器を採用しなかったため，数多くの繰り返し作業は実装班に多大な労力を強いることになった．MDD によりモデル変換やソースコードの（半）自動生成が可能になり，それらの間の追跡可能性が高まれば，実装班の負担に関して大幅な軽減が期待できるであろう．

☆ 複数の版の構成要素間の関係を正確に保持可能な（追跡可能性の高い）UML ツールがほしいと強く感じた．

実際に繰り返し型開発を実施して，今回の開発の成功は，チーム内のコミュニケーションが円滑であったことと実装班の実装技術の高さにあると強く感じている．開発初期のモデリングは，主にホワイトボードを用いて設計班と実装班の合同で行った．開発中期からは，JUDE により作成した最新のダイアグラムをスクリーンに投射するなどし，ホワイトボード上で合同レビューを数回実施した．**図 17** にスケッチの一部と JUDE を用いたモデリングの様子を示す．実のところ，レビューの主な目的は学生への教育であったのだが，レビューが設計班と実装班での意識合わせに大きく貢献した．このことが，設計から実装に円滑に移行できた大きな要因だと感じている．また，教育的観点から，今回の開発では，大学の講義にも使える教育用モデル，さらには，来年度以降参加するチームへの参照モデルとなることを目指してモデリングを実施した．このため，モデリング時において，飛行船システムを，いかに素直かつ単純に表現できるかを常に心掛けたつもりである．さらに，センサ情報取得処理のスレッド化やデザインパターンの採用を積極的に行った．このようなモデリング指針は，結果的に教育的効果だけでなく理解容易性の向上にも貢献し，チーム全員がモデルを共有するのに大いに役立った．モデルの理解に多くの時間を費やすようでは，繰り返し型開発は成功しない．繰り返し開発において，そのサイクルを短期化するためには，開発モデルに対するチーム内での情報共有あ

るいは意識合わせが重要であることを実感した．

今回のチームにおいて，モデル班の筆者から見ると，実装班の実装技術は称賛に値する．どのようなモデル変更を行っても，ほぼ 1 ～ 2 日以内に実装とテストが完了し，実行結果および変更の効果が得られた．このような環境は，モデリングにリズムを与えてくれると同時に，モデルを大胆かつ頻繁に変更する勇気を与えてくれる．このような経験は，繰り返し型開発を真に支援する MDD ツールの有用性を裏付けるものと捉えることができる．ただし，ここで述べている MDD ツールとは，単に上流のモデルを厳密に定義すれば，下流のモデルやソースコードを自動生成するものを指しているわけではない（設計者の負担を一方的に増加させるものであってはならない）．このような点から，より厳密なモデル（例えば，Executable UML[12]）の記述を仮定するのではなく，設計者の意図を汲み取り，それに応じた設計判断を取り入れた変換を備えるツールの登場を期待する．

ここで開発方法論に関して少し補足を述べる．**2.2** を注意深く読んだ読者の中には気が付いた人もいるかもしれないが，筆者らの実施した繰り返し型プロセスは，コンポーネントベース開発方法論の KorbA[15] を強く意識している．KobrA における繰り返し活動およびモデル洗練（Realization から Implementation への Refinement）は MDD と非常に相性が良い（と感じる）し，コンポーネント（KobrA では Komponent）の存在を前提したトップダウンとボトムアップの融合開発 (Balanced to-down and bottom-up development) は，組込みソフトウェアにおいて部品化再利用を促す（例えば，今回の開発では各種センサやモータが素直に部品化できる）はずである．さらに，KobrA がソフトウェアプロダクトライン (Software Product Lines)[13),14)] を明確に意識している点も重要である．今回の開発では時間がとれず，プロダクトラインの検討にまで及ばなかったが，プロダクト候補の洗い出しと決定モデルの作成にも取り組んでみたかった．

3.2.3 再利用可能なソフトウェアを目指して

再利用性に関する検討が必要なことは十分承知していたが，時間の都合上，今回の開発では，可変部分や拡張部分（カスタマイズポイントやパラメータ化）の検討によるフレームワーク化以外の作業を特に行わなかった．単純に考えても，各種センサやモータのようなデバイスは再利用可能部品の候補として有力であるため，それらに適切なインタフェースを規定し，利用に関する制約や文脈を文書化しておくべきであった．また，ドメイン分割に応じたパッケージングも考えられるだろう．さらには，MDD においてはモデル変換規則も再利用対象と見なせるため，これらに関しても異なるプロダクトに適用可能なように，変換の系統化や文書化をしておくことは興味深い．

3.3 課　　題

MDD2005 の特徴は，“MDD を用いて”，“組み込ソフトウェア” を開発することである☆．モデルを中心に開発を進めるというスタイルは，MDD の一側面ではあるが，モデルの自動変換やモデルにおける検査の導入は MDD の醍醐味の一つであろう．モデル構築にのみ注力し，開発を通して MDD に関する知見がほとんど得られなかった点に関しては，大いに反省すべきである．モデル変換規則の分類や表記法の検討，変換を意識したダイアグラムの拡張，自動変換によるコードの検査 (review) など，MDD を意識した開発を進めてみたい．

組み込みソフトウェア開発という点では，その開発スタイルの一部でも経験できたことは大きい．しかしながら，筆者らの開発したプロダクトは基地局 PC だけであり，また，ハードウェア特性を十分に検討した上での抽象化とは言えない．特に，ハードウェア特性（例えば，センサの精度や応答性）に関するモデリングが行われていない点が問題であろう．ハードウェアデバイスを単なる抽象的なコンポーネント（あるいはオブジェクト）とみなすには，機能特性だけでなくどのような非機能特性があるのか，非機能特性をどのようにモデルに反映させるのかを考えなければならない．

4. ま と め

本稿では，MDD2005 において筆者らのチーム (FSEL) が開発したモデルと飛行結果を紹介した．さらに，開発経験から得られた見識について述べた．本稿の内容が，MDD2005 の関係者および来年度以降の参加者に有益であることを期待する．最後に，今回のチャレンジへの参加者の所感をもって結論とする．

丸山　非常に忙しかったが，全体的に楽しい取り組みであった．開発を通してソフトウェア工学に関する研究テーマがいくつも見つかったので，今後研究発表していきたい．

山本　当初はソフトウェア，ハードウェア両方制作し参加する予定であったが，ハードウェアの知識不足によりソフトウェアだけでの参加になってしまったのが残念である．また，風や温度などの外的要因に非常に左右されやすいことが分かった．モデ

☆ 開場において実際に飛行船を飛ばすのも特徴ではあるが．

ルを作成する際には，それらの要因を織り込んで行わないと期待通りの結果に結びつかないことを痛切に感じている．

筏谷　組込みには全く縁がなかったのであまりチームに貢献はできなかったが，とてもいい経験になった．組込み開発というかソフトウェアの開発がいかに思い通りに進まないかを思い知った．今回は先生方の力のおかげで何とかなったが，次参加する事になれば学生主体でやっていけたらいいと思う．今度は実機で色々遊びたい．

石塚　このチャレンジを通して，システム構築とそのシステムからハードウェアに情報を伝達して正常に動作させることの困難さを痛感した．飛行船を飛ばすのに実装すべき事柄が多いため，どこから手をつければいいのか判らず四苦八苦することもあったが，これもまたいい勉強，いい経験になったと思う．

原田　ほとんどの学生はハードウェアに対する深い知識がない状況でのスタートだった．当初は飛行船のテスト環境を整えることすら苦労していた．このような状況での総合 2 位という好成績は先生達の指導力のおかげだと思っている．もし次回も参加する機会があれば今度は学生主体のチームで優勝を狙いたい．

平井　当初は，MDD という言葉自体知らなかったが，大会を通して，その考え方や一連の作業を理解することができたと思う．また，大会期間中は，普段は直に見ることのできない他校・企業の方々の頑張っている姿を見て，刺激を受けた．また，自身が参加することになれば，今回を上回る結果を残したい．

謝辞　MDD2005 ロボットチャレンジ推進委員会ならびに審査委員会の皆様，組込みソフトウェアシンポジウム 2005 実行委員の皆様に感謝いたします．このチャレンジを通して，筆者らは貴重な経験をさせて頂きました．また，チャレンジ当日において，情報交換および討論をして下さいました参加者の皆様に感謝致します．

参　考　文　献

1) Selic, B.: The Progmatics of Model-Driven Development, IEEE Software, Vol. 20, No. 5, pp.19–25 (2005).
2) OAKS16: `http://www.oaks-ele.com/`.
3) JUDE: `http://www.esm.jp/jude-web/index.html`.
4) 二上貴夫: チャレンジャへの司令，MDD ロボットロボットチャレンジ 2004, pp.95–98, 情報処理学会 (2005).
5) MDD ロボットチャレンジ 2005 推進委員会・審査委員会: MDD ロボットチャレンジ 2005 競技仕様書，2005 年 10 月 1 日改訂版 (最終版) (2005).
6) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley (1995). (本位田 真一，吉田和樹監訳: デザインパターン, ソフトバンク (1995)).
7) 結城浩: Java 言語で学ぶデザインパターン入門 マルチスレッド編，ソフトバンククリエイティブ (2002).
8) Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley (1999). (児玉公信，友野昌夫，平澤章，梅澤真史訳: リファクタリング, ピアソン・エデュケーション (2000)).
9) 鷲崎弘宜 他: MDD ロボットチャレンジ 2004：モデル講評，MDD ロボットロボットチャレンジ 2004, pp.31–40, 情報処理学会 (2005).
10) Martin, R. C.: *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall (2002). (瀬名啓介訳: アジャイルソフトウェア開発の奥義 ソフトバンク (2004)).
11) Tracz, W.: Software Reuse Myths, ACM SIGSOFT Software Engineering Notes, Vol. 13, No. 1, pp.17–21 (1988).
12) Mellor, S. J. and Balcer, M. J.: *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley (2002).
13) Clements, P. and Northrop, R.: *Software Product Lines: Practices and Patterns*, Addison-Wesley (2001). (前田卓雄訳: ソフトウェアプロダクトライン, 日刊工業新聞社 (2003)).
14) Gomma, H.: *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Siftware Architectures*, Addison-Wesley (2004).
15) Atkinson, C. et al.: *Component-based Product Line Enginnering with UML*, Addison-Wesley (2001).

付　　録

A.1　モデル一覧

本稿で参照しているモデルの一覧を以下に示す．

(1)　要求モデル 1/2 ～ 2/2
(2)　分析モデル 1/10 ～ 10/10
(3)　設計モデル 1/15 ～ 15/15
(4)　モデル変換図 1/4 ～ 4/4
(5)　詳細設計モデル 1/13 ～ 13/13
(6)　フレームワーク 1/1

(7)　最終版設計モデル 1/1

紙面の都合上，上記のモデルのすべてのダイアグラムを論文に掲載することを避け，モデル一覧の pdf ファイルと JUDE[3] により作成したモデルファイルを立命館大学情報理工学部ソフトウェア基礎技術研究室の Web サイトに配置した．それぞれのファイルは，
`http://www.fse.cs.ritsumei.ac.jp/mdd/appendix/`
からダウンロード可能である．

A.2　開発ソースコード一式

チャレンジで用いた最終版のソースコード一式も上記 Web サイトに配置した．最終版のソースコードでは，GUI による飛行船状況表示機能 (GTK++使用) が追加されている．また，コンパルソリー競技における動作を踏まえ，チャレンジ当日午後にソースコードの一部変更を行ったため，付録 A.1 の最終版設計モデルと一部違うところがある．ソフトウェアの利用方法に関しては，README を参照して頂きたい．