Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

オブジェクトの自律化と競合解決に基づく組込みオブジェクト指向開発手法の提案

岩橋　正実[†]　満田　成紀[††]　鰺坂　恒夫[††]　中島　毅[†††]

概要：機器組込みソフトウェア開発の生産性と品質を向上させるためのオブジェクト指向の開発方手法の提案とその有効性について述べる。組込みシステムにオブジェクト指向を適用する際には、要求からクラスの抽出方法、状態の抽出方法が開発者によりバラツキがある。要求から設計/実装の双方向のトレーサビリティの確保と動的分析、時間制約などのリアルタイム制御システム特有の問題を解決することで生産性と品質の確保を可能にした。本稿で提案する手法論は、1998 年に発表した自律オブジェクト指向技術を更に研究を進めたものである。その中で特に本稿では分析手法と設計手法を中心に述べる。

Proposal on an Object-oriented Development Technique for Embedded Systems using Autonomic Objects and Contention Resolution Methods.

Masami Iwahashi[†] Naruki Mistuda[††] Tsuneo Ajisaka[††] Nakajima Tsuyoshi[†††]

Abstract : This paper proposes and evaluates an object-oriented development method to increase the productivity for the embedded software development. Extraction of classes and their states from the specification is the key to developing a good design for embedded software. The method introduces the bidirectional traceability between specification and design and program components, and solves peculiar problems to real-time control systems, including time constrains and resource contentions. As a result, the method allows us to improve the productivity and quality of the overall system development. The AOO (Autonomic OO) technology presented by the first author in 1998 has been enhanced to acquire the method explained in this paper, in which the techniques for analysis and design are highlighted.

† 三菱電機メカトロニクスソフトウエア(株)和歌山支所/和歌山大学大学院システム工学研究科
†† 和歌山大学大学院システム工学研究科
††† 三菱電機株式会社　生産本部

1. はじめに

機器に組み込まれるソフトウェアは、年々開発量が増加すると共に高信頼性が要求されるようになってきている。生産性と品質を確保するためにはオブジェクト指向技術が有効な手段であるが、組込みソフトウェア開発の領域では、有効な実現手法が一般化されていない。オブジェクト指向技術の有効活用が進まない状況は2008年版組込みソフトウェア産業実態調査報告書のプログラミング言語の使用率から72.2%がC言語やアセンブリ言語の非オブジェクト指向言語の使用という実態からも読み取れる。[5)] これらの課題を解決するために本稿では、組込みソフトウェア開発に適用するオブジェクト指向の開発手法としてオブジェクトの自律化と競合解決の仕組みを提案する。

2. 提案する組込みソフトウェア開発手法

2.1 解決する課題

1) バラツキ

ソフトウェアアーキテクチャ設計のバラツキが課題である。要求からソフトウェアへの変換パターンが開発者により異なり同じ機能を実現するのにソフトウェア構造やインタフェースが異なってしまう。ソフトウェア構造やインタフェースが安定しないと開発のたびに異なる不具合を作り込んでしまう。不具合の再発防止を実施しても、また異なる不具合を作り込むような悪循環が続き品質に影響を与えるばかりではなくソフトウェアの共通化も進まないため生産性にも影響を与える。

2) 競合と時間制約

状態遷移、機能競合、出力競合、時間制約の分析設計の困難さが課題である。機能を組込みシステムに搭載する場合、状態遷移の分析設計が困難であり品質に影響を与えてしまう。また機能が複数ある場合は、機能の競合の課題が発生する。更に機能が並行して動作すると複数の機能による同一出力の競合の課題が発生する。これらを解決するた

Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

めには、機能実行の優先度や時間制約の分析設計が必要であるが、有効な手法が一般化されていないばかりではなく開発者ごとに実現手段にバラツキが発生して品質に影響を与える。

2.2 提案手法の構成

2.1 で示した課題を解決するために、我々は、自律オブジェクト指向手法(AOO: Autonomic Object-Oriented Techniques)を提案した[1)]。AOO は、大きく以下の２つの部分から構成されている。

1) 静的分析設計手法

自律オブジェクトを用いた**モデル化手法**、**カテゴリ化手法**、及び**クラス分析設計手法**を提供し、開発者によるソフトウェア構造及びインタフェースの設計におけるバラツキが生じる課題（2.1-1）を解決する。

2) 動的分析設計手法

自律オブジェクトを用いた**自律オブジェクトの状態分析設計手法**、**競合分析設計手法**、及び**絶対時間分析設計手法**を提供し、機能の状態遷移、機能の競合、出力の競合、時間制約の分析設計の困難さの課題（2.1-2）を解決する。

2.3 静的分析設計手法

1) モデル化手法

2.1 で示したバラツキは、ソフトウェア開発で最初に実施する要求分析定義から発生する。機能間の相互関連を持たせず１つの機能に１つの機能目的のみ持たせるという指針で機能項目のバラツキの抑制をはかる。この指針に基づいた要求分析定義のバラツキの抑制方法について述べる。組み込みシステムに搭載する要求を物理項目と機能項目に分離して相互干渉を受けないように要求を定義する。ゆで卵調理器の例で説明する。ゆで卵調理器は調理モードに応じて湯温を計測して火力を調整してゆで卵を茹でる調理器である。「標準ゆで卵制御」の仕様記述は他の機能項目に依存することなく自律した仕様を記述する。しかし複合した機能を１つの機能項目に記述してしまい要求分析定義にバラツキが発生する場合がある。例えば、「標準ゆで卵制御」の仕様記述の中に「温度センサが 150℃を超えると停止させる」という記述があったとする。この記述部分の目的は明らかにゆで卵を茹でる「標準ゆで卵制御」の目的と合致せず「火災異常制御」という別目的を記述していることが分かる。このように機能を混在して記述した場合、「火災異常制御」と他の機能項目との競合分析設計を実施するための項目の軸が無いため競合分析設計の品質に影響を与えてしまう。そのために１機能項目に１機能目的のみ定義して機能項目間の関連を考えず機能単体の要求を定義することで要求分析定義のバラツキを抑制した。

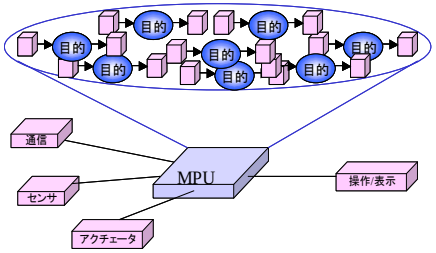

図１ 組込みソフトウェアと目的

次に要求をモデル化するときのバラツキの抑制方法について述べる。現実世界のモデルをソフトウェアのモデルに等価変換するモデル化のルールを確立させた。人が制御を実施するのには目的があり、目的達成のために入力デバイスの情報を参照して出力デバイスを操作して目的を達成する。この現実世界をモデル化すると物理的な物に等価なオブジェクトと人の意思決定部分と等価なオブジェクトでモデルが構成される。更に人の思考モデルでは、複数の目的を持つと、どの目的を優先して実施すべきかを思考している。この人の思考モデルもオブジェクトに変換することでモデル化のバラツキを抑制した。

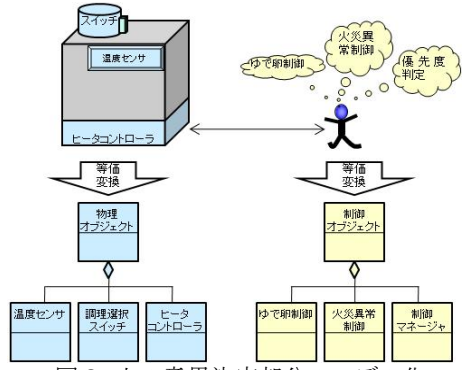

図２ 人の意思決定部分のモデル化

2) カテゴリ化手法

Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

2.3-1)で示したバラツキの抑制を更に進めるためにカテゴリによるバラツキの抑制を実施する。物理項目、機能項目を階層的にカテゴリで分類整理した物理項目リスト、機能項目リストを提供することでバラツキの抑制を可能にした。

まず物理項目のカテゴリ化について説明する。物理項目をカテゴリに整理すると物理的な物でカテゴリが階層化される。例えば「入力」のカテゴリの下位には、「センサ」があり更に下位には「温度センサ」がある。この分類は人によるバラツキは発生しにくい。

次に機能項目のカテゴリ化について説明する。機能項目を分類する場合にバラツキが発生しやすい。そのためにバラツキを抑制するカテゴリ階層のフレームを開発した(図3)。図2に示したゆで卵調理器の例で解説する。組込みシステムに搭載する機能項目は、機能目的を実現させるために外部デバイスの状態を参照して制御実行の条件を判定して外部デバイスを操作して制御している。このカテゴリを「制御判定」と定義した。更にカテゴリ「制御判定」を目的単位に整理する。組込みシステムが動作する場合に基準となる動作状態が必ず存在する。この基準となるトップレベルの目的を「運転」というカテゴリとした。この目的ベクトルは様々な外乱要因で目的達成のベクトルが傾く。この傾きを検知してトップレベルの目的を達成させるための支援機能が必要になる。このカテゴリを「支援」と定義した。目的ベクトルが更に大きく傾くと通常の制御では最終目的を達成することが困難になり異常を回避する制御が必要になる。このカテゴリを「回避」と定義した。そして異常回避を実施しても正常に動作し続けることが出来ない場合は、安全に停止させる必要がある。このカテゴリを「異常」と定義した。これにより機能目的を整理するカテゴリ階層のフレームを構築することができ開発者によるバラツキを仕様レベルから一掃することを可能にした。

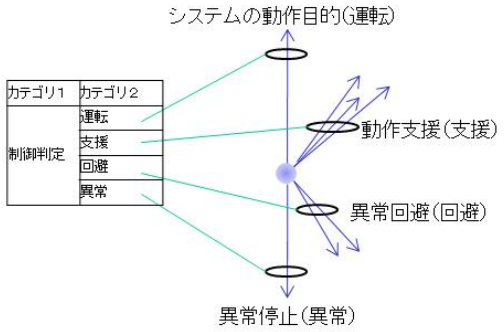

図3 カテゴリの階層化

上記の概念でゆで卵調理器の物理項目リストと機能項目リストを定義した例を図4に示す。この例を基にカテゴリ階層のフレームについて説明する。ユーザはゆで卵を茹でたいという目的があり、ゆで卵調理器の電源をONにしてゆで卵調理の種類(標準、半熟、硬い)を選びスタートスイッチを押すことで動作「運転」を開始する。「標準ゆで卵制御」で動作させた場合、ゆで卵調理器の動作状態は、標準ゆで卵制御状態となる。この「標準ゆで卵制御」がトップレベルの目的であるが様々な外乱要因で目的達成のベクトルが傾く。目的達成のためにゆで卵調理特有の課題が発生する。その課題解決のための制御「支援」が必要になる。例えば水温、周囲温度、水質等の違いにより湯温の上昇スピードが異なると美味しいゆで卵を茹でることができなくなる。この課題を解決するために湯温の上昇スピードをコントロールする機能「最適温度制御」が必要になる。また調理中に冷水を入れると急激に湯温が低下する。この急激な湯温変化に対応するための機能「湯温低下防止制御」が必要になる。更に目的ベクトルが大きく傾くと異常停止につながるのを回避するための制御「回避」が必要になる。例えば、湯温が120℃を超えてしまうとヒータを一時的に停止して異常停止を回避する機能「湯温上昇抑制制御」が必要になる。通常は、異常回避を実施することで正常運転の範囲に入り「支援」機能で最適制御を実現可能であるが、異常を回避しきれない場合は安全に停止させる機能「異常」が必要になる。例えば温度センサが150℃を超えると火事にならないように異常停止させる機能「火災異常制御」が必要になる。このように要求を分析するためのカテゴリ階層のフレームを開発することで要求分析定義のバラツキの抑制を可能にした。

3) クラス分析設計手法

最後に要求からソフトウェアの変換のバラツキの抑制方法について述べる。非オブジェクト指向言語での組込みソフトウェアの開発は、標準的なクラスの抽出の手法が一般

Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

化されていないため開発者によるクラス分析設計にバラツキが発生するばかりではなく、共通部分の抽出が困難でクラス間の依存関係も複雑に構築してしまう。本稿で提唱するクラス分析設計手法は、要求分析定義で定義した**機能項目リスト及び物理項目リストをクラスに１対１に変換**する。カテゴリ名称及び機能項目名称をクラス名称と合致させることで双方向の一貫性の確保を可能にした。上位のカテゴリをスーパークラスに下位のカテゴリをサブクラスに変換する。カテゴリ内のオブジェクトは同種の目的で分類されるため共通な機能の抽出の促進が可能になりオブジェクト構造のパターン化が進み可読性の向上に繋がった。この技術は、ソフトウェアの自動生成、自動検証、ソフトウェアプロダクトライン実現のための重要な技術要素になる。

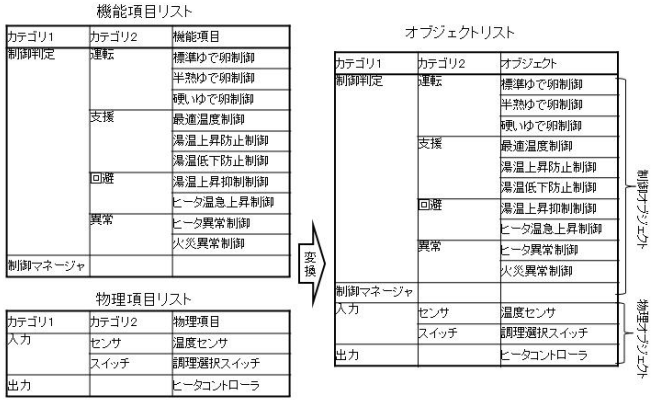

図4 オブジェクトへの変換

2.4 動的分析設計手法

1) 自律オブジェクトの状態分析設計手法

2.1で示した競合と時間制約の分析設計の難しさの課題を解決する。オブジェクトの相互関連を持たせず１つのオブジェクトには、１つの目的のみ持たせる。更にオブジェクトを自律化するという指針で競合と時間制約の分析設計の難しさの課題を解決する。まずオブジェクトの自律化について述べる。機能項目リストと物理項目リストから変換したオブジェクトには、カテゴリ内のオブジェクト間の関連を持たさずオブジェクトを自律させ自分自身の目的のみ実行するアーキテクチャを採用した。機能項目に定義した機能仕様から出力操作に関連する部分を、出力オブジェクトに抽出すると動的な振る舞いの部分である状態遷移部分が残る。この部分を状態遷移マシンとして制御判定クラスを継承するオブジェクト（以下、制御判定オブジェクト）に分離した。この制御判定オブジェクトの属性に機能目的達成の状態を定義する。制御判定オブジェクトの状態遷移は他のオブジェクトに依存しないで自律した状態遷移を定義する。自分自身の目的達成のための状態遷移のみ定義するだけで良い。このように自律したオブジェクトの状態遷移を厳密に定義することでオブジェクト単体の品質確保を可能にした。

次に状態分析設計手法について述べる。制御判定オブジェクトは、入力デバイスを参照して制御開始条件を判定して条件が成立すると出力デバイスの操作を実施して目的を達成する。この開始条件の成立から目的達成までの間を最上位の状態として定義する。機能実行に手順があれば、状態を階層的に定義する。例えば「標準ゆで卵制御」の最上位の状態は「標準ゆで卵制御状態」となる。「標準ゆで卵制御」は、最初に湯温を最適な調理温度まで到達させるための「標準ゆで卵制御初期調理状態」、次に余熱で調理する「標準ゆで卵制御余熱調理状態」が階層的に定義される。更に状態名称は機能項目名称に合致させることで分析、設計、実装の双方向の一貫性を確保させた。機能項目ごとに機能項目実行中の状態が定義される。そのため機能項目の競合の分析は、状態の組み合わせの分析を実施すれば論理的に問題ないことの立証が可能になる。

2) 競合分析設計手法

前節ではオブジェクト単体の分析設計の課題の解決策を示した。本節ではオブジェクトの競合の課題解決の方法について述べる。機能項目リストには、システム全体の動的な振る舞いが含まれている。この機能項目リストに定義した機能項目は、オブジェクトリストの制御判定オブジェクトに１対１に変換される。この制御判定オブジェクトの競合の問題を解決する。競合の解決は、カテゴリ内のオブジェクトの優先度を定義して競合の分析設計を実施する。次にカテゴリ間の競合をマトリクスで定義することで競合問

Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

題を解決する競合分析設計手法を開発した。

最初にカテゴリ内優先度の設定による競合分析設計手法について述べる。オブジェクトリストのカテゴリ内のオブジェクトに対して優先度を設定する。優先度とは、オブジェクトが同時にアクティブになった時にどちらを優先するかを指定するものである。ゆで卵調理器の事例のカテゴリ「支援」を例に競合分析設計手法を説明する。カテゴリ「支援」は、「最適温度制御」「湯温上昇抑制制御」「湯温低下防止制御」の 3 つのオブジェクトがあり優先度は表 1 に示す通りで、数値が小さい方が優先度を高く設定している。複数のオブジェクトがアクティブになると優先度に基づきオブジェクトに対して許可/禁止を設定することでカテゴリ内の競合の問題を解決した。

カテゴリ1	カテゴリ2	優先度	オブジェクト	優先度
制御判定	運転	2	標準ゆで卵制御	1
			半熟ゆで卵制御	2
			硬いゆで卵制御	3
	支援	4	最適温度制御	3
			湯温上昇抑制制御	1
			湯温低下防止制御	2
	回避	3	湯温上昇抑制制御	2
			ヒータ温急上昇制御	1
	異常	1	ヒータ異常制御	2
			火災異常制御	1

表1 カテゴリ内優先度

次にカテゴリ間の競合分析の手法について述べる。カテゴリ間の競合分析を実施することで組み合わせ数を削減した。表 1 に示した項目数は 10 個であり全ての組み合せでは 100 通りの組み合わせになる。これをカテゴリ単位の組み合わせで競合問題を解決すると表 2 に示すようにカテゴリ数が 4 であるため組み合わせ数は 16 になる。カテゴリの競合分析設計の方法は、行方向に現在アクティブになっているカテゴリを設定して列方向に新たにアクティブになるカテゴリを設定して交点にカテゴリに対する許可/禁止を定義する。カテゴリ間の競合分析設計で全ての機能項目の競合の解決が可能になる理由をゆで卵調理器の例で説明する。分析対象のカテゴリは、「運転」「支援」「回避」「異常」がある。「運転」中に他のカテゴリ「支援」「回避」「異常」は全て許可になるが、「回避」中の「運転」「異常」は許可であるが「支援」は禁止である。禁止にする理由は、異常回避「回避」を実行している時に制御目的の達成を支援する動作「支援」を実施すると異常回避できなくなるために禁止にする。禁止にするカテゴリ「支援」に定義した機能項目は、全て制御目的の達成を支援する制御動作するためカテゴリ内の機能項目が全て同じ禁止に設定される。カテゴリ内のオブジェクトは同種の目的で分類しているためオブジェクトの動作も類型化するためカテゴリ単位の競合分析が可能になる。

現状態＼新状態	運転	支援	回避	異常
運転		許可	許可	許可
支援	許可		許可	許可
回避	許可	禁止		許可
異常	禁止	禁止	禁止	

表2 カテゴリ間の競合分析

最後に出力の競合の課題の解決方法について述べる。機能の競合分析手法で複数の制御判定オブジェクトに許可を設定すると許可に設定した制御判定オブジェクトが並行して動作する。出力オブジェクトが参照する制御判定オブジェクトが複数あり且つ制御判定オブジェクトが並行動作する場合は、複数の制御判定オブジェクトが 1 つの出力オブジェクトの出力操作が競合していると判断できる。この競合問題はカテゴリ間優先度、カテゴリ内優先度に基づき出力操作の優先度を決定する。ゆで卵調理器の例で出力競合の解決方法について説明する。「運転」の「標準ゆで卵制御」、「支援」の「最適温度制御」、「回避」の「湯温上昇制御」と「ヒータ温上昇制御」が、「出力」の「ヒータコントローラ」を競合する場合、カテゴリ間で最も優先度の高いカテゴリ「回避」が選択され、カテゴリ「回避」の中のオブジェクトで最も優先度の高い「ヒータ温上昇制御」の出力操作が最優先で選択される。このようにして機能の競合問題、出力の競合問題の解決を可能にした。

3) 絶対時間分析設計手法

最後に時間制約の課題を解決する方法について述べる。要求には全て時間制約があり、時間制約の分析設計を実施せずソフトウェアを作り込むと、発生確率が極めて低いが発生するとシステムダウンなど致命的な問題を引き起こす要因となる可能性がある。これらの課題を解決するための有効な実現手段が一般化されていない。本稿で提唱する絶対時間分析設計手法は、時間制約のオブジェクトへの変換ルールと時間制約管理部分のカテゴリへの分離で時間制約の課題を解決する。

機能項目、物理項目には絶対時間が存在する。機能項目は、制御対象物の特性や環境

Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

特性等から機能を実現するために絶対守らなくてはならない時間制約が特定される。物理項目は、物理的なハードウェアの物理特性から時間制約が特定される。これらの時間制約を機能項目リスト、物理項目リストに定義する。このリストをオブジェクトリストに変換することによりオブジェクト単体の時間制約に変換する。各オブジェクトの時間制約を解決するためのカテゴリをオブジェクトマネージャとして分離する。オブジェクトマネージャは、オブジェクトの時間制約に対応してオブジェクトに起動周期を与えることで時間制約の問題を解決する。本稿では実現手段まで紙面の都合で述べる事ができないため別途論文にまとめる。

3. フレームワーク化

3.1 ＡＯＯフレームワーク

本稿で述べた静的分析設計手法、及び動的分析設計手法をフレームワークに組み込み開発の安定化に成功した。図 5 がドメイン依存性を可能な限り削減した AOO フレームワークで、現実世界をソフトウェアに等価変換する基本的な変換ルールにより人によるアーキテクチャのバラツキをなくし、ソフトウェアの開発ラインの統合化を可能にした。

3.2 高信頼性確保の仕組み

全ての要求事項は、機能項目と物理項目に定義される。機能項目から抽出した状態遷移部分は、フレームワークの「Control Judge」に実装される。物理項目は、フレームワークの「Input」「Output」に変換される。制御競合の問題は「Control Manager」に実装される。機能項目、物理項目に定義した時間制約は「Object Manager」がオブジェクトに対して起動周期を与える。ハードウェアとの界面は「Kernel」と呼ばれるカテゴリ内にハードウェアの特性を吸収するオブジェクトを設けることでハードウェアの変更をカーネルオブジェクトで吸収する。それぞれのカテゴリ内のオブジェクトの相互関係を持たさない事で組み合わせ問題の削減を実現している。

カテゴリ名称		説明	変換ポイント
入力	センサ	カーネルオブジェクトからの情報を取得して制御に必要なデータに変換して公開する。	物理的な物（最終制御対象物、又はソフトウェア内部の不変の情報）
	スイッチ		
出力		制御マネージャが公開する制御状態に応じてカーネルオブジェクトに対し出力操作を実施。複数の制御状態に応じて出力操作を実施する場合は、制御状態のマトリクス（現状の制御状態と新たに発生した制御状態のマトリクス）で分析をして出力操作を実施する。	
制御判定	運転	入力オブジェクトの入力値を取得して、制御実行条件を判断し、制御状態を生成して公開する。	人の意思決定部（機能目的）
	実験		
	回避		
	異常		
制御マネージャ		制御判定オブジェクトの優先度を判断して制御判定オブジェクトに許可/禁止を伝える。競合は全ての制御状態に対して現状の状態と次に発生した状態とのマトリクスで競合を制御する。	人の意思決定部（機能目的）の優先度判定）
カーネル		ハードウェアとの入出力を実施する。	物理的な物（ハードウェアデバイスとのインタフェース）
オブジェクトマネージャ		オブジェクトが保有する時間的な制約を満足できるように、オブジェクトに対して起動イベント（起動順序と起動周期）をオブジェクトマネージャが与えオブジェクトの起動とオブジェクト間のインタフェースを確定する。	時間制約部分（オブジェクトの起動イベントとインタフェース）

表3 カテゴリの役割と変換ポイント

A00 フレームワークで高信頼性を確保する仕組みについて述べる。カーネルオブジェクトは、ハードウェアのデバイス毎に搭載される。例えばアナログ入力オブジェクトでは A/D 変換のノイズ処理を実施して A/D 変換データを生成して公開する。ポート入力オブジェクトでは、ポートの入力のチャタリング処理を実施して安定したポートデータを公開している。これらのアナログ入力オブジェクトやポート入力オブジェクト間は、関連を持たしていないため自律したオブジェクト単体の品質を確保することでカテゴリ内のオブジェクトの品質確保が可能になる。

入力オブジェクトは、カーネルオブジェクトの情報を参照して自分自身の達成のための振る舞いを実施する。例えば、温度センサはアナログ入力オブジェクトの値を参照して温度データに変換して公開する。入力のカテゴリもカーネルと同様にカテゴリ内のオブジェクト間の関連を持たさない自律したオブジェクト単体の品質を確保することでカテゴリ内のオブジェクトの品質確保が可能になる。

制御判定オブジェクトは入力オブジェクトの情報を参照して制御開始条件を判断して条件成立で状態を生成して公開する。制御判定オブジェクト間

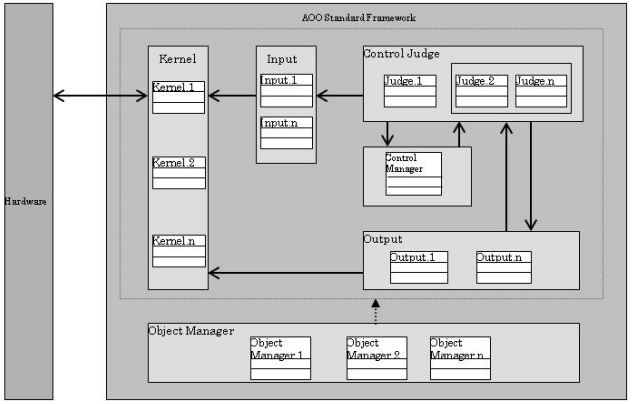

図5 AOOフレームワーク

Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

の関連を持たさないため１つの機能目的を保有する制御判定オブジェクト単体で問題ないことを立証することができる。次いで複数の制御判定オブジェクトの優先度を判定する制御マネージャの役割は制御判定オブジェクトの状態の競合を判定して許可/禁止を制御判定オブジェクトに伝える役割のみ保有する。そのため制御マネージャ単体の品質確保も可能になる。

次に出力オブジェクトは､制御状態に応じて出力操作を実施するオブジェクトである。出力オブジェクトは、自分自身の出力操作に必要な制御状態を知っていて制御状態に基づき出力操作をカーネルオブジェクトの出力ドライバのオブジェクトに要求する。出力オブジェクトでは､同時に許可になる制御状態の優先度を整理して出力操作を実施する。出力オブジェクトに関してもカテゴリ内のオブジェクト間の関連を持たさない自律したオブジェクト単体の品質を確保することでカテゴリ内のオブジェクトの品質が確保可能になる。

最後に機能項目、物理項目で定義した時間制約を解決するためにオブジェクトマネージャがオブジェクトに起動周期を与える事で静的にも動的にも問題ないことを立証することができる。時間制約が共通のオブジェクトは同じタスクに実装される。時間制約が異なれば異なるタスクに実装される。時間制約が定義されているために時間制約内にタスクが実行されれば時間制約に問題ないことを立証することを可能にした。

4. 成果分析

AOO 手法論で要求からソフトウェアへの変換が論理的に問題なく実施できるようルールを設け開発作業のパターン化を実現した。これにより高信頼性のある組込みソフトウェアの開発が継続的に実施できるようになった。その結果を以下に纏める。

4.1 コード品質分析

量産製品の制御機器の組込みソフトウェアを対象とした。対象とした製品のプログラムコードのサイズはC 言語で約 200Kline で改善前と改善後のプログラムコードがどのように改善したかを分析する。

1) コードクローン分析

同種の目的のオブジェクトをカテゴリで整理するとカテゴリ内の複数のファイルに同じコードのパターンが増加することに着眼した。非オブジェクト指向言語で開発する場合はクラスからインスタンス生成を実施できないため同じカテゴリ内のオブジェクトのソースコードの類型化が進むとコード品質レベルが高いと判断できる。またカテゴリ内のオブジェクトの共通部分は上位のクラスに抽出が進む事になるためオブジェクト（ファイル）内のソースコードの共通パターンが減少することになる。これを検証するためにコードクローン検出ツール（CCFinderX）を使用してコード品質を計測した[4)]。対象のファイルが、そのファイル以外との間のコードクローンによって占められている割合の RSA (ratio of similarity between another files)は、改善前が 0.053 に対して改善後は 0.145 で約 272%と増加している。この結果からファイル間でソースコードのパターン化が進んだと判断できる。

対象範囲	Name	改善前	改善後	割合
全体	RSA	0.053	0.145	272
	RSI	0.432	0.265	61
カテゴリ（制御判定）	RSA	0.165	0.557	338
	RSI	0.368	0.024	7
カテゴリ（出力）	RSA	0.024	0.176	737
	RSI	0.550	0.213	39

表4 コードクローン分析

ファイルのテキストが、そのファイル内のコードクローンによって占められている割合の RSI (ratio of similarity within the file)は改善前が 0.432 に対して改善後は 0.265 で約 61%と減少している。この結果からファイル内の共通ソースコードは共通化されスパークラスへの抽出が進んでいると判断できる。

更に同種のカテゴリ（制御判定、出力）でコードクローンの分析を実施した。その結果 RSA は、更に増加、RSI は、更に減少している。計測結果から同種の目的で構成されるカテゴリ内のファイル間に共通のパターンが進んだ。更に、同種の目的で分類整理した結果、同じファイル内の共通部分の抽出が進んだと判断できる。

Vol.2009-SE-164 No.11
Vol.2009-EMB-13 No.11
Vol.2009-CSEC-45 No.11
2009/5/28

2) サイクロマティック複雑度

目的単位に分類するとカテゴリ内のソースコードが類型化されソースコードの経路複雑度も低下することになる。QAC を用いて改善前と改善後のサイクロマティック複雑度を計測した[3]。改善前のサイクロマティック複雑度の平均値が 9.48 に対して改善後は 4.36 で約 46%に減少している。この結果から複雑度の低減が進んだことが判断できる。

4.2 品質レベル分析

ソフトウェア開発部門から依頼部門にリリースするソフトウェアの品質レベルを分析する。AOO 手法論を全面適用できない過去の資産をベースに開発しているプロジェクトが半数以上あるが AOO 手法論を適用することで品質改善が進んだ。その結果を分析する。

1) 不具合残存率

AOO 手法論を適用するとソフトウェア開発部門から依頼元にリリースした後に検出されたプログラムの不具合の残存率は 1/8 となり改善が進み、その後も品質が安定していることが分かった。また AOO 手法論の完全適用で流出した不具合は 0 件という結果となった。

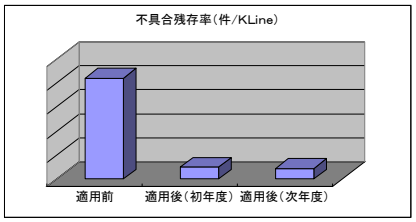

図 7 不具合残存率計測

2) 上流検出率

AOO 手法論を適用すると分析、設計、実装の双方向の一貫性が確保され更に論理的に問題ないことを立証させるためにソフトウェアの不具合は、分析、設計、実装の検出割合が高くなると想定される。不具合検出工程を分析するとソフトウェア開発の全工程で検出される不具合の 90%以上を上流（分析、設計、実装）工程で検出できた。分析設計で論理的に問題ないことを立証する仕組みが機能していると思われる。しかし、開発段階での工程残存不具合に関しては、要求分析定義で約 23%残存。ソフトウェア設計で 15%残存、ソフトウェア実装で 5%残存している。上流での残存率を下げることが今後の課題となる。

5. まとめ

要求から設計、設計から実装の変換のルール化を実施できた。この変換ルールで状態遷移部分を制御判定オブジェクトに分離、競合問題は制御マネージャオブジェクトに分離、時間制約はオブジェクトマネージャに分離することで高信頼性を確保する手法を確立した。更に AOO フレームワークを用いると開発者が考えないと実装が出来ないため分析、設計の作業の定着化に成功した。ソフトウェアがカテゴリで整理され可視化が進み、ソフトウェアの開発ラインの作業標準も構築することができた。しかし定型化したとしてもヒューマンエラーは取り切れないため残存不具合を 0 にすることは極めて困難な状況である。分析、設計、実装のルール化を更に進めて定型作業を確立させることでツールによるソフトウェア設計の検証、コード生成が可能になると考える。今後は、分析、設計、実装の双方向のトレーサビリティを確保してヒューマンエラーを一掃して生産性を改善するための研究を進める。

参考文献

1) 岩橋 正実, TECHI Vol.12 リアルタイムシステム実現のための自律オブジェクト指向, CQ 出版, 2002.4
2) ジェームズ・ランボー,イヴァー・ヤコブソン,グラディ・ブーチ, UML リファレンスマニュアル, ピアソン・エデュケーション, 2002.1
3) http://www.toyo.co.jp/ss/qacpp/product_summary4.html
4) http://www.ccfinder.net/ccfinderx-j.html
5) METI, 2008 年版組込みソフトウェア産業実態調査報告書