社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ＳＥ－156 (11)
2007－EMB－ 5 (11)
2007／5／29

組込みソフトウェアの再利用を支援するモジュール間相互関連表示法

神戸 英利† 永松 博子†† 三井 浩康†† 小泉 寿男††
†三菱電機株式会社 ††東京電機大学理工学部

現在，組み込みソフトウェアの需要は増大し，開発ステップ数も増大化の傾向にあり、かつ開発期間は短期化の傾向にあるため再利用が必要不可欠となっている．しかしながら、再利用対象の既存ソフトウェアの構成管理が不十分であると，再利用部品適用の際に他のどのモジュールに影響を及ぼすかがわからなくなってしまい，新たなバグを増やす要因となってしまう。本稿では、既存ソフトウェアのモジュール間関係を可視化できる構成管理ツールを開発し、効果的な再利用を実現する手法を提案する．本稿ではさらに、本手法を用いた開発例に関する評価について述べる。

A Visualizing Method of Relations between Modules to Support Re-use of Embedded Softwares

Hidetoshi Kambe† Hiroko Nagamatsu†† Hiroyasu Mitsui†† Hisao Koizumi††
†Mitsubishi Electric corporation ††Tokyo Denki University

Currently, demand for software increases, and the number of development steps tends to increase with shorter development term, then the software re-use is becoming essential. However, when the configuration management of existing software for re-use is insufficient, it becomes a factor to increase new bugs because the influence of applying re-use parts on other modules becomes unpredictable. In this paper, we develop a configuration management tool which visualize relations between modules of existing software, and propose an effective re-use method. Furthermore, in this paper we describe the evaluation of an example of a development using the method.

1. はじめに

昨今の携帯電話、メモリ音楽プレーヤ、デジタルカメラ、プリンタ、ＶＴＲ、ＨＤＤレコーダなど、いわゆるデジタル家電の制御には機器搭載型の制御ソフトウェアが搭載され、高機能化、高性能化が非常に速い速度で進んでいる。これらの機器に搭載されるソフトウェアは、当初はハードウェアの制御のための簡単なマイコンファームウェアであったが、現在では専用ＯＳやＬｉｎｕｘ等が採用され、高度な画像処理や通信制御等を搭載した一昔前のパソコンを上回る機能が実現されるようになった[1][2][3][4][10][11][12]。

これら機器搭載型ソフトウェアの開発においては、開発規模の拡大、商品投入サイクルの短期間化に伴う開発期間の短縮、低コスト化、高品質化が求められている。組み込みソフトウェアについては、プラットフォームが独自であることが多く、開発支援環境の整備が追いつかないケースが多い。そのため、ソフトウェア開発人員の対象プラットフォームに対する教育が行き届かなくなり、生産効率の悪化や品質のバラツキが発生するとともに、不具合の収束にも時間がかかる。開発者間の情報伝達漏れ等による仕様抜けや実装漏れ、それを防止するためのレビュー等のチェック作業が煩雑かつ大きな工数を占める。

現在，ソフトウェアの需要は増大し、開発ステップ数も増大化の傾向にあり、かつ開発期間は短期化の傾向にあるため再利用が必要不可欠となっている[5][6][7][8][9]。しかしながら、再利用対象の既存ソフトウェアの構成管理が不十分であると、再利用部品適用の際に他のどのモジュールに影響を及ぼすかがわからなくなってしまい，新たなバグを増やす要因となってしまう。

本稿では、既存ソフトウェアのモジュール間関係を可視化できる構成管理ツールを開発し、効果的な再利用を実現する手法を提案する．本提案では、開発する際、最小単位となるソフトウェアモジュールのソースファイル単位で、それぞれのモジュール間すべての依存関係、ソフトウェア規模等を抽出する。それに開発者、開発期間等の情報を必要に応じて追加して、管理情報のデータベースを作り出す。これらを人間が管理しやすい機能レベルへ階層的にまとめて整理し、ブロック図レベルに可視化する。新規追加機能や既存部分への仕様変更を発生させたときに、依存関係のあるモジュールが全て抽出できるかどうか、開発者がその情報を有効に利用できるかどうかを評価する。

入力として利用する情報は、ソフトウェアのソースコードやオブジェクト、その他コンパイルに使われる管理情報、バージョン管理情報等、既存の開発環境内に存在する情報を最大限使い、特別な労力やコストの追加を出来る限り抑えることを重視した。本稿ではさらに、本手法を用いた評開発例に関する評価について述べる。

本論文では、2章で現状のソフトウェア開発プロジェクトの課題と対応策を整理し、3章で本提案の支援方式について述べる。4章で実装、5章で評価を行う。

2. 組み込みソフトウェア再利用開発の現状問題点と対応策

2.1. 現状の問題点

携帯電話やカーナビ等の高機能化が顕著に進む機器における搭載ソフトウェアの開発においては、ソフトウェアをゼロから開発するということはほとんどない。既存のプラットフォームを導入するか、すでに導入済みの場合は使用中のプラットフォーム資産の再利用を行うことで、開発コストの削減や開発期間の短縮、品質向上を目指している。ソフトウェア資産には、仕様を記述したドキュメント類、ソースコードやオブジェクトなどのプログラム資産、評価方法やテストデータ等がある。しかし、再利用による開発に当たっては以下のような問題を抱えている。

2.1.1. 開発者による全体把握の限界

(1) ソフトウェアの大規模化に伴い、多くの機能単位に分割して開発評価を実施する必要がある。分割した機能が何百，何千にもおよぶと、各ソフトウェアの版管理と分割された機能全体を開発者が把握することが出来ない。そのため、開発が必要なソフトウェアモジュールの洗い出しに、手間と時間がかかる。

(2) 再利用対象部分を担当した開発者が当該開発時にはすでに不在であることが多い。そのため、ソフトウェア構成やモジュール間相互関係でドキュメントだけでは把握が困難であった場合、担当者からの聴取で補うことは難しい。

(3) ソフトウェアモジュールの構成管理では、ソースファイルの版管理は可能であるが、ソフトウェアモジュール間の依存関係や、仕様変更、修正変更後の確認にかかわる作業には利用出来ない。

2.1.2. 関連するソフトウェアの把握難易化

開発者は、規模が大きいソフトウェア内での開発では、自分の担当以外のソフトウェアモジュールの情報は多くを持ち合わせられない。担当のソフトウェアモジュールを中心に見たときに、周辺に存在するソフトウェアモジュール数が膨大となるため、関連を持つソフトウェアモジュールの洗い出しが困難となる。

2.2 対応策

既存のソフトウェア資産を流用しつつ開発を効率よく実施するためには、以下に示すような対応策が必要と言える。

(1) 開発者に対する情報把握の支援

① 流用するソースファイルを含む開発環境内から、開発に必要なソースファイル等の情報を自動的に抽出整理して、開発者に提示し事前調査時間を短縮する。

② ソフトウェアの構成をブロック図の形式で開発者にビジブル化して提示する。

(2) ソフトウェア関連性情報の把握支援。

① 開発/修正対象のソフトウェアモジュール間の連携関係をブロック図上で確認可能とすることで開発者の作業を援助する。作業者情報を同時に提供することで複数の開発者で作業する場合、相互の情報確認の支援をする。

② ブロック図からソースファイルへリンクさせるで、プログラミング作業の迅速化と、必要な情報の抜けや漏れを防止する。

③ ソフトウェアモジュールを新規開発する際に，ブロック図レベルで機能追加し、関連するソフトウェアモジュールとの関係の定義もブロック図レベルで同時に可能とすることで作業の効率化を図る。

④ 新規追加ブロックに対して、自動的にソースコードファイルのテンプレートを生成してソフトウェア環境内への組み込みを簡略化する。

3. 再利用を支援するモジュール間依存表示法

2章で述べた対応策を具体的に実現する方法として、本論文では，再利用を支援するモジュール間依存表示法の提案を行う。いまプロダクトAのアプリケーションソフトウェアを再利用して、新たにプロダクトBを開発するケースを図1に示す。

(1) プロダクトB仕様と再利用ソフトウェアの準備

・ 今回開発するプロダクトBの仕様を用意する。

・ 再利用するプロダクトAのソフトウェアを一式準備する。この中には、ソースファイル、オブジェクトファイル（バイナリ形式で提供されるソフトウェア)、プロジェクトファイル（ソースファイルやオブジェクトファイルの階層構成を示すデータでソフトウェアビルド時に利用される管理情報)、コンパイラ等の開発環境全てを一式用意してコピーし（図1の①)、プロダクトBソフトウェア開発環境とする。

(2) 関連解析

・プロダクトBソフトウェア開発環境内のソースファイル、オブジェクトファイル、プロジェクトファイルを読み込んで関連解析を実行する。ファイル間のシンボルレベルの関連情報を解析し、プロジェクトファイルからブロック図レベルのソフトウェア構成データを生成する。

・生成した関連情報ならびに、ソフトウェア構成のデータを関連解析データとして格納する。(図1②)

・追加修正される前の最初に生成される関連解析データは、プロダクトAと同一である。

(3) 調査

・関連解析データからソフトウェア構成をブロック図表示して、プロダクトBの仕様を実現するために新規追加が必要なソフトウェアモジュール、修正して再利用するソフトウェアモジュール、流用するソフトウェアモジュールの調査を行う。(図1の③)

(4) 関連情報の表示

・前記の調査で探し出した、修正して再利用するソフトウェアモジュールに関して、修正した際に影響が及ぶ周辺のソフトウェアモジュールを調査する。(図1の④)

・修正して再利用するソフトウェアモジュールを中心にして、関連を持つソフトウェアモジュールを全てブロック図レベルで関連情報表示する。

・ソフトウェアブロック図上でシンボルレベルの入出力関係の表示、修正前後の比較表示をさせて、関連を持つソフトウェアモジュール全てにつき修正の要否を確認し、開発/修正の必要なソースファイルの範囲を抽出する。

(5) 設計製作

・設計製作ではソフトウェアモジュールの新規追加と、既存ソフトウェアモジュールの修正の2つの作業がある。

・修正して再利用するソフトウェアモジュールは、関連情報を確認(図1の⑤)しながらソースファイルを修正し(図1の⑥)、所望の仕様のソフトウェアに変更する。

・新規追加では、プロダクトBで新規追加するソフトウェアモジュールをブロック図レベルで追加し、機能に基づいて周辺のソフトウェアモジュールと関連付けを行う。(図1の⑤)

・新規追加分のソースファイルをスケルトンソースコード追加機能を利用して作成する。(図1の⑥)

(6) ビルド

・コンパイル、リンクを実行して、ソフトウェアをビルドし、プロダクトBのソフトウェアの実行コードを得る。エラーが残る場合は、ビルド時のコンパイル、リンクエラーの情報と合わせ、再度②から⑥の手順を繰り返し、エラーを取り除く。エラーがなくなった時点でプロダクトBのソフトウェアの製作が完了する。

4. 本方式の構成と実装

4.1. 構成管理システムの構成

今回設計した構成管理システムの構成を図2に示す。構成管理システムは、関連解析部と関連表示部の2つに分かれる。再利用するソフトウェア開発環境は、前章3の(1)でプロダクトAをコピーして作成したプロダクトB開発環境である。関連解析部は3章の(2)を実現し、関連解析データを生成する。関連表示部は、3章の(3)から(6)を実現する。

4.2. 関連解析部

(1) 関連解析部は、パーサと、RCNAから構成される。パーサはソースファイルからクロスリファレンス情報を生成する。今回 Source-NavigatorTM を利用した。

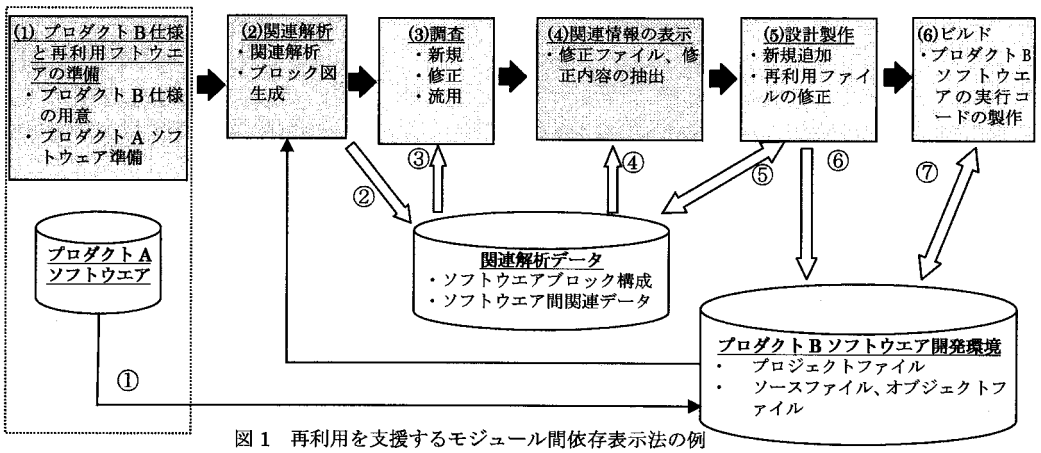

図1 再利用を支援するモジュール間依存表示法の例

Source-NavigatorTM は各ソースのシンボルをカテゴライズしてそれぞれデータベース化し、独自の解析エンジンでデータベース間にリレーションを持たせることでクロスリファレンスを作成する。

(2) RCNA はプロジェクトファイルからソフトウェアブロック構成を生成する。またパーサより取得したクロスリファレンス情報と併せて各ソフトウェアモジュール間の関係を解析するJavaとC言語で作成されるアプリケーションである。

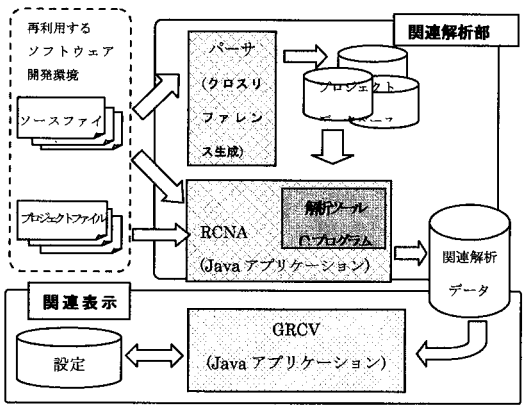

図2　構成管理システム構成図

4.3.　関連表示部

関連表示分は、設定情報を保存するデータベースと、関連表示部（GRCV）から構成される。

(1) 関連表示部は，関連解析部（RCNA）で分析したソフトウェアモジュールを設定ファイルの指定に従ってソフトウェアブロック図の形式でグラフィカルに表現するJavaアプリケーションである。また、各ソフトウェアブロックダイアグラム間の関連性を図的に表現する。主な機能はソフトウェアブロック表示機能，関連表示機能，開発規模見積機能などがある。

(2) 今回Ｗｅｂアプリケーションの開発で利用される、ソフトウェアの設計モデルのひとつ　ＭＶＣモデルの考え方を踏襲した実装を検討とした。

① 関連表示部のＭＶＣモデル適用検討

MVC モデルは一般的にウェブアプリケーションに適用される。関連表示部は通常のアプリケーションの範疇であり、MVC モデルの形式にそのまま適合しない。そこで拡張 MVC モデルをケ検討し関連表示部への実装を試みた。以下図３に関連表示部拡張ＭＶＣの実装構成を示す。各構成要素の内容を以下に示す。

・Model

主にデータベースから関連解析データの取得、外部ファイルアクセス、及び、各種インスタンスの生成を実行する。基本的には Controller からのイベントを受理して動作する。ストレスの発生するユーザ操作の一部イベント関係には View から直接呼び出す形式とする。

・View

画面構成、表示を実行する。ユーザ操作の各種イベントを受理し、Controller 経由または一部直接 Model に伝達する。

・Controller

Model、View を制御する。View から上がってきたイベントを必要に応じて、Controller に渡す。

・Resource

文字ラベル、パラメータ、各種プロパティを管理する。

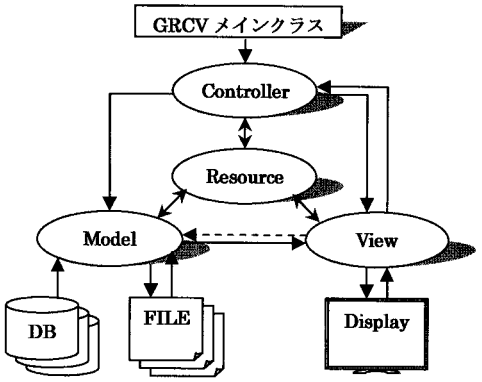

図3　関連表示部拡張ＭＶＣ構成

(3) 関連表示部で実装する関連表示機能につき以下表１に示す。

表1. 関連表示部実装機能

機能	内容
ソフトウェア構成ブロック表示機能	関連解析部での解析結果をソフトウエアブロック図で表示する機能
関連表示機能	分割単位（ファイル、モジュール等）毎の入出力関係や関連性の有無ををを矢印や色で表示する機能
関連データ比較機能	画面上で選択したブロックの関連解析データと、例えば前バージョンのソフトウエアにおける同一ブロックの関連解析データの比較をして、ブロックの追加削除についての差分情報を表示する機能。
新規ブロック追加・削除機能	新規のソフトウエア（機能）追加や削除を関連表示部内からトップダウンで開発できるようにする。
関連追加・削除機能	ブロック図レベルで関連情報の追加削除を編集できるようにする。
ソースコードビューア	指定したブロックのソースコードファイルを開く。
スケルトン生成機能	新規ブロックを追加した際に、ソースコードファイルの雛形を生成する。

5. 評価

４章で設計した内容を評価するため、関連解析部と関連表示部を実装した。以下に図１の流れに沿った評価結果を述べる。

(1) プロダクトＢ仕様と再利用ソフトウェアの準備

評価用に再利用するプロダクトＡとして、ソースファイル数、約6,900ファイル、オブジェクトファイル数、約2,000ファイル、プロジェクトファイル数520ファイルから構成されているソフトウェアを用意した。この中のアプリケーションの一部であるEcrioPocLibSignaling というライブラリを１つ修正し、NewSoftというアプリケーションを新規追加して、プロダクトＢを開発する例を評価対象とした。

(2) 関連解析

最初に準備した修正前のプロダクトＢ（プロダクトＡと同一）のソフトウェに関連解析を実施した。関連解析では、解析がきちんとできることと、解析時間の性能が実用上の評価ポイントとなる。

修正前のプロダクトＢ(内容はプロダクトＡと同一)の関連解析を実施して以下の結果を得た。

・ 解析は約６０分（クロスリファレンス生成：３２分、関連データ生成：２８分）で終了した。

・ 解析したシンボル数は全部で 677,405 個、ブロック図は 9,800 個（データの大きさ約 1.7MB)、関連解析データの大きさは約 140MB となった。

・ この規模のデータ量のソフトウェアであれば、約１時間程度で解析が終わり、その後ブロック図で表示をすることが可能になることから解析時間は実用範囲内にあることがわかった。

・ 解析したシンボル数と生成されたソフトウェアブロックの数は、設計者が手作業で個別に内容を見てチェックするには量が多すぎることがわかり、関連表示部の必要性が確認された。

(3) 調査

最初に開発者が理解しやすいように、関連解析されたデータからソフトウェアの構成をブロック図で表示して、必要な機能のソフトウェアが既存の修正で可能か、新規作成しなければならないか調査した。

この段階では、ソフトウェアの機能の有無を検索して調べるソフトウェアの構成をブロック図で表示した例を図４に示す。例では、人が全体を把握できる程度の比較的大きな機能単位でブロック図表示している。それぞれ吹き出しに示すように、下位のOS、ドライバ、ミドルウェア、アプリケーションと、よく使われるレベルに整理して表示した例である。表示の分解レベルは、階層化されたレベルで自由に表示可能であり、最小分割は、ソースファイル単位となる。今回修正が入る部分は図中２点鎖線で示した部分で、後の図５にアプリより詳細な階層で表示した例を示す。

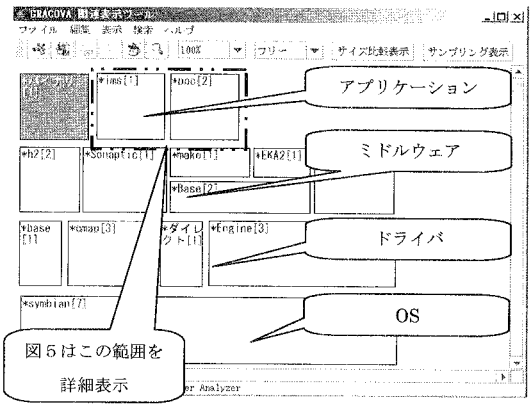

図４ ブロック図表示例

(4) 関連情報の表示

修正するソフトウェアモジュールを抽出したあと、関連しているソフトウェアモジュールに修正等の影響がないか関連情報を表示することで非常に簡単に確認ができ作業時間短縮に効果があることがわかった。一方、オブジェクトのみでソースコードが存在しない部分では、モジュール間の関連性の有無は自動的に判明できるが、入出力の方向は判別できず、入出力の確認が必要な場合は、オブジェクトの中を開発者が見る等して、手間がかかることがわかった。

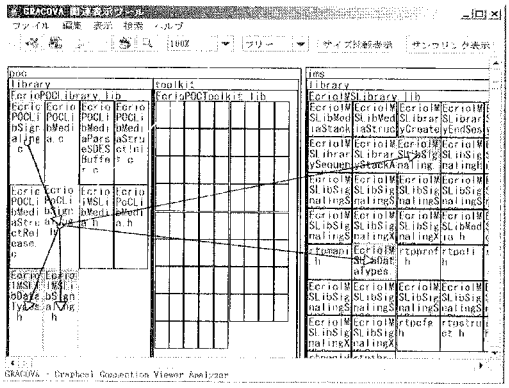

図５ 関連方向表示詳細時

① 関連方向表示

図５において、矢印は、指定した修正ライブラリブロックを中心に入出力の関係を表示させたものである。ブロック図上でどの機能ブロックと関係を持つか関連の方向性含め開発者にイメージが伝わりやすい。さらに、関連性があるブロック図のみをサンプリング表示

すると、図６のように表示され、チェック作業時に不要なブロック情報を隠すことが出来る。たくさんの関連をもつ場合に確認しやすいことが分かる。

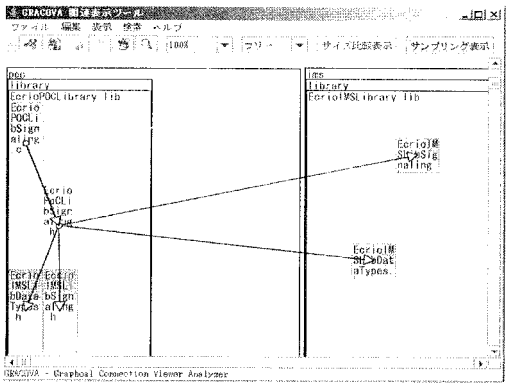

図６ 関連方向表示（サンプリング）

② 関数レベルの入出力表示

ブロック図を最小のソースファイル単位で関連表示するとファイル単位の実装レベルのチェックが可能である。関数レベルで入出力の関係をリスト表示すると図７のように表示される。中心に指定したブロックの関数がリスト化され、それぞれの関数を選択すると、左に入力として関連を持つ関数のブロック、ソースファイルの情報、右には出力結果が利用されるブロック、ソースファイル、関数の情報がリストアップされて表示されるため、関連先のチェックが効率よく実施できることが分かった。

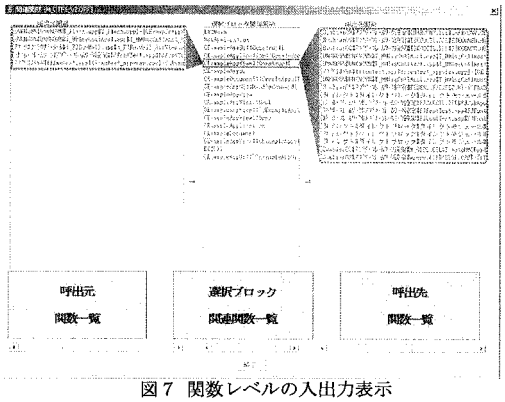

図７ 関数レベルの入出力表示

(5) 設計製作

設計製作時には、関連情報表示を利用して洗い出したソースファイルを修正するケースと新規ソースファイルの作成を行った。また修正時の新旧差分チェックの機能を確認した。

① ソースファイルへのリンク

修正するために抽出したブロック図からソースファイルへリンクして、プログラム内容の確認と編集が実施できることを確認した。図８に関連表示で指定したブロックのソースファイルを呼び出して編集する例を示す。ソースファイルを探し出す手間が省けるため作業効率の向上が可能となった。

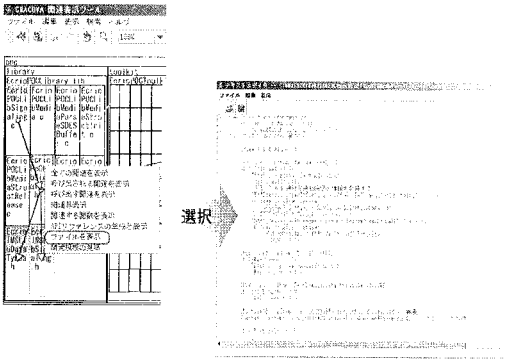

図８ ソースファイル表示

② プログラムの新規追加

プロダクト B において、新規アプリケーション（NewSoft）を追加した。ブロック図からトップダウンで関連付けの定義とソースファイルの組み込みを実施した。既存のソフトウェアの情報を確認しつつ、新規プログラムの作成が可能となるため、作業時間短縮に効果がある。

a) ブロック図上で、NewSoft 新規ブロックを追加する際の表示を、図 9 に示す。ブロック図上で新規ブロック作成機能を選択して必要なブロック情報を同時に入力することで、機能ブロックを追加することが出来た。

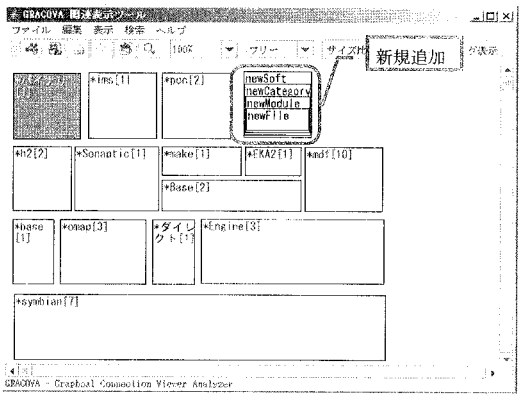

図 9 新規ブロック追加

b) 次に関連ブロック（ソースファイル）との関連情報を定義する機能を使い、ブロック図上で機能の入出力レベルをトップダウンで設計した。定義する際

の表示例を図 10 に示す。

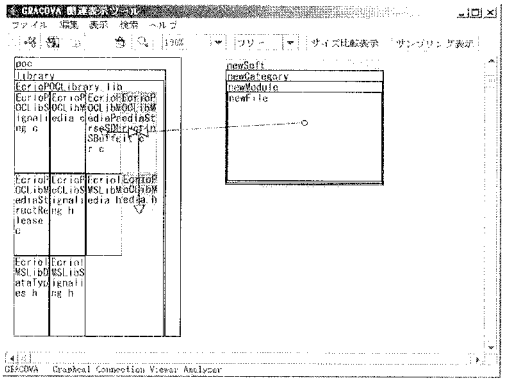

図 10 新規ブロック関連付け

ここでは、関数レベルで関連性の設計は可能であるが、ソースファイルに自動的にプログラムのコーディングとしては反映されない。そのため、設計後は通常のプログラミングで設計内容を開発する必要がある。

c) ブロックレベルの設計後、スケルトンコードを呼び出してソースふぁいるのプログラミングを実施する。図 11 に示すようにソースファイルを作成する際のヘッダ部分のコメント記述等が仮入力されたプログラムテンプレートが提示されることで、プログラミング作業の頭出しを簡単にすることができる。

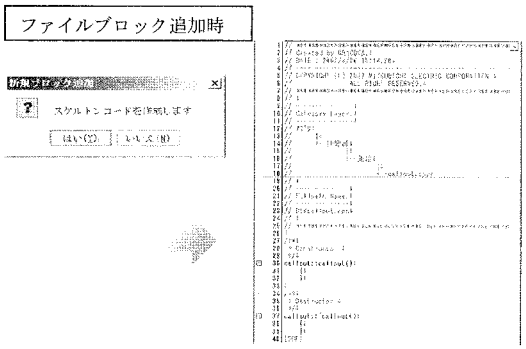

図 11 スケルトンコード

③ 差分チェック

ブロック図レベルで設計したブロック図間の関連情報を保存することで、開発作業中の新旧の差分を表示することができ、変更が発生した部分の確認が簡単に実施できた。ソースコードレビューや不具合発生時のチェック時、変更された部分が自動的に抽出できることから、確認抜けや漏れを防止する上で有効である。図 12 にライブラリ修正に伴い、追加、削除が発生した関連ソフトウェアの差分検出による確認を実施した時の表示画面を示す。図において、追加されたブロックには画面上赤のマーカが、削除されたブロックには黒のマーカが入って差分が一目で認識できる。

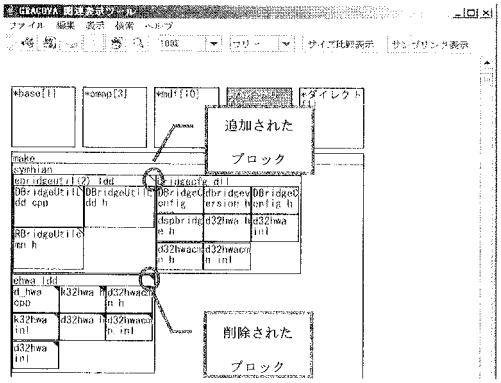

図 12 差分ブロック図表示

ただし、関連解析後にソフトウェア追加等で、関連情報を修正した場合は、ソースファイルに関連情報は自動的に反映されないため、ソフトウェアの実態と関連解析データは一意ではなくなる。その結果不具合が内在する可能性がある。最終的にソースファイルを修正して一度ビルドを行った後に、再度関連解析を実施して関連解析データとソフトウェアが一致した状態で確認が必要となる。

(6) ビルド

新規アプリケーションNewSoftのソースファイルを製作し、修正再利用するソフトウェアモジュールならびにその影響を受け修正が必要な関連ソフトウェアモジュールのソースファイルを修正し、ビルドを実施してプロダクトBのソフトウェアを製作することができた。

6. まとめ

本稿では、再利用を支援するモジュール間依存表示法を提案し、その仕様を実現するモジュール相互関連表示システムを構築した。さらに本稿では、構築したシステムの評価を行った結果、次のようなことを明らかにした。

- 大規模のソフトウェアプロジェクトで、ソースコードが存在する部分は、依存関係が全て抽出できることが分かった。
- 一方、オブジェクトのみ存在しソースコードが存在しない部分は、モジュール間の関連性の有無までは自動的に判明できるが、入出力の方向は自動で判別できず、オブジェクトの中を開発者が見る等して、手間がかかることがわかった。この部分を自動化で

きるか今後の課題となる。

・関連解析部で今回評価した規模のソフトウェアでは１時間程度の解析時間で実用範囲であったが、商用機器レベルのソフトウェアでは、５倍程度のサイズが想定されるため、大きな規模のソフトウェアで性能評価が今後必要と考える。

・ブロック図による開発者への情報提示と、ソフトウェアモジュール間の関連性の表示は非常に有効であることがわかった。ソースコードが存在しているソフトウェアモジュール間であれば、受け渡し情報の入出力の方向やＡＰＩレベルまで抽出/整理できるため、開発者の確認の手間と見落としの回避等に非常に有効である。

・新規追加時にテンプレートによるソースファイル追加は、新規参入の開発者には手っ取り早く環境に慣れるには有効であるが、関連設計情報に関する関数の呼び出し等が自動的にはソースファイル内に追加できないため、再度関連解析実施後に間違いの有無を確認する必要がある。人為的ミスの可能性が残るため、今後の改善の課題となる。

＜参考文献＞

[1]組込みソフトウェア 2006, 日経 BP 社 (2006)

[2]ソフトウェア・エンジニアリング・センター：“http://sec.ipa.go.jp/index.php”

[3]2006年度組込みソフトウェア産業実態調査報告書:“http://sec.ipa.go.jp/download/200606es.php”

[4]ソフトウェア・エンジニアリング・センター：組込みソフトウェア開発向けコーディング作法ガイド, 翔泳社 (2006)

[5]高田広章：「組込みシステム開発技術の現状と展望」, 情報処理学会論文誌, Vol.42, No.4 pp930-938 (2001)

[6]鷲澤暢亮, 神戸英利, 小泉寿男：構成管理をベースとしたソフトウェアの再利用法, 平成 18 年電気関係学会関西支部連合大会, G12-4, p.G289 (2006)

[7]岡本鉄平, 小泉寿男：組込みソフトウェア開発に XML 用いた再利用支援方式, 情報処理学会 ソフトウェア工学会, Vol.2002, No.136-18 pp.135-140 (2002)

[8]小池誠, 並木美太郎, 岩澤京子：分散環境における再利用性の高いオブジェクト作成を支援する共同開発環境の研究, 情報処理学会 ソフトウェア工学会, Vol.1999, No.122-6 pp41-48 (1999-3)

[9]新屋敷泰史, 三瀬敏郎, 江浦洋平, 畑中久典, 橋本正明, 鵜林尚靖, 片峯恵一, 中谷多哉子：組込みソフトウェア非正常系の概念モデル, 情報処理学会 研究報告, Vol.2004, No.145 pp.105-112 (2004)

[10]M.sugaya, S.Oikawa and T.Nakajima：Accounting system : A fine-grained CPU protection mechanism for embedded systems, Proceedings of 9th IEEE International Symposium on Object and Component-oriented Real-time Distributed Computing (2005)

[11]S.Oikawa, H.Ishikawa, M.Iwasaki and T.Nakajima : Providing protected execution environments for embedded operating systems using a u-Kernel, Proceedings of International Conference on Embedded and Ubiquitous Computing (EUC 2004) (2004)

[12]Saeid Nooshabadi, Jim Garside：Modernization of Teaching in Embedded System Design - An International Collaborative Project, IEEE Trans. on Education, VOL.49, NO.2 (2006)