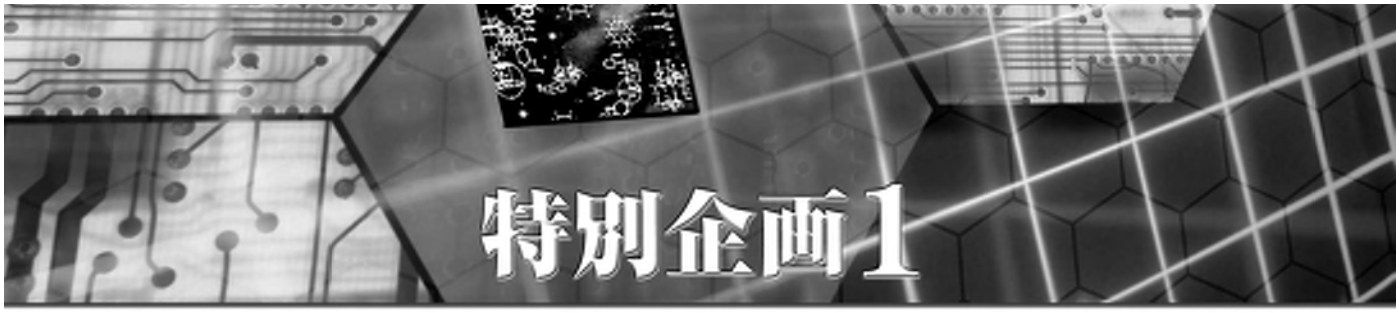


特別企画1

ユビキタスコンピューティング時代の組込みリアルタイムソフトウェア開発のアプローチ

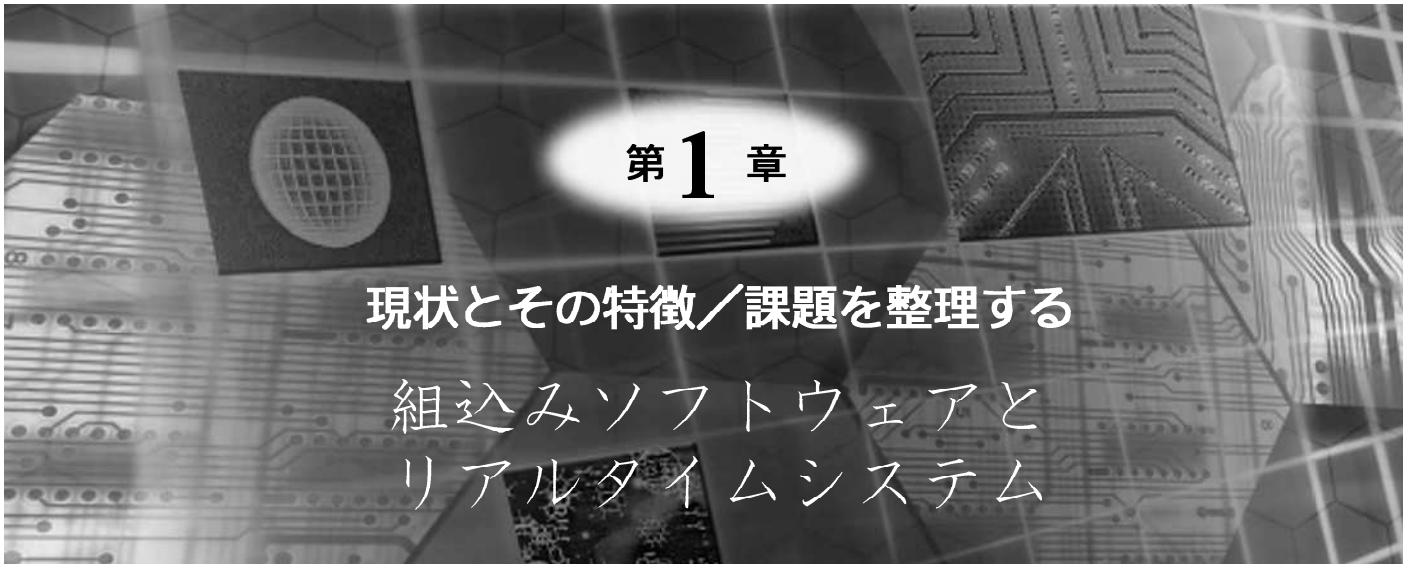
[基礎]

最近の組込みシステムは大規模化，複雑化の傾向は著しいものがあります．ユビキタスコンピューティングが，組込みリアルタイムシステムのソフトウェア開発ターニングポイントを迎えていると言ってよいでしょう．ネットワーク機能，ストレージ機能，セキュリティ機能など，10年前はエンタープライズ系の機能として考えられていたものが，組込みリアルタイムシステムに機能追加されるようになりました．また，組込みリアルタイムソフトウェアを開発する環境もここ数年急激に変化しています．製品のライフサイクルの短縮化と低価格化です．

本特集では，ユビキタスコンピューティング時代の大規模化，複雑化が著しい組込みシステムにおけるリアルタイムソフトウェア開発の開発アプローチについて，基本的なことから具体的な方法論まで解説します．

HASHIMOTO Software Consulting
橋本隆成　HASHIMOTO Takanari

SEEI認定CMMIインストラクター．護衛艦の艦船搭載用弾道計算プログラム，新型戦闘指揮システム，新型射撃制御システムなどの大規模ハードリアルタイムシステムのソフトウェア開発に10年間従事．以後，日本ヒューレット・パッカード，オージス総研，ソニーにて製品開発業務，開発手法・方法論のコンサルティングおよびCMMIによるプロセス改善業務に従事．



第1章

現状とその特徴／課題を整理する

組込みソフトウェアとリアルタイムシステム

はじめに

家電機器，情報家電，自動車など多くの製造業の製品にとって，組込みリアルタイムソフトウェアの重要性はますます高まっています．一方で，ソフトウェア開発を取り巻く状況は年々厳しい状況（激しい価格競争，短縮化されていく開発期間）になっています．ところが，開発規模は膨れ上がり，技術的な要求は年々高度かつ複雑化しています．

組込みシステムはその性質上，高度な品質が要求されます．ビジネス系のソフトウェア開発と比較して，技術的にもコスト的にも，いろいろな制約の中で開発することが要求され，高度な専門性ときめ細かな開発作業が要求されます．特に組込みリアルタイムソフトウェアは製品に組み込まれて出荷することもあり，出荷後にソフトウェアに問題が見つかった場合は，最悪製品の回収となります．回収にかかるコストもさることながら，企業のイメージダウンなど深刻な事態をもたらします．多くの企業が，今後の組込みソフトウェア開発の現状に対して何らかの対策を講じた開発戦略を取らなければ，確実に行き詰まる状況に直面するでしょう．

表1■組込みリアルタイムシステムの主な特徴

No.	特徴
1	機器の数だけ組込みリアルタイムソフトウェアの種類が存在する
2	処理時間に時間制約を課せられる
3	リアルタイムOSを用いることが多い
4	ミッションクリティカルなシステムが多い
5	メモリなどリソースの制約，ハードウェアの制約を強く受ける
6	機能および非機能的（セキュリティ，リアルタイム性，高信頼性，ROM/RAMリソース制約，etc）に高度な品質が要求される
7	テレビ，DVD，自動車，携帯電話などには，機能的に共通性が多いラインナップが存在し，シリーズ化される製品が多い

※組込みリアルタイムシステムにはいろいろな特徴がありますが，多種多様なため自分の開発するシステムの特徴をよく理解することが大切です

組込みリアルタイムソフトウェアの特徴から見た開発アプローチと課題

ここからは，組込みリアルタイムソフトウェアの特徴を考慮して，その開発アプローチを見ていきましょう．組込みリアルタイムシステムには，多くの場合ビジネス系のソフトウェア開発とは大きく異なる特徴があり，そのことがソフトウェア開発の手法やプロセスに大きな違いをもたらすことになります．

何が異なる特徴なのでしょうか？組込みソフトウェア開発に今後，科学的なアプローチを導入・実践するためには，まずは組込みリアルタイムシステムの特徴を俯瞰してみましょう．

組込みリアルタイムソフトウェアに要求される機能と制約

近年，従来の組込みリアルタイムシステムからユビキタス時代に進み，ネットワーク，セキュリティ，データの永続化機能を伴い，組込みリアルタイムシステムの対象範囲が拡大・発展してきています．何をもって「組込みリアルタイムシステムか？」という定義が困難になりつつあるほど，組込みリアルタイムシステムは大規模・複雑化，そして多様化し始めています．

その点から言えば定義は難しいのですが，ここでは，組込みリアルタイムシステムの学術的な定義や厳密な理論は避けて，まずは押さえておくべき組込みリアルタイムシステムの共通的な特徴（表1）を中心に整理してみましょう．表1に記述した以外にも，取り上げ方次第でほかにあるでしょう．しかし，表1に取り上げた特徴は多くの組込みリアルタイムシステムで

共通して見られるものです．

課題と開発アプローチ

組込みリアルタイムシステムといっても，それこそいろいろなシステムがあるのはご存知のとおりです．思いつくままに，具体的に挙げてみます．医療機器，携帯電話，デジタルカメラ，航空宇宙機器，自動車の車載システム（エンジン制御システム，カーナビゲーション，ABSシステム[編注1]など），電子レンジ，炊飯ジャー，オーディオ機器などの家庭電化製品，さらにエレベーターや自動販売機なども組込みリアルタイムシステムです．他にもまだまだあるでしょう（表2）．

こうしてみると私たちの利用する電化製品，工業機器製品には何らかの形で組込みリアルタイムソフトウェアが搭載されていることに改めて気づかされます．改めて理解できることは「組込みリアルタイムソフトウェア」は航空機，自動車などの乗り物から家庭用電化製品，工業機器製品などを制御するソフトウェアであることから，課せられる要求や制約は，組込みリアルタイムソフトウェアが搭載される機器製品に大きく依存することです．これはとりもなおさず，組込みリアルタイムシステムの開発のアプローチを画一的に述べることができないことを意味します．この点については次章で考えていきますが，ハードウェアで実現されていた機能をソフトウェア化するメリットについては図1に示します．

組込みリアルタイムソフトウェアがこれだけいろいろな種類があることから，表1の特徴をすべて満たしているシステムもあれば，いくつかの特徴を備えるものもあるということを理解しましょう．読者の皆さんが開発されているシステムはいかがでしょうか？

リアルタイムシステムとは？

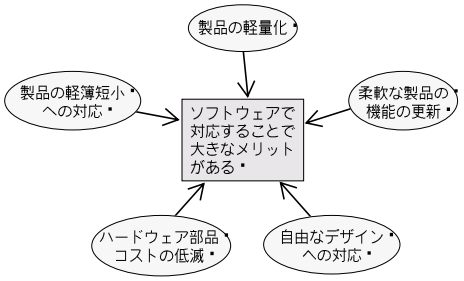
組込みリアルタイムシステムのソフトウェアには，要求仕様で求められる機能の処理時間に，時間制約が課せられることが一般的です．このような性能要求を「非機能要求」と呼び，処理に実時間制約があるシステムを「リアルタイムシステム」と呼びます．

表2■組込みリアルタイムシステムの一例

例
飛行場などの航空管制システム
エンジン制御システムなど自動車の車載システム
工場のファクトリーオートメーション
原子炉の制御システム
心臓ペースメーカーなどの医療機器
軍事兵器
家庭用を含む電化製品
産業用ロボット
航空宇宙システム
プラント機器
：

※私たちの身の回りに密着しているものばかりです

図1■ソフトウェア化のメリット



組込みリアルタイムソフトウェアが利用される機器を眺めてみると，実時間制約が課せられる理由は自明です．医療機器，携帯電話，航空宇宙機器，自動車の車載システム，電子レンジ，エレベーターや自動販売機など，どれをとっても，利用者の操作，他のシステムからの信号・イベントなどに対する処理応答時間がいつもバラバラでは大変なことになるからです．原子炉の制御をするシステムも組込みリアルタイムソフトウェアが使われていますが，デリケートな原子炉の制御にはミリ秒，マイクロ秒単位レベルの処理精度が要求されるでしょう．このような，実時間制約は組込みリアルタイムシステムの開発者にとって非常に大きな課題の1つです．通常ビジネス系（非組込みリアルタイム系）のソフトウェア開発では，ここまで厳しい実時間制約はありません．

ですから，一般に組込みリアルタイムシステムは，非組込みリアルタイムシステム（エンタープライズシステム）に比べ，課せられる非要求が数多く存在し，

編注1）Anti-lock Brake System．

かつどれもが開発者からみて厳しい要求が多くなります．ここで非機能的要求とはセキュリティ，リアルタイム性，高信頼性，リソース制約など性能や制約的要求を意味します．

しかも，組込みリアルタイムシステムで求められる要件では，非機能的要求で求められる性能が機能と同じように重要です（表3）．エンタープライズ系のソフトウェア開発では，非機能要求は補足的に扱われ，機能要求が主体で開発が進められることが多いからでしょう．

リアルタイム性，フォールトトレランス性，可能な限りコンパクトにするコードサイズなど高度な要求が課されるために，組込みリアルタイムシステムのソフトウェアには，他のソフトウェアとは異なった分析，設計，テスト方法が要求されます．なお，リアルタイムシステムについては，次章で改めて取り上げることとします．

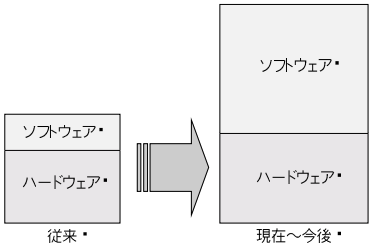
メモリなどリソースやハードウェアの制約を強く受ける

組込みリアルタイム系ソフトウェアに限らず，従来であればハードウェアで実現されていた機能が，ソフトウェアで実現する機能になっていますし，まったく新しい機能の追加も急激に増えています（図2）．このことは，つまりソフトウェアの性能，品質が今後の製品の優劣を決める時代であることを意味します．PCやUNIXなどのワークステーションが，ソフトウェアがなければ，“ただの箱”の状況になっていくことから，ソフトウェア主導の設計が可能になります．

表3■組込みシステムで求められる主な非機能的要求

No.	非機能的要求
1	プログラム言語の指定
2	処理応答時間などのリアルタイム性
3	利用するリアルタイムOS
4	ROM・RAMメモリ制約
5	保守性や拡張性への要求
6	フォールトトレランス
7	堅牢性

図2■ハードウェアとソフトウェアで実現される機能の割合増加



しかしながら，組込みリアルタイムソフトウェアの特徴，開発の制約という観点から見ると，ハードウェアがあくまで主役の状況は続きます．理由については，携帯電話を例にして考えましょう．携帯電話は内蔵している機能もさることながら，デザイン性や携帯性が重視されます．重い携帯よりは軽くて，操作性に優れ，デザインもおしゃれなほうが商品価値は高くなります．もちろん価格も低ければ，より商品の競争からみて有利です．

逆の例は自動車です．自動車に搭載されるソフトウェアは車載システムと呼ばれますが，エンジン制御，ABSシステム，カーナビゲーションなどが代表的なものとしてすぐ思い浮かびます．自動車も従来であればハードウェアで実現していた機能をソフトウェアで実現すれば，機能の追加もハードウェア的に大きく変更することなく可能になります．また，ソフトウェアで実現すれば大きく軽量化に寄与しますから，エコロジーで注目されている燃費などの問題にも有利です．

デザイン性や製品の重量は多くの場合，ハードウェアの制約を強く受けます．昨今の携帯電話が急激に薄くなり軽量化できた理由の背景はハードウェアの技術革新がありますが，さらにデザインや軽量化に有利なソフトウェアを使用しなければならない制約もあります．加えて，製品価格の内訳の多くはいまだ使用するCPUやROM，RAMなどの価格による影響が大きくなります．なにしろ，一部の特殊なものを省いて，白物家電，情報家電，自動車などは何万台というレベルで販売しますから，使用するハードウェアの価格を低く抑えることは利益を出すうえで極めて重要な選択となります．当然，低価格なCPUやROM，RAMであればあるほど良いわけですが，反面性能や容量が限られたものになります．

このような制約の影響を吸収するのはソフトウェア側で行います．つまり，組込みリアルタイムソフトウェアの場合，ソフトウェアの都合で決められることが限られており，なおかつ技術的に重要な影響をおよぼすことを，ソフトウェアの設計で解決していかなけれ

ばならないわけです．

テレビ，DVD，自動車，携帯電話などシリーズ化される製品が多い

組込みリアルタイムソフトウェアを開発するうえでの大きな特徴として，組み込まれる製品のシリーズ化があります．

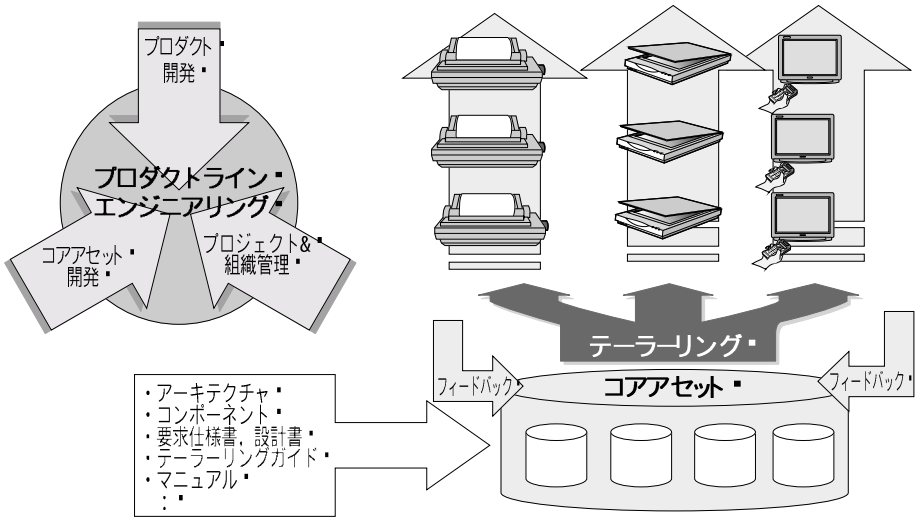
白物家電，情報家電，自動車，航空・宇宙など多くの製品やシステムは1回限りの製品もしくはシステムの開発であることは少なく，多くの場合製品は機能拡張をしながら市場へと供給されていきます．特にデジタルテレビ，DVDレコーダー，携帯電話，プリンタなどに代表されるコンシューマ製品の製品寿命が年々短くなり，新製品を短い開発サイクルの中で開発し，市場投入しなければならない中で，機能追加／変更をハードウェアで行うのは開発コスト，製品設計上でも困難です．製品にもよりますが，製品にはハイエンド，ローエンドなどいくつかのバリエーションがある場合があります．このような場合にはソフトウェアで機能を実現することで，製品に求められる機能の柔軟な対応や開発コストの大きな低減が期待できます．

また，今後開発予定の製品戦略の将来に渡るロードマップから，あらかじめ複数の製品対応で，共通に利用できるソフトウェアアーキテクチャを開発／準備し，組織の資産とすることも可能であり，これを戦略的かつ体系的に実現しようとしている方法論が「プロダクトラインエンジニアリング注1」です．またその戦略によって何が実現できるのかについては表4，図3にまとめます．

表4■プロダクトラインエンジニアリング戦略によって何が実現できるのか？

戦略の狙い	説明
開発期間の短縮	「市場調査～要件開発～テスト・保守」までの製品開発の全ライフルサイクルを戦略的にプロセス定義し，実践する
生産性の向上	製品間に共通のソフトウェア部品を構築・再利用することで，システマティックなソフトウェア再利用の組織戦略を実現する．新規開発・テスト作業を大きく短縮することを狙う
品質の向上	ソフトウェア資産の再利用により新規開発は最低限にすることで，新たな不具合を軽減する．ソフトウェア資産の再利用を繰り返すことで品質の向上が期待できる
開発エンジニア，テスト担当者の不足の軽減	重複や無駄な作業の排除によって，担当者の拡散を防止し，ソフトウェア資産の再利用とカスタマイズによる作業負荷の削減

図3■プロダクトラインエンジニアリングの戦略イメージ



注1）ソフトウェア集約的なシステム開発を狙い，共通で管理された一連の機能の資産を組織で共有して開発するしくみを指します．そのためには，事前に明示的かつ戦略的に，共通で核となる再利用するアーキテクチャやソフトウェアコンポーネントなどの一連の資産（アセット）の構築と利用のインフラが重要になります．多くの場合，オブジェクト指向技術によって再利用の技術的しくみは実現されます．

組込みソフトウェアを取り巻く情勢の変化

加速する国際標準化の波

現在のように「グローバリゼーションの時代」には，国際企業として活動していくことが求められることになるでしょう．開発も国内だけでなく，海外企業との協業・提携も多くなっています．そのため国際標準および業界標準となるプロジェクト管理や企業の開発プロセスのモデルが重要視されています．そこで各国の企業ともSEI[注2]のCMMI[注3]やPMI[編注2]のPMBOK[編注3]など世界標準の取り組みに熱心になっています．米国の国防総省に至っては，開発を委託する際の企業の基準をレベル3と発表してから，民間企業間でもレベル3が業務委託をする際の基準として広まっていますが，昨今では自動車分野，医療分野が製品やソフトウェアなどの成果物だけでなく，開発体制，環境，各種ドキュメンテーションおよび開発・管理プロセスも厳しく評価していく時代になりました．品質は開発プロセス，管理プロセスに大きく影響を受けるということがわかっているからです．評価方法は決められた厳密なフレームワークによって実施されます．

「職人芸」による開発から「ソフトウェアエンジニアリング」に基づく開発への移行

組込みシステムは，以前は航空宇宙，防衛システムなどを除いて，小規模の開発規模が多かったのは事実です．そのためでしょうか，すでに紹介したように組込みリアルタイムシステムは非常に数多くの製品が存在することもあり，組込みリアルタイムシステムの開発を行う企業は大企業ばかりでなく，中小および中堅企業など多くの企業が関わっています．

このような状況の中，組込みリアルタイムシステムを受託開発する企業は，請け負う製品を長年開発してきている企業が多く，頻繁に開発する製品を変えることはありません．開発者は高度な技術を身に付けた経験者が多く，小人数でも開発する製品の専門家として業務を行います．

デリケートな設計を要求される組込みリアルタイムシステムは，ソフトウェア開発者であっても，ハードウェアの知識を必要とすることも多く，一人ひとりの技術者の専門性で成り立っています．言わば，職人芸的な開発を要求され，事実，エンジニアとしての本領発揮というべき，職人技を駆使して組込みリアルタイムエンジニアは作業します．

一方，職人芸的な開発にはデメリットもあります．開発者一人ひとりの経験とスキルに依存するため，ソフトウェア開発の生産性は高くありません．また，どうしても開発者の能力に依存するため，開発の進捗や品質にバラツキがあります．加えて，インド，アジア諸国への開発コスト競争の波にさらされることになる今後は，開発エンジニアの人件費も重要な検討内容です．今後の組込みリアルタイムソフトウェアが大規模・複雑化し，開発期間の短縮が厳しくなるうえでこの問題は極めて大きいでしょう．

まとめ

以上により，職人芸による開発からソフトウェアエンジニアリングに基づく開発への移行は，好む好まざるにかかわらず“待ったなし”の状況になりました．昨今は日本の組込みリアルタイムシステムでも要求される機能が大規模し，ソフトウェアが複雑化しています．そして，以前のようなROM，RAMなどのリソース制約が緩やかになってきています．読者の皆さんがオブジェクト指向技術や反復開発などの開発・管理プロセスを導入するチャンスが多くなったと言えるでしょう．組

注2）米国カーネギーメロン大学のソフトウェア工学研究所（Software Engineering Institute）．
注3）CMMI（Capability Maturity Model Integrated）は，SEIが開発したプロセス能力の成熟度のモデル．SEIがCMMIを策定した当初の目的は，米国の国防総省が企業に開発を委託する際に，企業の開発能力を判定（監査）する手段として利用するためでした．しかし，その後多くの企業において，社内のハードウェアおよびソフトウェア開発プロセスを改善するための手段としてCMMIが利用されるようになり，現在に至っています．
編注2）プロジェクトマネジメント協会．
編注3）Project Management Body of Knowledgeの略で，ピンボックと読む．プロジェクトマネジメントのための知識体系．

第2章

方法論，手法，プロセスの基礎

組込みリアルタイムシステムの開発手法と戦略

はじめに

組込みリアルタイムソフトウェアの開発を効果的に進めるにはどうしたらよいのでしょうか？

この答えは誰もが知りたいものです．しかし，この質問には画一的な解答はないと考えたほうがよいでしょう．前章で説明したように，組込みリアルタイムソフトウェアは多様な機器に組み込まれるうえに，要求される内容，制約条件が多岐にわたるからです．そこで，本章は組込みリアルタイムソフトウェアの開発アプローチについて考えていきたいと思います．

手法・方法論の多様化と拡大

組込みシステムの開発手法と戦略

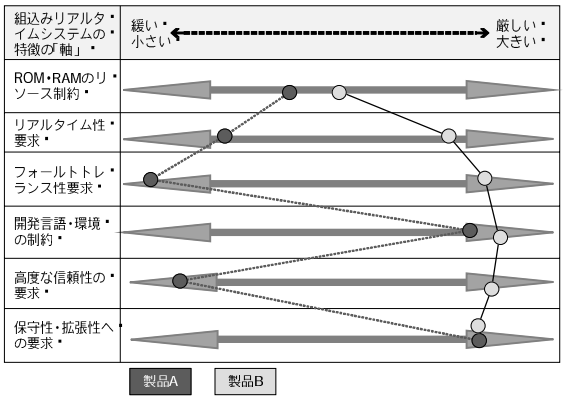
組込みリアルタイムソフトウェアの開発では必ずと言ってよいほど頻繁に語られることに，「ビジネス系ソフトウェアのような新しい手法や方法論はなかなか導入できない」があります．理由はいろいろあるでしょうが，組込みリアルタイムソフトウェアの開発エンジニアに話を聞くと，「我々の開発している組込みリアルタイムソフトウェアには，リソースの制約，リアルタイム性など数多くの制約があり，オブジェクト指向技術をはじめ，新しい手法や方法論には無理があるように感じる」，「効果的な手法や方法論を用いたいが，事例の多くは自分たちと異なるシステムで，どのように用いるのかわからない」というのが多いようです．まずは，この問題から考えていくことにします．

組込みリアルタイムソフトウェアがなぜ難しいのか？

組込みリアルタイムシステムは“多種多様”です．このことについては説明を要しないでしょう．筆者も経験がありますが，同じ企業でも部署が異なり，開発している製品が異なれば開発の方法も異なっています．隣の部署が開発しているシステムと同じような方法で自分たちの製品のシステムが開発できるとは限らないところが，組込みリアルタイムシステムの難しさです．では，具体的に何が異なるのでしょうか？筆者なりに図1に整理をしてみます．

組込みリアルタイムシステムの特徴を分類・整理する視点はいろいろとあると思われます．ですから，図1の分類は一例ですが，「開発するシステムに課せられる制約」を意識して分類してみました．組込みリアルタイムシステムの開発に多くの制約を与えるものとしては「組織の文化・風土」や「開発プロジェクトの予算」「開発期間」「開発エンジニアのスキル」など，管理面の課題もありますが，ここでは対象外にしていま

図1■製品ごとにそれぞれの制約のウエイトが存在する



す．

また，重要かつ組込みリアルタイムソフトウェアの開発を高度にしていることに，各特徴（軸）が，直交性を持ちつつも，相関関係にあるということです．フォールトトレランスとリアルタイム性は基本的に異なる要求ですが，要求されているフォールトトレランスを実現するには，リアルタイム性を実現するうえで欠かせない並行性のタスク設計に大きく関わってきます．また，高度な信頼性は，検証や妥当性確認といったレビューやテストを適切に実施することで実現されますが，それ以前に開発するソフトウェアのアーキテクチャが効果的に設計され，実装されていないといけません．

以上から，組込みリアルタイムソフトウェア開発を効果的・効率的にするには，開発アプローチを検討する前に，要求定義・分析作業で機能要求はもとより非機能要求も明確にしておく必要があります．

■組込みリアルタイムシステムのコンテキスト

前章で組込みリアルタイムシステムにはいろいろなシステムがあると書きました．ここでは改めて繰り返して解説しませんが，組込みリアルタイムシステムの特徴である，リソースの制約の厳しさ，リアルタイム性の条件，拡張性・保守性への要求，利用するリアルタイムOSの種類，開発言語，開発期間，システムの寿命，そして開発予算などいろいろな課題・条件・制約が存在します．開発するシステムごとにそれぞれの重みづけも異なるでしょう．

これまでの話からソフトウェア設計を行うとき，どのようにソフトウェアを構造化すれば効果的かという答えは，すべての組込みリアルタイムソフトウェアの開発において画一的に決定できません．図1で示した制約条件などの非機能要求や開発組織・プロジェクトの都合や状況が大きく影響するからです．ここではこのようなことを，開発における「コンテキスト」と呼ぶことにしましょう．このコンテキストは製品ごと，プロジェクトごとに大きく異なりますから，このコンテキストを考慮しない開発手法・方法論は意味が薄れてしまうのです．

組込みリアルタイムシステム開発の基礎

課題は大きく2つに分けられ，技術的な課題と開発予算，プロジェクト管理，開発エンジニアの人材確保・育成など非技術的な課題が存在します．どちらもシステム開発を確実に完遂させるためには重要な課題であり，互いに深く影響をおよぼすものもあり，必ずしも分離・独立させて解決できるとは限らない課題も存在しますが，今回は技術的な課題に注力することにしましょう．

課題を解決するためにはいろいろなことに取り組まなければなりません．手法・方法論ばかりでなく，開発言語や採用するリアルタイムOSなどの性能やメカニズムにも大きな影響を受けます．ここからは，組込みリアルタイムシステムが取り組む課題をどのように解決していくのかをいくつかの視点から見ていくことにしましょう．

リアルタイムとは？

リアルタイムシステム

リアルタイムシステムは，日本語では実時間システムとも言われますが，高速処理システムとは訳されていません．リアルタイムシステムは確かに要求された処理に対して，短い時間で処理をし，応答しなければならないことが多いのは事実です．その意味では「高速処理システム」と呼ぶことも性質的な一面をよく表現しています．英語でもリアルタイムシステムとリアルという用語が用いられているのは，コンピュータの中の時間間隔と外の世界，つまり私たちの生活している時間が一致していることに由来します．

コンピュータの中には，CPUを動作させるためのクロックと呼ばれるものがあり，これはコンピュータの世界の時間間隔を司っています．つまり，コンピュータの世界は閉じており，時間は私たちの世界とは別の時間が流れています．実世界では1年の時間をコンピュータのシミュレーションでは数秒以下であることは普通にありえます．これは，特にコンピュータの処理

が私たちの世界に影響を与えないシステムであるからです．

■組込みにおけるリアルタイムシステム

一方，組込みリアルタイムソフトウェアの多くは，自動車の車載システム，原子炉の制御を行うソフトウェア，航空宇宙のシステムなど私たちの操作や組込みリアルタイムソフトウェアとつながっている他の装置やシステムとのやり取りを行い，正確に同期することが要求されます．そのために，実際の実時間と一致するということで，リアルタイムシステムといわれます．

「ハードリアルタイムシステム」と「ソフトリアルタイムシステム」

演劇やコンサートの劇場のチケット，飛行場での旅客機のチケットの予約・購入などを処理するシステムも，広い意味ではリアルタイムシステムと呼ばれることがあるようです．これらのシステムはカウンター越しにお客さんからチケット購入の要望をうけて，リアルタイムで空席情報を確認し，予約できるようになっています．しかし，組込みソフトウェアの開発でのリアルタイムシステムとは区別しています．チケットの予約・購入などを処理するシステムは多くの場合，以前のバッチ処理に対してリアルタイムと呼ばれますが，具体的な時間制約を課せられているわけではないですし，時間制約が仮に要求されても，かなり緩やかな制約と言えるからです．

私たち組込みリアルタイムシステムの開発者が扱うリアルタイムシステムはもう少し複雑です．それでも，組込みリアルタイムシステムで扱うリアルタイムシステムは，時間制約の厳しさという視点から見ると「ハードリアルタイムシステム」，「ソフトリアルタイムシステム」の2つに大別できます（表1）．

一般的に，大半の組込みリアルタイムシステムは，すべての処理をハードリアルタイム制約が課せられることは多くなく，システムが通常実施しなければならない処理は，「ハードリアルタイム」処理と「ソフトリアルタイム」処理が混在していることが多くなります．ただし，システムに1つでもハードなリアルタイム制約が課せられる処理が存在するときは，システムの設計に大きく影響を与えます．それだけ，リアルタイム性の制約は大きいのです．

イベントと種類

■イベントをなぜ意識するか

大部分の組込みリアルタイムシステムは，電気的または機械的なデバイスと接続されています．システムは多くの場合，ハードウェアの監視や制御を行うために組み込まれているので当然でしょう．センサはシステムの外部環境の状態を信号でシステム情報を与えます．システムはアクチュエータを使ってその外部環境を制御します．

このことから，組込みリアルタイムシステムは接続されている外部機器の信号によるイベントが処理の起動のきっかけとなることがわかります．携帯電話，家電製品のようなシステムの場合，人間がシステムに対してボタン操作を通じて処理を要求することもありますが，その場合でも，操作がボタンのデバイスから信号となりシステムに伝えられることになります．

組込みリアルタイムシステムの開発ではシステムが外部とどのような電気的または機械的なデバイスと接続されているかを理解し，次に電気的または機械的なデバイスとどのようなイベントのやりとりをするかを調査する作業から始めます．システムが外部とどのような電気的または機械的なデバイスと接続されているかを示す図を「システムコンテキストダイアグラム」と呼びます．一方，外部のデバイスとどのようなやイベントのやりとりをするかを調査する作業を「イベント分析」と呼びます．

表1■リアルタイム制約

リアルタイム性	制約／説明
ハードリアルタイム	決められた時間制約内に処理が確実に終了しなければならないシステム．特に人命に関わることもある航空宇宙や軍事防衛システム，医療機器のシステム，自動車の車載システム，原子炉の制御システムが代表的
ソフトリアルタイム	処理が決められた時間制約内に，原則的に各処理が終了することが基本であるが，ときどき，かつわずかであれば時間制約から遅れてもかまわないシステム．多くの場合，処理の平均的な応答時間を重視する．特定のアクションの時間制約の厳密性ではなく，平均的スループットに制約を与えているともとれる

ユビキタスコンピューティング時代の組込みリアルタイムソフトウェア開発のアプローチ

■イベントの分析

リアルタイムシステムの分析作業で大切なことの1つは，システムが課せられている時間制約の要求を明確にすることです．この時間制約は必ずしも要求仕様書には記載されていません．システムに要求されている機能や特徴から，「システム要求分析」作業を通じて，要求されている処理の具体的かつ詳細な時間制約を洗い出します．このとき実施する作業の1つがイベント分析です．

組込みリアルタイムシステムでは外部機器の信号によるイベントが処理の起動のきっかけとなるわけですから，外部からのイベントにどのような処理を実施し，応答時間はいつまでなのかを明確にします．さらに，イベントごとにどれくらいの間隔でイベントが入力されるのかも調査します．分析結果は「イベント分析仕様書」（図2）に記録していきます．

イベント分析仕様書がひととおり完成されれば，システムが外部からどのようなイベントが入力され，システムが行うべき処理の応答時間のパフォーマンスが明らかになります．ソフトウェアの処理の応答時間のパフォーマンスを保証するようにソフトウェアの設計することが作業となります．

■イベントの種類

イベントは，どれくらいの間隔でシステムに入力されるかという到着パターンで分類されます．最も一般的なカテゴリは，「周期定期」と「不定期」の2つです．

周期定期イベントは，50ミリ秒なら50ミリ秒間隔でシステムにイベントが到着します．周期性があるというのは実はリアルタイム制約を設計に反映するうえで大変有利となります．あらかじめイベントが入力されることを予期することができるからです．時間制約のあるリアルタイムシステムは，優先度の高い処理や時間のかかる処理をどのように効率的に行うかが大きな設計テーマですから，「どういうイベントが，どういうタイミングで入力されるか」は重要な設計情報なのです．しかも，複数のイベントが同時にあるいは，次々と入力される場合もあるかもしれません．そうなると，実時間を満たすことは，より一層難しくなるからです．

一方，不定期イベントは，文字どおり周期性がないわけですが，到着を周期的にうまく扱えることもできます．一般的には，最短到着間隔を調べることでわかります．すなわち到着するメッセージインスタンス間の最短時間を調べることになります．また，外部からのイベントの数が多いほど，さらに発生頻度が高いほど，複数の外部割り込みが集中して発生する可能性が高く，システムの処理能力に大きな余裕がないと処理しきれなくなりますから，イベント分析の作業がいかに重要かわかっていただけるのではないでしょうか？

それ以外には，ソフトリアルタイム制約の場合には，平均時間間隔や標準偏差を使った確率プロセスから到着間隔を行うことも可能です．統計的に，あるイベントと次のイベントとの間には相互関係があるということも利用されます．

マルチタスクとスケジューリング

システムが外部イベントを受け取ったあと，その応答処理が行われるまでの間の時間に時間制約が与えられています．システムは多くの外部機器と接続されていることがありますが，イベントがどのように続けて入力されても時間制約を満たせなければなりません．イベント分析を通じてイベント仕様書を作成して，イベントの到着間隔などを調べましたが，システムに入るイベントの到着間隔がわかるだけでは時間制約を満たす設計を行うにはまだ情報が足りません．

図2■イベント分析仕様書（例）

イベントID	イベント名	処理内容	発生元	入力・出力	同期・非同期	時間制約	備考

※外部からイベントに対する処理を整理します．この表の情報は機能分析，タスクの並行性の設計などに利用されます．

■処理時間の見積もり

イベントに対してシステムが行う処理にどれだけの時間がかかるかを計算しなければなりません．ただし，処理が必ずしも一定の時間で完了するとは限らないこともあり，時間制約が厳しいハードリアルタイム制約では処理間見積もりは，最も処理時間がかかる「最悪ケース」の実行時間で持つことが多くなります．しかし，常に処理は最悪ケースではありませんから，最悪ケースがどれくらいの頻度で起こるかにもよりますが，最悪ケースでの見積もりは，ハードウェア／ソフトウェア的にも冗長な設計を行わないといけないことがあります．これは開発コストに跳ね返るということです．

さて，実装レベルで見ると，処理には1つあるいは複数のサブルーチンや関数あるいはクラス（インスタンス）のメソッドが呼び出され実行されますが，これらの処理動作全体のパフォーマンスを見積もることになります．

また，見積もりにはボードのバス処理動作，メモリへのアクセス予測時間など全部が含まれます．この時間の見積もりができなければリアルタイム性を保証するなど不可能になるということです．

しかし，実際には単純ではないために，リアルタイム制約を実現することが難しいというわけです．処理時間の見積もりと，システムの設計を複雑にしているのは，処理の多くが明確には独立していないためです．つまり，ある処理がメモリのあるセクションなどリソースを共有する場合，一方の処理は他方が完了するまでブロックされることがありますが，実行時間見積もりはこのブロック時間（ある優先度を待つアクションが，必要なリソースを所有する優先度の低いアクションによりブロックされる時間）も考慮に入れなければなりません．

さらに，たいていのシステムでは単純に入力イベントを処理することはできません．次々と入力イベントが来ますから処理しきれず時間処理を満たせないからです．そこで，マルチタスクの機能を提供するリアルタイムOSを用いて，タスクの抽出，タスク間通信，タスクの優先度付与とスケジューリングの設計作業を行います．リアルタイムOSは設計者がイベントおよびに対応した処理から検討したタスクの優先度を検討し，タスク優先度に基づいてスケジューリングを実施します．

組込みリアルタイムシステム開発方法論「DARTS」

昨今は「ユビキタス時代」と言われ，今後組込みリアルタイムシステムの大きな需要が見込まれています．それと同時に大規模・複雑化していくことが必至です．一方で組込みリアルタイムシステムの効果的な開発アプローチを紹介した書籍はあまり見かけません．

特に開発エンジニアは，組込みリアルタイムシステムで大きな課題である実時間制約を満たすためには，リアルタイム性をシステムに実現しなければなりません．このためには多くの場合リアルタイムOSを利用することになりますが，その際並行性について取り扱うことになります．つまり，リアルタイムOSが提供する並行性のサービス（スレッド，プロセス，通信メカニズムなど）を用いて並行性を実現させます．具体的に言えば，実行単位であるタスクの抽出，タスク間の通信のやり取りのデータの決定と通信メカニズムの方法の決定およびタスク間の優先度です．

しかし，多くの場合実時間制約を満たすためには，どのような並行性を設計すれば良いのか？という問題に，日本国内で明確に応えてくれる手法・方法論をあまり見かけない印象があります．

米国やEU諸国では航空宇宙・軍事防衛分野のソフトウェア開発が盛んであることから，歴史的にも長いリアルタイムシステムのソフトウェア開発手法・方法論が数多く存在します．ソフトウェアの歴史を見てもコンピュータが軍事的な利用から始まったことを考えると当然と言えるかもしれません．実際に過去の手法・方法論にもリアルタイムシステム用のソフトウェア開発手法・方法論はかなり存在します．

これらの手法・方法論は組込みリアルタイムシステムでしたから，当然，リアルタイムOSが提供する並行性のサービス（スレッド，プロセスなど）を用いて並行性を実現さるアプローチを扱っています．

今現在も過去も組込みリアルタイムシステムの開発では，実時間制約は極めて大きな技術的な課題です．言い換えれば，ソフトウェアアーキテクチャ構築に最

も大きな影響を与えます．並行性の扱いは非常にデリケートな問題です．それに加えて，ソフトウェアの“生産性”を向上し，かつその“信頼性”を高めるという両立しがたい困難な問題を解決しようとしているのが組込みリアルタイムシステムの手法・方法論と言えるでしょう．昨今は，オブジェクト指向全盛でかつ，再利用性が大きなトピックとして扱われ，組込みリアルタイムシステムの扱いがまったく見られない手法・方法論があります．ビジネス系の手法・方法論では特に顕著ですが，「組込み」あるいは，「リアルタイムシステム」とタイトルが付く手法・方法論でも，要件定義，オブジェクト指向・分析による解説が中心で，まともに扱っていないものがあります．

そこで，“温故知新”の諺(ことわざ)どおり，過去から現在まで利用されてきた組込みリアルタイムシステム向けの手法・方法論を見ていくことにしたいと思います．特に「組込みリアルタイムシステム向けの手法は並行性をどのように扱っているのか」をテーマにしていくことにしましょう．

DARTS方法論とは？

実時間を取り扱った手法・方法論を具体的に紹介しましょう．ここでは「DARTS」を取り上げることにします．DARTSを今回取り上げるにはいくつかの理由がありますが，DARTS（Design Approach for Real Time Systems）は，組込みリアルタイムシステム用の方法論ということと，ソフトウェアにおけるアーキテクチャの開発において，実時間制約を意識したアプローチを取っています．

組込みリアルタイムシステム開発において，リアルタイム性や並行性を実現するプログラムの開発・保守・変更にはコストがかかります．そのため，DARTSを改めて理解することは，非常に参考になるアプローチだと思います．まず，最初に簡単にDARTSの紹介を兼ねて，特徴を表2に整理してみます．

■DARTS兄弟シリーズ

方法論DARTSは，Hassan Gomaa博士により提唱されています．そして自身がさらに発展／派生版である「DARTS/DA」（DARTS for Distributed Application）「ADARTS」（Ada-based Design Approach for Real Time Systems），「CODARTS」（Concurrent Design Approach for Real Time Systems）を提唱しています．

さらに，同じくGomaa博士により，オブジェクト指向ベースのアプローチを追加した分析アプローチの「COBRA」（Concurrent Object Based Real-Time Analysis）や最近のプロダクトラインエンジニアリングのアプローチを技術的視点に重点をおいて解説した「PLUS」（Product Line UML based Software engineering）も発表されています．こちらはUMLによるオブジェクト指向によるコンポーネントベースのアプローチによる再利用をプロダクトラインエンジニアリングのコンセプトに基づいて実現する手法です．

DARTSは，組込みリアルタイムシステム向けに，ソフトウェアのモジュール化を詳しく解説しています．DARTS自身も実際のシステムで実用されていますが，多くの他の研究者の手法・方法論に影響を与えています．特にデータフローダイアグラムからソフトウェアのモジュール化およびタスクへの抽出とモジュールとの対応などが詳しく解説されています．

ADARTSはプログラミング言語Ada(エイダ)への設計・実装の展開を意図したもので，Ada言語の持つ情報隠蔽

表2■DARTS（Design Approach for Real Time Systems）の特徴

No.	特徴
1	航空宇宙・軍事・防衛分野を中心として実績がある
2	外部イベントから起動される処理やデータ変換処理から，タスク抽出・合成およびタスク間通信の具体的な作業手順・ガイドラインが与えられている
3	外部イベントから起動される処理やデータ変換処理から，ソフトウェアのモジュール構成にする手順が解説されている
4	タスク抽出とモジュールの統合手順が解説されている
5	DARTSは「CODARTS」「ADARTS」，さらには「PLSU」と発展した手法・方法論が存在し，これらのベースとなっている
6	特定のプログラミング言語，リアルタイムOSを指定しない（つまり汎用性がある）
7	DARTSを解説した書籍・参考文献が存在する[編注1]

編注1）参考文献は章末（135ページ）を参照ください．

機能のカプセル化の単位となるモジュール化の機能である「パッケージ」，Adaの並行性を実現する言語仕様が持つ機能「Adaタスク」への展開を意図したものです．

また，CODARTSは，組込みリアルタイムシステムのうち，複数のシステムによって実現される分散システムへの拡張が盛りこまれています．この各方法論DARTS，DARTS/DA，ADARTS，CODARTSはいずれもいくつかのステップから構成されていますが，共に作業の大半はDARTSと同じ作業となっています．ADARTSは詳細設計から実装でAdaへの展開，DARTS/DAは分散環境に対する分析・設計が冒頭と最後に，CODARTSは並列環境への配置が拡張されています．ゆえにDARTSを理解することはADARTS，DARTS/DA，CODARTSを理解するうえでもその基礎となります．

特に，コンテキストダイアグラム，データフローダイアグラム，モジュールダイアグラム，タスク構成・通信図を作成するまでの作業は同じ手順になります．

なお，章末の参考文献に示した書籍には，DARTS，DARTS/DA，ADARTS，CODARTSの解説がすべて掲載されていますから，興味がある場合は参考にしてください．

■DARTSのカバー範囲

DARTSの直接的なカバー範囲は，システム要件から設計までです．基本はプロジェクト管理や構成管理，品質保証などを省く範囲となっています．提唱者であるHassan Gomaa博士の書籍の中では，ほかにも開発プロセスのライフサイクルや他の手法との比較などに言及していますが，DARTSシリーズがカバーする範囲は，ソフトウェア開発のシステム要求分析から設計までとなっています．最上流の作業となる要件獲得・要件管理をはじめ，ソフトウェア開発にはほかにも必要な作業が存在しますが，それはDARTS利用者が適切な方法や手法を用いて補うことが必要です．

また，昨今注目されているような「プロセス」や「ワークフロー」の解説はありません．DARTSを解説している書籍の中で，「ウォータフォール」型，「インクリメンタル」型などの紹介はありますが，これはDARTSの利用者が自分たちのプロジェクトに合わせてDARTSを適用するようになっています．

■DARTSを理解すると何が得られるか

DARTSは採用実績もかなりあるくらいですから，昨日今日できた最新の手法・方法論ではありません．Gomaa博士によりオブジェクト指向を取り入れた，DARTS以降の手法も提唱されています．昨今の新しいキーワードやトレンドから見るとやや古い手法なのでは？という疑問がある方もいるかもしれません．しかし，今現在の手法・方法論を見渡しても，リアルタイム性を考慮した並行性の分析・設計やモジュール化の作業ステップは，DARTSが一番詳しく思えます．また，実際のところ，今現在のオブジェクト指向やコンポーネントベース開発を利用した分析・設計作業にも役に立ちます．

DARTSは，ワードメラー法，ハトレー法などの構造化分析の考え方の資産を活かし，オブジェクト指向的はモジュールの考えを加えた，実時間システムや制御システム向きにタスク概念を主体にしている点が大きな特徴の手法です．特にDARTSを有名にしている点は，タスク抽出，タスク間通信，タスクの優先度付与とモジュールの切り出しに対する設計に対する基準が明記されている点です．昨今のUMLによるオブジェクト指向開発が主流の方法論とは違い，時代的には以前の手法ですが，今現在でも十分役に立つ手法と言えるでしょう．

なぜなら，昨今の多くの手法・方法論がソフトウエアアーキテクチャを検討する際には，オブジェクト指向による視点のみが強調されたアーキテクチャ設計の手法が多く，リアルタイム性やタスク設計への配慮がまったく，あるいは非常に軽視されたものが目立つからです．いったいいつ，どのように仕様として要求されているリアルタイム性を実現するのかまったく示していない手法・方法や根拠の弱いアプローチが多い印象さえ持ちます．

加えて，DARTSは他の多くの手法・方法論に影響を与えていますが，逆にDARTS自身も他の多くの手法・方法論およびそれまでのソフトウェア工学の成果が盛り込まれています．データフローダイアグラムや

コンテキストダイアグラムは，ハトレー法およびワードメラー法からさらにDARTSへのステップとして引用しています．以上からDARTSを理解しておくことは大きな意味を持つと私は信じています．

DARTSの作業手順

ここからは具体的にDARTSの作業手順を解説していきましょう．誌面の都合でDARTSのアプローチをすべて掲載することは無理ですので，要求定義以降の分析・設計に的を絞って紹介していくことにします．なお，説明のためにいくつかのダイアグラムを用いますが，これはDARTSの文献の図の表記を用います．

図3■DARTSの作業ステップの主要部分

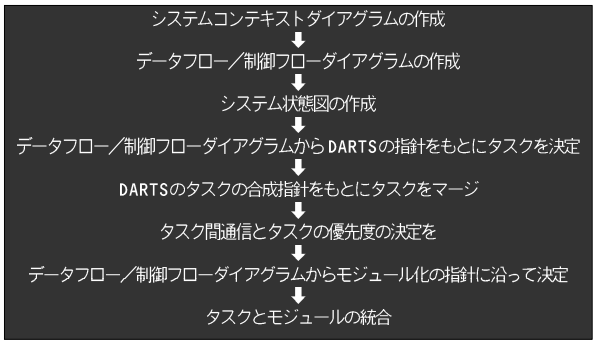
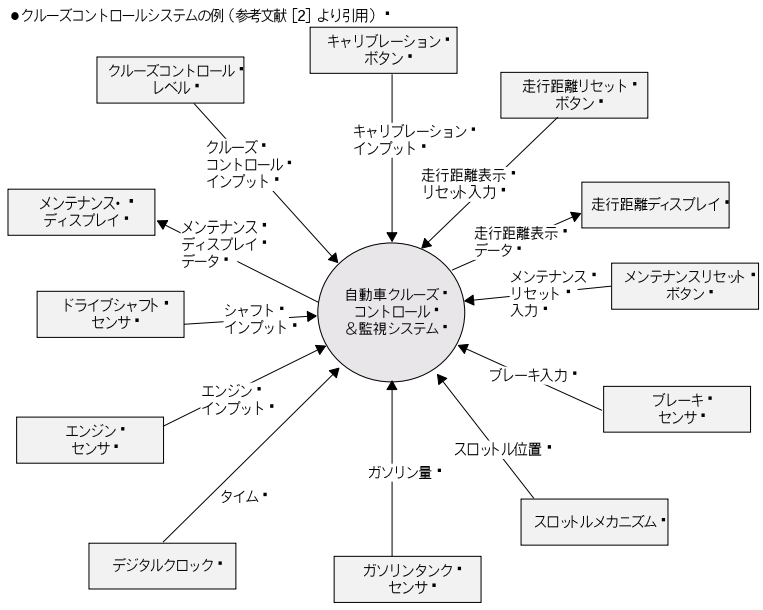


図4■システムコンテキストダイアグラム



またダイアグラムに出てくる例は「グルーズコントロール」と「エレベーターコントロール」のシステムの例ですが，どちらも単純かつわかりやすい例ですので，要求仕様は改めて説明しません．説明を補助するサンプルとしてご覧ください．図3にDARTSの作業手順の概要を示します．

■システムコンテキストダイアグラムとデータフロー／制御フローダイアグラム

ソフトウェア開発では，漠然としている顧客要求，あるいは潜在的・暗黙的といった隠れた要求が，システム開発の出発点になります．このとき顧客の要求をSEなどのエンジニアが，システム実現に向けて専門的な視点から支援し，精緻な要求仕様書を作成していきます．

しかし，大規模なシステムになれば要求を取りまとめ，システムへの要件，制約を矛盾，欠落なく仕様書にまとめるのは困難になっていきます．しかも，顧客要求は変化することが普通です．要求が変化してもソフトウェアに大きな修正をかけないように設計することが，顧客の要求として明示的にも暗黙的にも課されると言っても良いでしょう．ここでは，顧客要件の深い理解，システムが実現しなければならないシステム要件を明確にするために，下記の作業を行っていきます．

■システムコンテキストダイアグラム

DARTSは，対象とするシステムの範囲をシステムコンテキストダイアグラム（図4．System Context Diagram）によって作成し，システム開発のスコープを決定するところから始まります．システム開発のスコープを決定し，システムがハードウェアや他のソフトウェアサブシステムとの関係を明確するためです．

システムコンテキストダイアグラムを作成するための入力情報は，顧客の言葉で表現された「（顧客）要求仕様書」はもちろん，「ハードウェア構成図」「ブロック図」などこれから開発するシステ

ムが，どのような処理を要求されているか，外部とどのような関連を持っているかを理解できるものならすべて情報になります．

■データフロー／制御フローダイアグラム

次に「システムコンテキストダイアグラム」をもとに，ハードウェアや他のソフトウェアサブシステムからどのようなイベントおよびコミュニケーションを行うかを検討し，「データフロー／制御フローダイアグラム」を作成します．ここで，DARTSには明確に作業として記載されていませんが，「システムコンテキストダイアグラム」を参考に，外部装置とどのようなイベントの入出力が実施されるかを明確にするために「イベント分析」を実施します．この作業の成果物は「イベント仕様書」になります．イベント仕様書には，システムが外部から刺激を受けるイベントやハードウェアや他のソフトウェアサブシステムとデータの送受信のコミュニケーションを明確に分析し，記載します．

「データフロー／制御フローダイアグラム」を作成する際は，「イベント仕様書」の各イベントがシステムに入力されたら，どのような処理を行うのかを見極めて作成していくことになります．データフロー／制御フローダイアグラムの作成は，システムの外部からのイベントに対し，システムの振る舞い（処理）を検討していくことになりますから，イベント仕様書を作成するときに，できる限り正確に分析しておく必要があります．

データフロー／制御フローは抽象度の大きいものから描き，必要に応じて階層的に抽象度の低いデータフロー／制御フローへと展開させていきます．十分に単純な処理まで分解できれば終了です（図5）．

システムに課されることは，すべてが要求仕様書に明示的に記載されているとは限りませんから，暗黙的な要求や制約は，コンテキストダイアグラム，データフローダイアグラムを作成しながら明確にしてくことが必要となります．

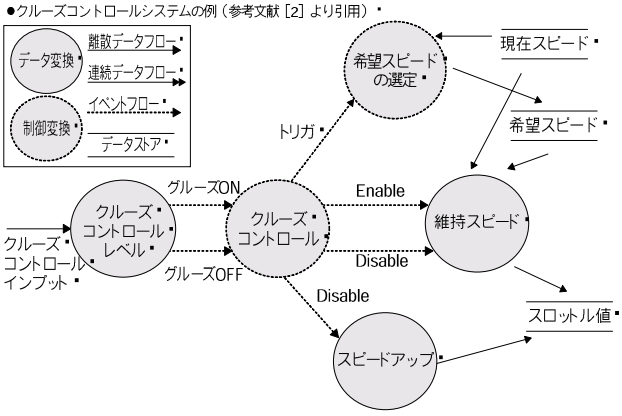
データフロー／制御フローダイアグラムを作成するポイントは，システムが最初から利用できるデータは何か，およびシステムにより何が生成されなければならないかを，システム設計者が各プログラムの外部仕様でその入出力を詳細に記述することと並行して検討していくことになります．なお，データフロー／制御フローダイアグラムは，イベントや処理内容など別に何枚も作成することになります．

また，この後の作業でプログラムのタスクの決定，モジュール構造（UMLでいうコンポーネント）は，データフロー／制御フローダイアグラムから決定していくので，データフロー／制御フローダイアグラムを作成する作業は大変重要です．システムを変換処理，データおよびデータフローに分割することは，システム設計の第1ステップです．この分割は，本質的には，システムの特徴を把握し，抽象的な構造を認識し，そのアーキテクチャ構造を明確にしていく作業となります．なお，データフロー／制御フローダイアグラムを作成するうえで簡単なガイドがありますので紹介します．

「動詞（～するなどの語）は一般に変換処理を示唆し，名詞はデータを示唆する」

なお，DARTSでは，システムコンテキストダイアグラム，データフロー／制御フローダイアグラムにハトレー法およびワードメラー法からアプローチを採用しており，記法も準拠しています．現在はUMLによるモデリングが主流ですので，アクティビティ図やコ

図5■データフロー／制御フローダイアグラム



ミュニケーション図などを用いて表現することもできるでしょう．

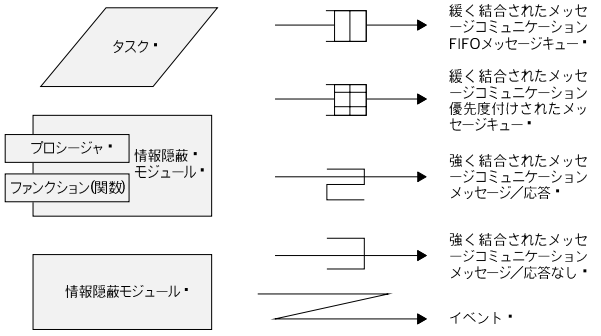
タスクアーキテクチャダイアグラムの作成

DARTSが最もその特徴を表しているアプローチが以下の作業です．ここまでのシステムコンテキストダイアグラムやデータフロー／制御フローダイアグラムおよびイベント仕様書から，要求されている実時間制約を検討し，タスク形成指針に従ってシステム内のタスクを決定していく作業に入ります．

作業は大別すると次のステップで進めます．「タスクの抽出」，「タスクの合成」，「タスクの優先度付与」，「タスクのスケジューリング検討」です．これらのステップを実施して最終的にタスクとタスク間通信，スケジューリングを決定します．

図6に「タスクアーキテクチャダイアグラム」の表記を紹介しておきます．ここで，情報隠蔽モジュールには2つのアイコンが存在しますが，ただの四角（左下）はまだモジュールが詳細になっていないときに用いるアイコンです．

図6■タスクアーキテクチャダイアグラムの表記（参考文献[2]より引用）



■タスクの抽出

DARTSは作成したデータフロー／制御フローダイアグラムから，処理とデータの入出力の特徴に応じてタスク決定していくのですが，このときのガイドラインが与えられています（表3）．そのガイドラインに応じてタスクを決定していくことになります．なお，タスク化を検討する際にタスク構成を記述したダイアグラムを作成しますが，これを「タスクアーキテクチャダイアグラム」と呼びます．

次ページにガイドラインの中からいくつかイメージを掴むために図を用意しました（図7〜9）．なお，データフロー／制御フローダイアグラムの「データストア」は「タスクアーキテクチャダイアグラム」になると「情報隠蔽モジュール」として表現されることになります．

■タスクの合成

次はタスクの合成について検討します．この作業は必ず実施しなければならないというものではないですが，タスク数が多数存在しタスクの数を減らしたいとき，処理のパフォーマンスの最適化が必要なときに実施することになります．

表3■DARTSのタスク抽出のガイドライン

タスク抽出のカテゴリ	タスク抽出の視点	説明
入出力装置に注目したタスク化	非同期な入出力（I/O）装置とのやり取りを行うイベントを制御する処理	入出力装置は処理時間がかかり，処理速度や優先度が異なることが多いため
	イベント入力の時間間隔に周期性がある処理	並行性を制御するうえで扱いやすい．タスクのデッドラインを満たすスケジューリングを計算しやすい
内部処理に注目したタスク化	周期的な処理	周期的に処理を必要とするか，タイマーイベントで起動される処理をタスク化する
	非周期的な処理	処理は必要に応じてオンデマンドでイベントが到着したときに処理する性質がある．処理後は次のイベントが来るまでウエイトする
	制御タスク	システムやある処理群を制御する処理．典型的にはシステムやサブシステムの状態を持ち，状態に応じた制御を行う
	ユーザロールタスク	ユーザがシステムに関わる場合の処理．これは，UNIXやVAX/VMSのようなマルチユーザ／マルチタスクシステム環境を用いたシステムで用いる
	同じタイプのオブジェクトを制御するタスク（同じタイプの多数のタスク化）	エレベーターを制御するシステムを例にすると，各エレベーターは「位置，速度，ドアの開閉」など異なる状態を持つ．このように同じタイプのオブジェクトを制御するには，それぞれのタスクを割り当てる

タスク抽出のガイドラインに従ってタスクを抽出すると，何十あるいは何百といったイベントが発生し得る複雑なシステムでは，タスクが多くなりすぎうまくいかなくなります．また，タスク時間（コンテキストスイッチ）が応答タイミングに対して大きく関係してくるシステムでも問題が出てきます．そこで，タスクの合成を検討します（表4）．タスクを合成した場合

図7■入出力装置に注目したタスク化

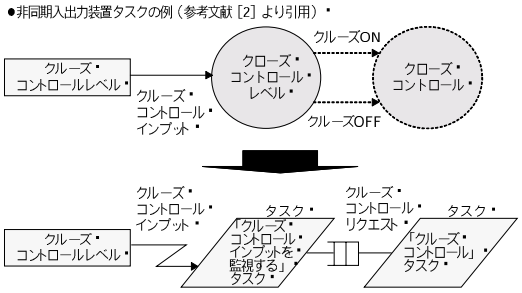


図8■内部処理に注目したタスク化1

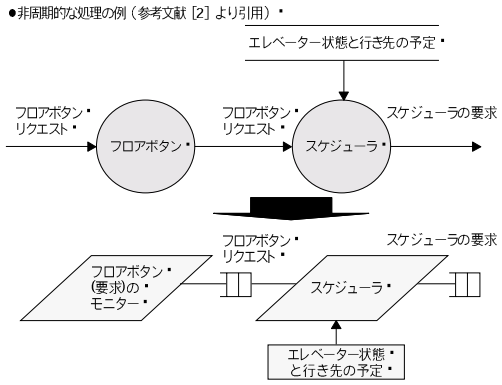


図9■内部処理に注目したタスク化2

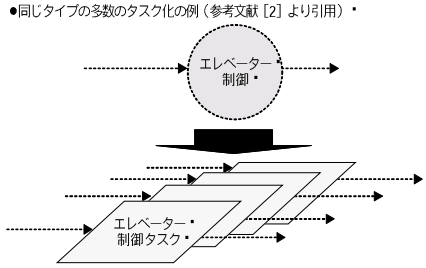


図10■逐次処理に着目した視点

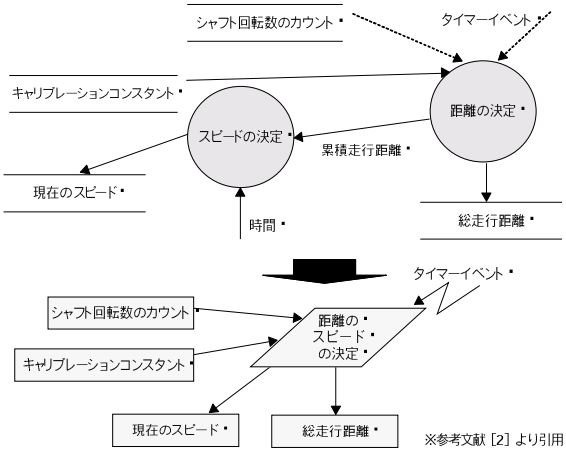


図11■制御処理に着目した視点

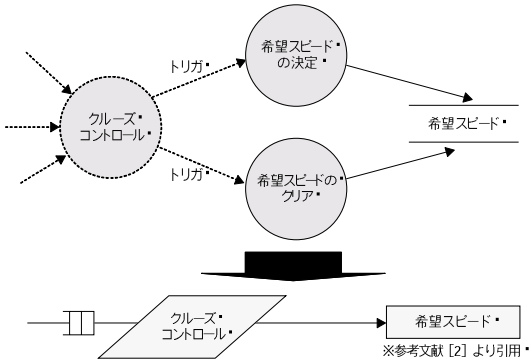


表4■DARTSのタスク合成のガイドライン

タスク合成の視点	説明
時間に着目した視点	イベントの時間間隔や受動的な装置をモニターする場合，同じ時間間隔で始動されるタスクをグループ化．つまり時間的強度があると見受けられるタスク群をまとめる．設計制約上タスクを合成し最適化することが必要な場合
逐次処理に着目した視点	複数のタスクが処理を順次行う逐次性が明確な場合，タスクの処理順序が一定であるのでタスクを1つにまとめることを検討する．グループ化は，逐次的処理のタスク群の中で，データを書き込むタスクがあったらそこまでグループ化する
制御処理に着目した視点	制御変換とその状態遷移によって直接始動されるデータ変換をグループ化した制御の関連性がある場合の基準
密接に関連した機能に着目した視点	機能に強い関連がある場合は，タスク群をグループ化した機能的依存性に着目する．これは自明と言える基準
タスクのオーバーヘッドに着目した視点	タスク数が多数存在する，またはガイドラインに従ってタスク分割をしたが，タスクのオーバーヘッドを削減するために，あえて独立したタスクとしなくても済むものや，優先度の不要やスケジューリングを簡単にするために最適化としてタスク数を減らす

は，タスクアーキテクチャダイアグラムを更新します（図10，11）．

■タスクの優先度

DARTSはタスク優先度については，実のところそれほど詳細なことをガイドラインとしては提供していません．組込みリアルタイムシステムの開発者なら，どれも一般的であると考えられるものです（表5）．また，相対的なタスクの優先度のガイドラインであり，どれを一番高いタスクにすべきか，低くすべきかの基準はガイドラインにはありません．もっとも，タスクの設計はシステムの経験が大きく影響を持つところであり，ガイドラインに限界があるのは当然でしょう．

■タスク間コミュニケーション（通信）

タスク間コミュニケーションは，タスク間通信と呼ばれることもあります．タスク間コミュニケーションは，タスク間の情報のやり取りを行うために発生します．タスク間コミュニケーションが必要となるのは，各タスクが独立していないからで，外部イベントの処理を複数のタスクで実現するためです．つまりタスク間には依存関係が存在します．

タスク間コミュニケーションは，タスク間でやり取りする情報と，タスク間の同期タイミング，タスクの時間制約などを総合的に考慮したうえで決定しなければなりません．デリケートさが必要とされ，決して単純な作業ではありません．

たとえば，タスクがあるデータを共有する場合，一方のタスクは他方が完了するまでブロックされることがあります．特に書き込みの場合は排他制御が必要となります．タスクは優先度に基づいて実行順序が決まってきますから，タスクの実行時間の分析を行った結果，ある優先度を待つタスクが，必要なリソースを所有する優先度の低いタスクによりブロックされる時間もあり得ます．本来，共有のリソースを優先度の低いタスクがアクセスする優先権を得て，高い優先度のタスクが待たされるのは好ましくないタスク間通信の設計なのですが，必ずしも避けることができない場合もあります．タスクのスケジューリングなども考慮に入れなければなりません．

タスク間コミュニケーションの最終的な判断は事実上，タスクとモジュールを統合し，タスクの優先度，スケジュールを検討しないと決定できません．ですから，タスク間のコミュニケーションはタスクとモジュールを統合する「ソフトウエアアーキテクチャダイアグラム」を作成する時点でも間に合います．そのときに，どのような種類のタスク間コミュニケーションを用いるかを検討します．

タスク間通信の呼び方はいくつかあり，代表的なものを表6に示します（表6の分類がすべてではありません）．また，章末の参考文献［4］，［5］を参考にしてみてください．タスク間コミュニケーションと設計方法およびノーテーションが詳細に解説されています．

■タスク関連作業のまとめ

タスクの抽出からグループ化，タスク間コミュニケ

表5■タスク優先度のガイドライン

No.	ガイドライン
1	時間制約が厳しいタスクには高い優先度を付与．たとえばI/Oハンドラーのタスク
2	周期処理のタスクのうち，周期間隔が短いタスクは高優先度を付与
3	時間がかかる処理のタスクには，低い優先度を付与

表6■タスク間通信の種類（代表的なもの）

通信の種類	説明
同期呼び出し	呼び出し側のタスクAが，呼び出されたタスクBの処理の結果を利用するまでAの処理が待たされ，Bの処理が完了し結果を受け取るとAは処理を再開する．実装には関数呼び出し，共有メモリのデータのやり取りなどいくつかの方法がある．これは「強い結合されたコミュニケーション」に分類される
非同期呼び出し	呼び出し側のタスクAは，タスクBの処理結果を待つ必要がない，あるいは，すぐに回答を使わない場合などに使う．実装にはOSの提供する通信機能を使うことが多く，パイプ，メッセージボックスなどがある．ほかにも大域変数，共有メモリを利用することもある．これは「緩く結合されたコミュニケーション」に分類される
ポーキング型同期呼び出し	受信側タスクが呼び出し側のタスクAからのメッセージをすぐに受け取れない場合には，呼び出し側Aはそのメッセージの送信を中断する
待機型同期呼び出し	呼び出し側タスクAは受信側タスクBがメッセージを受け取るのを指定された時間だけ待つ．その指定時間が経過すると，呼び出し側Aはメッセージの送信を中断する

ーションまでを実施し，ある程度作業が安定したと思えたら，システム内のタスク構成の初期版ができたことになります．製品開発が進展するにつれて，見落としていたイベントが追加されたり，逆に削除されたりすることがあります．それに応じてタスクの見直しがあるでしょう．こういった場合には，その都度，設計の見直しが必要になってきます．

（情報隠蔽）モジュール

タスクに関する一連の作業の次は，データフロー／制御フローダイアグラムから，モジュール構造を決定する作業に移ります．モジュールはタスクと異なりデータのまとまりやデータをカプセル化する観点からモジュール化を検討します．

タスクがイベントによる処理や時間制約などからの検討による「アーキテクチャの動的構造」に対して，モジュール化は「アーキテクチャの静的構造」に該当します．

■モジュール設計

図6（130ページ）の長方形はUMLで呼ぶサブシステムやコンポーネントで，DARTSではモジュールと呼んでいます．モジュールの粒度はその時々に応じて異なりますが，あまり大きくならないようにすることが基本です．

なお，モジュールは，データフロー／制御フローダイアグラムだけからはすべて見つけることができません．初期化処理および終了処理に関するものは，処理を見直して別途モジュールを検討することになります．また，機能変更やアーキテクチャの可読性，保守性からみてモジュール化することもあります．

同じく大きな長方形は，パッケージモジュールの境界を表し，これらのモジュールは外部から呼ばれますが，外部からデータは隠蔽されていることを表しています．長方形から突き出た細長い長方形は，公開するインタフェースで，このインタフェースを通じてのみ隠蔽されているデータにアクセスが可能となります．

原則としてガイドラインに従うと，優れたモジュール化を可能としますが，複数の変換を1つの一般的なモジュールで実現したり，1つの変換を1つのモジュールで実現したり，複数のモジュールで1つの変換を実現したりといくつか設計的な選択肢があります．開発全般を通じて必要に応じて適宜モジュールの再構成を検討することもあります．

■DARTSのモジュール化のガイドライン

モジュール形成指針に従ってシステム内の情報隠蔽モジュールを決定し，さらに，モジュール階層を決定します．ガイドラインではモジュールの種類が示されており（表7），データフロー／制御フローダイアグラムから検討していきます（図12～14）．

タスクとモジュール統合

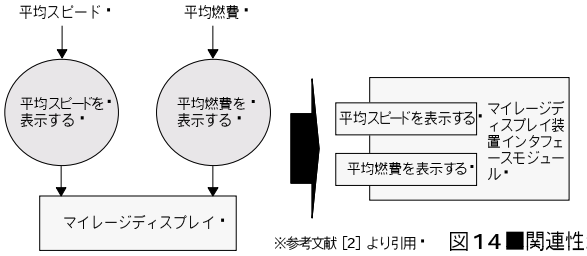
■モジュール／タスク間の関係の基礎

モジュールやタスクを結ぶ矢印はモジュールから他のモジュールの呼び出しを表していることを表現して

表7■DARTSのモジュール化ガイドライン

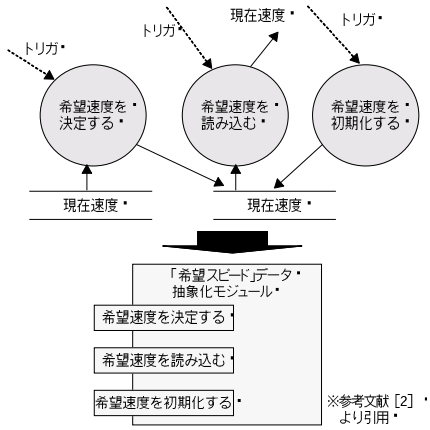
モジュール化の視点	解説
外部装置（DIM：Device Interface Module）	外部（I/O）装置と詳細なやり取りを隠蔽して抽象化し，外部（I/O）装置へのアクセスを公開インタフェースとするモジュール
データとデータへの読み書き操作（DAM：Data Abstraction Module）	データを抽象化し，データへの読み書き操作を公開インタフェースとするモジュール
振る舞いを隠蔽した状態遷移（STM：State Transition Module）	状態遷移の「振る舞い」を隠蔽し，状態遷移を起こすトリガ用の公開インタフェースとなるモジュール
関連性が密な外部装置のグループの振る舞い（FDM：Function Driver Module）	関連性が密な外部装置のグループの振る舞いを1つに抽象化し，アクセス用の公開インタフェースとなるモジュール
アルゴリズム（AHM：Algorithm Hiding Module）	アルゴリズムを隠蔽し抽象化し，アクセス用の公開インタフェースとなるモジュール
ソフトウェア（の戦略的）決定モジュール	ソフトウェア（の戦略的）決定モジュールは，データフロー／制御フローから見つけるのではなく，ソフトウェアの変更可能性，保守性のために設計者がモジュール化したほうが良いと判断してモジュール化する．通常設計の後半にいろいろなことを考慮して決定する

図12■デバイスインタフェースのモジュール化の例



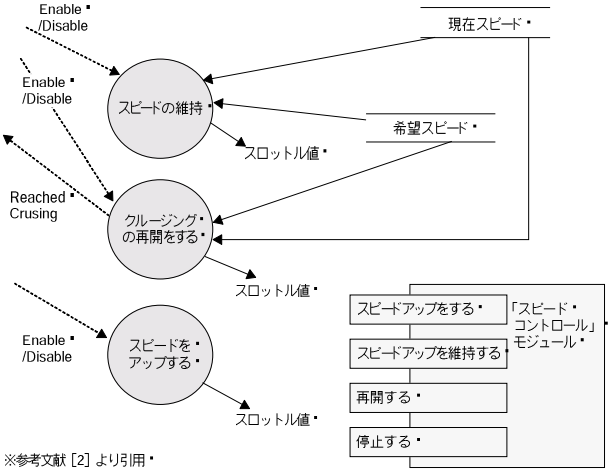
※参考文献 [2] より引用

図13■データ抽象化モジュール化の例



※参考文献 [2] より引用

図14■関連性が密な外部装置グループの振る舞いの例

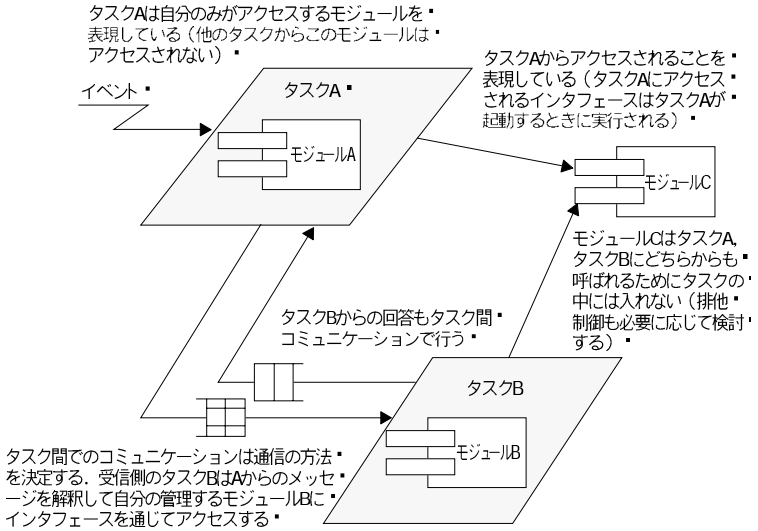


※参考文献 [2] より引用

表8■ソフトウェアアーキテクチャダイアグラム作成のガイドライン

No.	説明
1	時間的なトリガで起動するタスクは，そのタスクだけにアクセスさせ，起動されるモジュールをそのタスクの内部に包含し，タスク自身はイベントによって起動する
2	制御タスクは，状態遷移モジュール（STM）をそのタスクの内部に包含する
3	データ抽象モジュールは，制御タスクによって起動される操作に対応して，モジュールの公開するインタフェースを決定する

図15■タスクとモジュール統合の例（ソフトウェアアーキテクチャダイアグラム）



います．

モジュール間，タスク間あるいはモジュールとタスクの結合は，ソフトウェアの保守やテストの効率に大きく影響を与えます．モジュールダイアグラムでは，矢印がこれら関連を表し，モジュールAがモジュールBに大きく依存（いくつものインタフェースを呼び，データを参照しているなど）しているとき，AはBと密結合（tightly coupled）であると呼ばれます．AからBに依存が単純であるとき，AはBと疎結合（1oosely coupled）であると呼ばれます．ソフトウェア設計の世界では，疎結合は密結合よりも望ましいのです．結合は依存があるということですので，依存されている側を変更すると，依存している側にも変更がおよぶなどの問題があり，ソフトウェアの可読性が低下します．

どんなにうまく設計しても，依存をなくすことを避けられないこともあります．しかし，モジュールの結合が密になるほど，モジュールの実現，試験および変更が難しくなるので，保守性などのために依存をいろいろ工夫する必要があります．

1つのモジュールが別のモジュールを呼び出しているとき，疎な形の結合が発生しますが，呼出しに付随するデータフローは，結合を少し強めます．なぜならば，両方のモジュールのデータフローの型および方向が一致しなければならないからです．

■タスクとモジュール統合のガイドライン

DARTSはタスクとモジュールを別々にデータフロー／制御フローダイアグラムから検討したあとは，2つを統合した「ソフトウェアアーキテクチャダイアグラム」を作成します．タスクとモジュールの視点を統合化し全体のソフトウェア構造を決定することになります．ガイドラインを表8に示します．非同期の入出力装置のタスクは，対応する装置の抽象化モジュールを包含し，タスク自身は割り込みによって起動されることが基本です（図15）．

まとめ

DARTSの紹介は以上です．より詳しい解説は参考文献を参照してください．

次の第3章では，より包括的なプロセスである組込みリアルタイムシステム用のオブジェクト指向開発プロセスを紹介していきます．

■参考文献

[1]E.Yourdon，Modern Strudured Analysis，Yourdon Press，Englewood Cliffs，N.J（1989）

[2]『Software Design Methods for Concurrent and Real-Time Systems (SEI Series in Software Engineering)』Hassan Gomaa 著，Addison-Wesley Professional，ISBN：0201525771，1993/7/31

[3]『Designing Software Product Lines With UML: From Use Cases to Pattern-Based Software Architectures (Addison-Wesley Object Technology Series)』Hassan Gomaa，Maryann Barber 著，Addison-Wesley Professional，ISBN：0201775956，2004/7/9

[4]『Use Case Maps for Object-Oriented Systems』R.J.A. Buhr, R.S. Casselman 著，Prentice Hall College，ISBN：0134565428，1995/11

[5]『Introduction to Real-Time Systems: From Design to Networking with C/C++』R.J.A. Buhr，Donald L. Bailey 著，Prentice Hall，ISBN：0136060706，1998/8/21

[6]『ADA—言語とプログラミング方法論（ワークステーションシリーズ）』David A. Watt，William Findlay，Brian A. Wichmann 著，藤田 昭平，細谷 僚一 監訳，丸善，ISBN：4621034529，1990/4

[7]『ソフトウェア設計—オブジェクト指向からエージェント指向へ（システム制御情報ライブラリー）』松本 吉弘，大槻 繁 著，システム制御情報学 編集，朝倉書店，ISBN：425420972X，1995/3

第3章

統合的な開発アプローチが求められる時代

組込みリアルタイムシステム方法論「HARMONY」

はじめに

欧米では航空宇宙産業や防衛産業が盛んなために，組込みシステムやリアルタイムシステムの開発プロセス，方法論がかなり存在します（表1）．

本章では組込みシステムやリアルタイムシステムの開発方法論「Harmony」を取り上げます．誌面の都合上，詳細にはご紹介できませんがポイントを解説していきたいと思います．

リアルタイムシステムのオブジェクト指向開発方法論「Harmony」

昨今はオブェクト指向開発が主流です．理由は明確で，オブジェクト指向技術によるシステムの保守性・拡張性の向上が期待できるからです．オブジェクト指向の現場利用の歴史を見ても，オブジェクト指向の分野で著名なBrady Booch氏が，軍事分野のリアルタイムシステム開発でAdaによる開発，オブジェクト指向の研究等で多くの実績を上げています．著書も数多く執筆しています．その後，Booch氏の研究は方法論として体系化されBooch法[注1]として発展しています．

組込みリアルタイムシステムも大規模化・機能の複雑化が目覚しい一方で，開発期間は年々短くなっています．しかし，組込みリアルタイムシステムは，一度開発したら，莫大な人的資源を投入して開発された大規模なプログラムは長期間にわたって使用しなければなりません．このためには，使用環境の変化ある

表1■組込みリアルタイムシステム向けの手法・方法論

手法・方法論	特徴
ジャクソン法（JDS：JacksonSystemDevelopment）	組込みリアルタイムのシステムに向けた手法を展開している．事例も組込みリアルタイムシステムが多い（参考文献[11]）．EUでは過去に開発にJDSが主流となっていたこともあり，オブジェクト指向の考えを強化した派生方法論が数多く存在する
DARTS，ADARTS，CODART，PLUS	PLUSはプロダクトラインエンジニアリングの実現を，UMLによるコンポーネントベース開発で解説している（参考文献[14]）．DARTSなどの提唱者のHassan Gomaa博士による方法論．なお，DARTS，ADARTSなどについては第2章を参照（参考文献[6]）
Harmony	本章を参照（参考文献[1]，[2]）
ROOM（現在はIBM-RationalのRational-Unified Processに取り込まれている）	参考文献[10]を参照してください
HOOD（HierarchicalObjed-OrientedDesign）	HRT-HOOD（階層的オブジェクト指向設計）は，初期はAda言語を対象として1986～87年において欧州宇宙機関（ESA）によって使われている．現在は他の言語への展開も考慮されている．航空宇宙，防衛産業で利用が多い（参考文献[7]）
Octopus	携帯電話で有名なノキア社のオブジェクト指向開発方法論．オブジェクト指向で検討してアーキテクチャの静的構造とマルチタスクの動的構造の統合についてのアプローチが興味深い（参考文献[5]）．また，Octopusによる開発環境Rhapsodyを用いた際の評価・考察についてのホワイトペーパーも存在する（参考文献[15]）
ExecutableUML	オブジェクト指向開発方法論としては，Booch法，コード・ヨードン法，OMTなどとともに歴史のあるシュレイアー＆メラー法に，MDAのアプローチを導入したUMLによるリアルタイム向けオブジェクト指向方法論（参考文献[12]）

※参考文献は，章末（146ページ）を参照してください．

注1）初期はAda言語をターゲットとし，設計を中心に置いたオブジェクト指向の開発手法から，言語非依存の開発全般における方法論として発展しています（参考文献[8]，[9]）．

統合的な開発アプローチが求められる時代
組込みリアルタイムシステム方法論「HARMONY」

第3章

いは使用目的の変更等に対処できる柔軟性が必要です．また，通常はシステムの保守・変更作業は，このプログラムの最初の開発者とは異なる人々によりメンテナンスが行われる場合がほとんどです．

加えて，プロジェクトが巨大になるにつれ，システム分析，要求定義，システム設計の重要性が増してくることは理解できると思います．実際，開発プロジェクトでは，純粋な技術のために消費される時間より，作業の手順，開発者の意思疎通，ドキュメントなどの問題に労力のほとんどを占めることも珍しくありません．つまり，ソフトウェア設計にあたっては，保守しやすいように多大な注意を払わない限り，保守には非常にコストがかかってしまうのです．このことから，優れたプログラム設計の主要な基準として，正当性とともに変更の容易性を重視するオブジェクト指向開発が主流になっていくのは自明でしょう．

本章で取り上げるのは組込みリアルタイムシステムの開発方法論「Harmony」です．米国で航空宇宙，軍事防衛，医療分野などで実績があります．

組込みリアルタイムシステムは，機能的・非機能的の両方で求められる内容が技術的に高度です．そのため，組込みリアルタイムシステムの開発方法論は，求められている要求を分析・設計を通じてどのように作業するかが比較的詳細に述べられているのが特徴です．ビジネス系の開発手法や方法論でももちろん詳しく書かれているものがありますが，開発全体を巨視的にプロセスとしてまとめたものが多い印象を得ます．

組込みリアルタイムシステムの開発手法や方法論は，ビジネス系に比べリアルタイム性，並列・並行性などに伴うタスク抽出，タスク間通信，排他制御，タスク優先度づけ，タスクスケジューリングなど緻密な設計を必要とするため技術的なノウハウがプロセスとして体系化されているものが多くなっています．

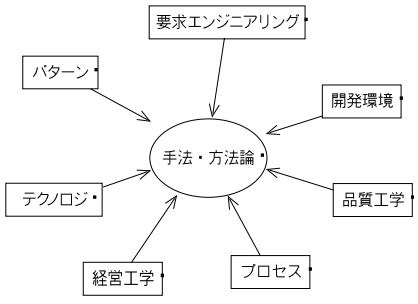
統合的な開発アプローチが求められる時代

「Harmony」も含め，代表的な組込みリアルタイムシステムソフトウェア開発の手法や方法論の多くは，オブジェクト指向技術やコンポーネントベース開発などを前提としているものがほとんどです．組込みリアルタイムシステムソフトウェアの分野でも，ソフトウェアの効率的な開発，ソフトウェアアーキテクチャの優れた保守性・拡張性を実現するためには，オブジェクト指向のアプローチが重要です．もっとも，最近の手法・方法論などは日々新しい技術や考え方を取り入れていますので，単純にオブジェクト指向技術の手法・方法論と呼ぶのは不適切でしょう．ソフトウェア開発のライフサイクル全般をカバーし，要求獲得からテストまで扱うために，いろいろなアプローチを効果的に取り入れています．たとえば，要求定義／管理ではユースケースによるアプローチ，ソフトウェアアーキテクチャの優れた保守性・拡張性を実現するために，上流の分析やモデリングとして再利用可能なソフトウェア部品を開発するドメインエンジニアリング，設計ではパターンなどの利用です（**図1**）．

さらには，開発規模の増加，市場ニーズの多様化による製品ラインナップの増加，短縮化する開発期間など，難しい要求を確実に実現するために，上流から必要な作業を明確し，効果的に進めて品質を作り込んでいくなど，さまざまなことを検討しなければなりません．昨今は，従来に比べてビジネス系，組込みリアルタイム系にかかわらず手法・方法論およびプロセスも上流が最重要視されています．

たとえば，最近注目を集めいているプロダクトラインエンジニアリング[注2]は，組込みソフトウェアにオブジェクト指向開発を活用し

図1■現代の手法・方法論が扱う内容



注2）プロダクトラインエンジニアリングは，プロダクト開発を機能に共通性のある製品間でソフトウェア集約的なアーキテクチャ開発を行い組織の資産とします．端的に言えば，組織で一元管理されたソフトウェア資産を組織で共有して開発するしくみです．そのためには，事前に明示的かつ戦略的に，共通で核となる再利用するアーキテクチャやソフトウェアコンポーネントなどの一連の資産（アセット）の構築と利用のインフラが重要になります．多くの場合，ドメインエンジニアリング，オブジェクト指向技術，機能指向分析などの手法によって再利用の技術的しくみが実現されていきます．プロダクトラインエンジニアリングを実施するには，従来の開発から大きく組織のしくみを再編成することもありうるために，並行してSEIのCMMIなどを用いたプロセス改善活動も行うことがあります．

てソフトウェア資産の再利用を促進する取り組みですが，市場調査，ビジネス戦略から始まり，要求獲得など，非常に上流の活動が多くなっています．一方，テクノロジ的には同じく最近注目され，実際に活用されつつあるMDA[注3]も，多くの事例は組込みシステム開発やリアルタイムシステム開発であることを考えると，今後ますます手法・方法論，およびプロセスの価値と需要は高くなるでしょう．

しかし，その一方で組込みリアルタイムシステムソフトウェア分野のROM・RAMの制約，リアルタイム性の実現，並列・並行性の問題は，オブジェクト指向のアプローチとは，単純には馴染まない技術的課題が存在するのも事実です[注4]．オブジェクト指向技術を用いて保守性・拡張性を実現し，さらにはソフトウェア部品として再利用可能なコンポーネント，ソフトウェアアーキテクチャを実現するには，オブジェクト指向技術を利用しながらも，組込みリアルタイムシステムの多くの課題をどう実現するかが開発手法や方法論の大きな課題です．

Harmonyのプロセスの概要

RopesからHarmonyへと進化

「Harmony」とは，米国I-Logix社[注5]が提唱している組込みリアルタイムシステム用の開発方法論です．モデリング言語であるUMLを組込みリアルタイムシステムへどのように活用し，効果的なソフトウェアの分析・設計の開発アプローチを行うかを展開しています．

実はこのHarmonyは，以前まで「Ropes」と呼ばれてた方法論を発展させたものです．Ropesはユースケースを用いて要求定義から詳細設計までの範囲を，UMLを利用した反復開発のアプローチで体系的にまとめていました．『Real Time UML』（参考文献[3]），『DoingHardTime』（参考文献[4]）などの書籍では，ユースケースの抽出からアーキテクチャの構築，そして詳細設計までの反復型開発を，ケーススタディを用いて解説しています．

Harmonyは，上流のシステムエンジニアリングが追加・強調され，従来のRopesをHarmony/SWとして踏襲し，そのまま包含した形です．新たに定義・追加された上流のシステムエンジニアリングは「Harmony/SE」として呼び分けていますが，方法論としては統合化されています．

Harmonyの特徴と他の手法・方法論との比較

まずは，Harmonyの概要を理解するところから始めたいと思います．Harmony全般の特徴を表2にまとめてみました．簡単に特徴を見ていくことにします．

組込みリアルタイムの手法・方法論・開発プロセスと呼ばれるものは，非常にユニークなものが多いのが特徴です．表1で紹介したROOM（現在はIBM-RationalのRational-Unified Processに取り込まれています），欧州の航空宇宙・防衛産業で使われたHOOD，MDAの先駆的な開発で知られるExecutableUMLは，あらかじめ対象としている製品，採用する特定のテクノロジが存在していると考えてよいでしょう．たとえば，ROOM，HOODはともに航空宇宙，軍事防衛分野にどちらかというと向いています．方法論の持つ特徴を最大限に活かせるからです．そのために，基本的に8ビットなどのマイコンで動作させる小さなシステムでは不向きと言えるでしょう．ExecutableUMLはMDAをアプローチのコアにしていますから，MDAを開発の戦略に組み入れないプロジェクトでは無理があります．

各手法・方法論がカバーする作業のスコープ，プロセスおよび重視しているテクノロジなど，各手法・方法論の哲学・コンセプトなどの違いにより，それぞれ独特の手法・方法論を作り上げています．理由は組込みリアルタイムでは，開発対象の製品の多様性から万能的な手法・方法論よりも，特定のテーマや問題解決のアプローチを明確にし，可能な限り具体的にし

注3）MDA（Model Driven Architecture）は，UMLなどによるモデルを中心として，プラットフォームに依存しないPIM（Platform Independent Model），依存するPSM（Platform Specific Model）を意識して，ソースコード生成まで実現する開発方法です．多くの研究者，ベンダが盛んにMDAのアプローチを提唱しており，サポートするツールも近年積極的に現場に導入され始めています．
注4）オブジェクト指向採用の技術的課題については，「補足」の部分で説明します（144ページ）．
注5）現在はTelelogicの買収によりTelelogic AB, System and Software Modeling Business Unitとして活動．

たほうが，効果があると考えるからでしょう．あるいは，航空宇宙・防衛産業のように，最初から特定の分野の開発が深刻な状況であったために，Ada言語の開発や，DOD-STD-2167A[注8]などの基準が定められ，最初からその分野をターゲットとした手法・方法論が確立したというのも理由でしょう．確かに汎用的になりすぎて，抽象的なものでは意味がないと考え具体的にするという意味では1つの考えです．

一方，その反面サポートする開発環境を利用しないと事実上適用できない，あるいは手法・方法論が想定しているシステム以外では適用が難しくなるために，手法・方法論側で開発対象の製品や分野を選ぶという欠点が各手法・方法論には同居します．この点を考慮して組込みリアルタイムの手法・方法論・開発プロセスを自組織・プロジェクトに適用することが重要です．

前章で組込みシステムの仕様や開発・運用上の制約は製品ごとにさまざまで，開発手法やプロセスを画一的には扱えないことはすでに話をしました．さらに，それが理由で組込みリアルタイムの手法・方法論・開発プロセスと呼ばれるものは，非常にユニークなものが多いのが特徴であることにつながるでしょう．

さて，それではHarmonyはどうでしょうか？Harmonyは“良い意味”で他の組込みリアルタイムの手法・方法論のように，特別独特な印象を与えません．つまり，特定の製品分野，利用するテクノロジ，あるいは特定の開発環境を指定するという傾向は見られないということです．Harmonyは非常にオーソドックスな開発プロセスである印象を受けます．具体的なことはおいおい説明することとして，Harmonyのプロセスの全体構成を見てみることにしましょう．

Harmonyのライフサイクルとプロセスアーキテクチャ

■マクロサイクル

Harmonyの構成を見ると，大きくシステムエンジニアリングとソフトウェアエンジニアリングから成り

表2■Harmonyの主な特徴

No.	特徴	説明
1	組込みリアルタイムシステムを対象としている	対象分野を特定しているため，組込みリアルタイムシステムの開発者にはプロセスおよび作業成果物などが具体的
2	独自の3階層粒度の反復型開発プロセスを提唱	作業内容，フェーズなどを軸に反復型を3階層の段階に分けて行うことで明確な作業手順と計画立案が可能
3	Harmony/SEとHarmony/SWとシステムエンジニアリングとソフトウェアエンジニアリングから成り立つ	組込みリアルタイムシステムのシステムエンジニアリングとソフトウェアエンジニアリングの作業の進め方や内容から，Harmony/SEとHarmony/SWから構成．Harmony全体は反復型であるが，Harmony/SEはウォータフォール的な作業のアプローチのために「セミスパイラル」と呼ぶ
4	組込みリアルタイムシステムに向けたデザインパターンを多く紹介している	ビジネス系のデザインパターンを効果的に組込みリアルタイムシステムに適用する事例や組込みリアルタイムシステム用によく見られるパターンを積極的に紹介，利用している
5	プロジェクト管理，構成管理，製品戦略立案などはスコープの外にあり，技術的なテーマにフォーカス	プロジェクト管理，製品戦略については軽く記述が見られるが，システムエンジニアリングとソフトウェアエンジニアリングのフォーカスしている
6	従来のオブジェクト指向技術の作業アプローチを踏襲し，同時に組込みリアルタイムシステムの従来のアプローチも踏襲	組込みリアルタイムシステムを対象としているので，リアルタイム分析・設計が詳しく扱われている．イベント分析[注6]からスケジューリング理論やRMA[注7]まで紹介されている
7	Harmonyを支援する開発環境「Rhapsody」が存在する	Harmonyはオブジェクト指向によるMDD（モデル駆動開発）を原則としている．開発をより効果的にするために，Harmonyを支援する開発環境Rhapsody（後述）をI-Logix社が開発・提供している
8	書籍・ホワイトペーパーなどが多く存在している	Harmonyの解説・ケーススタディを詳しく解説した書籍，組込みリアルタイムシステムについて要件開発，技術解説などのホワイトペーパーが充実している

注6）第2章（124ページ）を参照．
注7）Rate Monotonic Analisysの略で，レートモノトニックスケジューリング（RMS：Rate Monotonic Scheduling）に基づいて行うスケジューリング分析．すべてのスレッドの優先度はその周期に反比例で，周期が短ければ短いほど，優先度を高くするという考え．RMSの当初の枠組みでは，デッドラインは周期に等しいと仮定するが，該当しない場合は優先度は周期ではなくデッドラインに基づいて割り当てなければなりません．
注8）DOD-STD-2167Aは，ミッションクリティカルなソフトウェア開発に適用されることを目的とした標準．http://www.software.org/quagmire/descriptions/dod-std-2167a.asp

立ちます．Harmonyでは反復型プロセスを3つの粒度で階層化して構成しています．作業を単純に反復させるのではなく，システマティックかつ効率的に開発をドライブするには，反復にもいくつかの作業の特徴から見た粒度があり，階層化することで明確になると捉えているからです．

図2で示されている作業の流れが最上位の大きな反復型のフローになっており，この最上位が「マクロサイクル」と呼ばれています．このマクロサイクルが反復するのは，システムのアーキテクチャや開発スケジュールに影響を与える要求変更が起きた場合と考えてよいでしょう．

図2には，最上位から見たプロセスの構成，製品を開発するのに必要な各エンジニアリング活動を示します．図2 - ❶にある2つ（「要求分析」と「システム分析&アーキテクチャ分析」）が，システムエンジニアリングの作業（表3）になります．

ここでの作業は，一般的な要求エンジニアリング，システムエンジニアリグが実施されると考えてよいでしょう．また，システムエンジニアリングの中のアーキテクチャ分析作業でレイヤ構成，サブシステム構成，サブシステム間の依存関係やリアルタイム分析作業を検討しておきます．なお，Harmonyでは下流のソフトウェア開発の反復をユースケースドリブンで進めますので，ユースケースにより要求仕様は定義しています．

特徴的なのは図2を見てわかるとおり，システムエンジニアリング（Harmony/SE）の部分とソフトウェアエンジニアリング（Harmony/SW）の部分とでは作業の進め方が異なっています．この理由はHarmonyではシステムエンジニアリングで，システムの分析作業を明確に行うことを重視しており，中途半端な分析では，下流の作業で大きな手戻りや非効率な開発になると考えられているからです．そのため，システムエンジニアリング（Harmony/SE）は，図2が示すとおりウォータフォール的な作業フローになり，反復のフローをとりません．ただし，これは絶対に作業を後戻りしないというわけではありません．

Harmonyのプロセスが重視する他の点として，静的な構造であるシステムのアーキテクチャを構成するサブシステムとサブシステムごとに用意するサブシステムの構成および内容を記述するサブシステム仕様書があります．

サブシステム仕様書には，システムアーキテクチャの中で識別された各サブシステムについて，別途システム全体とは独立したサブシステムの仕様の詳細が記載されていきます．この定義の詳細は反復型開発を進めるうえで重要になってきます．すなわちサブシステムが作動する状況，それが付着するに違いないインタフェース，およびサブシステムに割り当てられた責務が満たす必要条件などが記載されますので，複数のチームで並行して開発するときにも重要ですし，サブシステム間の独立性，堅牢性を評価・実現するうえでも重要です．反復開発を通じてアーキテクチャを評価しながら優れたアーキテクチャを作り上げていきます．

図2■Harmonyプロセスオーバービュー（マクロサイクル）

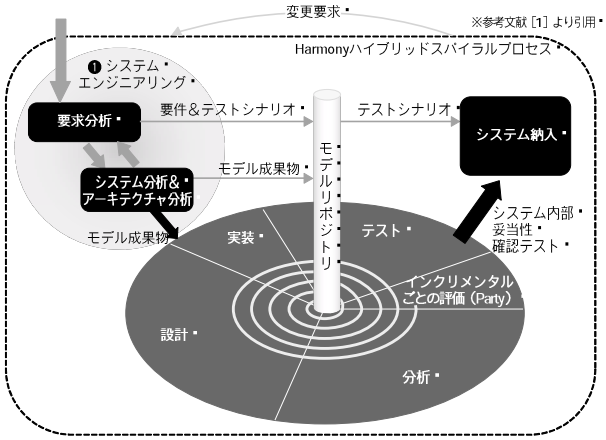


表3■Harmonyのシステムエンジニアリングの主な作業

No.	内容
1	システムが実現するうえで，前提となる必要条件の識別および分析
2	UMLによって表現したシステムの機能的・非機能的な振る舞いの検討と定義
3	システムアーキテクチャの構築，異なるモデルの構築によるハードウェア／ソフトウェアトレードオフの評価，下流のエンジニアリングプロセスの検討と決定など

■マイクロサイクル

Harmonyの中で実際に反復プロセスの要になるのは，この「マイクロサイクル」です．マイクロサイクルの中のフェーズはソフトウェア開発のためのプロセスです．そのため，マイクロサイクル以降のプロセス

をHarmony/SWと呼び分けています．

図3で示すように「プロトタイプ定義」から「反復ごとのレビュー」まで明確なフェーズが定義されています．これをあらかじめ計画したスケジュールで反復していきます．

Harmonyのシステムエンジニアリングのアーキテクチャ分析を経て，マイクロスパイラルでサブシステムの分割が行われれば，各サブシステムソフトウェアの開発チームは，サブシステムの詳細な要件定義から作業を始めます．

プロトタイプの定義や作業計画は，組込みリアルタイムシステムの開発作業の特徴を反映したものになります．すなわち，初期のプロトタイプは，最終的なターゲットマシンであるハードウェアを開発ホストマシン上にソフトウェアでシミュレーションしたうえで動作検証されるかもしれません．次の反復のプロトタイプは，開発用の評価ハードウェアボードで評価することも考えられます．さらに以降のプロトタイプは，さらに実機に近いハードウェア上で動作をさせるプロトタイプを実行・評価する計画となるでしょう．

マイクロサイクルの各反復のプロトタイプは，要求仕様で要求されている機能・非機能を反復ごとに付加的に追加するだけでなく，評価方法もより効果的に進めていくように計画することになるのです（図4）．

■各フェーズのワークフローとナノサイクル

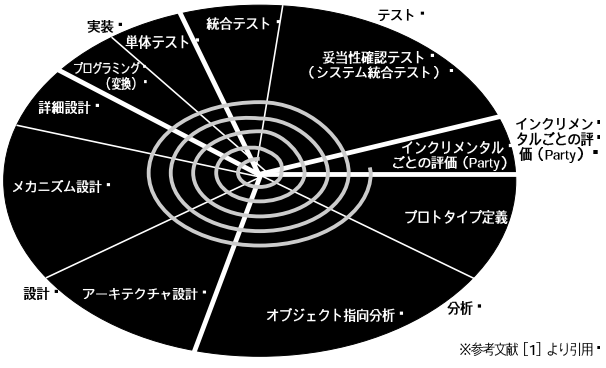
Harmonyで最下層の反復は，図3で登場したマイクロサイクルの各フェーズを詳細に記述したワークフローの中に登場するサイクルで，「ナノサイクル」と呼ばれます．Harmonyではマイクロプロセスは一連のフェーズとして構成されていますが，その各フェーズは，ワークフロー（ニナノサイクル）の中で詳述されます．これらのワークフローは開発作業がどのように進歩するか示すためにモデル化されています．マイクロサイクルの各フェーズのワークフローのすべての例を示すのはページの制約上無理ですので，ここではマイクロサイクルの各フェーズ全体の流れを示したフロー（図5．次ページ）とアーキテクチャ設計ワークフロー（図6．同）の2つを示します．図の中で大きな長方形はフェーズを示し，その中にある長方形は詳細に定義される成果物を表わします．小さな角のとれた丸い長方形の四角は主要な作業を表現しています．各フェーズは，それぞれさらに詳細に定義されていますが，各作業に参考になるガイドや例を示しているものもあります．たとえば，「オブジェクト分析」では，オブジェクトの抽出方法のガイドとしていくつかのやり方が示されています．

◇ ◇ ◇

HarmonyはI-Logix社でDouglass博士が中心となって体系化しています．Harmonyの解説やケーススタディの詳細な情報は，彼の著書やホワイトペーパーを見ると，より深く理解できます．言えることは，Harmonyのオブジェクト指向を利用した開発プロセスの手順は，ビジネス系の開発で用いるオブジェクト指向の手順と大きな変化は見られない標準的なオブジェクト指向開発アプローチを踏襲しています．

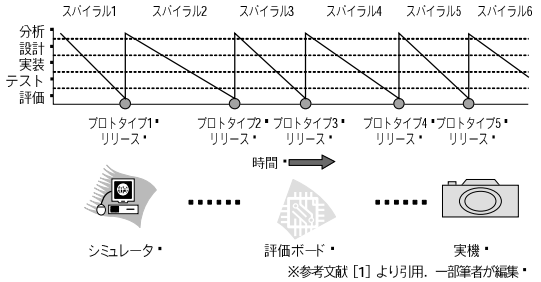
一般に，システムのアーキテクチャを検討・定義する作業では，アーキテクチャ設計には大きく「静的構造」と「動的構造」があります．ごく簡単に解説すると「静的構造」は，レイヤ構成，サブシステム（コン

図3■統合プロダクト開発プロセスのマイクロスパイラル



※参考文献［1］より引用・

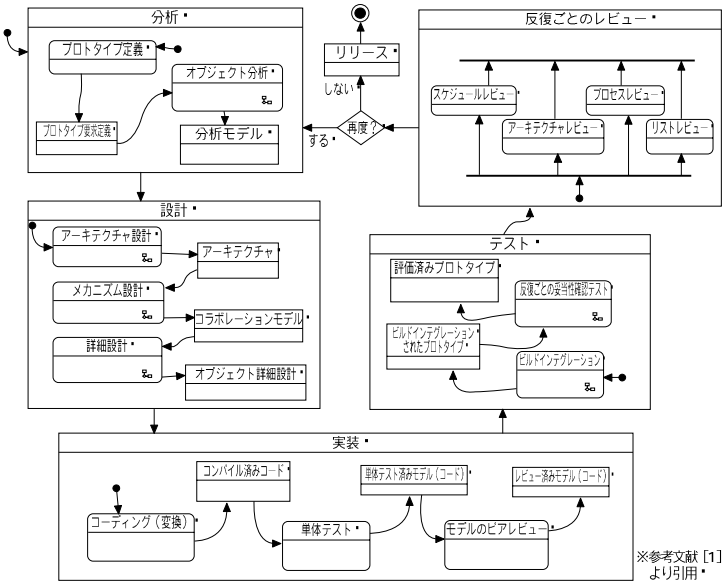
図4■ミクロサイクルと反復ごとのプロトタイプのリリースイメージ



※参考文献［1］より引用．一部筆者が編集・

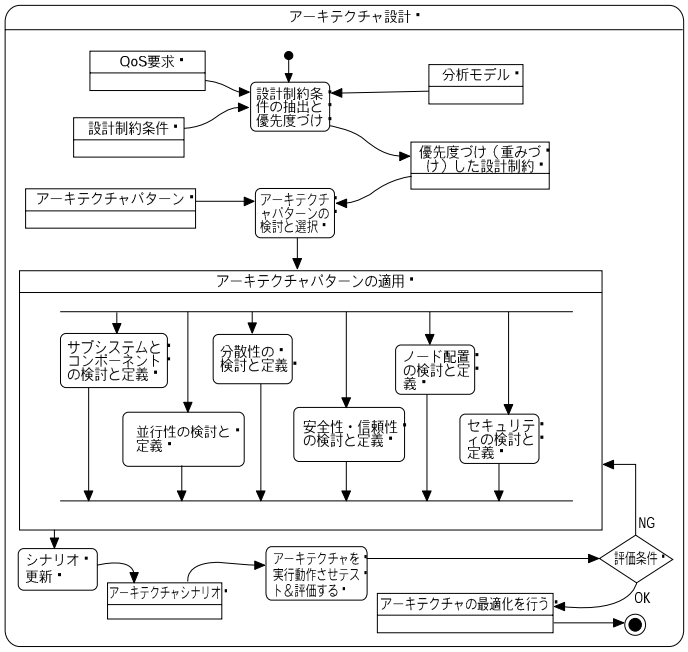
ポーネント）構成，クラスとクラス間の構成ですが，オブジェクト指向の性質を用いてシステムの堅牢性[注9]を実現するには適しています．Harmonyでは，このときの作業プロセスが特に奇をてらったものでなく，オブジェクト指向の効果を最大限に活かせる進め方をとっています．機能の追加・削除など要求変更に強いアーキテクチャを定義するうえで非常に重要です．

図5■統合開発サークルプロセスワークフロー



※参考文献［1］より引用

図6■アーキテクチャ設計ワークフロー



※参考文献［1］より引用

■リアルタイム性，並行・並列性の扱い

一方，組込みリアルタイムシステムで重要となるリアルタイム性やパフォーマンスの性能の要求はどのように分析・設計作業を進めるのでしょうか？アーキテクチャの「動的構造」は粒度の大きいものとしてはマルチスレッド／マルチプロセスなどの並行性／並列性です．中・小粒度としてはクラスの状態図やクラス間，サブシステム間のコミュニケーションがあります．

組込みリアルタイムシステムは，多くのシステムでリアルタイムOSを用いたマルチスレッド／マルチプロセス構成が多くなります．それはハードウェアから非同期でいつ入ってくるかわからないイベントを受け取り，限られた応答時間内で以下に求められた機能の処理を達成させるのは，リアルタイムOSの提供するメカニズムを利用してマルチスレッド／マルチプロセスなどのマルチタスク構成が必要だからです．しかし，システムの適切なマルチタスク構成を設計することは簡単なことではありません．

リアルタイム性やパフォーマンスの性能を保証するには，マルチスレッド／マルチプロセスなどの並行性を可能な限り正確に分析・設計することが必要です．採用するリアルタイムOSで提供される並行性のメカニズム（スレッド，プロセスなど）と，タスク間通信の性能を考慮し，並行性の設計をしますが，Harmonyのアーキテクチャの動的構造はどのように進めるのでしょうか？実のところHarmonyでは，アーキテクチャ

注9）ソフトウェアの信頼性の1つに取り上げられることが多く，不正な操作や不適切なデータの入力があってもシステムがトラブルを避けることができることを指します．

の動的構造を検討する作業では，オブジェクト指向で進める静的構造の分析・設計とは別の作業が必要です．

先ほどの「マイクロサイクル」のアーキテクチャ設計でマルチタスク／マルチプロセスの並行性，リアルタイム性の設計を行うのですが，詳しくは説明をしていないので，簡単にポイントを説明することにします．

Harmonyのリアルタイム性とタスク抽出の考え方

システムの静的構造はオブジェクト指向で開発しますから，オブジェクト（クラス）として抽象化されたハードウェアからシステムはイベントを受け取ります．システムはハードウェアと密接に連動して，低レベルの割り込みやクロック割り込みを処理しなければならないのですが，限られた応答時間（リアルタイム性の要求）で確実に処理を実現するには分析・設計作業として，技術的観点から言えば，オブジェクト分析・設計とは異なる別の緻密な分析・設計作業も用いなければなりません．

近年まで，組込みリアルタイムシステムで本格的に導入されずにいた大きな理由の1つとして，オブジェクト指向の概念とリアルタイムの概念の両方を支援し，オブジェクト指向の性質を活かし，同時にリアルタイム性を満たすシステム開発を実現することが可能なのかという疑問が大きかったためと考えられます．

伝統的な組込みリアルタイムシステムの開発では，要求として求められる最も重要なものはリアルタイム性やフォールトトレランス性であるために，リアルタイム性やフォールトトレランス性を開発の視点にして行ってきました．これはリアルタイム性が厳しくなればなるほどこの傾向にあります．リアルタイム性の分析・設計では，時間要件，同期・非同期イベント，通信機能，並行・並列性に対する分析・設計を正確に行う必要がありますが，システムは往々にして，CPUや動作するハードウェアが複数に分散されることもあります．アーキテクチャを設計するにはシステムの分散も考慮することになり，こうなるとさらに複雑です．複数のCPUでシステムが構成されれば，データや情報のやり取りにはバスを通じて共通メモリが利用されます．しかし，バス上を流れるデータ間のコンフリクトの際のオーバーヘッドや共通メモリを利用した場合には，排他制御による待ち時間が発生し，処理時間を正確に見積もることが困難になってきます．

従来の組込みリアルタイムシステム開発でよく見られた方法は，システムはイベント分析を行い，可能な限り時間制約を満たすことを重視した開発が実施されます．それには，システムを複数のタスク（スレッド／プロセス）に分割させることになりますから，ソフトウェアアーキテクチャの構造化の基準の中心をリアルタイム性においていました．言い換えれば，要求されているリアルタイム性を満たす処理時間を最大限に優先して並行性（タスク：スレッド／プロセス）を検討するやり方です．各タスクは内部構造を設計するために，段階的に，アーキテクチャや詳細設計を通してC言語・アセンブリ言語など用い，リアルタイムOSのシステムコールを利用し，実装されていきます．非同期通信，並行性を初期段階でモデル化し，段階的に実装環境のサービスや能力に変換を考慮し，最適化します．

Harmonyの並行性の分析・設計の考えの基本は，オブジェクトなどの静的なシステムの構成とは別に，タスクの抽出は前章で紹介したHassan Gomaa博士のDARTSなどのタスク抽出基準や設計アプローチなどを参考にしています．最終的に各タスクで振る舞いオブジェクトを検討し，オブジェクトとタスクをマッピングしていきます．このマッピングは競合，スケジューリング性などを考慮して行わなければならず，簡単な作業ではありませんが，オブジェクト指向の持つ保守性・拡張性の利点とリアルタイム性の実現の両立させるためには必要な作業となります．

組込みリアルタイムシステムでオブジェクト指向が敬遠されがちなのは，パフォーマンスを引き出すためにクラスやコンポーネントでモデル化した理想的なアーキテクチャを，実装作業で最適化しなければならない場合が多く発生するからでしょう．なぜなら，多くのレイヤ階層による抽象層，クラス（オブジェクト）のメソッドを通じてのデータコミュニケーションを行うことと，イベントを処理するのに多数のオブジェクト間でやり取りが生じ，そのオーバーヘッドによってパフォーマンスが悪くなるということです．この場合

は，残念ですがシステムの保守が難しくなるという対価の換わりに，パフォーマンスを重視することもあります．

【補足】オブジェクト指向とリアルタイム

オブジェクトと並行性

クラスモデルとオブジェクトモデルは，論理的には本来並行的なもの，さらにすべてのオブジェクトが本質的に並行的であると考えています．

つまり，実装の点で見てみると，リアルタイムOSの提供するタスクが独自にすべてのオブジェクトに割り当てられることを意味します．しかし，これでは実行環境の能力を超えてしまい非現実的です．そこで，実際は，適切なタスク設計をしたあとに，各タスクで動作するオブジェクトを見つけ出し，タスクとオブジェクトを対応づけます．特に，時間制約が厳しい組込みリアルタイムシステムでは，DARTSなどの考え方を利用してタスクに分割することは，常に設計上の決定事項になります．オブジェクトをタスクに適切に割り当てることは，システムのパフォーマンス上，きわめて重要です．タスク間の関係はアーキテクチャ上の基本的な決定事項で，システムのパフォーマンスとハードウェアからのイベントによる要求に多大な影響をおよぼしますから，タスクとその関係を識別するだけでなく，メッセージの特性自体も十分考慮して設計することになります．前章でDARTSを紹介しましたが，タスク間の通信にいくつかのメッセージの種類を用いて設計したのを思い起こしてください．

タスクとオブジェクトの統合

適切なタスクが識別されたら，次にそれらへオブジェクトを割り当てますが，厄介なのはオブジェクトとタスクは直交しているので，同じクラスのオブジェクト（インスタンス）が，異なるタスクに現れたり，タスク間のインタフェースとして現れたりします[注10]．

一般的には，クラスはいくつものオブジェクトにインスタンス化されオブジェクトになりますが，各オブジェクトが現れるタスクはいくつでもありえます．どのタスクにどのクラスのオブジェクトが関係するかは，じっくり設計の段階で検討しなければわかりません．

「暗黙的タスク分割モデル」と「明示的タスク分割モデル」

実はオブジェクト指向とタスクなど並行性を扱う問題として「明示的タスク分割モデル」と「暗黙的タスク分割モデル」の2種類が存在します．どちらにもメリット，デメリットが存在します．ビジネス系のシステムや時間制約をあまり気にしなくてよいシステムで，オブジェクト指向開発を行う場合は，事実上「暗黙的タスク分割モデル」が採られますが，時間制約を気にしなくてよいため，開発者は「明示的タスク分割モデル」と「暗黙的タスク分割モデル」の区別や使い分けを気にする必要がありません．

「明示的タスク分割モデル」と「暗黙的タスク分割モデル」を気にするのは，時間制約が課せられている組込みリアルタイムシステムの開発にオブジェクト指向技術を用いるときです．

明示的タスク分割

明示的タスク分割では，オブジェクト指向のアプローチの分析・設計と並行して，タスクモデル（DARTSのタスクアーキテクチャダイアグラム）を決定するために分析・設計を別途検討します．タスクモデルの決定はシステム要求分析の結果から時間制約，実行効率，タスク間の結合度などを考慮し決定されます．タスク間の通信は原則このタスクモデルを作成するときに決めておき，タスク間でやり取りするデータの種類や構造も決めます．ソフトウェアの静的なアーキテクチャはオブジェクト指向の分析・設計を適用し，並行して「分析」「設計」「実装」作業を進めていくことになります．システムの時間制約の厳しいハードリアルタイムシステムの場合には，時間制約を満たすうえで重要な並行性の設計を重視し，最優先して行う場合にも明示的タスク分割が向いていると言われています．欠点は，分析の初期にタスクを分割するので，オブジェクトでシステムをモデル化する上で，オブジェクト指向の持つ柔軟性と自由度が制限される点ですが，こ

注10）オブジェクトを通信によってやり取りする設計もよく用いられます．

れも設計次第です．

また，マルチタスクの組込みシステムでCPUや要求されている機能が変われば，システムのパフォーマンスに影響を与えますから，タスク構成を変更する必要がでてくる場合があります．加えて，機能に拡張を加える場合でも，デバイスなどの追加が多く，機能追加によりパフォーマンスや競合，デッドラインなどを検討しなおすためにタスク構成が変更になった場合もインパクトを受ける可能性はあります．タスク構成が変わるたびに，各タスクとオブジェクトを統合し，リソース競合の問題やハードリアルタイム性を満たすスケジューリング設計をやり直すのは大変な労力です．再利用にも制約を与えることがあります．

暗黙的タスク分割

暗黙的タスク分割の場合は，システムをオブジェクトで構造化させることを最優先させ，オブジェクト指向のメリットを考慮しつつアーキテクチャ設計で並行性を検討することです．Java言語で開発をした場合は，スレッドクラスやインタフェースクラスを利用しますが，タスクの分割の境は意識しないで開発をすることがあります．UMLのアクティブクラスを検討し，そこにスレッドクラスやインタフェースクラスを割り当てるからです．並行性はJavaのスレッドクラスのインスタンスなどの処理シーケンスによって実現されることになります．暗黙的タスク分割の長所は，「明示的タスク分割」のように開発初期に問題空間をタスクで分割する制限がないため，より自然かつ保守性，拡張性が良い構造を得ることができると言われている点です．

しかし，時間制約が課せられていない場合はともかくとして，オブジェクト指向で設計したアーキテクチャの視点でタスクを割り当てるので時間制約が厳しくなると，このアプローチでは時間制約を保障できなくなります．イベント分析などを通じて，時間制約を満たすような並行性を明示的に切り出さないからです．

開発環境「Rhapsody」

本章の最後にHarmonyの開発環境について紹介します．HarmonyはUMLを用いてMDDで開発を進めていきます．開発環境は特に限定されないHarmonyですが，Harmonyのプロセスやコンセプトを支援した開発環境が存在します．I-Logix社が開発・提供している「Rhapsody」（表4，図7～9．次ページ）です．

RhapsodyはHarmonyをサポートしている開発環境ですので，Harmonyのプロセス作成するさまざまなモデルなどの成果物を作成し，管理できます．Rhapsodyについては操作トレーニングコースも用意されているので導入することを検討する際は受講をお勧めします．また，日本国内でRhapsody購入も可能であり，日本語版も用意されています[注11]．

Harmonyのプロセスで「コーディング」の記述がありますが，そこでは「コーディング（変換）」となっています．これは，Rhapsodyのような開発環境を用いて，設計情報のモデルから，ターゲット言語でソースコードを自動生成するときに「モデルからコード生成して変化」という意味です．Rhapsodyのような開発環境を用いれば，ソースコードを自動生成，コンパイル，ビルドそして動作確認まで行えます．言語はC++，C，Java，Adaに対応しています．

まとめ

いかがだったでしょうか？誌面の都合もあり，ちょっと急いだ感もありましたが，みなさまの今後の開発にお役に立てれば幸いです．組

■参考文献

[1] The Harmony Process-The Development Spiral，Bruce Powel Douglass，Ph.D. Chief-Logix
[2] The Harmony Process Bruce Powel Douglass，I-Logix
[3]『Real Time UML: Advances in the UML for Real-Time Systems, 3rd Edition』Bruce Powel Douglass 著，Addison Wesley Professional，ISBN：0321160762，2004/2/17
[4]『Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns』Bruce Powel Douglass 著，Addison-Wesley Professional，ISBN：0201498375，1999/5/11
[5] Octopus Concurrency Design with Rhapsody，Eran

注11）Rhapsodyの日本代理店：伊藤忠テクノサイエンス株式会社（http://www.ctc-g.co.jp/ EDA/ilogix/）．

表4■Harmony支援環境「Rhapsody」の特徴

特徴	説明
Harmony支援ツール	Harmonyに準拠したプロセスでは作業効率に優れている．また，Harmony以外のプロセスでも特に問題になることはなく，利用者の意図した使い方が可能
MDDを実現	UMLによるモデル中心開発．UML 2.0に準拠しており，反復型の開発を効果的に実施できる．クラスの状態図に開発言語でアクションを記述することで，コードの自動生成が可能．このときRhapsodyのアーキテクチャがビルドされ実際に動作可能な実行コードが生成される
アニメーション機能によるモデルのシミュレーション	アニメーションにより，分析・設計で意図した動作をするかを自動検証可能．上流の時点で検討を行いながら作業を進めることができる
リバースエンジニアリング機能	ラウンドトリップ開発を可能とするために，ソースコードの中の構造からリバースエンジニアリング機能を用いて，設計情報を引き出し，設計情報へとモデルを更新する
モデルチェック機能	コード生成を行う前にモデルの完全性を検証可能

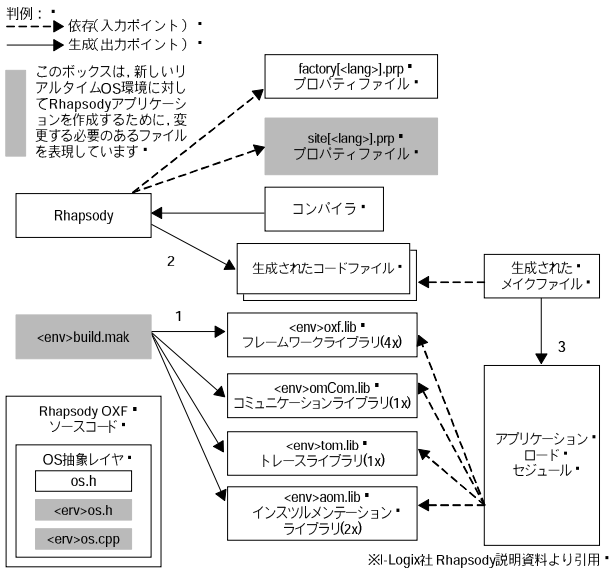
図7■Rhapsodyのシミュレーション画面

※参考文献［13］より引用

図8■Rhapsodyのコード生成の様子

※参考文献［13］より引用

図9■Rhapsodyの開発環境のメカニズムの概要



※I-Logix社 Rhapsody説明資料より引用

Gery, Ran Rinat I-Logix Inc. Jurgen Ziegler Nokia Research Center，2000/9/6

[6]『Designing Concurrent, Distributed, and Real-Time Applications With UML』Hassan Gomaa 著，Addison Wesley Professional，ISBN：0201657937，2000/1/15

[7]『オブジェクト指向設計の標準 HOOD3.1』HOOD技術グループ 著，立田種宏 翻訳，海文堂，ISBN：4303726109，1994/8

[8]『Object-Oriented Analysis and Design With Applications (The Benjamin/Cummings Series in Object-Oriented Software Engineering)』Grady Booch 著，Benjamin-Cummings Pub Co，ISBN：0805353402，1993/10

[9]『Software Engineering With Ada (The Benjamin/Cummings Series in Object-Oriented Software Engineering)』Grady Booch，Doug Bryan，Charles G. Petersen 著，Benjamin-Cummings Pub Co，ISBN：0805306080，1993/8/31

[10]『Real-Time Object-Oriented Modeling（Wiley Professional Computing)』Bran Selic，Garth Gullekson，Paul T. Ward 著，John Wiley & Sons Inc，ISBN：0471599174，1994/4/22

[11]『System Development (Prentice-Hall International Series in Computer Science)』M. A Jackson 著，Prentice/Hall，ISBN：0138803285，1983

[12]『Executable UML MDAモデル駆動型アーキテクチャの基礎 Object Oriented SELECTION』スティーブJ.メラー，マークJ.バルサー 著，二上 貴夫，長瀬 嘉秀，Executable UML研究会 翻訳，翔泳社，ISBN：479810602X，2003/12/16

[13] Rhapsody:Collaborative Model-Driven Development for Software, Systems and Test，I-Logix

[14]『Dedigning Software Product Lines With UML』Hassan Gomma，Maryann Barber 著，Addison-Wesley Pub Co，ISBN：0201775956，2004/7/9

[15]『Object-Oriented Technology for Real-Time Systems』Maher Awad，Juha Kuusela，Jurgen Ziegler 著，Prentice Hall Pub Co，ISBN：0132279436，1996/3/8