JIPDEC 先端情報技術レポート　03-01 (2003.07)

これからのソフトウェア研究の目指すものは何か。

米国 SDP 計画における研究テーマの

策定ワークショップにおける議論の紹介

（財）日本情報処理開発協会　調査部

先端情報技術調査・普及グループ（JIPDEC-AITRG）

http://www.icot.or.jp

最近、わが国においても、インターネット時代のソフトウェア開発の基盤となる新しいソフトウェア理論やソフトウェア工学、ソフトウェアの検証方法、開発ツールなどを、再構築すべきであるとの意見を耳にするようになった。

米国では、わが国に先んじて、このような内容を目指す計画が、NITRD 計画のテーマ（PCA）のひとつとして開始されている。これが、SDP(Software Design and Productivity)である。しかし、米国といえども、どのような研究を行えばよいかは明確ではない。そこで、どのような研究をすべきかを議論するワークショップが開催され、そこで多くの興味深い議論と提案がなされた。本ワークショップの内容は論文にまとめられて公開されている。

JIPDEC-AIRTRG では、3 月に廃止された先端情報技術研究所（AITEC）の活動を引き継ぎ、本ワークショップ紹介論文の日本語訳の校正、概要編の追加を行い、公開することとした。わが国のソフトウェア研究者の活用を期待している。尚、この資料は、www.icot.or.jp に公開されている。

第１編　SDP 計画、および、ワークショップの概要

1999 年 2 月に、米国の PITAC（IT に関する大統領諮問委員会）は、IT に関す米国政府支援の研究開発に対し、長期的基礎研究とソフトウェア研究の重要性訴える提言を行った。この提言を受けて、米国の情報技術研究開発政策の柱であるネットワーキング及び情報技術研究開発（NITRD：Networking and Information Technology Research and Productivity）に「ソフトウェアの設計と生産性」SDP：Software Design and Productivity）という研究領域が追加された。

本章は、この SDP が何を目指しているかについて、2001 年 4 月アーリントンで開かれたワークショップの内容から紹介し、2003 年度の SDP の活動計画として、米国どのような研究に取り組もうとしているかを紹介する。

1.1　SDP 発足の経緯と PITAC の提言

1999 年 2 月に、PITAC（IT に関する大統領諮問委員会）は、IT に関する政府支援の研究開発に対し、次の示すように、長期的基礎研究とソフトウェア研究の重要性を訴える提言[1]を行った。

●1992 年以来、IT 関連の生産高は GDP の 1/3 を占め、数百万ドルの雇用機会を提供してきた。

●しかし、現状の政府の投資は、長期的なハイリスクな研究開発よりも、各省庁のミッションを追求する短期的な課題に偏りすぎている。

●このままであると、過去 20-30 年に渡って続いてきた IT 革命へのアイデア供給の流れは途切れてしまう。

●従って、2000-2004 年の 5 年間に下記の研究開発テーマに 4,743M ドルを長期的基礎研究開発に追加投資すべきである。

- ソフトウェア
- 大規模な情報化のためのインフラストラクチャ
- 最高レベルのコンピューティング
- 社会経済的影響
- 連邦政府の情報技術研究の管理と実現

このPITACの提言を受け、政府は、2000年には「21世紀に向けての情報技術」IT²計画として366Mドルの研究開発投資を追加し、2001年にはこれまでのCIC計画（2000年の予算は1,089Mドル）と統合して、ITR&D計画として、1,928Mドルと大幅に増額している。そして、現在は、ITR&D計画はNITR&D計画と名前を変更し、予算規模は1,830Mドル（2002年）となっているPITACは、追加投資すべきとしたソフトウェアについては、以下のような現状、課題、であるとの調査結果を発表し、

●ソフトウェアに対する要求が、国家的なソフトウェアの生産能力を上回っている。

●脆弱なソフトウェアに国家が依存している。

●信頼性と安全性に優れたソフトウェアを構築するためのテクノロジが不十分である。

●ソフトウェア・システムは、急速な勢いで多様化と洗練化が進んでいる。

●一般的な市民にとって、ソフトウェアへの依存度はしだいに高まりつつある。

●ソフトウェアの基礎研究に向けた国家の投資が不足している。

その対策として、次のことを提案している。

●ソフトウェアの基礎研究を最優先課題とする。

●ソフトウェアの開発手法、およびコンポーネント・テクノロジに関する基礎研究に向けた投資を強化する。

●ヒューマン/コンピュータ・インタフェースと相互作用に関する基礎研究を支援する。

●情報管理の方法に関する基礎研究を強化する。

●主な情報テクノロジ関連の研究構想の中に、ソフトウェアの研究を主要な課題として位置付ける。

1.2 NSF SDP ワークショップの概要

PITAC の提案により、SDP の研究開発予算はついたものの、PITAC の提案は、具体的にどのような研究を行えばよいか必ずしも明らかではない。そこで、NSF の Frank Anger が座長となって、後述する米国の著名な研究者（他に省庁の研究機関担当者も出席）が、2001 年 4 月 18~19 日にアーリントンに集まって開催されたのが、本ワークショップ[2]である。

1.2.1 SDP ワークショップの狙い

本ワークショップでは、Frank Anger がイントロとして、ソフトウェアの現状は、以下のようであり、

●全米でソフトウェア技術者 200 万人

●サポートスタッフ 35 万人

●まだまだ空きポスト多数

従って、以下のことを達成しなければならない。

●ソフトウェア生産性の劇的向上

●高いソフトウェア品質

と説明し、前述した PITAC の提言を説明した後、本ワークショップでは、以下の点をあきらかにするのがねらいであるとした。

●解決すべき基礎的な問題は何か

●真の障壁と挑戦とは何か

●どの方向がもっとも有効か

●何が理想的な成果か

●どのように研究を進めるべきか

この「どのようなソフトウェア研究をすればよいか」の議論を基にして、SDP（Software Design and Productivity）の研究計画を作ろうということで、以下の 4 つのグループに分かれて検討がなされた。

●ソフトウェア研究の将来

●新ソフトウェア開発パラダイム

●実世界ソフトウェア

●ネットワーク中心のシステム

これら四つのグループの参加メンバを表 3.1 に示す。

表 3.1　SDP ワークショップ参加メンバ

ソフトウェア研究の将来	新ソフトウェア開発パラダイム
Barry Boehm, USC Benjamin Pierce, Univ. of Pennsylvania Doris Carver, Louisiana State Univ. Shankar Sastry, UC Berkeley Bonnie John, Carnegie Mellon Univ. Kevin Sullivan, Univ. of Virginia William Mark, SRI International	Grady Booch, Rational Software Ralph Johnson, Univ. of Illinois Gregor Kiczales, Univ. of British Columbia Charles Simonyi, Microsoft John Vlissides, IBM
実世界ソフトウェア	ネットワーク中心のシステム
Don Winter, Boeing Phantom Works Martin Feather, Jet Propulsion lab. Gabor Karsai, Vanderbilt Univ./ISIS	Ian Foster, Argonne National lab. Doug Lea, State Univ. of New York at Oswego Adam Porter, Univ. of Maryland

Patrick Lardieri, Lockheed Martin Cleve Moler, MathWorks, Inc. Edward Lee, UC Berkeley	Rick Schantz, BBN Technologies Jim Waldo, Sun Microsystems, Inc

1.2.2　各検討グループからの提言

(1) ソフトウェア研究の将来

現状は、

●現在のインフラは、低い信頼性、安全性、可用性で、まったく不適切である。

●大規模ソフトウェアプロジェクトの半数は失敗している。

●応用ソフトウェア同士が情報をやりとりできない。

●過去のソフトウェア資産が１兆ドルもある。

との仮説に立って、以下の目標を立てている。

●各領域の専門家がソフトウェアを使いこなせるようにしよう。

●既製品ソフト業界の共通基盤となるアーキテクチャを開発しよう。

●ソフトウェア業界に対し、鍵となる概念・技術を提供しよう。

●プログラマの意図を機械が理解できる形で記録する方法を開発しよう。

●ソフトウェア開発の現状をしっかり認識しよう。

この目標実現に向けて、特に注目すべき領域は、以下であり、

●ネットワーク埋め込み型

●対人インタラクション

●グループインタラクションの計算機支援

●領域専門家のための環境：教員、エンジニア、金融、科学者

その領域で基礎となる科学は、以下であり、

●ソフトウェア物理：並行処理のモデルなど

●ソフトウェア経済学：柔軟性の価値を定量化など

●ソフトウェア行動科学：人間の認知・記憶に関する知見をソフトウェア設計に応用したり、ソフトウェアと人間の役割分担を明らかにしたりすることなど

そのキーポイントは、以下であると提言している。

●ソフトウェア開発のプレイグラウンドを変えよう。

- 「プログラム」の意味の変更:
  Ptolemy（同期データフロー、状態遷移マシン、ランデブー方式同期メッセージング、連続的な時間、離散イベントといったコンピュテーションモデルのビジュアル化イメージでシステムを作成するアーキテクチャデザイン言語）
  Simulink（ブロックダイアブラムを作成することによって、システムのシミュレーションや性能の評価、デザインの改善が行える Mathworks 社のモデリングツール）
- 「プログラマ」の意味の変更:practitioners （開業医などの専門家）

●以下のキーとなる技術を進展させよう。

- 軽い形式的枠組み
- 並行性・耐故障性・安全性などのモデル
- 「価値」に基礎を置いたデザイン
- 分散オープンソーステスト方式

●ソフトウェア経済学を変えよう

●ソフトウェアでハードウェア を定義しよう。
（ハードウェアがソフトウェアをではなく）

(2) 新ソフトウェア開発パラダイム

新ソフトウェア開発パラダイムでは、現状は、以下のような要因によって、複雑さが指数関数的に増加している。

●組込型、モバイル、浸透性、大域性

●長寿命、複数バージョン混在

●共同開発

●利用者ごとのカスタマイズ

この難問題を以下のような革新的概念によって、一挙に解決することが必要である。

●国立ソフトウェア考古学センター構想

●多面的ソフトウェア

●協調開発環境

[国立ソフトウェア考古学センター構想]

ソフトウェアの古典を蒐集および保存し、それをリバースエンジニアリングによって、その古典の芸術的本質を抽出し、Web で公開する。これにより、参考となるソフトウェアアーキテクチャの宝庫として、将来の研究の基礎として、特許取得のための事前芸術研究として、役立て、芸術的表現を奨励することをねらう。

[多面的ソフトウェア]

高級言語、オブジェクト指向プログラミング、デザイン・パターン、特定領域用言語といったこれまでの歩みは、ある側面をより理解しやすくした、あるいは、ある側面をうまく表現したというものにすぎない。本来、ソフトウェアは、様々な異なる意図を表現するものであり、新たなシステムは、共存する異なる側面が互いに影響しあって生まれるものである。このような多面的ソフトウェアについて、従来は認識不測であったが、デザイン・パターン、協調開発環境、アスペクト指向プログラミング、特定領域用言語といった研究分野では、一部、多面的ソフトウェアの考えが取り入れられ始めている。今後、多面的ソフトウェアとしてすべきことは以下である。

●どの側面が肝要か
　性能、価格、使い勝手、過去からの連続性…

●各側面をどう表現するか

●諸側面をどう変形・統合しシステムとするか

●特定領域用言語のためのツール整備の加速

[協調開発環境]

　協調開発環境とは、ソフトウェア開発のプラットフォームとしてWebを仮想的な議論の場にするものであり、研究すべきことは以下である。

●ソフトウェア開発の社会学

●ソフトウェアプロセスの利用

●考古学センターや多面的ソフトウェアを利用するプラットフォームの創造

これらの革新的概念の研究計画例には、以下のようなものがある。

●側面中間形式（特定領域用言語のための汎用ツール、Weavers?）

●対応代数　（Correspondence Calculus）

●デモプロジェクト　（Demonstration Projects）

●領域回復（Domain Recovery）

●ソフトウェア保存　（Preservation）

●ソフトウェア開発民俗学　（Ethnography）

(3) 実世界ソフトウェア

　今日のソフトウェアは、偶然的複雑性、予測不能性、合成不能性、脆弱性、実世界との相互作用に向かないといった問題点を抱えている。

また、ソフトウェアは「コードの集まり」という考え方にハマッテおり、大規模にするには人力を注ぎ込む、システム全体像の理解が喪失している、仕様とその実現が分離している、といった事態を招いている。

10 年後には、どこでもコンピュータだらけになって、これまでの世界とは全く変わった世界になるというのに、今日のようなソフトウェアの作り方をしていたら、世界は危険極まわりない場所となってしまう。それを避けたければ、技術の実施を遅らせねばならず、その場合には、自動車は、コンピュータ制御とはならず、交通渋滞にはまり込んでしまうだろう。

これを解決するには、モデリング言語、システムの合成、変換、過去のソフトウェア遺産の扱い、といった課題に取り組まねばならない。

[モデリング言語]

ソフトウェアの問題点は、「プログラムコード」、あるいは「仕様」にあるのではなく、「モデル」あるいは「設計」にある。すなわち、複雑な機能の実現、設計の理解、質問の定式化、振舞いの予測、といったことを行う際に、それを表現するモデリング言語を我々が持っていないことである。

これを解決するため、以下のような研究を行うべきである。

●システムのモデリング「言語」

●有用な抽象化を見つけること

●計算的システム理論

●組合わせて使える抽象化

●時間・並行・能力などの表現

[システムの合成]

システムを組合せて作る組織的方法、例えば、要素の枠組、組合せの意味論、その場での（on-the-fly）合成、既成のソフトウェア要素との統合、といったことが欠如している。

これを解決するため、以下のような研究を行うべきである。

●意味の枠組みと理論

●方法論 （methods） とツール

●実験台と挑戦的問題

●参照実装 （reference implementation）

●アーキテクチャ枠組みの定義

●分散・分割の戦略

●粒度とモジュラー性の制御戦略

[変換]

異なる抽象間の変換についての理論,例えば、異なる抽象の間の関係、生成器（変換器）、複数の視点を許す抽象化（multi view abstraction）、物理的環境の抽象化、高信頼性ソフトウェアとシステム（HCSS：High Confidence Software and Systems）との関係: 証明つきの変換、といったことが欠如している。

これを解決するため、以下のような研究を行うべきである。

●オープンな生成器インフラ（方法論、ライブラリ）

●生成器の理論

●Methods for correct by construction transformers （?意味不明?）

●変換器のパラメタ化

[過去のソフトウェア遺産の扱い]

過去の遺産を扱う方法論,例えば、システムを徐々に現代化する方法、新しいものを古いものと統合する方法論、といったことが欠如している。

これを解決するため、以下のような研究を行うべきである。

●遺産コードの部品化

●遺産からの抽象化の抽出

●漸次現代化、リバースエンジニアリング

●再設計を安価に ⇔ ソフトウェア遺産からの進化

(4) ネットワーク中心のシステム

今後のシステムは、以下のような状況に進展すると予想される。

●すべてがコンピュータ

●すべてがネットワークコンピュータ

●すべてが相互依存する可能性

●実世界と接続

●ますますヘテロに

●接続はますます多様で不安定に

その状況下で、以下のような応用システムが登場してくるであろう。

●交通制御: 自動車数千台からセンサーデータ

●戦域管理: 敵対環境で多様粒度の調整・任務

●サプライチェーン管理

●科学データの共同解析

数千ユーザ、数千資源に対し、ソフトリアルタイムで問合せ最適化の要請

●家庭内電力管理

このような状況、応用システムにおいては、以下のような技術的課題が生じる。

●資源（QoS, 時間・相互依存・エネルギー消費・大きさ）を意識したプログラミング

- 工学的トレードオフを考慮する計算モデルの欠如
- 複雑なシステムでは端点間の特性が不明

●動的な資源の振舞い: 故障・不均一性 等

- 計算モデルや端点間特性の欠如

●ソフトウェア遺産

- 設計時に想定しなかった利用形態への対応

●危機・信用・制御の管理

- ポリシー・保障・管理のドメイン
- 極度に動的なシステムの安全性確保と確認手法
- プライバシ

●スケールの問題

- 構成要素数、構成要素の大きさ
- 単一の計算に関係する要素数
- ネットワーク中心システムは長時間無停止動作

これらの問題を解決するには、以下のような研究方向とアプローチが必要である。

●「問題」を扱うプログラミングモデル

- 工学的トレードオフ
- 大規模システムの運用
- 動的な資源の振舞い
- 危機・信頼・安全の管理

●「問題」の基本機構

- 資源のトレードオフ: QoS、実時間性、等
- 適応的振舞い
- さまざまな次元への拡大
- 分散管理

●「問題」の全体を扱える、多レベルの資源管理

●ネットワーク中心の（小規模な）要素を、動的に大規模システムに合成する技術、相互運用性

この他、次のような研究も必要である。

●新たな計算モデルを使うためのツール

●高度に複雑化したシステムの検証・シミュレーション・理解

●大規模分散応用システムの （自動） 構成・管理

これらの研究によって、以下のような効果が期待される。

●社会的には

- 現状では作れないものを作れるように
- ネットワーク化システムの信頼性・制御性向上
  工学的設計によって実現　デバッギング
- ソフトウェア開発コストの抑制

●教育

- 新たなソフトウェアの作成・システムの設計にあたれる人材の供給

1.2.3 SDP ワークショップ提案のまとめ

4 つのグループからのそれぞれの提案は、別々の方向を向いているのではなく、ほぼ同一方向を向いており、概ね、以下の提案にまとめられる。

●有用な抽象化を含んだモデリング

- モデリングは、対象とする特定領域のモデルを記述できるものとし、その特定領域の専門家が使いこなせる。
- モデリングは、資源のトレードオフ・動的管理、危機・信頼・安全の管理といった、設計の意図を表現できる。
- ある特定の側面からのモデリングを組み合わせ・合成・変換・統合して、様々な側面からのモデリングができ、システムの複雑さを軽減できる。
- モデリングは、科学的、工学的設計に基づいた軽い形式的枠組みを持ち、これからコードの自動生成ができると共に適合性の検証も行える。

●オープンな協調開発環境

- Web をベースとしたオープンで協調して開発できるプラットフォーム。
- この環境には、芸術的ソフトウェアが蒐集され、ソフトウェアの宝庫となる。
- 柔軟性の価値なども定量化され、ソフトウェアの経済的社会的評価ができる。

●既存ソフトウェア資産を生かす仕組み

- システムを徐々に現代化する仕組み
- 新しいものと古いものとを統合する仕組

第２編　SDP ワークショップ　詳細編

ソフトウェア設計・生産性に関する新構想:研究とアプリケーション

情報技術研究開発（ITRD）のための省庁間ワーキング グループ：
ソフトウェア設計及び生産性（SDP）調整グループによるワークショップ
2001 年 12 月 13、14 日
ヴァンダービルト大学
テネシー州ナッシュビル

まえがき

80年代後半から90年代は、情報技術（IT）の「ビッグバン」が起きた。過去数10年間にわたる研究開発への投資をその礎とし、また、コンピューティング ハードウェア、ソフトウェア、ネットワーキングに於ける研究のブレークスルーを契機として、ITがこの10年余りのうちに急展開を遂げるのを我々は目の当たりにしてきた。そして、その結果「IT 大爆発」が起こり、多方面に同時に広まってきた。

以下に挙げるとおり、この過程の効果は大きい。

- 一番面白い開発は「波打ち際」、つまりITが他の分野と交わるところで行われる。科学と技術の新しい分野での事例には事欠かない。バイオ情報学、ナノテクノロジー、コンピュータ支援学習（CAL）、組み込みソフトウェアとリアルタイムシステムを始めとする多くの分野がある。これらの分野から新しい専門分野が生まれ、重要な新産業をさらに創り出す可能性を生み出した。
- ITが世の中にあまねく浸透していくにつれ、IT研究とIT産業の昔ながらの構造は急激に変化してきている。たとえば、組み込みシステムの物理プロセスと情報プロセスとを緊密に統合するには、コンピュータ的であると同時に物理的な新しいシステムサイエンスの開発が必要である。このような進歩には、既存モデルからの徹底した「決別」を意味する新しい教育アプローチやプロジェクト管理構造が最終的に必要となるだろう。

IT主導へと様変わりした経済の中心となるのが、複雑なコンピューティング情報システム向けの高品質のソフトウェアである。世界における米国の技術支配が、今後も大いに成功し、進歩するかどうかが問われている。高品質のソフトウェアを迅速かつ効率的に構築することが困難なことから、大統領直属情報技術諮問委員会（PITAC）は、米国が

「基礎的なソフトウェア研究に最高の優先順位を付ける」ことを提言した。したがって、米国の安全と継続的な経済発展を維持するためには、ソフトウェア設計と生産性における実質的な研究のイニシアチブを確立し、この提言を現実のものとすることが肝要である。

ワークショップ「ソフトウェア設計及び生産性に関する新構想」ではフォーラムを開催し、ソフトウェアの品質面で妥協せず、ソフトウェアの生産性を飛躍的に向上させうるソフトウェア開発技術について、画期的な方法を科学者、技術者、ユーザが見い出す場を提供した。このワークショップは、「情報技術研究開発（ITRD）のための省庁間ワーキンググループ」の一つである「ソフトウェア設計及び生産性（SDP）調整グループ」が主催する2回目のイベントである。企画ワークショップは、2001年4月18、19日にバージニア州バーリントンで行われた。2回目のSDPワークショップは、企画ワークショップで得られた成果と洞察をベースに、それを発展させたものである。研究開発コミュニティの主要メンバーにあてた討議資料（ポジションペーパー）募集の呼びかけに応じてメンバーからホワイトペーパーが提出され、その評価に基づいて会議への招待状が発送された。ワークショップへの参加者は、企業あるいは大学の64名の研究者からなる招待参加者と、14名の政府機関の研究者で構成されている。SDPワークショップの詳細な資料は、http://www.itrd.gov/iwg/pca/sdp/sdp-workshops/Vanderbilt に公開されている。

ワークショップ共同議長:
Janos Sztipanovits 博士
ヴァンダービルト大学
Adam Porter 博士
メリーランド大学

SDP 共同議長:
Frank Anger 博士
NSF
Douglas C. Schmidt 博士
DARPA

注１： 本資料は、JIPDEC-AITRGが、編集者より、日本語訳作成の許可を得て、作成したものである。
注２： 翻訳文の監修、校正は、AITEC主任研究員 牧村信之による。

目次

エグゼクティブ サマリー

SDPワークショップ「ソフトウェア設計・生産性に関する新構想」の目的は以下のとおりである。

- それぞれの技術の最先端の研究者と実践者との出会いの場を提供する
- ブレーンストーミングと創造的な発想を奨励する
- 政府の研究課題の認知度を高める
- 政府機関と研究コミュニティの双方の参加を得る

特に重要な関心事は、今後数十年のソフトウェアの設計、製造方法を根本から変える技術の必要性と有望な解決策であり、製品化に要する時間や利益を重視する今日の研究開発プログラムの範囲では、到底達成できないことである。SDPワークショップは、パネル・ディスカッション、ブレークアウト・セッション、全体会議で構成された。パネル・ディスカッションとブレークアウト・セッションでは、ソフトウェア設計と生産性の要となる、以下の4つの課題を取り扱った。

1. ソフトウェアとソフトウェア研究の将来

経済活動のあらゆる分野にソフトウェアが急速に浸透したことは、ソフトウェア研究の土台となる基礎とソフトウェア研究をどう行うべきかを考えることに対して、大きな課題を投げかけた。今日では、大規模ソフトウェア・プロジェクトの過半数が失敗に終わったと評価されている。さらに、新旧のソフトウェア・アプリケーションがシームレスに連携できないことが多いという事態にもかかわらず、従来のソフトウェアに対する投資はやめる訳にはいかない。明らかに、こうした現状を打開するには、新しい考え方が必要である。そこで、ワークショップでは以下の内容について議論した。

- 将来のソフトウェアとソフトウェア開発チームはどうあるべきか。
- プログラミング言語の役割は何か。
- ソフトウェア研究コミュニティはどのように発展すべきか。たとえば、より実践的な機関となるべきか、支援ツールに注力すべきか、個別の分野ベースの枠組みに分割すべきか、まったく新しい抽象化を推進すべきか、これまでとまったく異なる方向に進むべきか、など。
- 科学者からプログラマ、エンドユーザまで幅広い層にわたる「将来のソフトウェア実践者」をどう訓練するか。

2. 新しいソフトウェア開発の枠組み

ソフトウェアは、大規模システム全体を統合する役割をますます担い始めている。大規模システムは、それ自体ネットワーク中心の「システムからなるシステム」である。慎重なエンジニアリング・プロセスと系統立った検証手法とを包括する枠組みが必要である。以上について、次の問題が提起された。

- ソフトウェア開発は、柔軟性を欠く「規則優先」のものから、よりダイナミック、そして流動的なものへと、どのような方向性で、変わらなければならないか。
- 形式的枠組みと非形式的枠組み、エンジニアリングと芸術性、進化と再構築、作って直す方針と合意して直す方針、これらの均衡をどう図るべきか。
- オープンな標準とオープンソースのソフトウェア開発、エンドユーザによるプログラミング、根本から異なる開発モデル、といったものの役割は何か。
- 法的、社会的な要請は、ソフトウェア開発、テスティング、検証の枠組みにどのような影響を及ぼすか。
- 分散化した共同開発環境は、ソフトウェアの設計、生産性、品質にどのような影響を及ぼすか。

3. 実世界のためのソフトウェア

マイクロプロセッサ全体の実に90パーセント以上が、組み込みシステムで使用されている。このような組み込みシステムにほぼ共通する特徴は、実世界の制約を受ける点である。従来からの機能要件も満たしつつ、一方で物理的な制約を受け入れた組み込みシステム向けソフトウェアを仕様決定し、プログラミングし、構成し、統合し、検証するための理にかなった手法が必要である。これらに基づいて、以下の問題が提起された。

- 無数にある実世界の制約を、ソフトウェア設計時にどのように機能的制約に統合するか。
- ハードウェアとソフトウェアとの協同設計は、問題解決にどのように貢献できるか。
- モデルとジェネレータは多くのレベルでソフトウェア設計のソリューションとなり得るが、これ以外に有望なアプローチとその展望はないものか。
- 今日の社会のかなりの部分を占める従来のシステム（と従来の開発者）をどのように扱うべきか。

4. ネットワーク中心の大規模システム

次世代アプリケーションはさらにネットワーク化が進み、構成と負荷に関する難しい課題を背負うことになるだろう。遅延の隠蔽、部分的な故障、因果関係、動的サービス分配、分散デッドロック回避などの課題が挙げられる。これらの難題を解決するには、サービス・フレームワークのエンド・ツー・エンド品質を実現する技術が必要であり、マルチレベルの分散リソース管理や適合型ミドルウェア・アーキテクチャを理解することが求められる。これらについて、以下の問題が提起された。

- 今後加速するネットワーク中心の世界で解決すべき基本的な設計上の課題は何か。
- 多種多様なネットワーク、OS、アプリケーションを包括するサービス要求のエンド・ツー・エンド品質をいかに効果的に指定し、実践するか。
- 常時稼動の基幹システムを設計、実現するにはどのような解析手法、ツールが必要か。
- リスク管理がシステムの一部である場合、それがソフトウェア製品にどのような影響を及ぼすのか。

招待されたパネリストらは、初日の全体会議で以上の問題を議論した。パネル・ディスカッションに続いて、各パネル議長の主催によりブレークアウト・セッションが開かれた。ブレークアウト・セッションの議論は、SDPワークショップの提言案の元となるインプットを参加者全員が提供する機会となった。セッションの議長らは、ブレークアウト・セッションで得られた最終提言案を、ワークショップを締めくくる全体会議の場で発表した。

SDPワークショップの全文書は、（a）エグゼクティブ・サマリー、（b）ブレークアウト・グループ・サマリー、（c）セッションの議長による要約発表、（d）パネリストの発表、（e）参加者の所信表明で構成されており、ワークショップのWebサイト http://www.itrd.gov/iwg/pca/sdp/sdp-workshops/Vanderbilt に公開されている

以下に、SDPワークショップの主な成果と提言案の要約を示す。

今日の我々の位置付け

1960 年代半ばにソフトウェア分野が誕生してからというもの、「ソフトウェア危機」は避けられないものとして存在し続けてきた。これは、「エンドユーザの要求が技術の進歩を上回ることが避けられない」という事実に起因する。過去数10年にわたってソフトウェア研究が続けられ、エンジニアリングの改善が実践されてきた結果、とりわけ業

務アプリケーション分野において、生産性が驚異的に向上したことは明らかである。実際、我々は現在、毎日の生活の隅々にまでソフトウェアが浸透していることを「当たり前」と捉えがちである。家電、自動車、街灯、オフィスの警備システム、ビジネスや個人のコミュニケーションから、銀行業務、医療機器、緊急サービス、電力供給、科学上の発見、国防、軍事システムに至るまで、ソフトウェアは我々の生活の中に深く浸透している。

皮肉なことに、今なお信頼できる投資効果のある複雑なソフトウェアを高い品質で開発できないことを根拠に、「過去四半世紀には大した進歩はなかった」とする批判の声もある。しかし、この批評は「10年前と比べて、はるかに複雑な大きく異なるシステムが今日作られている」という基本的な観点を忘れている。ITがますますエスカレートするユーザの要求に翻弄されるため、ソフトウェア研究は構築するシステムの複雑性の枠を広げようと絶えず苦労が続いている。ソフトウェア研究開発のコミュニティは、以下に挙げる分野で、実践的な技術の進歩を担った功を認められてしかるべきである。

- モデル・ベースのコード生成、自動テスティング・スーツ、高級プログラミング言語、再利用可能なパターンやオブジェクト指向のツールなどに代表される近年の開発方法の進歩によって、今や小規模のチームで10万行を超えるコードからなる高品質のソフトウェア・システムを、以前の何分の一かの時間で作り上げることができる。わずか10年前までは、この規模のソフトウェアを開発するのは大変な難題と考えられていた。

- Webブラウザ、ユーザ・インタフェース・ジェネレータ、標準Javaクラス・ライブラリなどの適用範囲の広い商用アプリケーション・フレームワークは、CORBAコンポーネント・モデルやJ2EE（Java 2 Platform Enterprise Edition）や.NET Webサービスなどのプラグアンドプレイ・コンポーネント技術と結合し、データウェアハウジング、エンタープライズ・リソース・プランニング（ERP）、電子商取引Webサイトといった業務用デスクトップ・アプリケーションと企業アプリケーションの生産性を驚異的に向上させた。

- リアルタイムCORBA などの分散型オブジェクト コンピューティング ミドルウェアにおけるサービス品質（QoS）のサポートは、分散リアルタイム組み込みアプリケーション向けの設計生産性と信頼できるエンド・ツー・エンドQoSの実現とに大きな影響を及ぼし始めている。

- 大成功の事例として、エンドユーザがプログラムできるアプリケーション、ア

プリケーション固有のモデル・ベースのフロントエンド、プログラムジェネレータなどがあり、このような方向性への強力な動機付けとなる。

- ソフトウェア・ライフサイクル・プロセス・モデルが成熟し採用され、様々なプロジェクトの質が向上している。たとえば、Rational Unified Process やエクストリーム・プログラミング（eXtreme Programming：XP）のような標準的なプロセスは、開発時間とコストが削減され、大規模から比較的小規模なものまで幅広い規模のソフトウェア・システムの品質が向上することが予想される。

今後の課題

ビジネスサイクルがもたらす周期的景気の減速にもかかわらず、IT が今後勢いを持ち続けることは明らかである。プロセッサやネットワーキング技術の不断の進歩に後押しされ、ITの指数関数的急発展は今後10年間は続くと予想される。最も大切なのは、アプリケーション分野やグローバルな競争激化に対して、ITは大きな影響力を持っており、ITの浸透力と変革を起こす力を増大し続けることである。SDPワークショップのパネル・ディスカッションとブレークアウト・セッションにおいては、ソフトウェア技術をさらに大きく発展させるのに必要な、以下のような新しく台頭するアプリケーションの課題を明らかにした。

- **システムの統合**　おそらく、この10年で起きた「ITビックバン」の最大の影響は、「システム全体の統合者」という新しい役割をコンピューティングとソフトウェアに与えられたことである。ソフトウェア技術にこれが与える影響には、次の 2つの要素がある。1つは、コンピューティングとソフトウェアがアプリケーション分野とさらに強く融合していくことである。もう 1 つは、個々のソフトウェア技術間の大きな乖離であり、個々のソフトウェア技術が、新しいアプリケーションごとに異なる方向に成長し始めていることである。

- **インフラの重大な役割**　大規模ソフトウェア・システムは、銀行業務、医療機器、緊急サービス、電力供給、通信、輸送、国防、軍事などのインフラストラクチャとして、ますます重要な役割を担い始めている。これらは、そのほとんどが一瞬たりとも全体をシャットダウンしてはならないものであるため、監視機能、自己修復機能を備え、中断することなく常に進化するシステムを考案しなければならない。

- **リアルタイム組み込みシステム**　ペースメーカー、発電所の制御装置、飛

行技術に欠かせない航空電子工学システムなどの急成長アプリケーション分野は、物理装置やシステム内にそのインテリジェンスが組み込まれる。これらのアプリケーションは極めて密接に物理環境と繋がっているため、力学、騒音、電力消費、物理装置のサイズなどの物理面の要求や条件を満たしつつ、タイムリーに設計する必要がある。

- **ダイナミックに変容する分散型モバイル・アプリケーション**　コンピュータ間、あるいはコンピュータと物理装置間の接続が急増するにつれて、システムは「ネットワーク中心」の性格を強めてきた。ネットワーク中心の分散型アプリケーションは動的であり、常にトポロジーを変化させ、変化する環境に応じて機能と相互作用のパターンを適合させる。

新しい研究の方向性

主要なアプリケーション分野間の乖離は大きく、固有の課題が数多くあるため、すべての問題を解く「魔法の杖」とでも言うべき技術が存在しないのは当然である。それでも、複雑性の限界を押し広げようとする普遍的要求によって、新しい種類のアプリケーションが創られ、それがもたらす多種多様な課題は、今後10年間にIT研究開発コミュニティが扱うべき中核となる研究を定義することができる。以下にその課題を挙げる。

- **多面的プログラミング**　SDPワークショップ全体を通して浮上した一般的なテーマは、「階層的でモジュール化された今日の構成を多面的構成に拡張する必要性」である。アプリケーション分野がコンピューティングと緊密に融合し、深く統合されてきたことは、システムの根本的な特質がソフトウェアの影響を強く受ける、あるいはソフトウェアによって決定されることを意味する。その結果、ソフトウェア要件は多面的となる。つまり、コンピュテーションとソフトウェア・アーキテクチャは、多数の機能要件と物理要件を「同時に」満たさなければならない。多面的プログラミング構成の目的は、複数の設計上の側面を同時に利用、管理することを可能にし、それによって設計上の関心事を分離することにある。多面的プログラミング構成の主な課題は、信頼性、拡張性、効率性、安全性、柔軟性などの異なる設計上の側面の間で相互依存性を明快に表現すること、そして、様々な状況下で様々な目的に合ったシステムを自動的に構成する際に利用することである。多面的プログラミング構成は、手続き型言語とオブジェクト指向言語、あるいは宣言型モデリング言語という異なるレベルの抽象化に取り組む開発の枠組みを統合することが可能であり、それが実現されなければならない。

- **モデル・ベースのソフトウェア開発**　　ワークショップでの議論の結果、「ソースコードだけで設計、保守の工程を記述、管理するのは適切ではない」という統一見解が参加者の間で得られた。研究の主な目的の1つは、高レベルで分野に特化した抽象化を開発工程で利用し統合することである。高レベルで分野固有のモデリング言語は、設計の解析やソフトウェアの生成で直接利用できる形式的なものでなければならない。さらに、モデルを開発したり、モデルのないままに作られた膨大な量のソフトウェアに、モデルを後から関連付けたりすることを支援するツールが必要である。モデル・ベースのソフトウェア開発技術は、自分自身のモデルを利用して、自己監視、自己修復、自己適合、自己最適化など広範囲の新しい機能を提供するような、システム生成するための土台となるべきである。

- **構成可能でカスタマイズ可能なフレームワーク**　　ソフトウェア アプリケーションの開発の伝統は、融通のきかないばかでかい機能を実現することであった。そして、それは、理解、保守、拡張するのが困難な代物だった。こうした問題を軽減すべく出現した重要な技術が、「オブジェクト指向のフレームワーク」である。このフレームワークは再利用可能なコンポーネントで構成され、コンポーネントはカスタムアプリケーション用に構成したり、特化したりすることができる。また、フレームワークは、検証済みの設計やパターンを実際のソースコードに具体化することにより、コストを抑えてアプリケーション・ソフトウェアの質を改善できる。そして、それは、個々のクラスや個別の機能を再利用して実現できるものよりも大きな規模でソフトウェアの再利用を実現することができる。その結果、フレームワーク・ベースのアプローチを大幅に拡張し改善することが新たに重要な研究目的として浮上してきた。中心となる研究課題の1つには、通信、電子商取引、ヘルスケア、プロセスオートメーション、航空電子工学などのキーとなるアプリケーション分野を、再利用可能なフレームワークに再構成できるツールと技術を開発することが挙げられる。

- **インテリジェントで堅牢なミドルウェア**　　大規模でネットワーク中心のシステムについて議論した参加者は、このような種類の複雑なシステムを扱うには、新世代のインテリジェントなミドルウェア技術を開発、評価する必要があるという合意に達した。そのミドルウェア技術は、システムニーズをサポートすべく、利用可能なコンピュータとネットワークのインフラを常に最大限に活用することを目指さなければならないが、それによって生じるダイナミックに変わり続ける状況に対して、信頼して適応できるものでなければならない。新世代のミドルウェアとは、機能とQoSに関連した属性を静的にも動的にも変更できるソフトウェアである。ここで、「静的な変更」とは、たとえば、フットプリントを減らす、特定のプラットフォー

ムにある機能を活用する、機能をサブセット化する、ハードウェアとソフトウェアのインフラ依存度を最小限に留めることなどである。また、「動的な変更」とは、たとえば、変化するコンポーネント相互接続、電力レベル、CPU/ネットワークの帯域、レイテンシ・ジッター、信頼性ニーズ、といった変化し続ける環境や要求に対するシステムの対応を最適化することである。

- **ネットワーク化された組み込みシステムの設計**　　ITの基本的な動向の1つに、増大する組み込みコンピューティング[1]の影響力が挙げられる。組み込みシステムのソフトウェア開発は、その物理的な特性のために、より複雑となる。「実世界」に役立つプログラミングに注目する参加者は、この基本的に重要なシステム分野に関わる難しい課題を解くための大きな研究テーマを見つけた。その研究テーマには、開発工程にモデルを広く利用することや、自己適応をサポートし、危険な振る舞いを予防することを保証する新しい実行フレームワークが含まれている。また、軽いプロトコルや電力が限られたサービス、フォールト・トレラント（無停止型）アーキテクチャの開発を通して、リソースの限界の影響を管理することについて、よりよく理解するのも研究テーマの1つである。

- **ソフトウェアの協同開発**　　ソフトウェア市場の基盤である経済の変化に伴って、企業は組織を再編し、事業目標を再定義し、個別にソフトウェア開発を行う。そのツールと枠組みは、次のような内容をサポートするものでなければならない。すなわち、（1）様々な専門分野や地理的に分散した開発チームによるソフトウェア協同開発、（2）複数の時差の異なる地域に分散しているのが日常化しているソフトウェア開発チーム、また、エンドユーザとそれぞれの分野の専門家（生物学、心理学、医学、財政学など）とで構成されているソフトウェア開発チーム、（3）分野モデルとビジネスモデルの強化をサポートし、かつ多面的な設計や価値重視の開発を追求することを支援する開発工程、（4）初期の要求収集からリリース後の保守まで、ソフトウェアのライフサイクル全体に渡って、これら協同開発の発展、継続に対する支援、などである。

- **システムとソフトウェアの共同設計環境**　　実社会のシステムにおいて増大しているコンポーネントや相互作用がますます「コンピュテーショナル」になるにつれ、システムやソフトウェアの設計は、非常に複雑に結合してきている。設計の生産性を高めるには、システムとソフトウェアの共同設計の基礎となる新しい科学を開発

[1] 『Embedded Everywhere』、National Research Council（NRC）著、D. Estrin 編集、2001 年

する必要がある。また、広範なシステムやソフトウェアを解析、統合する様々なツールと、分野固有のシステムやソフトウェアを共同設計する環境を整えるための技術も開発する必要がある。さらに、この新しい設計環境では、ソフトウェアやシステムの開発者がすばやくソフトウェアを変更し、それを実施あるいはシミュレータで検証し、更新後のソフトウェアを試し、そして、その工程を繰り返すことができるものでなければならない。

結びの言葉

ソフトウェア開発の究極のゴールは、許容される範囲内の時間と労力で高い品質のシステムを作り出すことにある。ソフトウェア研究は、コストの制約があるため、純粋に理論的なままである。つまり、実験データに基づかず、現実の状況下で検証されていないため、科学的な信憑性を欠き、実践者からは理論の域を出ないとみなされてしまうだろう。我々は、実験的で強力なコンポーネントを充分に盛り込むように、国家のソフトウェア研究戦略を再構築しなければならない。ソフトウェア研究者は、自らの研究を現実的な設定で評価し、より複雑なシステムに適用させることを容認されるべきである。また、人が関与するシステムにもっと注目すると共に、現存あるいは計画されている大規模システムを詳細に研究し、多様な技術が様々に作用する実態とその原因の解明にも努めなければならない。

この10年間、ITは米国の経済競争力を強める上で、（おそらく最も）重要な要素であった。ITを利用して積極的に企業活動を自動化したことが、この期間の米国経済に驚異的な生産性の向上をもたらしたというのが大方の見方である。世界の他の先進諸国も、米国に追いつこうと努力しており、ITの広まりとその変革を起こす力が今や企業における自動化にとどまらず、製品やインフラ、教育をはじめとする多くの分野にも急速に広まりつつあることは、もはや疑いの余地がない。

米国のITがもたらす経済的メリットの維持、発展に向けて、新しいアプリケーション分野とIT技術との関係を理解するための投資を増やさなければならない。さらに重要なのは、新たな機会を開拓し我々の戦略の優位性をさらに高めるため、新しい手法やツールを積極的に追求していかなければならないという点である。エンドユーザをとりまく業界には、そのような変革を自ら起こす専門知識もリソースもない。その一方で、優位に立つソフトウェア業界には、現状を変えるリソースがない、あるいは変えることに関心がない。政府がより多くの資金をITに投じることが、この動きを推進する上で不可欠なのである。米国の長期的な経済繁栄と安全保証のために、活力があり、積極的なIT研究コミュニティを米国に創出し、維持することが肝要である。

ソフトウェアとソフトウェア研究の将来

ブレークアウト グループ サマリー

作成者: Adam Porter

パネリスト: 議長 Adam Porter、
Barry Boehm、Bob Neches、Daniel Jackson、Gabor Karsai

ブレークアウト セッション参加者: Richard Buchness、Laura Dillon、Susan Gerhart、Susan Graham、Alkis Konstantellos、Steven Ray、Rose Gamble、Helen Gigley、Steven Reiss、William Pugh、Eric Dashofy、Philip Johnson、Marija Mikic-Rakic、Kenny Meyer

はじめに

ソフトウェア研究の焦点は、複雑なソフトウェア・システムの設計、構成、実現、進化の質と生産性の向上に役立つ手法、技法、工程にある。今日構築されているソフトウェア・システムは、20年前のものとは劇的に変化している。したがって、その変化を反映するために、どれほどソフトウェア研究が進化してきたかは驚嘆に値する。このパネル・ディスカッションでは、以上の内容について調査し、ソフトウェア研究が中・長期的にどのように変化していく必要があるかを考察した。

今日の我々の位置付け

今ではあまり意識されることもないが、ソフトウェアは我々の日々の生活全般に浸透している。今やソフトウェアは家電から自動車、街灯、オフィス警備システム、ビジネスや個人の通信、銀行業務、医療機器、救急サービス、電力供給、科学上の発見、国防、軍事システムの中にまで入り込んでいる。30年前は、ソフトウェアはこれほどまで普及していなかった。このこと自体、ソフトウェアが大成功を収めたことの「証し」である。しかし今後を展望すると、ソフトウェア技術の抜本的な進歩なくしてこれまでのような成長が続きえないことは明らかである。特に、現在のソフトウェア開発技術は複雑性を極めており、投資効果があって安全で信頼できる、または変更が容易で可用性の高いシステムを提供することに限界がある。だが、そうしたシステムは我々の生活の質を高める上で必要不可欠であり、いずれは自動交通管理、相互連結型の医療機器（人の体内に埋め込まれる装置を含む）、緊急サービスと非常時のサービスの協調管理、微小規模のロボット利用サービス、戦場の自動管理などを実現してくれる可能性がある。

成功を収めたこと

このようなシステムを実現するためには、どのような進歩が必要かを理解するには、まず、ソフトウェア研究の現況を評価する必要がある。現在のソフトウェアの実情を20年前と比べると、ソフトウェア研究が劇的なプラス効果を及ぼしてきたことは明らかである。例を挙げると、以下のようなものがある。

- **著しい生産性の向上**　高品質で複雑なソフトウェア・システムの構築にかかる時間は、以前に比べて短縮されている。実際、現在の開発の現場では、10万行を超えるコードからなるシステムを日常的にごく当たり前に扱っている。このようなシステムを作ることはほんの10年前まで大変な難題と考えられていた。こうした生産性の向上は、繰り返し行われるベスト・プラクティスによって支えられている。開発者は、様々なクラスのシステムに対して、予測できるスケジュールと質でもって、標準的な工程にて作業することができる。このことは、ソフトウェアのライフサイクルを通して用いられる工程、手法、ツール類の研究の成果である。たとえば、次のようなものである。

  - 要求把握のための技法と言語（ユーザ中心の設計、Z、DOORS など）
  - 設計原理、言語、記号（情報隠蔽、オブジェクト指向設計、Java、UML など）
  - ソフトウェア・アーキテクチャ・スタイル（C2、クライアント/サーバ、イベントベース プログラミング）
  - 分野に特化した言語（Matlab、Simulink、Splus、Genoa、Lex＆Yacc など）
  - ソフトウェアの検査とテスティングのための技法とツール（FTArm、選択的リグレッション・テスティング、プロトコル適合テスティングなど）
  - ソフトウェア工程サポート（構成管理システム、CMM、プロセス・モデリング言語など）

- **新しいコンピュテーショナル モデルへの適応の成功**　コンピューティング・ハードウェアの進化と共に、ソフトウェア開発技術も進化してきた。その結果、バッチ処理から対話処理、分散システムからシングル・プロセッサ・システム、テキストベースのインタフェースからグラフィカル・インタフェースなど、新しいコンピューティング環境に適したソフトウェアをより簡単に作ることができるようになった。以下のテーマに関するソフトウェア研究によって、こうした機能が適用できるようになった。

  - オブジェクト指向のモデリングと実現（UML、パターンとパターン言語など）

・ ミドルウェア（DCE、CORBA、COM+、J2EE、.NET など）
・ GUI ジェネレータ（PowerBuilder など）

- **ソフトウェアの浸透**　今やソフトウェアは、国家の基本インフラストラツチャであり、エネルギーや道路、上下水道と同じくらい重要なものである。日常生活で必要な無数の装置が、コンピュータの力で稼動している。航空管制や医療技術、工場の自動化など、安全第一のアプリケーションにまで利用され、その一部として受け入れられている。安全第一のソフトウェアの成功は、次のようなソフトウェア研究が直接要因である。

  ・ ソフトウェアの安全性（状態図、エラー ツリーなど）
  ・ 分析と検証（モデル チェッカー、タイプ チェッカー、Lint、Purify など）
  ・ ソフトウェア・テスト（信頼性モデリング、テスト・カバレージ アナライザなど）

- **限定分野での（部分的）自動化ソリューション**　ソフトウェア開発者は、コードの生成、再利用によって、少ない時間と労力で開発ができるようになってきている。設計ツールを用いて、システムのかなりの重要部分を作ることも少なくない。データベース管理やネットワーク通信のような仕事では、ソフトウェア開発者は高品質で系統立った再利用可能なフレームワークとコンポーネントを利用することができる。たとえば、次のようなものである。

  ・ モデル・ベースの開発ツール（Promula、Rational Rose、ObjecTime、GME、MatLabなど）
  ・ アプリケーション・フレームワーク（JavaBeans、SAP、ACEなど）

勝利宣言するには何が必要か

この20年間の著しい発展にもかかわらず、ソフトウェア開発は依然として難しい。構築したいソフトウェアと、今日の技術の成熟レベルで構築できるソフトウェアとのギャップにはいくつかの要因がある。その要因は、技術面と社会面、理論面と実践面など多岐にわたっている。具体的には以下のようなものである。

- **ソフトウェアの普及と重要性**　ソフトウェアは、国家のインフラストラクチャの根幹をなすようになった。その結果、「書かなければならない」ソフトウェアの分量が著しく増加した。さらに、ソフトウェアの大部分に求められる信

頼性と安全性の要件は、ますます厳しくなってきている。

- **変化のスピードの速さ**　情報技術は大変なスピードで変化している。そのことはソフトウェアにも当てはまる。たとえば、スタンドアロンのコンピューティング・システムのネットワーク化が進むにつれて、分散型のコンピューティング環境をサポートするソフトウェア開発技術が開発されてきた。同様に、価格の低下により、組み込み装置やセンサーの普及が進むにつれて、これらの分野をサポートする開発技術や開発環境が生まれた。さらに、ソフトウェア開発の経済環境にも影響を与えており、これらの変化がより新しいソフトウェア開発技術を生んでいる。ソフトウェアは、今や「日常品」となりつつある。利ざやは縮小され、ライフサイクル・プロセスと開発手法は、「モノリシック（堅牢）」であることよりも「アジル（敏捷）」であることが求められる。

- **プログラムの規模**　ソフトウェア・システムの規模と複雑性は増大している。特に、ソフトウェア・システムは地球規模でグローバルに接続され始めている。その結果、多数の相互接続されたシステムからなるシステムが作り出す膨大な同時並行処理を管理する必要性が生まれる。さらに、これらのシステムからなるシステムを構成する個々のシステムをコスト効果の高い方法で同時に設計することはできないため、その統合が非常に大きな課題となってきた。大規模ソフトウェア・システムの統合にからむ様々な問題により、将来のシステムは白紙の状態から作るのではなく、変化し続けるコンピュータ環境や実行時の構成に適合するように設計にしなければならないことが多くなるだろう。

- **コンポーネントに対する理解不足**　大規模ソフトウェア・システムは、ますます既存のコンポーネントで構成されるようになってきている。コンポーネントを再利用すれば、初期システムを短期間で構成することができるが、各コンポーネントが柔軟でなかったり、カスタマイズ可能でなかったりすると、別の問題を抱えることになる。たとえば、コンポーネントを利用する開発者は往々にして、全体のシステム性能を予測することができない。これは、コンポーネントが予測されない作用を相互に及ぼすことがあるためである。また、コンポーネントは平均的な条件に合わせて最適化されることが多いが、その条件は個々のアプリケーションを実行する条件とは異なるものである。また、コンポーネントのインタフェースがアプリケーション要件の様々な変更をサポートしていなければ、コンポーネント・ベースのシステムを変更することは困難となる。

- **ベスト・プラクティス採用の遅れ**　研究の結果、ソフトウェア開発

の実践の本流は、ソフトウェア研究あるいはトップのソフトウェア実践者が採用するベスト プラクティスからはるかに遅れをとっていることが繰り返し明らかになっている。実際のところ、個々の技術が研究から実践に問題なく移されるまでに20年以上もの歳月を要することも少なくない。これではエンドユーザのますます高まる要求に応えるには、あまりにも遅すぎる。

- **ソフトウェア・テストの生産性の低さ**　現在の開発技術では、ソフトウェアの振る舞いが保証されないため、テストは今もソフトウェア開発の中で重要な役割を担っている。「作って直す」開発技術あるいは「自分で直す」（自己修復型）ソフトウェアを生み出せない限り、テストの低い生産性がソフトウェア開発工程において、深刻なボトルネックとなる状況は今後も続くであろう。

重要な研究開発の課題

これまで述べてきた一般的な問題を解決するには、ソフトウェア研究でいくつかの重要な課題に取り組まなければならない。

- **コンピューティング環境の進歩**　ソフトウェア システムが相互に作用する物理的な要素と、ソフトウェア システムの実行環境は、ますます複雑化している。ソフトウェアは将来、従来のシングルCPUによるデスクトップ環境から、マルチホストによるネットワーク化されたシステム、モバイル システム、組み込みシステム、ウェアラブルコンピュータ（着用するだけでなく、人工内耳やペースメーカーなど体内組み込み型のものも含む）に至るまで、幅広いコンピューティング環境の中で実行されることになるだろう。このような進んだ環境には、次のような特徴がある。

  - 「高度な分散化と同時性」　ソフトウェアは、ますます多数のホストマシンで実行され、ホストマシン同士はネットワーク間、あるいはネットワークから成るネットワーク間で通信を行う。

  - 「実世界を感知し場所を認識するコンピューティング」　ソフトウェア・システムは、物理的な状況を認識するようになる。実世界を感知するセンサーからのインプットを受けて、実世界の環境を管理するアウトプットを出す。また、他のコンピューティング リソースが近くにあればそれを認識し、その情報を使って周りのリソースがとる振る舞いを誘導する。

- 「プラットフォームの多様化」　　ソフトウェアは、多様な言語、OS、ハードウェアなどの特性を持つコンピューティング・プラットフォーム上で実行され続けるだろう。
- 「コンポーネント・ベースのシステム開発」　　地球規模の競争激化と製品化に要する時間のプレッシャのために、次世代の複雑なシステムの開発者はこれまで以上にソフトウェア・コンポーネントを購入し、構築し、それらを統合してシステムを作るようになるだろう。ソフトウェア・コンポーネントは、比較的小さな再利用可能なクラスのものから、大規模のフレームワークやサブシステムまで様々である。
- 「多様な方式」　　ソフトウェア・システムは日常的にインプットを受け取り、音声、触覚、電気インパルスなどのテキストベース以外の方式でアウトプットを書き出す。

- **統合のメカニズム**　　大規模システムは、複数のシステムで構成される傾向を強めており、このような「システムからなるシステム」の統合を理解し促進することが非常に重要になっている。特に、「システムからなるシステム」全体としてのサービスの質など、システム全体の特性を保証する手法を開発する必要がある。

- **可変性**　　システム統合の必要性に加えて、コンポーネントあるいはコンポーネントの相互作用を変更するためのサポートを改善する必要がある。たとえば、システム全体を停止させることなく、あるコンポーネントをアップグレードしたり、現在のバージョンのコンポーネントをシステムから削除して（どこでどれほどの規模で実行されていようとも）、コンポーネント・システムのインスタンスをすべて置き換えたりできなければならない。

- **フォルトトレランス**　　各コンポーネントや接続されたシステムに問題があり、その振る舞いが停止してしまった場合でさえも、迅速に回復する堅牢なシステムを作れるプロトコルやメカニズムを考案する必要がある。コンポーネントは、相互作用のあるコンポーネントが停止したり、性能が低下したり、無効なインプットを与えられた場合であっても、動作し続けることができなければならない。

- **変化するソフトウェア開発者、工程、ツール**　　ソフトウェア市場の基盤となるエコノミクスは変容し続けており、企業組織のあり方やビジネス・ゴール（目標）の設定の方法、ソフトウェアの開発方法などを変化させている。以下にその例を挙げる。

- 「様々な専門分野の、地理的に分散した開発チーム」　　ソフトウェア開発チームは今や複数の時差の異なる地域に分散し、エンドユーザとその分野の専門家（生物学、心理学、医学、財政学など）とで構成されるのが常である。開発チームは、分野モデルとビジネスモデルを一層強化しなければならない。
- 「製品化に要する時間の重視」　　機能性と正確性は重要であるが、ソフトウェアの購入者の多くは、待たずに今すぐ手に入るソフトウェアの方を選ぶ。
- 「段階的に開発し、変更の伝達を頻繁に行う」　　製品化に要する時間のプレッシャが大きく、コードの更新が分散されるために、開発者は変更を小さくしてシステムを頻繁に更新し、リグレッション・テスティングを行い、変更したコードをユーザに伝達（配布）しようとする。
- 「迅速な開発と試験」　　ソフトウェア開発者は、ソフトウェアをすばやく変更し、それを実施やシミュレータで検証し、更新後のソフトウェアを試さなければならない。さらに、この工程を一貫して繰り返さなればならない。

- **ツールのサポート**　　大規模なコンピューティング力を手にしたプロの開発者は、非常に強力なサポート ツールを適用することができる。たとえば、次のようなものがある。

  - 「テスト・ツール、解析ツール、検証ツール」　　ソフトウェア開発者は、増大するコンピューティング力を活用することができる。ツールには、モデルチェッカー、レグレション・テストや機能テストの生成ツールと実行ツール、設計評価、プログラム解析ツール、および意味認識ツールなどがある。
  - 「コラボレーション用ツール」　　開発者には、協同作業、設計、変更の統合、自動化されたテスティング、解析と検証、そして要求分析と設計仕様や再利用などについて、より多くのサポートが必要である。
  - 「ソフトウェア工程のサポート」　　ソフトウェア開発組織は、開発工程をモデリングし、理解、改善するためのサポートを必要としている。性能データを記録、解析し、性能を改善し続けるためのより一層のサポートも必要としている。

- **完全に捨てられない既存コンポーネント**　　技術が進歩しても、既存のソフトウェアと開発ツールへの投資をやめることができないために、ソフトウェア開発者が技術革新のメリットを直ちに受けることができないことは多い。その結果、新しい技術で書かれたソフトウェアを古い技術で書かれたソフトウェアと連携しなければならないことも多い。従来のソフトウェアの保守は欠かせないのである。

研究の課題

これまで述べてきた研究開発の課題に取り組むためには、ソフトウェア研究者は次の分野で成果を上げなければならない。

- **複数の目的を持つ技術の設計**　ソフトウェア研究者はこれまでずっと、製品化に要する時間やコストの制約などよりも優先して、機能の正しさを重視してきたが、現在多くの技術が別の基準から設計し直されている。たとえば、次に挙げる新技術では、経験則を導入することを認め、むしろ奨励し、理論上の正しさよりも実際に有用であることが優先されるのである。

  - 「価値志向の技術」は、開発者が利益がコストを上回るような場面、配分で、つまり経済的に意義のあるような形態で、技術を適所に活用することをサポートする。たとえば、ソフトウェア設計上の結論を今すぐ出すことの価値と、留保することの価値とを定量化する方法を研究し始めた研究者らがいる。
  - 「人志向の技術」は、人間の限界や強みを考慮する。たとえば、形式的仕様の技術は昔から複雑な数学の記号に頼ってきたが、最近になって、新しい形式的仕様の記述は、コンピュータの専門家以外の人が簡単に読めることを目的として、考案されている。
  - 「機を見て適用される技術」は、ライフサイクルを通して断片的な軽い技術を戦略的に適用する。そのような技術は大抵、本質的に経験主義的であり、理論的な裏付けもなく、適用範囲が限定されたものである。たとえば、設計リファクタリング技術では、最初から一度にシステムを設計するのではなく、コーディング開始後にソフトウェアの設計を改良することを重視している。
  - 「部分的に適用される技術」は、プログラム全般に対してではなく、ソフトウェア工芸物のある部分に対して適用される。たとえば、最近行われたソフトウェア解析では、システム動作モデル全体を捉えるのではなく、注目されるごく一部のモデルを調べることに集中している。

- **実証的な評価を大規模に実施**　ソフトウェア開発は、実践で行われるものである。開発の現場と接点を持つことで、ソフトウェアの研究上得られるものは大きい。ソフトウェア研究者は、自らの研究を現実的な状況の中で評価し、大規模システムに適用することに一層努めなければならない。また、人に関わるシステムにより注目し、既存の大規模システムを徹底的に研究すべきである。

- **分野固有の機能を開発言語に組み込む試み**　開発者は、プログラムの抽象化のレベルをひたすら高めることが求められる。プログラミング言語の研究が進み、信用管理、セキュリティ、電力管理、タイミング重視の組み込みシステムなど、ハイレベルなアプリケーション固有の概念を持つ高級言語が開発され始めている。

- **協同作業によるソフトウェア開発**　離れた場所に分散する開発者が同時に作業しやすくするような新しい技術やツールが考案され始めている。たとえば、分散する開発者が不都合な競合を発生させることなくシステムをより簡単に変更できるように、システムを自動分解する方法を検討しようとする研究が行われている。また、マルチプラットフォーム・システムのテストとプロファイリングの分散化をサポートする研究も進められている。

- **向上したマシン性能を活用**　コンピュータに大きな負荷のかかるモデル チェッキングなどの技術に関わる研究は、ソフトウェア分野ではまだ実務規模のプログラムにまで発展しておらず、あまり成功しているとは言えない。しかし、コンピューティング性能の向上により、こうした技術をより多くの場面に適用することが可能になってきている。実務規模のプログラムに使える技術までもっていく方法に集中して、新たに研究されている。

- **設計とコードの橋渡し**　ソースコード・レベルのプログラミングには、拡張の余地がない。設計レベルのソフトウェア工芸物のモデリングと解析のためのツール研究が行われている。分野ベースのモデルやツール（つまり、アプリケーション固有の情報を表現、操作するもの）は、組み込みシステムのような研究の進んだ分野で一定の成果を上げている。

- **新しいコンピューティングの枠組みを探求**　生物学、量子学、市場ベースのコンピューティングなど、新しいコンピューティングの枠組みの開発が進められている。ソフトウェア研究では、これらの枠組みを研究し、枠組みの中でのプログラミングを支援するツールを開発している。

- **ユーザがアクセスできる技術**　「プログラマではない人」によるソフトウェア開発が増えており、これに対するサポートが必要である。プログラマではない人が様々な視点からシステム要求を記録、検討したり、プログラムを開発、テスト、変更したりすることをサポートする技術が求められる。

- **SE教育と訓練技術の向上**　　ソフトウェア技術の学習曲線を短縮できる新しい技術を開発する必要がある。こうした技術は、様々なレベルのIT関係者に向けられなければならない。また、ソフトウェア設計や生産性の向上を目指して開発される新しい手法や技術をマスターするのに要する学習時間を大幅に短縮することにも重点を置かなければならない。

結びの言葉

ソフトウェア研究は、これまで重要で有用なソフトウェアを現場に供給する我々の能力に非常に大きなプラス効果を及ぼしてきた。しかし、その成功は、もっぱら消費者の欲求を増大させ、より困難な新しい課題を生むばかりだった。こうした課題に取り組むため、ソフトウェア研究者たちは壮大で野心的な研究課題をまとめ上げた。特に、大規模化するソフトウェア・システムに見合うように研究の規模を大きくする必要がある。たとえば、将来のソフトウェア研究には、次の各項について国の資金提供機関からの投資が必要である。

- **マルチ分野の大規模なテストベッド**　　ソフトウェア研究者は、ソフトウェアの設計と生産性を向上させる新しい技術について研究を行うために、非常に大規模なソフトウェア工芸物にアクセスできなければならない。

  - 「ソフトウェア考古学」　　研究者が問題がどこで実際に発生するか、どの位頻繁に問題が発生するかを理解するために、実務規模のソフトウェア工芸物が必要である。
  - 「ソフトウェア工芸物」　　ソフトウェア研究者の研究評価のため、ソースコードだけでなく、ソフトウェアのライフサイクル全体にわたる工芸物（要求仕様、設計、コード、テスト・ケース、テスト・ケースのアウトプット、構成管理システム（CMS）ログ、バグ・データベース）が必要である。
  - 「研究を可能にするオープンソース・ソフトウェア・プロジェクトへの出資」資金提供機関は、後の研究者達に有用な工芸物を作るという条件を満たせば、一部のオープンソース・プロジェクト（ともかく、工芸物の多くを公開している）に対して援助を与えてもよい。また、資金提供機関は、出資プロジェクトで工芸物が充分活用されるように働きかけることもできる。

- **実践パイロット・プロジェクト**　　実務開発者と交流できる大規模なデモンストレーション・プロジェクトに出資することは個々の有望な研究技術にとって望ましいことだろう。また、物理学、生物学、医学、MEMSなど、ソフトウェア

以外で出資対象の研究分野との共同研究にソフトウェア研究を受け入れることも望ましい。

- **研究員の増強**　　多くのソフトウェア研究プロジェクトは、資金がプログラミング支援要員を受け入れるレベルに満たないため、開始前からその成果が望めない状況にある。

- **プロを育て、増やす必要性**　　大学院生はソフトウェア研究を避け、サイエンティフィック・コンピューティングなどの潤沢な資金が手に入る他の分野を専攻する傾向が強い。このため、高度な訓練を受けた若いソフトウェア研究者が不足している。このワークショップの構想を実現するには、ソフトウェア研究に携わる大学院生の数を大幅に増やす必要がある。

新しいソフトウェア開発の枠組み

ブレークアウト グループ サマリー

作成者: Janos Sztipanovits、Doug Schmidt 、 Gregor Kiczales
パネリスト: 議長 Gregor Kiczales
Don Batory、Ira Baxter、James Larus、Charles Simonyi

ブレークアウト セッション参加者: Karl Crary、Shriram Krishnamurthi、Paul Hudak、Frank Sledge、Karl Lieberherr、Kurt Stirewalt、Spencer Rugaber、Tzilla Elrad、Benjamin Pierce、Prem Devanbu、Tom McGuire、Doug Smith、Joy Reed、Mike Mislove、Ralph Johnson、Janos Sztipanovits

はじめに

情報技術(IT)が科学、技術、経済のあらゆる分野に急速に広まるにつれ、数も種類も急増するコンピュータ・アプリケーション用のソフトウェアを作るプレッシャも激増した。この数十年間に、アーキテクチャのスタイルやパターン、再利用可能なオブジェクト指向のフレームワークやコンポーネント・ライブラリ、高級プログラミング言語、新しい開発プロセス（エクストリーム・プログラミング(XP)やオープンソース）など、ソフトウェア技術は著しく向上した。こうした技術の進歩は、ソフトウェア生産性を大幅に向上させたが、個々の進歩には限られた範囲の効果しかなかった。ますます高まるユーザの要求レベルが、進歩のもたらす恩恵をすぐに打ち消してしまうためである。
ソフトウェアの生産性と品質は、「ソフトウェア開発の枠組み」に大きく依存している。開発の枠組みは、設計手法、プログラミング言語、コンピュテーション・モデル、正しいことの証明と検証技術、開発プロセスで使用するツールなどの総体によって特徴付けられる。以下に例を挙げる。

- 設計手法には、流れ図のような非形式なものから、状態図のような形式なもので解析可能なものまである。
- プログラミング言語は、プログラムを書くプログラマの使用する記述や抽象化、構成技術を決定する。構成手法の違いによって、プログラミング言語は、構造化プログラミング、オブジェクト指向プログラミング、アスペクト指向プログラミングなどの大きく異なる様々なプログラミング・スタイルをサポートする。
- コンピュテーション・モデルは、通常、機能的、論理的で命令的である。正しいこ

との証明と検証の役割は、適用された環境でソフトウェアが機能要件とそれ以外の要件を満たしていると保証することである。
- モデリング、解析、デバッギング、テスティングは、正しいことの証明と検証の工程の中でいずれも重要な要素である。

無論、開発の枠組みの他の要素もソフトウェアの生産性と品質に強い影響を与える。現在の開発の枠組みでは、設計における予測可能性があまりにも低く、そのため、正しいことの証明と検証に要するコストは開発コスト全体の50パーセントないし75パーセントにも達することがある。したがって、開発の枠組みを改善することは、ソフトウェア開発者の生産性とアウトプットの質を向上させる鍵となる。

開発の枠組みが成熟し採用されるまでには莫大なコストがかかるため、標準化と長期的な安定性が求められる。成功した開発の枠組みは、長い年月をかけて進化するが、世の中に広く受け容れられても、10年を超えてその地位を維持できるとは限らない。不幸なことに、その有効性は適用対象ソフトウェアの特徴をいかにうまくサポートできるかに大きく依存する。たとえば、構造化プログラミングの枠組みは、中規模のビジネス・アプリケーションには有効であるが、大規模な分散化したリアルタイムの組み込みシステムに求められるサービスの機能や品質に対応してうまく拡張していくことができない。ありとあらゆる分野の隅々にまでITが普及し、各分野に変化を及ぼしたことを考えれば、構造化プログラミングやオブジェクト指向プログラミングなどの現在有力なプログラミングの枠組みが、多くの方面で課題を抱えているのも不思議ではない。

パネルの目的は、ソフトウェア開発の枠組みにおける新しい進歩の動向を見極め、それが新しいアプリケーションの課題にどう関係するかを議論することであった。

機会と課題

いずれにせよ、IT が広く浸透したことは、ハードウェアとソフトウェアの設計に関する長年の研究開発の大きな功績であり、賜物である。しかし、皮肉なことに、新しい機会を生んだ成功が、同時に既存のプログラミングの枠組みを急速に時代遅れにしてしまうという新たな難題をもたらしてしまった。そして、不幸にも、この点が「ソフトウェアは進歩していない」という印象の元凶となっている。かくして、コンピュータ・プログラミングの幕開けからずっと慢性的に「ソフトウェア危機」に悩まされるという図式ができ上がっている。以下に、その最新の例と関連する課題を掲げる。

- **アプリケーション分野への融合が進むソフトウェア**　多くのアプリケ

ーション分野で、コンピューティングは複雑なものを収める重要な貯蔵庫となり、新しい機能の主な源となった。たとえば、自動車業界の技術革新の90パーセント以上が組み込みコンピューティングによるものである。コンピューティングの重要性がますます高まっているということは、アプリケーション分野固有の特徴がプログラミングの枠組みにじかに反映されなければ、アプリケーション・エンジニアリングにおいて考える内容は汎用のソフトウェア エンジニアリングの概念やツールに手作業で割り付けなくてはならないことを意味し、そのような作業は実に骨の折れるエラーの起こりやすいものである。この手作業による割り付けプロセスは多難であるので、分野固有の言語やモデリング・ツール、ミドルウェアなどの念入りにカスタマイズされた機能がますます必要となってくる。

- **システム全体を統合するソフトウェア**　おそらく、この10年で起こった「IT ビックバン」の最大の影響は、コンピューティングとソフトウェアが「すべてのシステム統合者」という新たな役割を担うようになったことである。システムは相互に作用するコンポーネントで構成されている。新しい傾向としては、実世界に存在するシステムの多くのコンポーネントとコンポーネント間の相互作用とがますます「コンピュテーショナル」になってきていることである。今や SAP（ドイツ製の最も使われている業務用ソフトウェア・パッケージ）のような複雑な情報管理システムが組織を円滑に機能させ続ける「ITバックボーン」を提供し、分散管理されたプロセス自動化システムが製造業の生産ラインを統合している。また、航空管制と航空電子工学システムが航空機の運行を支えている。このような変化は次の2つの影響をもたらす。1つは、コンピューティングとソフトウェアがアプリケーション分野にさらに強く融合していくことである。もう1つは、現場が新しいアプリケーションの方向ごとにますます成長し続けるため、個々のソフトウェア技術間に大きな乖離がおこることである。

アプリケーション分野がコンピューティングとより一層融合し、深く統合されてきたことは、システムの根本的な特性がソフトウェアの影響を強く受ける、あるいはソフトウェアによってまさに決定されることを意味する。その結果、ソフトウェアの要件は多面的となる。すなわち、コンピュテーションとソフトウェア・アーキテクチャは多数の機能要件と物理要件を「同時に」満たさなければならないのである。

進歩した分野

この10年間はプログラミングの枠組みが顕著な成功を収めた時代であり、それがIT革命を牽引した。その主な進歩には次のようなものがある。

- **フレームワーク・ベースの設計**　　大規模アプリケーションのコスト削減に最も成功したアプローチは、新しいコードを書く必要性を最小限に留める方法である。大学の研究者と企業ベンダーは、この目的を達成するため、再利用可能なアーキテクチャを関連アプリケーション・ファミリーに提供し、協調動作する統合されたクラス群を備えたアプリケーション・フレームワークを作り上げた。Visual Basic、Java、C++、PERL など様々な言語を使って少ないライン数のコードを書くだけで、ユーザは再利用可能なフレームワークをカスタマイズして、複雑なアプリケーションを構築できる。大規模なフレームワークを高度に再利用した例では、SAPをR3（データベース・システムの名称）上で動作させるものがある。そのベースコードは4.5GB（ギガバイト）にも及ぶ。そのカスタマイズ作業は、そのおよそ1パーセントあるいはそれよりも少ない分量のコードに変更を加え、1パーセント未満のプラグインを使用するだけでよい。興味深いことに、通常のアプリケーションに使用するコードの量は、ベースコードの約10パーセントにしか満たない。フレームワーク・ベースの設計は、テストされ尽くした既存のコードベースを大々的に再利用することで成功している。比較的小規模の拡張を慎重に行えば、設計全体の整合性を損なうことはありえないため、正しいことの証明と検証の作業量を最小限に抑えることができる。

- **コンポーネント・ベースのミドルウェア**　　コンポーネント・ベースのミドルウェアは、それを再利用して大規模アプリケーションを構成できる特殊なビルディングブロック サービス群をカプセル化している。プログラミング言語レベルでは、コンポーネントはモジュール、クラス、オブジェクト、さらには関連する関数群として記述することができる。コンポーネント技術は、大規模アプリケーション分野において、構成可能性と相互運用性に著しい進歩をもたらした。商用コンポーネント・ミドルウェア（CORBA、.NET、J2EE など）は、「水平」インフラ・サービス（ORB、インタフェースとサーバ・リポジトリ、トランザクションなど）、分野共通の概念である「垂直」モデル（コンポーネント間で共有のドメイン・セマンティクス）、そして、コンポーネント同士の「コネクタ」メカニズム（メッセージ、イベント、ワークフロー、位置の非依存性）を提供する。現在のコンポーネント・ミドルウェア技術は、小規模アプリケーション（GUI など）内や、いくつかの粗粒子コンポーネント（2層ないし3層からなるクライアント・サーバ型業務アプリケーション）ではうまく機能する。

- **モデル・ベースのソフトウェア・エンジニアリング**　　設計を記述する道具としては、ソースコードは貧弱であることが次第にわかってきた。モデル・ベースのソフトウェア・エンジニアリングは、流れ図などの設計を記述する非形式的な技術に始

まって、より形式的で豊かな意味を持つ高級な設計言語へと移行している。また、パターンやパターン言語、アーキテクチャ・スタイルを通して設計の中核となる側面を系統立てて捉える方向へと移行している。ごく最近では、モデリング技術はアプリケーションの機能だけに注力するのではなく、アプリケーションのサービスの質（QoS）の要件（たとえばリアルタイム処理の処理時間の制約、依存性の制約など）をも規定するようになってきている。こうしたモデル・ベースのツールは、アプリケーションの開発者やインテグレータに、従来の命令的なプログラミング言語よりも高レベルの抽象化と高い生産性をもたらしてくれる。

- **生成的プログラミング**　　高級な設計言語やモデリング ツールへと移行すれば、当然、コードの生成と統合の過程をますます自動化していくことが生ずることになる。生成的プログラミングの目的は、仕様と実装との間のギャップを埋めることである。それには、個々のミドルウェアやアプリケーションの特徴（たとえば、トランザクションの分離レベル、障害時のバックアップ サーバの性質、認証と承認に関する方針など）に合うようにカスタマイズされ、プラットフォーム依存のコードを合成することができる洗練されたアスペクト・ウィーバーや生成ツールが利用される。この分野の初期の研究成果が、限られた、明確な分野をサポートする商品として、現在利用され始めている。たとえば、MathWorks社のツールであるSimuLinkとStateFlowは、高度なモデルで信号処理を生成して、アプリケーションを制御する。生成的プログラミングがもたらす生産性の向上には、目を見張るものがある。部分的に導入した場合でさえ、ユーザの生産性は40パーセントないし50パーセントも向上している。

勝利宣言するには何が必要か

一昔前のドナルド クヌースの言葉[2]に次のようなものがある。「おそらく最も重要な教訓は、ソフトウェア開発は難しいということである。良いソフトウェアを作るには、ほかのどんな仕事よりもおそろしく高度な正確さが求められ、ほかの知的な仕事よりも長い期間、細心の注意が要求される。」クヌースの言葉は、現在にも当てはまる。我々の開発の枠組みは、現在および将来のアプリケーションが求める水準に達していない。我々の設計の枠組みが高水準の正確性を欠くため、正しいことの証明と検証が極めてコストのかかる非効率なものとなっている。その上、対象は常に変わり続けている。たとえば、業務ソフトウェアの難しい問題を解決しても、必ずしも組み込みソフトウェアの

[2] 「第 11 回 World Computer Congress（IFIP Congress 89）」での基調演説

難しい問題を解決することにはならない。さらに、今日有効な解決策は新たなニーズを生み、我々が経験したことのない新分野や別次元の複雑な明日のシステムへの広がりをもたらす。

以下に、新興の最重要分野で、ITを活用するのにボトルネックとなることが現在懸念される分野を挙げる。

- **エンドユーザが開発、保守できる複雑なアプリケーション**　ITがまたたく間に普及したことは、誰もが訓練を受けたプログラマになるか、あるいは誰もがプログラミングの成果を手にできるようになることを意味する。過去から現在まで、ITの研究開発は最初の方針、つまり「どんな職業を選んでも最後はプログラミングに行き着く…」という方針に沿って、大きく進んできた。しかし残念ながら、このアプローチには拡張性がない。そこで、経済的なプレッシャの元で技術的な議論を盛んに行い、研究者がいかにプログラミングをエンドユーザが理解できるものにするかを検討しなければならない。エンドユーザ・プログラミングに非常に大きな影響を及ぼす散発的な例に次のようなものがある。スプレッドシートからCADパッケージまで、あるいは回路シミュレータからワークフロー管理システムまで、ますます多くのアプリケーションがエンドユーザ向けのAPI（アプリケーション・プログラム・インタフェース）で構築されている。今日エンドユーザが行うプログラミングを今後も継続することに対する主な障壁は、エンドユーザがプログラムできるシステムの開発と保守にかかるコストが高いことである。エンドユーザがプログラムできるアプリケーションを開発し、進化させる技術が必要である。

- **多数の問題を同時に解決**　プログラミングは、投資効果、拡張性、柔軟性、可搬性、予測可能性、信頼性、規模の拡大性など多くの異なる要件を満たさなければならない。先に述べたように、現在の傾向（たとえば分野との融合が進むことや、物理システムを統合する役割、大規模化など）は、こうした多様な要求を同時に満たすシステムを構築する必要があることを示している。たとえば、業務アプリケーションは要求される機能を提供し、「かつ」安全で極めて信頼性の高いものでなければならない。同様に、組み込みシステム・アプリケーションは予測可能で信頼でき、安全で、セキュアで、規模が小さいことに加えて、様々な物理的な制約（力学、電力、サイズなど）を満たさなければならない。ある1つの視点で（通常は機能面から）ソフトウェア・システムを作ることは、モジュール間をつなぐ言語や階層構造を通して最近のプログラミングの枠組みで比較的よくサポートされている。だが、複数の視点から見ても信頼できるシステムを構築できる拡張性に富んだ解決策はまだ見つかっていない。

- **微粒子で構成された大規模分散アプリケーションの構築と理解**　IT の基本的な趨勢の1つに、ネットワーキングがコンピューティング・システムに及ぼす影響力の増大が挙げられる。こうしたシステム用のアプリケーションは、より複雑になるだけでなく、（人間を含む）環境と作用し合い、アプリケーション間でも相互に作用し合う。その相互作用は、要求が高く生活に重大な影響を及ぼすものである。こうした分散アプリケーションの多くは動的である。すなわち、環境の変化に合わせて常にその形を変え、相互作用のパターンを変え続ける。設計と実行のフレームワークは、これらのシステムの振る舞いと関連する開発の枠組みとを規定するような設計上の判断を含み、将来扱うべき大きな課題を形成する。

より一般的な表現を使えば、今日のソフトウェア設計には基礎となる科学がほとんど、あるいはまったくない。今のところソフトウェア設計は、一方で偉大なソフトウェア エンジニアが作る芸術品であり、他方ではその他大勢が作る駄作である。ソフトウェア設計が芸術品の域を出ない限り、前述した課題を解決する我々の能力は限られたままだろう。したがって、将来の研究開発の努力は、設計を厳密に科学的に捉えるように方向付けるものでなければならない。そしてそれが、将来の開発の枠組みの確固たる基礎となる。

重要な研究開発の課題

主要なアプリケーション分野と多数の固有の問題との間に大きな乖離があるため、すべての問題を解く「魔法の杖」と言うべき技術が存在しないのは当然である。それでも、新しい種類のアプリケーションが生んだ新しい課題と、複雑性の限界を押し広げようとする普遍的な要求とによって、今後10年間にIT研究開発コミュニティが扱うべき中核となる研究目的が定義される。以下に挙げる各項がそれである。

- **分野固有の言語による構成**　前述のとおり、ソフトウェア設計において分野固有の抽象化がますます重要になっている。その必要性は、次の大きな傾向によって導かれるものである。

  - 「さらなるコンピューティングの普及」は、分野のエンジニアリングと関連するソフトウェア設計とが密接に結合することを意味する。
  - 「エンドユーザが容易にプログラミングできることの必要性」は、分野固有の言語を使ってカスタマイズできる強力なアプリケーション・フレームワークが使用できることに帰着する。

残念なことに、現在の技術には限界があり、各分野固有のプログラミングの枠組みを広く導入することには抑制が働く。安全なアプリケーションの実現に必要となる、確保たる意味論の基盤や手頃なサポートツールを開発するのに高いコストがかかるためである。そこで、様々な抽象化レベルで短期間に「分野固有の言語の構成」ができる新しいインフラストラクチャを確立しなければならない。その言語には、宣言型の高級モデリング言語、異種多様なコンピュテーション モデルを表す言語、確保たる意味論をもつパターン言語などがある。

- **広く多様な分野での再利用を支援するツール インフラストラクチャ**　　自動化が進むと、各種の解析ツールや統合ツールを適用する必要が生じる。しかし、こうしたツール類は開発にも、習得にも高いコストがかかる。ツール開発の大部分が特定の分野や開発の枠組みとの関連で行われるため、再利用は今のところ極めて限られている。というのも、文書化されない前提条件や、分野の中だけで通じる暗黙の意味が多いためである。再利用を推進する方向に変えていくことをことさら正当化する必要性はないが、実践するとなると非常に難しい。なぜなら、個々のツールやプログラミングの枠組みで使われている多くの異なったサブ分野間に関係する意味と問題のすべてを把握し明確に表現しなければならないためである。その上、選択される表現形式は、サブ分野間の関係のモデル化をサポートするものでなければならず、かつ、これらの関係モデルを変換してツール間の「意味論的なインタフェース」を表現できるものにしなければならない。問題をより一層難しくしているのが、その結果を実際に使うことに大多数が賛成しなければ成功とは言えないことである。この分野を積極的に研究すれば文化を変えることになるだろう。そして、正確な意味論の仕様は、重荷になるどころか、安全なソフトウェア開発に必要な「健全性」とみなされるだろう。

- **多面的プログラミングの構成**　　前述のように、構成は拡張性の鍵となる。現代のプログラミング言語は、階層化されたモジュール構成をサポートしている。ただし、コンピューティングが進むであろう新しい重要な方面（組み込みシステム、大規模な分散アプリケーション、分野横断的な設計上の制約が多数存在するという特徴を持つその他のシステムなど）で、このサポートが不足している。これらの分野が進歩するには、まったく新しい構成の戦略を導入し、プログラミングの枠組みに変革を起こす必要がある。つまり、「多面的な構成」である。多面的な構成では、次のことが可能となる。

  1．プログラムの仕様を複数の視点から決定する。

2．異なる視点をサポートできる再利用可能なコンポーネントを開発する。
3．自動構成のメカニズム（プログラム・ウィービングと呼ばれる）によって全体を結合、生成する。

明らかに、多面的構成は、多様な抽象化のレベルを扱う開発の枠組み（手続き型のオブジェクト指向言語や宣言型モデリング言語など）と統合することが可能であり、それが求められている。

多面的なソフトウェア開発の大きな課題は、異なる視点に相互依存関係を与える「分野横断的な」制約の存在である。これが自動構成を極めて複雑なものにする。しかし、多面的な構成を自動化するためにリソースを投資することは賢明である。もう1つの方法、つまり手作業は、今日行われているような、単調でエラーが起こりやすく費用のかさむシステム統合に陥る可能性がある。

- **フレームワークの構成**　　動的で大規模な分散アプリケーションには、大きく異なる多様な課題がある。各ノードの振る舞いの複雑性は限られているが、相互に作用する多数のノードからなるシステム全体の状態や振る舞いは、限りなく複雑化する可能性がある。このようなシステムの問題は、特定のグローバルな振る舞いの設計の問題ではない。なぜなら振る舞いは完璧に正確であるがゆえに、監視あるいは意識すらされないことも多く、むしろ、動作環境全体の「安全な領域」での振る舞いに絡んでいるからである。この課題は、分散したコンポーネントの振る舞いやコンポーネント間の相互作用を制限する「設計と実行のフレームワーク」を導入することによって解決できる。この制約は、安全でない振る舞いを起こさないということを保証する要求レベルのものを選択しなければならない。今日、複雑な設計と実行のフレームワークを開発するのは非常に難しく、成熟するまでにはまだ長いプロセスが必要である。このプロセスの大部分を自動化し、各分野の特徴に合わせて高度に最適化されたフレームワークを自動合成できる技術が必要である。

結びの言葉

コンピュータの能力が飛躍的に向上し、ネットワークの帯域が大幅に拡大されたにもかかわらず、アプリケーション・ソフトウェアの設計と実装にはいまだにコストと時間がかかり、エラーが起こりやすい。ソフトウェアに対する要求は厳しさを増し、核となるソフトウェアの設計と実装の工芸品がソフトウェア業界全体で継続的に発見・発明し直されているが、このことがソフトウェアにかかる資金と努力の起因となっている。さらに、ハードウェア・アーキテクチャが一律ではなく、多様なOSとネットワーク・プラ

ットフォームが存在し、世界的に競争が激しいために、一からアプリケーション・ソフトウェアを構築し、以下の品質条件を満たすのは困難である。

- 「投資効果」　　ソフトウェアの取得と発展に要する総所有コスト（TOC）が受け容れがたい費用レベルにならないことを保証する。
- 「拡張性」　　新しい要求や市場に対応するため、ソフトウェアの迅速な更新、追加が継続的に行えるようにサポートする。
- 「柔軟性」　　ますます増えるマルチメディアのデータの種類、トラフィック・パターン、エンド・ツー・エンドのサービスの品質（QoS）の要求をサポートする。
- 「可搬性」　　多様なOS、プログラミング言語、コンパイラ、プラットフォームにアプリケーションを対応させるために必要となる作業を軽減する。
- 「予測可能性」と「効率」　　遅れに敏感なリアルタイムのアプリケーションには、低いレイテンシを提供し、帯域に敏感なアプリケーションには高い性能を提供し、無線リンクなどの狭い帯域幅のネットワークには使いやすさを提供する。
- 「信頼性」　　アプリケーションが堅牢でフォールト・トレラントで極めて利用価値が高いことを保証する。
- 「規模的拡大性」　　アプリケーションが多数のクライアントを同時に扱えるようにする。

このような品質のアプリケーションをタイムリーに開発できるかどうかは、もっぱらソフトウェア開発の枠組みにかかっている。開発の枠組みとアプリケーション分野との間には、重要な相互作用がある。以前はコンピューティング機器や通信機器というリソースが不足し、アプリケーションが比較的小規模だったため、開発の枠組みでは開発者というリソースよりもハードウェアとソフトウェアのリソースをいかに有効活用するかに注目を集まっていた。ハードウェアの性能が向上し、アプリケーションが複雑になるにつれ、コンピュテーション・モデルやプログラミング言語、関連ソフトウェア・ツールをアプリケーション分野に近づけ高めることが重要になってきた。ITが新しい分野で急速に広まり続ける今、再利用が可能なフレームワークとパターン、コンポーネント・ミドルウェア、モデル統合型コンピューティング、生成的プログラミングなどを基盤とするソフトウェア技術がさらに普及し、アプリケーション中心となり、最終的にはソフトウェアの生産性と品質の向上にもっと大きく役立つようになることを期待している。

実世界のためのソフトウェア （組み込み型システム）

ブレークアウト グループ サマリー

作成者: Robert Laddaga

パネリスト: 議長 Robert Laddaga
Paul Robertson、Scott Smolka、Mitch Kokar、David Stewart、Brian Williams

ブレークアウト・セッション参加者: Insup Lee、Kang Shin、John Reekie、Rajeev Alur、Calton Pu、Mitch Kokar

はじめに

将来ソフトウェア開発者が直面する最も困難な問題の一つとして、組み込み型システム用のソフトウェア開発に伴う問題が挙げられる。組み込み型システムとは、コンピュータ・プロセッサが物理的、化学的、生物学的にプロセスまたは装置を制御するシステムのことである。例えば、航空機、自動車、CDプレーヤー、携帯電話、原子炉、精油所、腕時計などが挙げられる。こうした多くの組み込み型システムには、数個、数百個、場合によっては数千というプロセッサが使用されている。今日、組み込み型システムの多くは比較的小さなサイズ（限られたメモリ容量、8ビット・プロセッサの使用など）になっているが、プログラミングの問題は残されている。近年は、メモリ容量と計算処理量が大幅に増強され、機能性が大幅に向上している。また機能性とハードウェア・リソースが強化されることで、複雑性も大いに増している。半導体業界においては、毎年地球上の人口よりも多い数のプロセッサが製造されており、その数は年々著しく増えている。そのプロセッサのほとんどが、従来からのなじみのあるパーソナル・コンピュータ、ワークステーション、サーバ、メインフレームシステム向けではなく、組み込み型アプリケーションで使用されている。

組み込み型システムで制御する実世界のプロセスでは、組み込み型プロセッサをプログラムする場合に、対処しなければならない多くの厳しい制約がある。以下にこのような制約の例を挙げる。

- リアルタイム要件。レイテンシやジッターの制限/抑制。
- 信号処理要件。打ち切り誤差、量子化雑音、数値的安定性など。
- 高有用性要件。信頼性、物理システムおよび計算システムの境界を越えた障害伝播/

回復など。

- 物理的な要件。消費電力、サイズなど。

上記の制約への対応が難しい理由は、これらの制約が組み込み型システム全体に影響しており、かつ組み込み型システムが問題全体に絡んでいるからである。このような実世界での問題は、物理プロセスを制御する場合には常に生じるもので、これら問題そのものが、組み込み型ソフトウェアの急激な複雑化の原因であるとは説明できない。次の2つの重要な傾向が、複雑化を招いている。(1) 組み込み型ソフトウェアを、複合システムの主要インテグレータとして使用する動きが強まったこと、(2) すばらしいけれども込み入った機能性を提供するために、組み込み型ソフトウェアに対する依存度が高まったこと。いずれの場合も、組み込み型ソフトウェアは他のものを代替している。前者の場合、複雑なシステムの主要インテグレータは、かつては開発者やユーザだった。今や、増加の一途をたどる統合作業の負担に対する扱いを、組み込み型ソフトウェアに頼っている。後者の場合、コンピュータ制御の電気機器システムが機械式連結や油圧連結に取って代わった。というのも、システムに設定された新しい機能と厳しいパフォーマンス要件を満たすのは、これらのシステムだけだからである。

ソフトウェア開発において、主に実施されている単純化の手法は、機能的、オブジェクト指向、コンポーネント・ベースの分解といったかたちの「抽象化」である。このような分解を通じてモジュラー化を広げていくことは、我々が書くソフトウェアの複雑性を効果的に削減することができる。モジュール性が損なわれると、異なるモジュールをコードする場合には別のモジュールの内部コードを考慮する必要が生じるため、複雑化する。このようなモジュール間に横断的な関心事がいくつも重なると、ソフトウェアの開発、検証、展開の作業は複雑性を増し、しばしばソフトウェアを完成させることが難しくなる。リアルタイム要件と、物理プロセスとの調整とにより、横断的な関心事が数多く引き起こされ、モジュール性をあきらめて破ることが発生する。これらの問題に対応するためには、関連分野のソフトウェア技術が必要である。中でも多階層分散リソース・マネジメント、多次元的な分解、自己適応ソフトウェア、許容力のある協調的なインタフェースが必要となる。

このレポートでは、実世界におけるアプリケーションと技術の両方の観点から、まず実世界でのソフトウェア開発の分野における目覚ましい進歩について検討する。次に、現時点では不可能な多くの課題のいくつかを取り上げて解説する。そして最後に、実世界でのソフトウェア開発における多数の有望な新技術について説明することにする。

成功を収めたこと

実世界のソフトウェアの研究開発では、以下の分野で目覚ましい進歩が見られている。

- **ツールの活用**　　ここ10年の間で、コード・ジェネレータ、モデリング・ツール、ソフトウェア検証およびモデル・チェッキングのツール、テスト・ケースを生成するツールが非常に完成度を高めた。このようなツールが相当数あるとはいうものの、自動車や航空電子工学の領域など、業務用組み込み型システムの実用が特定領域で広く普及し始めたのは、つい最近のことである。

- **リアルタイム・オペレーティング・システムの使用**　　リアルタイム・オペレーティング・システムを使用すると、スケジューリング、メモリ割り当て、割り込み処理、周辺機器ドライバーに再利用可能なサポートを実装し、組み込み型システム構築の作業量をかなり削減することができる。

- **メモリ/計算能力の制約範囲内での実装**　　メモリまたは計算能力が非常に制約を受ける場合でも、ソフトウェアをうまく開発することができる。このような場合には、問題の規模は、1人（または少数）の高度な技術を有するプログラマによって、ソリューションが得られる規模が適している。そのプログラマは、プロセッサ・サイクルまたはメモリ使用量を最小化するような精巧な短いコード・シーケンスを作成している。このようなシステムを開発できるようになったのは、決して小さな成果ではない。さらに、現在までに開発してきた組み込み型システムが、これらの制約をうまく処理できたのは、過去に財政的要因や特定のアプリケーションの特性がメモリまたは計算能力を制約してきたことに大きな要因がある。

- **耐障害性プロトコルの使用**　　TCPやSTCP（Secure TCP）、ネットワーク・プロトコルなど、有用性の高い耐障害性プロトコルが開発された。

- **シングル・プロセッサのスケジューリング**　　レート・モノトニック・スケジューリングや関連する分析手法などの技術を利用して、シングル・プロセッサのスケジューリングを非常に順調に行っている。

- **テストと検証**　　テストと検証の両方において著しい進歩がみられた。テスト用のツールおよび手法には、テスト・ジェネレータと共に用いられる、機能セットのクロス乗積の利用や精緻な統計的手法の利用などがある。小規模な組み込み型システムに限定されるが、生命およびセキュリティの面で重要なソフトウェアを検証する形式的枠組みの利用も著しく進展した。

- **固定ロケーションの自動化の実装**　　ロケーションが固定で、限られた範囲の活動ではあるが、オートメーション化された工場、化学プロセス、装置も、かなり順調に進んでいる。

勝利宣言するには何が必要か

これまでに述べたような技術進歩があるにも関わらず、現在必要とされ、現在の技術レベルを超えている重要な課題と機能を次に挙げる。

- **柔軟性のある高度なセンシング**　　組み込み型コンピューティングの究極の目的は、物理デバイスに知能を組み込むことにある。組み込まれた知能の重要な要素は、高レベルなセンシング機能で、実世界に存在するオブジェクトの行動と判断を決定する問題に関する、画像、音声信号、その他のセンサーデータを理解できる能力を指す。我々は、インテリジェント・プロセッシングと、シグナル・プロセッシングとを区別している。シグナル・プロセッシングとは、ノイズのある環境から低レベルでの信号抽出を行うに過ぎない。必要とするのは、処理する信号情報から、オブジェクトとその複雑な相互関係に関する情報を、現在と未来について、推測する能力である。また、関連する課題として、複数のセンサーおよびデータ・ソースからの情報を、リアルタイムで整合性のとれた世界像として融合させられない点が挙げられる。

- **制御と自動化**　　実世界の中を動き回るシステムを自動化する方法は、まだ見つかっていない。未知のコンテキストにおけるナビゲーションと堅牢な振る舞いもまた、難題である。同様に、自動コントローラ間、もしくは自動コントローラと人間による制御の間で、システムまたはプロセスの制御をシフトする必要がある場合に円滑に移行させる方法もまだ見つかっていない。制御のシフトに関するこれらの問題は、軍事行動や商業活動における共同提携を非常に難しくさせている。

- **容易に信用できる組み込み型システム**　　現在、生命およびセキュリティの面で重要なソフトウェアに対して受け入れられている認証プロセスはとてつもなく高コストだが、これらのプロセスがなければ、システムは非常に脆弱かつ複雑で信頼できない。組み込み型システムを「不可視型、かつ超安定型」にすることで、単なる信頼の域を超え、制御ソフトウェアを人々にまったく気づかれないというところまで到達させる必要がある。なぜなら、システムはまさに「正しいことを行う」だけのシステムになるからである。

- **変化する要件/環境への適応性**　　最後に、環境の変化や、さまざまな用途に適応できるシステムを作成する方法はまだ見つかっていない。

重要な研究開発の課題

今日、サポートできないアプリケーションおよびサービスは、基礎となる技術の不足によるものである。重要な作業を行えない、また前節で挙げた機能を提供できない理由は、以下のような技術がまだ開発されていなかったり、一般的に認められていなかったり、広範囲に展開されていないためである。

- **予測不能性とモデル選択**　　予測不能な世界に関して、正確かつ効率的に説明および計算するのは、至難の業である。さらに、モデルのパラメータを効率的に見積もるのが困難である上に、最初はどのモデルが正しいのか判断がつかない場合さえ頻繁にある。したがって、データを使用してモデルに関する決定を下してから、パラメータを見積もるか、もしくはこの両方の作業を同時に行う必要がある。

- **システム・ハードウェア/ソフトウェア共同設計**　　シーケンシャル・システム設計プロセスはソリューション代替案を過剰に制約するため、工学的トレードオフの品質は常に、あまり芳しくない。たとえば、組み込みシステムのハードウェアを先に設計した場合、ソフトウェア設計段階のみで明らかになるシステム内の選択肢や、これらの選択肢に基づくソフトウェア機能については、明らかにすることができない。このため、信頼できるシステム・ハードウェア/ソフトウェアの共同設計機能を開発する必要がある。

- **大規模な統合された組み込み型システムのテストと検証**　　密に統合された物理および計算システムに関するシステム理論がないため、大規模な統合された組み込み型システムを効率的にテストおよび検証することは不可能である。既存の理論は不完全であり、重要な振る舞いを予測することができない。さらに、検証のための証明はあまりにも大きすぎて、計算処理が困難である。また、テスト・ケースの空間が大きすぎて、実行可能ではない。検証またはテストされたシステムの構成ができ、サブシステムで対処された制約が構成システムでも確実に対処されている必要がある。テストが必要な場合は、（保証に加えて）適度なテストのみが要求されるべきである。

- **統合システムにおけるサービスの質(QoS)に関する系統的な制約への対処**　　統

合システムにおいて、系統的なQoS制約（リソース使用の制限、スケジューリングやタイミングに関する問題など）を満たすことは重要である。

- **統合におけるモデルの使用**　　システム構築のために、数学モデルのような実質的なモデルは使用できるが、モデルを使用してコードで複数のシステムやモジュールを相互に統合することは困難である。

- **既知の方法の自動化**　　メモリまたは計算の厳しい制約があるシステムは手動では構築できるが、そのプロセスを自動化する方法はまだない。自動化されれば、ツールが解決すべき問題の高レベルの仕様を改善することによって、この種のシステムを自動的に生成することができる。

- **動的な分散リソース割り当て**　　集中リソース割り当ては実行できるが、分散リソース割り当てに関しては、技術が不足している。分散リソース・ベースのシステムは、部分最適化または異常な割り当てを行ったり、ライブロックまたはデッドロックの状態になったりする可能性がある。

- **障害管理**　　障害の診断は困難で、不確実、かつ高コストである。そのため、現在は効果的な障害管理を実施していない。障害管理は、多重ユニットや投票方式などの静的設計、ループにおける人間の介在、障害ユニットのシャットダウンや交換などを介して操作している。稼動中の組み込み型システムに事前対応型の診断と修理を行う効果的な手法が必要である。

- **コスト効率のよい苛酷な環境への対処**　　苛酷な環境への対処法として、これまでに効果的に適用された手段は、物質的および構造的な過剰設計および強化など、高コストなものばかりだった。困難な環境にコスト効率よく対処する手段として、システムを苛酷な環境に適応させ、結果として起こるエラーから修復、回復させるようなコンピュータ能力の使い方は、まだ調査されていない。

- **機能的冗長性のエンコードおよび測定**　　解決すべき主要な問題の一つとして、組み込み型システムの優れた設計を適切かつうまく再利用する方法が挙げられる。ソフトウェアそのものは、組み込み型システム設計全体のわずかな部分にすぎないため、まず再利用したい部分を厳密に決める必要がある。また再利用できるものは、ソフトウェア工芸品以外にもパターンやアーキテクチャ・スタイルにまでおよぶ必要がある。効果的な再利用においては、ソフトウェアの変更やカスタマイズも含まれるため、機能的冗長性を測定し、プログラムが機能的冗長性を計算できる記述を

提供するような効果的な方法が必要である。このような記述は、耐障害性システムの構築でも役立つだろう。この種のシステム構築では、サブシステム障害を克服できるように十分な機能的冗長性を備える必要がある。

研究戦略

前節で挙げた技術不足に対応するため、ひいては前々節で列挙したサービスおよびアプリケーションのニーズに対応するために、以下の技術の研究開発に集中的に取り組み、その技術を完成・移行させる必要がある。

- **モデル・ベースのソフトウェア開発**　組み込み型システムは、センサーとアクチュエータの複雑なネットワークを注意深く調整することで、苛酷な環境下でも堅牢に動作させることができる。将来的に組み込み型システムが複雑化する点を考慮すれば、きめ細かい調整は、概念的にも、ソフトウェア工学の取り組みとしても、ほぼ不可能な作業である。モデル・ベースのソフトウェア開発では、システムのモデルを使用して、システム要件を捕捉・追跡し、自動的にコードを生成し、半自動的にテストや正しさの証明を行う。また、モデルは生成されたコードに対する検証の証明やテスト・スイートの作成にも活用できる。

  モデル・ベースのソフトウェアが開発されれば、きめ細かい調整はほとんど必要なくなり、プログラマは物理システム内に隠れている状態変数の展開を読み込み、設定することができるようになる。たとえば、あるプログラムは、ハードウェアをアクチュエートし、センサーする詳細 （つまり、「コントローラ1に信号を送り、バルブ12を開く」、そして、「圧力と速度を調べてバルブ12が開いているかを確認する」）を指定するかわりに、「35%の信頼性で10.3秒を実現」と指定するようになる。つまり、モデル・ベースのプログラムは、対象とする状態の展開に関するより高レベルの仕様となる。モデル・ベースのプログラムを実行するには、インタープリタは、制御されたプラントのモデルを使用して、常にプラントの状態を観測して推定し、制御動作を生成してプラントを指定された状態に移行させる能力を持つ。

  モデル・ベースのソフトウェア開発研究では、対象とする状態の展開とプラントの振る舞いを指定するためのより表現力の高い言語や、きめ細かい調整を全面的に実行するための自動化された実行方法を目指している。以下は、モデル・ベースのソフトウェア開発の研究課題における重要事項である。

  ・ モデルとコードの間にある整合性のギャップを埋める

- コードの構造設計特性を保持する
- 非形式的な要件を形式的な要件に変換する
- 要件を実装にトレースする
- 異なったモデルから形成されたサブモデルを統合する
- 非機能的な局面をサポートしている形式的枠組みを充実させる
- より効率的なテストを実施する
- 分散組み込み型システムのモデルを取り込む
- 予測不能性をモデル化し利用する
- 自己適応モデルを理解し構築する
- 以下の目的のために、組み込み言語をシームレスに拡張する
  - 組み込み環境の優れたモデルを受け入れる
  - プログラムの役割を命令から勧告に転換する
- 次のような高速オンライン推論とのインタラクションを管理する
  - 状態推定
  - 環境の再構成
  - プラニングおよびスケジューリング
  - 不連続イベントの制御と連続的な制御
- コンパイル時間と実行時間の各タスク間の調整を自動的に区分化する
- モデル化に関する一連の優れた形式的枠組みを取り入れて推論するための枠組みを使用する

組み込み型ソフトウェアのモデル・ベース開発は、いくつかの方法により、自律性に対する包括的なサポートも提供している。たとえば、プログラマと環境との振る舞いの単純なモデルを提供し､振る舞いに関する推論を言語のインタープリタ/コンパイラにわたすことで、自律性に対するプログラミングが簡素化される。また、系統的に、一連の実現可能なインタラクションと応答を幅広く考慮し、新規イベントにオンラインで応答し、証明できる正確なアルゴリズムを用いることにより、自律性がより堅牢になる。さらにプログラマが制御プログラム内で好ましいレベルの制御権を委任できるようになるため、自律性のレベルが調節可能になる。

モデル・ベースのソフトウェア開発の主要なメリットは、以下のとおりである。

- 合理化されたテスト、バグの早期発見、強力な検証手法により、システム開発および認証のコストが低減する。
- 組み込み型システムの安定性、堅牢性が向上する。
- 分かりやすさ、信頼性、認証が改善されるため、組み込み型フトウェアに対す

る信頼が高まる。

- **モデルの選択と推定**　　モデルの選択と推定に関する研究の目的は、センサーデータに対して、モデルを適用して高レベルな推測をサポートするアルゴリズムと方法論を形成することにある。その研究内容は、モデルのパラメータ推定と代替モデルとの比較とを同時に行う手法である。非常に割り切った世界観から見ると、科学者たちは実験を通して現象に適用できるパラメトリック・モデルを決定し、技術者たちはこれらのモデルを利用してシステムを構築し、環境に合わせてモデルのパラメータを調整する。つまり、科学者も技術者もパラメータ設定についてと同じ位、モデルの正確性についても推測し、これらを同時に行う必要がある。環境に関する重大な情報を得るために、センサーデータを解釈する工学的作業においては、特にその傾向が強くなる。モデルの選択と推定は、以下のような場合に役立つ。

  - 情報の融合。広い多次元情報空間や、分散した、相反する可能性のある情報源を統合するのに役立つ
  - 組み込み型システムの振る舞いに対する理解。たとえば、初期状態の検出によりマスク状態を検出したり、隠れ状態を検出したりする。
  - 細密にネットワーク化されたシステムの開発の効率化 （比較的小さなコンポーネントが極めて多数存在し、それらがすべてネットワーク化されたシステム）。
  - 適応性の高い障害管理の実現。適応性の高いモデルとパラメータを選択する。

  以下は、モデルの選択と推定の研究計画における重要な項目である。
  - 現実的な依存関係に関する仮説に加えて、扱いやすいコンピュテーションができる近似および最適化の手法。
  - 大量の分散されたパラメータ空間全体にわたるパフォーマンスの推定。
  - さまざまな表現による複数のモデルの統合。モデルには、制約、ロジック、ベイジアン・ネット、隠れマルコフ・モデル、常微分方程式などがある。
  - モデルの選択と推定に関する方法を、組み込み言語にシームレスに織りこむ方法。

- **動的なプラニングおよびスケジューリングに使用されるドメイン特有の言語**
  過去数十年にわたる組み込み型ソフトウェア開発の経験を通して、静的なプランやスケジュールは、実世界で予想外の推移や変化に一度直面すると、無効となってしまうことが明らかになった。その結果、使用中の状況変化に適応できるプランのみが有効であることが明確になった。人工知能もオペレーションズ・リサーチも、プ

ラニングおよびスケジューリングのアルゴリズムと表現の研究に貢献してきたが、これらの研究の大部分は、静的なプラニングおよびスケジューリングしかサポートしていない。動的なプラニングおよびスケジューリングでは、実行中の状況を判断してからプランを実施したり、必要に応じて事後対応的に再計画したりするようなプランを作成する。このような動的なプランを作成できるようなより優れた技術が必要である。ドメイン特有の言語または組み込み言語は、プログラマが計算された動作の高レベルな制約や条件を表現できるようにする。また、期待値を使用するプラニングや､多数の実行プランを追跡する機能がある｡オンラインのトラッキング、予測、実行、プランの立て直しなどを用いた、高度で動的な手法もある。以下は、動的なプラニングおよびスケジューリングに対するドメイン特有の言語の研究課題における重要事項である。

- 見込みのない実行プランのうち、どれを追跡するか決定する方法。すべての実行プランを追跡するのは難しいとはいえ、非常に見込みのある実行プランのみを追跡するだけでは予想外の壊滅的な障害を発見するには不十分である。
- 安全確保のため、追跡した実行プランの先々の影響を予測する方法。
- オンラインのモデル・チェッキング。
- 一時的な決定理論によるプラニングおよび実行を組み込み言語に織り込む方法。
- 事後対応的なタイムスケールにおいて、一時的な決定理論によるプラニングを実施する方法。
- プラニングと実行をオンラインで同時に行う方法。

一時的な決定理論による組み込み言語は、組み込み型システムの自動化を実現し、適応性と堅牢性を高める効果的な方法であり、主に、動的なプラニングおよびスケジューリングに対するサポートと、障害管理に対する柔軟で動的な回復戦略を可能とすることによって行われる。

- **組み込み型ソフトウェア開発の系統的なQoS推論**は､信頼性の高いコンポーネントとビルディング・ブロックを生成するためのボトムアップ・アプローチを提供し、自動化と認証を改善する上で、また低いレベルのコンポーネントの振る舞いを保証する上で役立つ。明示的なコード表現よりも、むしろコード本体への副作用など系統的なQoS制約に役立つ手法とツールの設計技術にポイントが置かれている。例として、ソースコードに明示的に表現されていない時間的および空間的制約が挙げられる。以下は、組み込み型ソフトウェア開発に対する系統的な QoS 推論の研究課題における重要事項である。

- システム/ソフトウェア協調設計。ソフトウェアの再設計および再構成など。
- 信頼性の高いデバイス・ドライバ。信頼性の低いハードウェアに対する高信頼性インタフェースなど。
- アスペクト指向ソフトウェアの開発。パフォーマンス・モニタリングなど。
- システムのQoS制約。時間、空間、不正確な計算、予測不能性、耐障害性の問題に対応。
- トレードオフ分析。
- 構成可能なハードウェア

- **自己適応型ソフトウェア**は高レベルなセンシング、適応、自動化に対応している。このソフトウェアは自らを監視し､変化や障害に対して自律的に修復と改善を行う。修復や改善は、フィードバック・コントロール・システムの振る舞いを利用して、自らのプログラムおよびサブシステムを変更または再合成することによって行われる。自己適応ソフトウェアの使用例は以下のとおりである。

  - 協調する車両のネットワーク
  - 空中、地上、海上、海中の輸送車両内部のハードウェアの再構成
  - 地上飛行および潜水艦に対する管理法への適応
  - 最適化またはシミュレーションに使用される数値コードへの適応
  - 高レベルなセンシングをしている間に生じた状況変更を追跡するための仮説への適応（例: 映像または言語）

  以下は、自己適応型ソフトウェアの研究課題における重要事項である。

  - 安定性を確保する方法の研究
  - 高レベルなシステムの目的が達成されているかを確認する方法の研究
  - システムのさまざまなクラスに対するモデルを表現し、追跡する方法の研究
  - プログラムを合成する方法の研究
  - 受け入れ可能なパフォーマンスを達成する方法の研究 （例: 十分な品質、十分な速度、QoS 測定基準の準拠など）
  - 自己適応型ソフトウェアのアーキテクチャおよび設計
  - センシングと適応に関するアイディアを取り入れた設計言語

結びの言葉

今日のコンピュテーショナル・サイクルのほとんどは、電気機械的な機器の制御に使用されている。これらの機器には、航空機、自動車のエンジン、カメラ、化学プラント、病院の患者監視装置、ミサイル、レーダー・システム、人工衛星、腕時計などがある。パーソナル・コンピュータは情報処理システムでもあると同時に、メイン・プロセッサ、メモリ、周辺装置の間の通信を制御する特別なチップを備えた組み込み型システムでもある。ディスク、プリンタ、オーディオ・カードなどの周辺装置にも、ソフトウェアが組み込みまれている。機械的な制御および連携がソフトウェアの制御に置き換えられている割合は高く、またその率は増加している。つまり、人の生活と経済の本質的機能は、ますます組み込み型ソフトウェアの質とコストに依存するようになっているのである。使用している機器がずっと正常に動くのか、また故障した場合は迅速かつコスト効率よく診断でき、修復可能かどうかを知っておく必要がある。

毎年、半導体業界では、さまざまなマイクロプロセッサやマイクロコントローラが製造されている。その種類は、4ビットから64ビット、また各種の特別なサイズのものにまでおよぶ。数年前、組み込み型システム業界は、マイクロプロセッサのバルクを4ビット・プロセッサから8ビット・プロセッサに移行した。現在はバルクを16ビット・プロセッサに移行しようとしているところであり、今後は急速に32ビット・プロセッサへと移行することが予想されている。これらのコントローラのメモリ・サイズも、同じような速さでおおよそ倍増している。最も普及しているプロセッサの容量が増加することにより、組み込み型ソフトウェア・アプリケーションの複雑性は、いとも簡単に、指数関数的に増大していく。手頃な価格の新しいプロセッサとメモリ・リソースを利用して、役割の数を増やして機能性を追加すること何回も行う。しかし、アセンブラ・プログラミング言語のハンド・コーディングや、手動による動作保証の検証という古い方法では、組み込み型アプリケーションの複雑性も数もスケールアップすることはできない。したがって、国家安全、そして経済的観点から考えても、このようなアプリケーションを構築するための新しい技術が必要不可欠である。

これまでに提唱した技術的研究のほとんどは、設計者やプログラマが考慮すべき抽象レベルを引き上げる手法を含んでいる。これこそが、指数関数的に複雑化する問題に対処できる唯一の方法である。しかし、組み込み型ソフトウェアの複雑性の原因は、実は部分的には抽象化の境界を越える相互依存性が高すぎるため、うまく分離された抽象層を構築できない点にある。これが、組み込み型システムの開発者たちが何十年も抽象化の使用を拒んできた一因である。ここで説明した有望な研究戦略には、抽象化の境界を横断する複雑性を管理するさまざまな方法が含まれている。こうした戦略を採用すること

により、実世界における組み込み型システムを効果的に構築し、保証していくことができる。

ネットワーク中心の大規模システム

ブレークアウト グループ サマリー

作成者: Rick Schantz、Doug Schmidt

2002年1月16日

パネリスト: 議長 Rick Schantz
Dave Sharp、Mike Masters、Prem Devanbu、Priya Narasimhan

ブレークアウト セッション参加者: Sally Howe、Kane Kim、Cordell Green、Gary Daugherty、Thuc Hoang、Joe Loyall、Betty Cheng、Jim Hugunin、Martin Rinard、Joseph Cross、Doug Schmidt

はじめに

次世代の商用および軍用アプリケーションは、大規模ネットワーク中心の構成で稼動することになるだろう。このネットワーク中心の構成では、多くのリモート・センサーから情報がインプットされ、地理的に分散したオペレータが収集された情報へアクセスし、リモートへ結果を出すのを制御できるようになる。人が介在することで高コストになったり、人の対応では非常に遅れてしまったりするような状況では、システムの稼動中に発生する数々の不測の事態に自律的かつ柔軟に対処していく必要がある。さらに、これらのシステムのネットワーク化が進み、ライフサイクルの長い「システムから成るシステム」を形成するようになっている。「システムから成るシステム」の大部分は、押し付けがましくなく、自律的に稼動し、オペレータに不必要な詳細情報を与えず(ただし、非常時に対応できるよう常に評価する)、同時に、以前は実現不可能だった速さで、ミッションクリティカルな情報に応答、処理する必要がある。以下に、このようなシステムの例を挙げる （ただし、これらに限定されるものではない)。

- 都市圏の交通制御システム。何千台もの車両からのセンサーデータを処理する
- 無人航空機群の調整
- 戦域レベルの戦闘管理に対する指揮および制御のシステム
- サプライチェーンマネジメント
- 科学データの共同解析
- 家庭内電力管理
- 医療提供システムの統合

- テロリスト追跡および識別システム

この種のシステムでは、実現可能な物理システム構成や作業負荷の構成のすべてを直感的に数え上げることは、概算ですら難しい。

増加しつつあるこうした大規模なネットワーク中心のシステムには、ネットワーク/バス相互接続、ローカルおよびリモートの終端システム、ソフトウェアの多重レイヤなど、相互に依存する多数のレベルがある。これらのシステムに望ましい性質としては、時間、情報量、正確性、信頼性、同期などの機能に関わるアプリケーションの操作特性の予測可能性、制御性、適応性が挙げられる。「システムからなるシステム」では、多くの相互接続されたパーツが動的に相互作用するため、これらの性質はすべて非常に不安定な状態になっている。多くの場合、これらのパーツは細かいパーツから同様の方法で組み立てられており、この種の複合システムをまったくの白紙から開発することは「理論的」には可能だが、経済的および組織的な制約、ますます複雑化している要求や競争プレッシャがあり、現実的に不可能である。

多くの競合する設計要求や実行時のサービスの品質（QoS）要求に対応するには、包括的なソフトウェア方法論と設計時／実行時環境への持続的な政府の研究/開発投資を受けて、再利用コンポーネントから、大規模で複合的な相互運用アプリケーションを確実に構成できるようにする必要がある。またコンポーネント自体も、パッケージ化される環境に細かく対応する必要がある。究極的には、異なったメーカーで、異なった製造時期に、別々に構築されたコンポーネントを組み立てて、要件や環境条件に合わせてカスタマイズして完全なシステムを構築することが望まれる。さらに長期的には、完全なシステムそれぞれ一つ一つは、より大規模な「システムからなるシステム」に組み込まれるコンポーネントとなることも考えられる。この取り組みの複雑性を考えると、これらのシステムを何層にもわたり構成および再構成するためのさまざまなツールや手法が必要である。そうしたシステムを構築できれば、基本的な設計と実装に変更を加えることなく、システムをよりグローバルな規模の多種多様な状況に適応させることができる。

今日の我々の位置付け

過去10年間、IT 研究者、実践者、エンドユーザは、ハードウェア（CPUやストレージ・デバイスなど）やネットワーク機器（IPルータなど）が日用品なみの相場で使えるようになった恩恵を受けてきた。さらに最近では、プログラミング言語（Java、C++ など）や実行環境（POSIX、Java仮想マシンなど）、新しいミドルウェア（CORBA、Enterprise Java Beans、.NETなど）が成熟の域に達し、多くのソフトウェア・コンポーネントとア

ーキテクチャ層の商品化が進められている。このようなソフトウェア製品の質は、概してハードウェアに遅れをとっている。また、アプリケーションへの要求が増々複雑化するにつれ、ミドルウェアは、一層、多面的になってきたと考えられている。そのため、実用システムの構築に必要な層に跨る完成度と機能にばらつきが生じて来ている。にもかかわらず、最近の改善されたフレームワーク、パターン、設計ツール、開発プロセスでは、市販（COTS）のネットワーク・ソフトウェアが開発され、統合され、増加する大規模な実世界アプリケーションで使用できるようにするために、必要な情報が外部から隠蔽されるようになってきた。実世界アプリケーションの例としては、eコマースWebサイト、家庭用電化製品、電子工学ミッション・コンピューティング、熱間圧延機、指令および制御のシステム、バックボーン・ルータ、高速ネットワーク・スイッチなどが挙げられる。

これまで10年間以上にわたり、ネットワーク・システム向けソフトウェアの開発に伴う多くの複雑性を軽減するために、さまざまな技術が開発されてきた。これらの技術により、システム・ソフトウェアという新しいカテゴリーが、オペレーティング・システム、プログラミング言語、ネットワーキング、データベースといった前世代のおなじみの各産物に加えられた。中でも、成果を上げた新しい技術のうちのいくつかは、ミドルウェアの分野でみられる。ミドルウェアとは、アプリケーションと、基盤となるオペレーティング・システム、ネットワーク・プロトコル・スタック、ハードウェアとの間にあるシステム・ソフトウェアである。ミドルウェアの主要な役割は以下のとおりである。

- アプリケーションのパーツの接続方法や相互運用方法を調整するために、アプリケーション・プログラムと、低レベルのハードウェアおよびソフトウェア・インフラストラクチャとの間にあるギャップを機能的に埋める。
- 複数の技術サプライヤーにより開発されたコンポーネントの統合を実現し、かつ単純化する。
- 機能性とパターンに対して、共通に再利用できるようにアクセス可能性を提供する。機能性とパターンとは、以前はアプリケーションに直接的に備えられていたが、実際にはアプリケーションから独立しており、新しいアプリケーションごとに別々に開発する必要はないものである。

ミドルウェア・アーキテクチャは、比較的自律性が高いソフトウェア・オブジェクトにより構成されている。これらのオブジェクトは、広範なネットワークに分散されたり、一緒に配置されたりすることもできる。クライアントは、インタラクションを実行するためにターゲット・オブジェクト上でオペレーションを呼び出し、アプリケーションの目的を達成するために必要な機能を呼び出す。ミドルウェアは適正に実装されると、以

下の場合に役立つ。

- ソフトウェア開発者に、低レベルで冗長でうんざりする、エラーの起こりやすいプラットフォームの詳細情報（例えばソケット・レベルのネットワーク・プログラミングなど）は与えない。

- 過去の開発に関する専門知識を活用し、フレームワークを用途に応じて手動で構築するのではなく、再利用フレームワークにキーパターンの実装を取り込むことによって、ソフトウェアのライフサイクル・コストを償却する。

- 分散システムや組み込み型システムの開発を簡素化するために、アプリケーション要件に近い、一連の一貫性がある高レベルなネットワーク指向の抽象化を提供する。

- ネットワーク化された環境で効果的に稼動するのに必要だと証明されたロギングやセキュリティなどのような、開発者指向の再利用サービスを豊富に提供する。

ミドルウェア領域における注目すべき成功事例は、以下のとおりである。

- Java 2 Enterprise Edition(J2EE)、CORBA、.NET 主流のITコミュニティに先進的なソフトウェア工学の機能をもたらし、さまざまなレベルのミドルウェアを開発プロセス全体の一環として連携している。ただし、パフォーマンス・クリティカルな組み込みソリューションに対する部分的なサポートのみに限られている。

- Akamai など ミドルウェア・サービスの方式を実現可能なビジネスとして合法化した。ただし、独自のものであり、オープンでない、ユーザではプログラムできないようなソリューションを使用している。

- Napster 非常に優れたシステムを迅速に（数週または数ヶ月で）開発できるのをはじめにとして、高機能な市販（COTS）のミドルウェア・インフラストラクチャの持つ能力を実証した。ただし、システム・ライフサイクルやソフトウェア工学の実践面などは、あまり考慮されていなく、一つの例にすぎない。

- WWW ワールド・ワイド・ウェブのミドルウェア/標準は、個別に開発されたブラウザとWebページを簡単に接続できる。ただし、エンド・ツー・エンドのサー

ビスの質を高めようとするシステム工学や配慮がなかったため、「World Wide Wait」とも呼ばれている。

- Global Grid　　科学者や高性能コンピューティング研究者が、世界の気候変動モデリングなど大きな難題に関する共同研究を行うことができる。ただし、使用しているアーキテクチャとツールは、メインストリームITの市販ミドルウェアと提携するものではない。

勝利宣言するには何が必要か

ここまでで挙げた成功事例は、完全性という観点から見た場合、部分的に成功していない点がある。例えば、最近のものでは、デンバー空港の開設時に航空交通管制で異常が発生した。軍用システムが統合されていなかったため、照準が狂う事態となり、無数の小規模で可視性の低いシステムが停止したり、適切に機能しない状態で戦場に配備されたりもした。より一般的なレベルでは、コンピュータ間や、コンピュータと物理デバイスの間の接続性は、接続性オプションと同様に、依然として急増している。このため、ネットワーク中心のシステムの規模を拡大し、精度を高めるべきだという社会的なニーズが高まっている。こうしたニーズを満たせば、拡大した接続性を活用して、集団およびグループのやり取りや振る舞いをより一層行えるようにできる。システムが発展し、今後も発展を続ければ、複雑化もさらに進む。そのため、アプリケーション・プログラミングを、分散とスケールという複雑な問題から（先進的ソフトウェア工学の実践とミドルウェア・ソリューションという形で）、切り離しておく必要がある。また、他にも米国航空交通管制システムや電力供給網などの国家規模のシステムは、不可避的に増加し、さまざまな組織が、まだ標準とはなっていない高レベルな共通プラットフォームと技術開発の枠組み上に、共通したソリューションを構築することによって、開発されるだろう。

過去数十年間にわたって技術の進展はあったが、再現可能、予測可能、コスト効率的な方式で大規模なネットワーク中心のシステムが構築、開発、検証、稼動し、改善され得ることが可能な、成熟した工学的法則やソリューション、確立した協定はまだない。つまり、我々は「複雑性」という限界点を目の当たりにしているのである。この限界点こそが、大規模なネットワーク中心のシステム開発の成功を妨げている。この複雑性の限界の一因に寄与している特有の複雑性は、以下のとおりである。

- シームレスなエンド・ツー・エンド・ソリューションを提供するために、適切な規模のプラットフォームを区別しなければならない

- 異なった形態のコンポーネントを、シームレスに統合する必要がある
- ほとんどの障害は、分散コンポーネントのサブセットに影響を与える、部分的なものである
- 実行環境と構成が動的に変化している
- 大規模システムは、アップグレードの間さえも絶え間なく稼動する必要がある
- エンド･ツー・エンドの特性は、時間もリソースも制約された環境の中で満たされる必要がある
- システム全体のサービスの質（QoS）を維持することが期待されている

これらの複雑性の問題に対応するためには、ミドルウェア指向のソリューションと工学的法則を、一般的に利用できる新しいネットワーク中心のソフトウェア・インフラストラクチャの一環として、構築、展開する必要がある。このようなインフラストラクチャは、多くの異なるタイプの大規模システムを効果的に開発するために必要である。

重要な研究開発の課題

これまでに概要を示した特有な複雑性を緩和する上で不可欠なのは、ネットワーク管理、データ管理、分散オペレーティング・システム、オブジェクト指向のプログラミング言語において、従来から見られる概念を統合し、拡張することである。すなわち、ミドルウェアの再利用という成果が上がれば、複合ネットワーク中心のシステムの開発と展開が著しく単純化するであろう。以下は、この成果の達成に伴う研究開発の重要な課題である。

- **単なるコンポーネント・レベルのQoSにとどまらないエンド・ツー・エンドの QoS サポートに対する要求**　　この分野は、先進的ミドルウェアの進化において次なる「大きな波」となる。最近では、大規模なネットワーク中心のアプリケーションを効果的に開発するには、市販のインフラストラクチャとサービス・コンポーネントを利用するという認識が広がっている。さらに、成果物としての製品の有用性は、構成される各パーツから受け継いだ全体の特性に依存するところが大きくなっている。こうした環境は、予測可能性と柔軟性の高い、統合されたリソース管理戦略が必要となり、その構成要素内、および構成要素間の関係は、開発者にとって理解可能であり、ユーザに可視的であり、システム所有者に認証可能でなければならない。しかし、ミドルウェアが提供する接続性は容易であるにもかかわらず、統合システムの構築は、予測可能なレイテンシ/ジッター/スループット、規模の拡大性、信頼性、セキュリティといった非機能的なQoS特性を有効にカスタマイズする必要があるため、依然として困難である。これらの特性は、最も有用性の高い形として、エ

ンド・ツー・エンドに拡張し、次のものに対して適用できるような要素を備えるべきである。

・ ネットワーク基盤
・ プラットフォームのオペレーティング・システムとシステム・サービス
・ 開発に用いられるプログラミング・システム
・ アプリケーション自体
・ これらの構成要素のすべてを統合するミドルウェア

ミドルウェアによってエンド・ツー・エンドでのQoSサポートを推進する上では、次の2つの基本的な前提条件が基礎になっている。

1. さまざまな条件やコストのもとで、多様なレベルのサービスが提供可能でかつ望ましい
2. 目標とする結果を全般的に達成するには、ある特性のサービスのレベルを、別の特性のサービスのレベルと協調および/またはトレードオフする必要がある

- **可変性と制御の双方に対処する、適応的かつ反応型のソリューション**　「一か八か」のポイント・ソリューションは避けることが重要である。今日のシステムは概して、設計された目的のリソースをすべてタイムリーに受け取る限りは正常に機能するが、わずかでも異常が発生すると完全に故障する。システムの振る舞いは柔軟性に欠け、多くの場合エンドユーザや管理者が対応する必要がある。ハード障害や不定な待機状態を避け、必要なリソースを自動的に再び獲得するために「再構成する」か、リソースが利用不可能な場合には、「適切に質を落とす」か、いずれかが必要である。最適ではない条件下で再構成および操作する際は、「個別の振る舞い」と「集合の振る舞い」という2つの点に注目すべきである。さらに、再構成するのに必要な制御および管理のメカニズムに加えて、それらの相互運用性も求められる。これまでの相互運用性の問題では、データの相互運用性とコンポーネント間の呼び出しの相互運用性を中心に扱ってきた。エンド・ツー・エンドでの統合システムの動作全般を制御するメカニズムに重点を置く研究は、ほとんどなかった。個別に動作するコンポーネントの単なる集合体以上のものを実現するのであれば、データと呼び出しの相互運用性を補完するために、「制御の相互運用性」を検討する必要がある。相互運用可能な制御機能に対する要件は、まず、個別リソースで実現され、その後、ミドルウェア・ベースのマルチプラットフォームがリソース管理サービスを集約することにより、これらのリソースを許容可能なグローバルな動作として集約することで実現されるべきである。

次世代ネットワーク中心のアプリケーションに対する幅広いQoS要求に対応するために、ミドルウェアの適応性と反応性を高める必要がある。

適応的なミドルウェアとは、機能的およびQoS関連の特性が以下のいずれかで変更できるソフトウェアのことである。

- *静的*　フットプリントを減らし、特定プラットフォームに存在する機能を活用し、機能的サブセッティングを有効にし、ハードウェア/ソフトウェアのインフラストラクチャの依存度を最小化する。
- *動的*　変化する環境や要件に合わせて、コンポーネント相互接続、電力レベルや、CPU/ネットワーク帯域幅、レイテンシ/ジッター、信頼性に対するニーズなどを変化させ、システムレスポンスを最適化する。

ミッションクリティカルなシステムでは、適応的なミドルウェアは、厳しいエンド・ツー・エンドのQoS要件を満たしながら、確実に上記のような変更を行う必要がある。

その上さらに、反応型のミドルウェアは、提供する機能の自動検証や、これらの機能を最適化するための自動調整を可能にする。反応的な手法は、システムの内部組織を、構築の際に使用されたメカニズムと同様に、ミドルウェアとアプリケーション・プログラムに対して可視化かつ操作可能にし、実行時に検査および変更できるようにする。したがって、反応型ミドルウェアは、現行の状況に合わせたより高度な適応的な動作と、より動的な戦略をサポートする。すなわち、システム内、システム環境内、エンドユーザにより定義されたシステムのQoSポリシーの条件に基づいて、必要な適応化を自律的に実行することができる。

- **標準ミドルウェアの汎用化に向けて**　最近の傾向として、アプリケーション開発の取り組みの大半が、アドホック的な独自のミドルウェア代替品の構築や、欠如しているミドルウェアの機能性の追加などに向けられている。その結果として生まれるこれらのアドホック機能による構成は、実行不可能か、手が出ないほど高コストになる。再開発が続く理由の一つには、アドホック・ソリューションは開発者以外のものにとってほとんど不可視であるので、最小限のアドホック・ソリューションでまとめあげることがしばしば比較的容易であるということが挙げられる。残念ながら、この方法は開発の下流段階において多くのコストをもたらす。複合的で長い寿命のネットワーク中心のシステムにおいてはことさらである。

したがって、最も早急な課題は、QoS属性を含むミドルウェア・インタフェースの確立し、最終的には標準化することである。QoS情報に対する明確な理解は、次のことを実現するために重要である。

1. いかなる時点においてもユーザ要件を確認する。
2. その要件が満たされているか（満たすことができるか）どうかを判断する。

要件を統合することも重要である。よりグローバルな情報管理組織への対応を開始する意思決定、ポリシー、メカニズムを形成することが可能となるからである。これらの要件を満たすには、異なったシステムから構成されるシステムの全体にわたって使用されるアプリケーション・コンポーネントと、リソース管理戦略の両方の柔軟性が必要となる。これらのニーズに対応するために向かうべき方向性は、適応的な振る舞いを管理し、常にすべての要件を満たせるわけではないことを認識し、それでも予測可能で制御可能なエンド･ツー･エンドでの振る舞いを確保するという概念を介して得られる。

- **設置基盤の活用と拡張**　　上記で概要をまとめた研究開発の課題に加えて、実際的な問題もある。たとえば、それぞれ独自に発展し続けているネットワーク、オペレーティング・システム、セキュリティ、データ管理インフラストラクチャなどですでに機能しているさまざまなビルディング・ブロックに、インタフェースを取り込むことなどである。最終的には、次の2つのタイプのリソースについて考慮する必要がある。

  1. アプリケーション開発の一環として組み立てられるリソース。
  2. 現在、提供されているリソースと、現在利用可能な基盤の一環としてみなされるリソース。

  現在導入されているハードウェアおよびソフトウェア基盤の方向性を転換するのは、短期間では無理である。しかし、合理的な方法は、より高レベルの（ミドルウェア・ベースの）抽象化をしたサービスを提供することである。このアーキテクチャにより、新しいコンポーネントのプロパティは制御可能なアプリケーションに容易に組み込んだり、容易に相互に統合できるようになる。これによって、アプリケーション開発者が対応すべき低レベルの複雑化した問題は減少し、システム開発コストと所有コストが削減される。したがって、次世代ミドルウェアの目的は、単に優れたネットワークやセキュリティを個別に構築することではない。むしろ、この

ような機能を統合してアプリケーションに導入し、さまざまなQoS属性間のトレードオフにより適応的な振る舞いのモデルを実現することである。基盤となるシステム・コンポーネントが制御性の高いものへと進化するにつれて、適応制御の強化を支える実装のリファクタリングが期待できる。

研究戦略

上記で説明した研究開発の課題に対応する上で、次の3つの概念が中心となる。

- *コントラクトベース、適応的メタ・プログラミング*　　ユーザ要件、リソース要件、システム要件に関連する特定のアプリケーションやアプリケーション・ファミリーのために情報が収集される必要がある。多様な要件のもとで、最適な振る舞いはなにかをベースにして、複数のシステムの振る舞いが利用できなければならない。ユーザと根底となるシステム基盤とのコントラクトに対する原則は、各要件下の最適な振る舞いに関する情報によって規定される。これらのコントラクトは、特定レベルのサービスを保証する程度を指定する方法だけでなく、指定された振る舞いの適応的変化の可視性を改善する明確な高レベルの抽象化ミドルウェアを与える

- *適切な質の低下*　　システム監視とコントラクト強化が可能なメカニズムを開発する必要がある。加えて、アプリケーション・サービスは、アクティビティを決定する予め決められたコントラクトに従い、状況変化に応じて、適切に質を落とす（または増強する）ことができるフィードバック・ループが、提供される必要がある。ここでの最初の課題は、「複数の振る舞いが実行可能かつ望ましい」という概念を、開発者とユーザの心の中に確立することである。次なるステップは、現行のシステム条件に応じて、正確な振る舞いを効果的かつ効率的に提供するのに必要なミドルウェア・サポートを（より低レベルのネットワークおよびオペレーティング・システムを強化するメカニズムへの接続も含め）追加することである。

- *優先順序付けと現実世界に制約される負荷不変のパフォーマンス*　　システムによっては、物理的な制約に強く縛られており、QoSなどのコンピュータ資産に対する要求の柔軟性が非常に低い場合がある。狭義に定義されたエラー許容範囲を越えた要求を逸脱すると、システムの壊滅的な障害につながる場合もある。今後は、さまざまな負荷条件のもとで、これらの不変性に対応することが課題となる。多くの場合、いくつかのリソースへのアクセスを保証する一方で、他のリソースは正常な振る舞いを確保するために転用されることを意味する。通常、このようなコンポーネントの集合に対しては、コンポーネントの（個別の）視点に加えて、システムの

（集合的な）視点からリソース管理する必要がある。

コントラクトを満たして適切な質の低下を行い、いくつかのリソースを広範に、システムの限定された範囲で限定されたレベルで管理することは現状では可能だが、まだ研究開発の余地がある。これらの目的を達成するのに必要な研究戦略は、以下の 7つの分野に分類できる。

1. **個別のQoS要件**　　個別のQoSは、エンド・ツー・エンドでの単一ユーザまたはアプリケーションの視点からのQoSニーズに関連するメカニズムを開発することを行う。その仕様要件には、複数のコントラクト、交渉、領域固有性などがある。複数のコントラクトは、常に変化する要件を処理し、そして、アクティビティの一部をそれぞれ規定している単一の視点から観た幾つかのコントラクトを連携づけるために必要である。同じアプリケーションを実行している別々のユーザが、別々のQoS要件をもつ場合もあり、そのQoS要件は現行の構成に依存して強調するメリットとトレードオフが異なる。たとえ同じアプリケーションを実行している同一ユーザでも、実行する時間によって異なるQoS要件を持つ場合がある。すなわち現行の操作モードや他の外的要因に左右されるのである。このような動的な振る舞いを考慮に入れて、次世代の分散システムにシームレスに導入しなければならない。

   通常の交渉機能は、交渉で取り決めされた動作（交渉で取り決められるサービスとは対照的に）を始動し、制御するような快適なメカニズムを提供し、市販のミドルウェア・パッケージを利用できるようにするものである。このような交渉に基づく適応メカニズムがQoSの不可欠な部分になるようにするには、「ユーザ・フレンドリー」になることが最も効果的である。つまり、ユーザまたは管理者に、単に優先順位のリストを提供するよう要請することである。これはドメイン特有またはユーザ特有になる可能性が高い分野である。そのほか、個別アプリケーションにQoSをもたらす上で対応すべき課題には以下のものがある。

   - 分散したエンド・ツー・エンド・パス上の、多様なエンティティの中あるいは間のサービスに対する要求の変換。
   - 「あいまいな」環境における、適切なアプリケーション機能性とシステム・リソース・トレードオフの定義と選択の管理
   - 構成可能な範囲内での適切な振る舞いの維持

   変換は、複雑なネットワーク中心のシステムが層ごとに構築されている事実に対応している。階層化アーキテクチャのさまざまなレベルにおいて、ユーザ指向のQoS

は、低レベルの他のリソースに対する要求に変換される必要がある。課題は、ユーザ要件からシステム・サービスへのこの変換を達成する方法を見つけることである。最初はアプリケーションとミドルウェアの境界線からとりかかるのがよい。これはアプリケーション・リソースと、適切な分散システム・リソースとのマッチングの問題と緊密に関わる領域であるからである。システム・リソースが、異常または負荷のために大幅に変化する場合、QoS属性（適時性、精度、正確性など）間のトレードオフを（再）評価し、状況に応じてQoSの効果的なレベルを確保する。これらのトレードオフを適切な時に認識し、実行するようなメカニズムを開発する必要がある。最後に、アプリケーション中心のエンド・ツー・エンドの特性を維持しつつ、個別コンポーネントからシステムを効果的に構成する理論を開発し、理論を効率的に実装できるように具現化することも重要である。

2. **実行時の要件**　　システムのライフサイクルの視点からみると、QoS管理に対する決定は、設計時、構成/展開時、実行時に行われる。そのうち実行時の要件は、意思決定のタイムスケールが最も短く、共同で適切なソリューションを開発した経験が乏しい分野であるため、最もチャレンジングな課題である。また、先進的ミドルウェアの概念に最も関連のある分野でもある。この分野の研究対象は、アプリケーションとシステムの振る舞いを変更するために必要とする実行時の監視、フィードバック、移行メカニズムであり、それは、動的な再構成、質を落とした振る舞いの調整、オフラインの再コンパイルを介して行われる。この際の主な要件は、測定、レポート、制御、フィードバック、安定性である。これらの要件はそれぞれ個別のアプリケーションのみならず、それらを集めたシステムに対するエンド・ツー・エンドでのQoSをもたらす上で、重要な役割を果たす。実行時環境の主要部分は、永続的で高度に調整可能な測定と、共通ミドルウェア・サービスとしてのリソース状況サービスに重点を置いている。このサービスは、さまざまな時間で多様な粒度に適応し、実行時の適応化の用途に対して適切な抽象化と集合を行う。

   適切に質を落とすことを可能とする機能に加えて、この基礎的メカニズムは、アプリケーションの基本的な実装構造を変えることなく、予想されうる多様な振る舞いをサポートするような柔軟性を提供する可能性も秘めている。反応的な柔軟性は、設計時に予測される実行環境だけに代わって、展開時の実際の実行環境に適応するように、最新で実行時のバインディング・オプションを設定することによって、初期設計における決定の重要性を減少させる。さらに、これらのバインディングの変化を予測して、新しい振る舞いに適合するようにする。

3. **集合的な要件**　　この研究分野は、システム全体にわたる一連のリソースから必

要な情報を収集し、システム全体の目的を達成するようなリソース管理のメカニズムとポリシーを提供するといったシステムの視点を扱う。ミドルウェアそのものは、システム・レベルのリソースを直接（低レベルのリソース管理、および実施メカニズムが提供するインタフェース経由を除いて）管理することはできないが、連携のメカニズムとポリシーを提供し、個々のリソース・マネジャーをドメイン全体で一貫して統合することができる。このようなリソース管理については、意思決定プロセスを導き、方針決定を実行するためのメカニズムが機能するような政策が必要である。

以下に、特に関心の高い研究開発の分野を挙げる。

- 予約。リソースが予約され、一定レベルのサービスが保証される
- 許諾制御メカニズム。特定ユーザのシステム・リソースへのアクセスを許可または拒否する
- 適切なスケール、粒度、パフォーマンスを伴う実施メカニズム
- よく連携された戦略およびポリシー。さまざまな特性を最適化するように分散リソースを割り当てる。

この他にも、それぞれのアプリケーションが、ベストエフォート、条件的、もしくは統計的なレベル、のいずれのサービスを受け取るかといったさまざまなレベルのQoSを実現する方針決定が必要である。プロパティ構成の管理は、コンポーネント・ベースのアプリケーションに対して個別のQoSを与える上で不可欠である。また、集合的なケース、特にドメイン内およびドメイン全体にわたって階層化されたリソース管理では、さらに重要性を増す。

4. **統合の要件**　　統合の要件は、OS、ネットワーク管理、セキュリティ、データ管理などシステム構築に使用される主要ビルディング・ブロックとのインタフェースを開発する必要性に対応する。これらの分野の多くは、部分的に個別の視点からQoSソリューションを行っている。今日の課題は、これらの部分的な成果を共通のインタフェースに統合しなければならないことであり、そうすれば、ユーザとアプリケーション開発者は情報を交換し、どの視点がどの状況下で有力なのかを判断し、属性の適切な構成を得ることができる境界を跨ったトレードオフ管理をサポートできる。現在、ミドルウェアと連動するオブジェクト指向ツールは、エンド・ツー・エンドでの構文的相互運用と、ネットワークとサブシステムにまたがる比較的シームレスな連携を提供している。しかし、管理QoSは存在しないため、これらのツールとミドルウェアはリソースが豊かなベストエフォートな環境でしか有用ではな

い。

統合された振る舞いに対するさまざまな要件に応えるために、この相互動作を規定している多様なトレードオフを持った多様なレベルの属性に対する要求を許可する高度なツール、そしてメカニズムが必要である。このシステムは、要求されたエンド・ツー・エンドでQoSを提供するために再構成されるか、そのレベルのサービスを提供できないことを示しておそらく別のQoSのサポートを提供するか、またはアプリケーション・レベルの適応化を開始するかのいずれかを行う。これらのすべてが適切に連動するために、多次元のQoS要求が、共通のフレームワークの範囲内で理解され、要求とサービスがそれぞれ関連するインタフェースで変換され、やりとりされる必要がある。高度な統合ミドルウェアは、この共通フレームワークを提供して、基礎となる機能の適切な構成を実現する。

5. **適応性の要件**　　次世代の情報環境における多くの高度な機能は、ユーザの期待に応え、需要と変化する環境のアンバランスを円滑にするような、適応的な振る舞いを必要とする。適応的な振る舞いは、上記の4つの分野の機能を適切に編成し、相互運用することで可能となる。必要とされる適応性には基本的に次の 2つのタイプがある。

   - アプリケーションの下位レベルの変化。リソース可用性の変化にかかわらず、要求されるサービス・レベルを維持するもの。
   - アプリケーション・レベルでの変化。現行の利用可能なレベルのサービスに対処するか、変化した環境下で新しいレベルを要求するもの。

   いずれの場合でも、システムは望ましいQoSを達成するために、リソースを再割り当てするか、戦略を変更するか（できるか）を判断する必要がある。アプリケーションは、実行している条件が変化した際に、QoS要求を変更できるように構築される必要がある。再構成のメカニズムを導入する際には、必要に応じて新しいレベルのQoSを実装できるように、個別と集合の双方の視点と、起こりうる競合に注意を払わなければならない。

   これらの目的を達成する取り組みの一部は、上記のように、収集された関連リソース情報を継続して収集し、瞬時に分析することである。それを補完する部分は、迅速に変化する要求とリソース可用性のプロファイルを扱うために必要なアルゴリズムと制御メカニズムを提供し、多様な環境に合わせて調整された種々のサービス戦略と方針とにこれらのメカニズムを構成することである。理想的には、

このような変化は、幅広い条件を扱う上で動的かつ柔軟であるべきであり、自動化された方法でインテリジェントに現れ、適合的なコンポーネントの構成から発生する複合的な問題が処理されるべきである。これらの機能を効果的で適応性の高いミドルウェアに統合するツールと方法論は、研究開発分野における最重要課題である。

6. **システム工学の方法論とツール**　　高度なミドルウェアは、それ自体では次世代の組み込み環境で想定される機能をもたらさない。複合的な分散コンピューティング・システムの構築に使用される環境が高度になるにつれて、システム工学の訓練とツールの状態も発展させる必要がある。この研究分野は、特に、差し迫ったシステム工学の方法とツールに対するニーズに対応し、高度なミドルウェア・ソリューションを補強する。以下のような課題がある。

   - *視点指向またはアスペクト指向のプログラミング手法*。システムが保有しなければならない属性の多様な投影や視点の分離（専門化と焦点）、構成（分離されたものを全体に網状にかみ合わせる）をサポートする。分離し、そして統合する能力と、多様な相互運用機能の実装は、変化する要件への適応をサポートするために必要である。
   - *設計時のツールとモデル*。システム導入後のコスト高の変更を避けるために、システム開発者が設計を理解するのを助ける（なお、前述のQoS決定に向けてのバインディングの遅延によって、部分的に解消されている）。
   - *インタラクティブな調整ツール*。システムの各部分がシームレスに連動する必要性に伴う課題を克服する。
   - *構成可能なツール*。個別コンポーネントを2つ以上組み合わせた結果のQoSを分析する。
   - *システム・パフォーマンス・モデル開発のためのモデリング・ツール*。試行錯誤に伴うコストを削減するために、リソース管理を監視し、理解するための補助的な（オンラインとオフラインの）手段である。
   - *デバッグ・ツール*。不可避な問題に対応する。

7. **信頼性、信用、検証、認証可能性**　　次世代の大規模なネットワーク中心のシステムで想定される、動的に変化する振る舞いは、これまでに構築・使用し、一定の信頼を置いていた現在のシステムとは大きくかけ離れている。したがって、システムを実際に展開する前に、適応的な振る舞いの的確な機能を綿密に検証し、大規模なシステムが現在の状況から評価して振る舞いを変更しようとすることを深く理解することに多くの努力を集中しなければならない。しかしそれ以前に、市販のコ

ンポーネントを使用する方法論と設計において重要因子となる十分な信頼性と信用という、長期にわたる課題が、まだ十分に解決しておらず、共通利用を達成できていなく、引き続き改善に努める必要がある。現行の戦略は、長期のライフサイクル、最小限の変更、網羅的なテスト・ケース分析を前提としたものだが、そのままでは次世代の厳しいQoS要件を伴う動的システムには明らかに不十分である。

結びの言葉

このITユビキタス、経済的激変、規制緩和、国際競争激化の時代には、高品質で柔軟性が高く、相互運用が可能な大規模ネットワーク中心のシステムを開発する上で、サイクル・タイム、労力レベル、複雑さを軽減させることが重要になっている。このタイプのシステムは、それぞれの用途に合わせてまったく初めから実装するのではなく、再利用ソフトウェア（ミドルウェア）・コンポーネント・サービスを使用して開発されるようになっている。ミドルウェアが考案されたのは、大規模ネットワーク中心のコンピューティング・システムのソフトウェア開発を単純化するため、そしてその時点で複雑な環境を使いこなせるごく少数のエキスパートだけでなく、より多くの開発者が手軽にこれらの機能を使えるようにするためである。複合的なシステム統合の要件は、難易度が高く、再利用できない「アプリケーションの視点」からも、必然的に通信および終端システム・リソース管理層の提供に関与する「ネットワークまたはホスト・オペレーティング・システムの視点」からも満足のいくものではなかった。
ミドルウェアは、過去10年間にわたって、特に異機種性と相互運用性に伴う問題を解決する一連のソフトウェア・サービス群として登場し、ネットワーク中心のアプリケーションを構築する環境や、分散リソースを効果的に管理する環境を改善する上で、多大な貢献してきた。結果として、研究者と実践者は、マルチレイヤ・アーキテクチャ（アプリケーション、ミドルウェア、ネットワーク、オペレーティング・システム・インフラストラクチャ）に向かう傾向にある。このアーキテクチャは、再利用コンポーネントからのアプリケーション構成に向かうものであり、アプリケーションがネットワークとオペレーティング・システムの抽象化の最上位レベルで直接開発され、より従来型のアーキテクチャとはかけ離れている。このミドルウェア中心のマルチレイヤ・アーキテクチャは、インターネットの出現とハードウェアおよびソフトウェアのコンポーネント化および必需品化により生まれた、ネットワーク中心の視点の導入に直接由来している。

初期の基本的なミドルウェアで成功したことで、もっと野心的な取り組みやこれらのミドルウェア指向アクティビティの範囲拡大に繋がった。そのため、現在では具体化したミドルウェア自体の多数の明確なレイヤを目にすることができる。結果として次の2項目の間に横たわる大きく高まりつつある課題と潜在的なソリューションに対する理解

を深めることができた。

- 複合的な分散アプリケーション要件
- 既存ネットワーク・システム、オペレーティング・システム、プログラミング言語のバンドルにより導入された単純なインフラストラクチャ

今日では、さらに複雑なシステムを構築する上で深刻な限界に直面している。たとえば、アプリケーションのQoS要件はますます厳しくなっている。また、安定したバックプレーンではなく、不安定で変化の激しいインターネットのように、あらゆる状況を横断的に統合することをさらに進めなければならないことが分かってきた。

問題の一つは、競争の環境がリソースと予測という点で常に変化していることである。もはやシステムを非常に高度な機能を実行するように設計し、変更を最小限に抑え、20年間のライフサイクルを全うさせることを期待するような贅沢さはできない。現実に我々は、異なる条件下で異なる振る舞いをするシステムを期待することを日常的に繰り返し、いつも通りに動作しないとクレームをつける。こうした変化は多くの問題をなげかける。たとえば、エンド・ツー・エンド指向の適応的なQoS、市販パーツから構築するシステムなどであり、これらの大半が新しい重要なミドルウェア・ベースの機能とサービスを備えた有望なソリューションを提供するものである。

このSDPワークショップでの短い時間では、過去と現在における多くの取り組みを要約し、展望を提供したにすぎない。だが、ミドルウェア技術が、現在大きなチャンスと未解決のやりがいのある問題を抱えた興味深い開発分野であることはお伝えできたと思う。このブレークアウト・グループ・サマリーは、SDPワークショップ参加者たちが、大規模ネットワーク中心のシステム向けのミドルウェアにおいて、将来の研究開発が向かうべき最も有望な方向性を示していると信じる一連の取り組みに関して、より詳細な議論を提供し、体系化したものである。最終的な研究開発における取り組みの目的は、以下のとおりである。

1. 多様化し、変化の激しい要件/環境に対応・適応することのできるネットワーク中心のシステムを、信頼性のある、再現可能な形で構築、構成する。
2. 社会のニーズに合った、それぞれ特定の領域に正確に適応するシステムを大量に、手頃な価格で構築・構成する。

この目的を達成するためには、技術的な課題だけでなく、教育的および過渡的な課題を克服する必要がある。そして、ハードウェアとソフトウェアのコンポーネントがミドル

ウェアを介して統合化される傾向がますます強まる中で、徐々にこのような環境に伴う非常に高度な複雑性を修得し、単純化させる必要がある。

参加者リスト

First Name	Last Name	所属
Rajeev	Alur	University of Pennsylvania
Don	Batory	University of Texas, Austin
Ira	Baxter	Semantic Designs
Barry	Boehm	University of Southern California
Richard	Buchness	Boeing
Betty	Cheng	Michigan State University
Karl	Crary	Carnegie-Mellon University
Joseph	Cross	Lockheed-Martin
Eric	Dashofy	University of California, Irvine
Gary	Daugherty	Rockwell Collins
Premkumar	Devanbu	University of California, Davis
Laura	Dillon	Michigan State University
Tzilla	Elrad	Illinois Institute of Technology
Kathi	Fisler	Worcester Polytechnic Institute
R.	Gamble	University of Tulsa
Susan y	Gerhart	Embry-Riddle Aeronautical Universit
Susan	Graham	University of California, Berkeley
Cordell	Green	Kestrel Institute
Paul	Hudak	Yale University
Jim	Hugunin	Xerox PARC
Daniel	Jackson	MIT
Ralph	Johnson	University of Illinois, Urbana-Champagne
Philip	Johnson	University of Hawaii
Samuel	Kamin	University of Illinois, Urbana-Champagne
Gabor	Karsai	Vanderbilt University
Gregor	Kiczales	University of British Columbia/Xerox
Kane	Kim	University of California, Irvine
Mieczyslaw	Kokar	Northeastern University
Alkis	Konstantellos	European Commission, IST
Fist Name	Last Name	所属
Shriram	Krishnamurthi	Brown University
Robert	Laddaga	MIT

James	Larus	Microsoft
Insup	Lee	University of Pennsylvania
Karl	Lieberherr	Northeastern University
Joe	Loyall	BBN
Tommy	McGuire	University of Texas, Austin
Marija	Mikic-Rakic	University of Southern California
Michael	Mislove	Tulane University
Priya	Narasimhan	Carnegie-Mellon University
Bob	Neches	University of Southern California
Dewayne	Perry	University of Texas, Austin
Calton	Pu	Georgia Tech
William	Pugh	University of Maryland, College Park
Joy	Reed	Armstrong Atlantic State University
John	Reekie	University of California, Berkeley
Steven	Reiss	Brown University
Robert	Riemenschneider	SRI International
Martin	Rinard	MIT
Paul	Robertson	Dynamic Object Language Labs
David	Sharp	Boeing
Kang	Shin	University of Michigan
Massoud	Sinai	Boeing
Douglas	Smith	Kestrel Institute
Scott	Smolka	Reactive Systems, Inc.
David	Stewart	Embedded Research Solutions
Kurt	Stirewalt	Michigan State University
Brian	Williams	MIT
政府関連招待者		
Abdullah	Aljabri	NASA
Frank	Anger	NSF
Fist Name	Last Name	所属
Daniel	Dvorak	NASA
Helen	Gigley	NCO/ITRD
Helen	Gill	NSF
Thuc	Hoang	DOE
Sally	Howe	NCO/ITRD

Michael	Masters	U.S. Naval Surface Warfare Center
Kenny	Meyer	NASA
Vijay	Raghavan	DARPA
Steven	Ray	NIST
Spencer	Rugaber	NSF
Doug	Schmidt	DARPA
Frank	Sledge	NCO/ITRD
プログラム委員		
Benjamin	Pierce	University of Pennsylvania
Adam	Porter	University of Maryland, College Park
Rick	Schantz	BBN
Charles	Simonyi	Microsoft
Janos	Sztipanovits	Vanderbilt University
Don	Winter	Boeing
ボランティア		
Aditya	Agrawal	Vanderbilt/ISIS
Mark	Briski	Vanderbilt/ISIS
Michele	Codd	Vanderbilt/ISIS
Brandon	Eames	Vanderbilt/ISIS
Jeff	Gray	Vanderbilt/ISIS
Jung-Min	Kim	U. Maryland
Lorene	Morgan	Vanderbilt/ISIS
Tal	Pasternak	Vanderbilt/ISIS
Jason	Scott	Vanderbilt/ISIS
Jonathan	Sprinkle	Vanderbilt/ISIS
Eric	Wohlstad	UC Davis